

COMMUNICATIONS

CACM.ACM.ORG

OF THE

ACM

06/2011 VOL.54 NO.6



Leslie Valiant

**ACM's A.M. Turing
Award Winner**

Scalable SQL

Advancing Home
Networking

10 Rules for Scalable
Performance

All the News
That's Fit for You

PageRank: Standing on the
Shoulders of Giants

Association for
Computing Machinery



8th ACM Conference on
Creativity & Cognition

<http://dilab.gatech.edu/ccc/index.html>

November 3-6, 2011

Georgia Institute of Technology,
Atlanta, Georgia, USA

The theme of Creativity & Cognition 2011 is Creativity and Technology. We seek to understand human creativity in its many manifestations, to design new interactive techniques and tools to augment and amplify human creativity, and to use computational media technologies to explore new creative processes and artifacts in all human endeavors ranging from the arts to science, from design to education. The conference features scholarly talks, artworks, tutorials, a graduate student symposium, and workshops.

Keynote Speakers:

Guy Claxton (Creativity and Learning)
Sara Diamond (Creativity and Visual Arts)
Atau Tanaka (Creativity and Music)

General Chair:

Ashok Goel

Program Co-Chairs:

Ashok Goel, Georgia Institute of Technology, USA
Fox Harrell, Massachusetts Institute of Technology, USA
Brian Magerko, Georgia Institute of Technology, USA
Yukari Nagai, Japan Advanced Institute of Science and Technology, Japan
Jane Prophet, Goldsmiths College, University of London, UK



THE ACM A. M. TURING AWARD

by the community ♦ from the community ♦ for the community



ACM, Intel and Google congratulate

LESLIE G. VALIANT

for transformative contributions to the theory of computation, including the theory of probably approximately correct (PAC) learning, the complexity of enumeration and of algebraic computation, and the theory of parallel and distributed computing.



"Leslie Valiant's research in the theory of computation has revolutionized both machine learning and artificial intelligence, making machines almost think. His approach invites comparison with Alan Turing himself—a novel formulation starting from a deep fundamental insight. Intel is pleased to support this year's ACM Turing Award."

Shekhar Borkar
Intel Fellow, Microprocessor Technology Lab
Intel Corporation

For more information see www.intel.com/research.



"Google joins in honoring Leslie Valiant for his profound impact on computer science research and his inspired innovations in machine learning, an area of growing importance in computing. We are pleased to sponsor this award, which motivates and recognizes the great advances in computing that together have had such a beneficial impact on the world."

Alfred Spector
Vice President, Research and Special Initiatives
Google Inc.

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



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

Departments

- 5 **Letter From ACM India Council Computing and India**
By P.J. Narayanan and Anand Deshpanda
-
- 6 **Letters To The Editor**
Why Concurrent Objects Are Recurrently Complicated
-
- 7 **In the Virtual Extension**
-
- 8 **BLOG@CACM**
Simple Design; Research vs. Teaching; and Quest to Learn
Daniel Reed writes about design and disruptive technologies. Mark Guzdial discusses the tension between teaching and research at U.S. universities. Judy Robertson argues in favor of educational video games in the classroom.
-
- 10 **CACM Online**
Say It With Video
By Scott E. Delman
-
- 27 **Calendar**
-
- 124 **Careers**

Last Byte

- 126 **Puzzled**
Solutions and Sources
By Peter Winkler
-
- 128 **Q&A**
A Lifelong Learner
By Leah Hoffmann

News

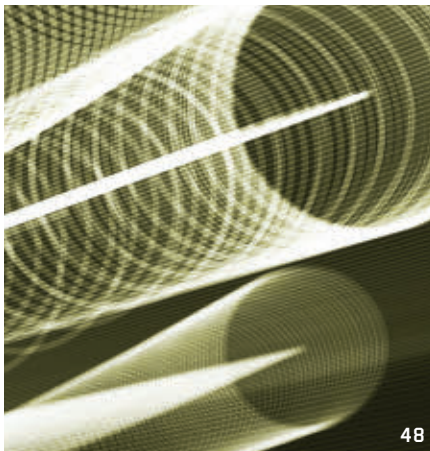
- 11 **Biology-Inspired Networking**
Researchers have developed a new networking algorithm, modeled after the neurological development of the fruit fly, to help distributed networks self-organize more efficiently.
By Kirk L. Kroeker
-
- 14 **Beauty and Elegance**
Leslie Valiant talks about machine learning, parallel computing, and his quest for simplicity.
By Gary Anthes
-
- 16 **The Promise of Flexible Displays**
New screen materials could lead to portable devices that are anything but rectangular, flat, and unbendable.
By Tom Geller
-
- 19 **Unlimited Possibilities**
M. Frans Kaashoek discusses systems work, “undo computing,” and what he learned from Andrew S. Tanenbaum.
By Gregory Goth
-
- 20 **All the News That’s Fit for You**
Personalized news promises to make daily journalism profitable again, but technical and cultural obstacles have slowed the industry’s adoption of automated personalization.
By Marina Krakovsky

Viewpoints

- 22 **Privacy and Security**
Identity Management and Privacy: A Rare Opportunity To Get It Right
The National Strategy for Trusted Identities in Cyberspace represents a shift in the way the U.S. government is approaching identity management, privacy, and the Internet.
By Ari Schwartz
-
- 25 **The Profession of IT**
Who Are We—Now?
Considerable progress has been made toward the formation of a computing profession since we started tracking it in this column a decade ago.
By Peter J. Denning and Dennis J. Frailey
-
- 28 **The Business of Software**
Practical Application of Theoretical Estimation
Estimation models at the extreme.
By Phillip G. Armour
-
- 31 **Inside Risks**
The Risks of Stopping Too Soon
Good software design is never easy, but stopping too soon makes the job more difficult.
By David Lorge Parnas
-
- 34 **Kode Vicious**
Think Before You Fork
KV’s thoughts on forking, config files, and using internal wikis.
By George V. Neville-Neil
-
- 36 **Viewpoint**
Computer Science Can Use More Science
Software developers should use empirical methods to analyze their designs to predict how working systems will behave.
By Clayton T. Morrison and Richard T. Snodgrass



Practice



48

- 40 **If You Have Too Much Data, then 'Good Enough' is Good Enough**
In today's humongous database systems, clarity may be relaxed, but business needs can still be met.
By Pat Helland

- 48 **Scalable SQL**
How do large-scale sites and applications remain SQL-based?
By Michael Rys

- 54 **Does Deterrence Work in Reducing Information Security Policy Abuse by Employees?**
Methods for evaluating and effectively managing the security behavior of employees.
By Qing Hu, Zhengchuan Xu, Tamara Dinev, and Hong Ling

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

Contributed Articles

- 62 **Advancing the State of Home Networking**
Before building the network or its components, first understand the home and the behavior of its human inhabitants.
By W. Keith Edwards, Rebecca E. Grinter, Ratul Mahajan, and David Wetherall

- 72 **10 Rules for Scalable Performance in 'Simple Operation' Datastores**
Partition data and operations, keep administration simple, do not assume one size fits all.
By Michael Stonebraker and Rick Cattell

- 81 **Specification and Verification: The Spec# Experience**
Can a programming language really help programmers write better programs?
By Mike Barnett, Manuel Fähndrich, K. Rustan M. Leino, Peter Müller, Wolfram Schulte, and Herman Venter

- VE** **Viscous Democracy for Social Networks**
Decision-making procedures in online social networks should reflect participants' political influence within the network.
By Paolo Boldi, Francesco Bonchi, Carlos Castillo, and Sebastiano Vigna

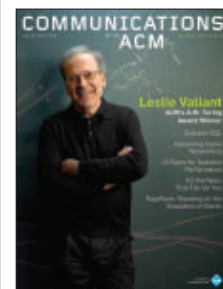
- VE** **Wireless on the Precipice: The 14th Century Revisited**
Business continuity plans for the wireless world must address solar activity.
By Denise McManus, Benjamin Adams, and Houston Carr

Review Articles

- 92 **PageRank: Standing on the Shoulders of Giants**
The roots of Google's PageRank can be traced back to several early, and equally remarkable, ranking techniques.
By Massimo Franceschet

Research Highlights

- 103 **Technical Perspective**
The Quest for a Logic for Polynomial-Time Computation
By Phokion G. Kolaitis
- 104 **From Polynomial Time Queries to Graph Structure Theory**
By Martin Grohe
- 113 **Technical Perspective**
Data Analysis at Astonishing Speed
By Michael J. Franklin
- 114 **Dremel: Interactive Analysis of Web-Scale Datasets**
By Sergey Melnik, Andrey Gubarev, Jingjing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, and Theo Vassilakis



About the Cover:
Leslie Valiant, ACM's 2010 A.M. Turing Award recipient, was honored for his fundamental contributions to theoretical computer science. His work, spanning over three decades, includes the theory of probably approximately correct (PAC) learning, the complexity of enumeration and of algebraic computation, and

the theory of parallel and distributed computing. See pages 14 and 128 for more on his illustrious career. Cover photograph by Larry Berow.



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 Marketing and Membership

David M. Smith

Director, Office of SIG Services

Donna Cappel

Director, Office of Publications

Bernard Rous

Director, Office of Group Publishing

Scott E. Delman

ACM COUNCIL

President

Alain Chesnais

Vice-President

Barbara G. Ryder

Secretary/Treasurer

Alexander L. Wolf

Past President

Wendy Hall

Chair, SGB Board

Vicki Hanson

Co-Chairs, Publications Board

Ronald Boisvert and Jack Davidson

Members-at-Large

Vinton G. Cerf;

Carlo Ghezzi;

Anthony Joseph;

Mathai Joseph;

Kelly Lyons;

Mary Lou Soffa;

Salil Vadhan

SGB Council Representatives

Joseph A. Konstan;

G. Scott Owens;

Douglas Terry

PUBLICATIONS BOARD

Co-Chairs

Ronald F. Boisvert; Jack Davidson

Board Members

Nikil Dutt; Carol Hutchins; Joseph A. Konstan;

Ee-Peng Lim; Catherine McGeoch;

M. Tamer Ozsu; Holly Rushmeier;

Vincent Shen; Mary Lou Soffa

ACM U.S. Public Policy Office

Cameron Wilson, Director

1828 L Street, N.W., Suite 800

Washington, DC 20036 USA

T (202) 659-9711; F (202) 667-1066

Computer Science Teachers Association

Chris Stephenson

Executive Director

2 Penn Plaza, Suite 701

New York, NY 10121-0701 USA

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

Association for Computing Machinery (ACM)

2 Penn Plaza, Suite 701

New York, NY 10121-0701 USA

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

COMMUNICATIONS OF THE ACM

Trusted insights for computing's leading professionals.

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

STAFF

DIRECTOR OF GROUP PUBLISHING

Scott E. Delman

publisher@cacm.acm.org

Executive Editor

Diane Crawford

Managing Editor

Thomas E. Lambert

Senior Editor

Andrew Rosenbloom

Senior Editor/News

Jack Rosenberger

Web Editor

David Roman

Editorial Assistant

Zarina Strakhan

Rights and Permissions

Deborah Cotton

Art Director

Andrij Borys

Associate Art Director

Alicia Kubista

Assistant Art Directors

Mia Angelica Balaquiot

Brian Greenberg

Production Manager

Lynn D'Addesio

Director of Media Sales

Jennifer Ruzicka

Public Relations Coordinator

Virginia Gold

Publications Assistant

Emily Williams

Columnists

Alok Aggarwal; Phillip G. Armour;

Martin Campbell-Kelly;

Michael Cusumano; Peter J. Denning;

Shane Greenstein; Mark Guzdial;

Peter Harsha; Leah Hoffmann;

Mari Sako; Pamela Samuelson;

Gene Spafford; Cameron Wilson

CONTACT POINTS

Copyright permission

permissions@cacm.acm.org

Calendar items

calendar@cacm.acm.org

Change of address

acmhlp@cacm.org

Letters to the Editor

letters@cacm.acm.org

WEB SITE

<http://cacm.acm.org>

AUTHOR GUIDELINES

<http://cacm.acm.org/guidelines>

ADVERTISING

ACM ADVERTISING DEPARTMENT

2 Penn Plaza, Suite 701, New York, NY

10121-0701

T (212) 869-7440

F (212) 869-0481

Director of Media Sales

Jennifer Ruzicka

jen.ruzicka@hq.acm.org

Media Kit acmm mediasales@acm.org

EDITORIAL BOARD

EDITOR-IN-CHIEF

Moshe Y. Vardi

eic@cacm.acm.org

NEWS

Co-chairs

Marc Najork and Prabhakar Raghavan

Board Members

Hsiao-Wuen Hon; Mei Kobayashi;

William Pulleyblank; Rajeev Rastogi;

Jeannette Wing

VIEWPOINTS

Co-chairs

Susanne E. Hambrusch; John Leslie King;

J Strother Moore

Board Members

P. Anandan; William Aspray;

Stefan Bechtold; Judith Bishop;

Stuart I. Feldman; Peter Freeman;

Seymour Goodman; Shane Greenstein;

Mark Guzdial; Richard Heeks;

Rachelle Hollander; Richard Ladner;

Susan Landau; Carlos Jose Pereira de Lucena;

Beng Chin Ooi; Loren Terveen

Q PRACTICE

Chair

Stephen Bourne

Board Members

Eric Allman; Charles Beeler; David J. Brown;

Bryan Cantrill; Terry Coatta;

Stuart Feldman; Benjamin Fried;

Pat Hanrahan; Marshall Kirk McKusick;

Erik Meijer; George Neville-Nel;

Theo Schlossnagle; Jim Waldo

The Practice section of the CACM

Editorial Board also serves as

the Editorial Board of *Queue*.

CONTRIBUTED ARTICLES

Co-chairs

Al Aho and Georg Gottlob

Board Members

Yannis Bakos; Elisa Bertino; Gilles Brassard;

Alan Bundy; Peter Buneman; Andrew Chien;

Peter Druschel; Blake Ives; James Larus;

Igor Markov; Gail C. Murphy; Shree Nayar;

Lionel M. Ni; Sriram Rajamani;

Marie-Christine Rousset; Avi Rubin;

Krishnan Sabnani; Fred B. Schneider;

Abigail Sellen; Ron Shamir; Marc Snir;

Larry Snyder; Veda Storey; Manuela Veloso;

Michael Vitale; Wolfgang Wahlster;

Andy Chi-Chih Yao

RESEARCH HIGHLIGHTS

Co-chairs

David A. Patterson and Stuart J. Russell

Board Members

Martin Abadi; Stuart K. Card; Jon Crowcroft;

Shafi Goldwasser; Monika Henzinger;

Maurice Herlihy; Dan Huttenlocher;

Norm Jouppi; Andrew B. Kahng;

Gregory Morrisett; Michael Reiter;

Mendel Rosenblum; Ronit Rubinfeld;

David Salesin; Lawrence K. Saul;

Guy Steele, Jr.; Madhu Sudan;

Gerhard Weikum; Alexander L. Wolf;

Margaret H. Wright

WEB

Co-chairs

James Landay and Greg Linden

Board Members

Gene Golovchinsky; Marti Hearst;

Jason I. Hong; Jeff Johnson; Wendy E. MacKay



ACM Copyright Notice

Copyright © 2011 by Association for Computing Machinery, Inc. (ACM). Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and full citation on the first page. Copyright for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or fee. Request permission to publish from permissions@acm.org or fax (212) 869-0481.

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

Subscriptions

An annual subscription cost is included in ACM member dues of \$99 (\$40 of which is allocated to a subscription to *Communications*); for students, cost is included in \$42 dues (\$20 of which is allocated to a *Communications* subscription). A nonmember annual subscription is \$100.

ACM Media Advertising Policy

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

Single Copies

Single copies of *Communications of the ACM* are available for purchase. Please contact acmhlp@cacm.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.

Computing and India

India is rightfully in the news in recent years, thanks to its impressive economic growth and its notable software industry. Modern computing and India have had a long

association. India's own computing efforts began in the early 1950s with the Tata Institute of Fundamental Research (TIFR) in Mumbai and the Indian Statistical Institute (ISI) in Kolkata building their own computers, primarily for scientific applications. Business process outsourcing started in the 1960s with commercial computer centers performing telephone and electricity billing, airline scheduling, and so on.

India, however, missed the rapid pace of growth in computing toward the end of the 1970s. That era found discrete computers replaced by VLSI circuits requiring cutting-edge technology and billions in investments. The software train also moved far ahead in that period. The country lagged seriously due to inadequate research activities in its academia and industry.

However, destiny was beckoning India. The rapid computerization worldwide in the 1980s brought software to far greater prominence. Small and large organizations began using computers to increase productivity. Development of new software and customization of existing software created a demand for a great number of programmers.

India was naturally poised to play a critical role at that stage, with a large number of young people trained in software. The demography was critical, as fast-changing tools and technologies could only be mastered comfortably by a young work force. India had many necessary ingredients: youth, English training, a hunger for learning, and, of course, a dire need for employment. The Y2K problem brought

unprecedented demand that was leveraged effectively by the Indian software industry. The sophistication of software tools empowered even those who didn't have top-class computer science training. The software industry grew rapidly, aided by high-bandwidth data lines. From nearly nothing in the early 1990s, the industry today contributes to nearly 7% of India's more than \$1 trillion GDP and employs over three million employees directly.

Despite the boom in the software industry, the country clearly missed investing in high-quality academics and research. It is puzzling how a country with more than three million IT professionals and over a million undergraduate computing students produces barely 100 quality Ph.D.s each year. Computer science has many Indians, but only a small fraction work in India!

India today has 1.2 billion individuals, of which about 400 million are in the 1–15 age group and another about 400 million in the 15–35 age group. India, therefore, has the opportunity and responsibility to provide the future intellectual work force in computing and other areas. The quality of primary education is highly variable and downright poor in all but a fraction of schools. Good teachers cannot be produced rapidly, but good teaching materials can be replicated economically using technology. The problem is truly Himalayan in proportions and a great challenge in scale and variety. It can provide wonderful opportunities to all researchers and technologists to work with their Indian counterparts for great mutual benefit.

Challenges for India include improving quality in its undergraduate programs as well as enrollment numbers in the research programs in computing. College enrollment numbers have multiplied in the last decade but the quality lags. Adoption of technology can help improve education, but technology cannot solve the problem. Trained college teachers are in great shortage and radical ideas are needed to create them. Creating enough Ph.D.s is practically impossible. A scalable master's-level course with sufficient research exposure can be designed to create a teaching force.

Increasing human resources for top-class research is essential for India to make its own mark in computing. This calls for more doctoral students and greater advising resources in the academia. Creating networked research groups from institutions in India and abroad is a way to approach this challenge. This is a great opportunity and an invitation to the global computing community to join hands with Indian institutions to establish multi-country research groups and doctoral programs. Let us learn from the industry, where multi-country groups are the norm today. The government and industry in India have recognized the problem and are eager to be part of any solution that can work.

ACM has recently established local presence in India. In addition to extending membership to the great number of software professionals here, ACM can help facilitate the expansion of computing education and research in India by initiating collaborations with Indian groups and bringing top ACM conferences here. India can gain from ACM's collective experience and ACM from the opportunities India presents. 

P.J. Narayanan and Anand Deshpanda are co-chairs of ACM's India Council.

© 2011 ACM 0001-0782/11/06 \$10.00

DOI:10.1145/1953122.1953124

Why Concurrent Objects Are Recurrently Complicated

NIR SHAVIT'S ARTICLE "Data Structures in the Multi-core Age" (Mar. 2011) was a thrill to read; so, too, was the timely Viewpoint "Managing Time" by Peter J. Denning, the article "Understanding Scam Victims: Seven Principles for Systems Security" by Frank Stajano and Paul Wilson, the Technical Perspective "VL2" by Jennifer Rexford, and the article "VL2: A Scalable and Flexible Data Center Network" by Albert Greenberg et al.

I was especially intrigued by Shavit's exposition on concurrent objects. I first encountered lock-free and wait-free protocols in a 1993 article by Maurice Herlihy,¹ using load-linked and store-conditional instructions. Thanks now to Shavit for explaining additional techniques, including exchangers and balancers.

Also important is to point out that Shavit's pseudocode for the lock-free stack included an unstated restriction—that a node, once popped, never goes back on the stack—the ABA problem; see the related *Communications* blog page <http://cacm.acm.org/magazines/2011/3/105308-data-structures-in-the-multicore-age/comments>, as well as Herlihy's and Shavit's 2008 book.²

The key point is that concurrent data structures are more difficult than they might first seem, as I discussed in my own 1995 article³ and that it is easy to introduce subtle bugs, especially when all important assumptions are not stated or enforced; in this case, that a node, once popped, never goes back on the stack.

Joseph P. Skudlarek, Lake Oswego, OR

References

1. Herlihy, M.P. A methodology for implementing highly concurrent data objects. *ACM Transactions on Programming Languages and Systems* (Nov. 1993).
2. Herlihy, M.P. and Shavit, N. *The Art of Multiprocessor Programming*. Morgan Kaufmann, San Mateo, CA, 2008.
3. Skudlarek, J.P. Notes on a methodology for implementing highly concurrent data objects. *ACM Transactions on Programming Languages and Systems* (Jan. 1995).

Origin of SQL

I would like to clarify Erik Meijer's and Gavin Bierman's statement concerning the origin of SQL in their article "A Co-Relational Model of Data for Large Shared Data Banks" (Apr. 2011) that said, "The landscape changed radically when Ted Codd proposed a new data model and a structured query language (SQL) based on the mathematical concepts of relations and foreign/primary key relationships." There were actually additional steps and additional people involved getting from Codd's influential 1970 *Communications* article "A Relational Model of Data for Large Shared Data Banks" (cited by Meijer and Bierman) to the SQL language. In it and in subsequent articles and papers, Codd discussed query languages based on both relational calculus and relational algebra. But it was Raymond Boyce and Donald Chamberlin who, after studying Codd's work, defined the SEQUEL language, later renamed SQL, as first documented by Chamberlin and Boyce.¹ For more on how Codd's ideas influenced Boyce and Chamberlin and the rest of the System R team, see Chamberlin.²

Paul McJones, Mountain View, CA

References

1. Chamberlin, D.D. and Boyce, R.F. SEQUEL: A structured English query language. In *Proceedings of the 1974 ACM SIGFIDET (Now Sigmod) Workshop on Data Description, Access and Control* (Ann Arbor, MI, May 1–3). ACM Press, New York, 1974, 249–264.
2. Chamberlin, D. Oral History, Paul McJones, interviewer. Computer History Museum, Mountain View, CA, July 21, 2009; <http://www.computerhistory.org/collections/accession/102702111>

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

© 2011 ACM 0001-0782/11/06 \$10.00

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

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



DOI:10.1145/1953122.1953125

In the Virtual Extension

To ensure the timely publication of articles, Communications created the Virtual Extension (VE) to expand the page limitations of the print edition by bringing readers the same high-quality articles in an online-only format. VE articles undergo the same rigorous review process as those in the print edition and are accepted for publication on merit. The following synopses are from articles now available in their entirety to ACM members via the Digital Library.

contributed article

DOI: 10.1145/1953122.1953152

Viscous Democracy for Social Networks

Paolo Boldi, Francesco Bonchi,
Carlos Castillo, and Sebastiano Vigna

On April 23, 2009, Facebook announced preliminary results of a vote in which users were asked about a change in the terms of use of the network. The vote was the result of a petition by thousands of users criticizing the site for claiming too many rights over user-generated content. Attempting to justify the change, Facebook let users vote, saying that if 30% of the then roughly 200 million “active” users would vote the decision would be binding on Facebook management. The outcome was 74.4% of the voters preferred the new rules. And while only 600,000 users voted (1/100th of the prefixed quorum of 60,000,000) the change in the terms of use were officially adopted.

The global privacy watchdog, Privacy International, called the vote a “massive confidence trick.” The low voting turnout was largely foreseeable. Only a small fraction of Facebook users have the time, patience, or dedication or take the service seriously enough to actively participate in its governance.

Whether to decide on a motion (such as pick one option among a set of alternatives) or elect representatives (such as constitute a senate), voting is a collaborative decision-making process, seeking a result that reflects as much as possible the opinion of the community as a whole.

Viewed this way, the failure of the 2009 Facebook voting experiment is explained by the kind of voting system Facebook adopted. Attempting this kind of direct democracy voting in large online communities is not necessarily the best choice; when public decisions reach a certain level of complexity, it is unrealistic to assume every participant is engaged and informed enough to contribute to the decision.

This opinion is shared by other authors observing that the degree of commitment of participants in online communities and collaboration systems varies greatly. Nielsen’s “90-9-1 Rule

for Social Design” says: “In most online communities, 90% of users are lurkers who never contribute, 9% of users contribute a little, and 1% of users account for almost all the action.”

contributed article

DOI: 10.1145/1953122.1953153

Wireless on the Precipice

Denise McManus, Benjamin Adams,
and Houston Carr

Effective communication is the cornerstone of a successful business continuity plan. Any cataclysmic event that causes the failure of communication and associated technologies will have a devastating impact on the global economy.

Today, a significant amount of global communication and business transactions are conducted electronically; a major part handled wirelessly. Almost all technology that controls our life requires electrical power, so just imagine the simultaneous loss of wireless and wired devices. Even a regional loss covering North America, Europe, or Asia could have catastrophic consequences in a global environment.

One event that can be a significant threat to wireless and electrical power is the occurrence of geomagnetic disturbances due to solar activity. These disturbances have historically

presented the risk of power disruption as well as infrastructure outages. Current potential disruptions include all services that rely on satellite navigation data, such as aircraft navigation, emergency location systems, global positioning systems, and cellular services. Organizations that have worldwide locations have increased risk to solar activities.

While there are a number of dystrophic predictions about the year 2012, they do not include what may be the most significant—the potential disruption of global communications and electrical power by geomagnetic radiation as a result of increased cyclic sunspot activity. We are currently approaching a period of maximum solar storm activity (2012–2013) that is certainly the first since the explosive growth in wireless technologies. The solar activity, at its highest, can produce powerful electrical surges that can disrupt any exposed wired or wireless technology. The very infrastructure of our society could be at risk.

How real is the threat from solar storms? What are the associated risks? How might we mitigate these risks? This article addresses the vulnerabilities of this threat and present actions that seem warranted. The authors warn organizations to prepare for geomagnetic disturbances in the upcoming period of high solar activity. Moreover, business continuity plans for the wireless world must address solar activity at all stages of the solar cycle.

Coming Next Month in COMMUNICATIONS

**Algorithmic Composition:
Computational
Thinking in Music**

Searching for Jim Gray

**A Decade of Model
Checking with SLAM**

Too Many Copyrights?

**Cellular Telephony and
the Question of Privacy**

**Non-Myths
About Programming**

**Driving Power in
Global Supply Chains**

Values in Design

Also, the latest news on autonomous cars,
social media usage, and Monte Carlo research.

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



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

DOI:10.1145/1953122.1953126

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

Simple Design; Research vs. Teaching; and Quest to Learn

Daniel Reed writes about design and disruptive technologies.

Mark Guzdial discusses the tension between teaching and research at U.S. universities. Judy Robertson argues in favor of educational video games in the classroom.



Daniel Reed
"Simple HPC Wins—Usually"

<http://cacm.acm.org/blogs/blog-cacm/98517>

In the 19th century, writing about his work on mechanical calculating devices, Charles Babbage noted, "The most constant difficulty in contriving the engine has arisen from the desire to reduce the time in which the calculations were executed to the shortest which is possible." Roughly a century later, Daniel Slotnik wrote retrospectively about the ILLIAC IV parallel computing design, "By sacrificing a factor of roughly three in circuit speed, it's possible that we could have built a more reliable multi-quadrant system in less time, for no more money, and with a comparable overall performance."

Babbage's design challenged the machining and manufacturing capabilities of his day, though others were recently able to build a functioning sys-

tem using parts fabricated to tolerances achievable with 19th century processes. Similarly, Slotnik's design challenged electronics and early semiconductor fabrication and assembly. Today, of course, parallel computing designs embodying tens of thousands of processors are now commonplace, leveraging inexpensive, commodity hardware.

There is a lesson here that systems designers repeatedly ignore at their peril. Simple designs usually triumph, and the artful exploitation of mainstream technologies usually bests radical change.

All of which is to say that incrementalism wins repeatedly, right up to the point when a dislocating phase transition occurs. There are, of course, many paths to failure. One can be too early or too late. Or to put it another way, you want to be the first person to design a successful, transistorized computer system, not the last person to design a vacuum tube computer. The same is true of design approaches such as pipe-

lining, out of order issue and completion, superscalar dispatch, cache design, and programming tools.

Any designer's challenge is to pick the right technologies at the right time, recognizing when inflection points, such as maturing, disruptive technologies, are near.

The shift from largely proprietary high-performance computing designs to predominantly commodity clusters a decade ago was but one of the most recent transitions. Arguably, we are near another disruptive technology point. The embedded hardware ecosystem offers one intriguing new performance-power-price point, particularly as we consider trans-petascale and exascale designs that are energy constrained. The experiences of cloud providers in building massive scale infrastructures for data analytics and on-demand computing are another possibility.

As I frequently told my graduate students at the University of Illinois at Urbana-Champaign, the great thing about parallel computing is the question ("How can I increase performance?") never changes, but the answers do. Babbage would have understood.



Mark Guzdial
"Can the U.S. Afford Universities Focusing on Research Over Teaching?"

<http://cacm.acm.org/blogs/blog-cacm/98702>

I direct an effort, which is funded by the National Science Foundation's Broadening Participation in Computing

program, called “Georgia Computes!” in which we are trying to improve and broaden computing education, across the pipeline, statewide. We spend a lot of effort offering professional development to high school teachers and undergraduate faculty. We are increasingly getting signals that the faculty in the University System of Georgia (USG) are turning more toward research, away from teaching.

Over the four years of the project, we have had fewer and fewer USG faculty attend our workshops. We have had many return visitors, and good coverage across the 29 USG institutions with computing departments. But overall nowadays, we are lucky to get a half-dozen attendees. Our external evaluator did interviews with faculty around the state to help us understand the attendance issue. The answer was pretty much the same from everyone, characterized by this interview quote: “In any department, only about 20% of the faculty care about undergraduate teaching. You got them all.”

One of my Ph.D. students, Lijun Ni, is studying computing teachers, e.g., their sense of identity and how they improve their practice. She interviewed USG faculty at institutions whose mission is primarily undergraduate teaching. When asked, “What do you do to improve as a computing educator?” one faculty member told her, “I’m not a computing educator. I don’t want to improve at it. My tenure case depends on my research work. I’m not going to spend any time working at being a better teacher.”

This tension has always existed in the American university system. In his book, *How Scholars Trumped Teachers*, Larry Cuban points out that the American land grant universities were designed in the late 1800s to merge the British focus on undergraduate education with the German research university. But nobody asked whether that was even possible, and Cuban argues that the structure of the American university pretty much prevents the potential synergy of research and teaching from ever working.

The question I’m raising here is whether we can afford a shift toward research and away from teaching in the United States. There is evidence suggesting that the increasing costs of higher education are not due to growth

in instructional costs, but in costs associated with sponsored programs and graduate education. In his “Why Universities Do Research” post, Rich DeMillo points that university research rarely pays for itself. Doing research is more expensive than doing education well.

Of course, we need research in American universities. It’s absolutely critical for graduate education. Do we need research in our undergraduate institutions? Rich also cites a new study in his “Damaged Pipelines and the Future of Innovation” post showing that many of our undergraduate institutions that produce the most future Ph.D. students do not have (large, expensive) research programs. Number two on that list is Harvey Mudd College, which prides itself on having a “liberal arts engineering” focus, where undergraduate education is the top priority. If we can have high-quality undergraduate education without research programs, and research programs are so expensive, and higher education costs are growing too quickly for our nation’s economy to absorb, does it make sense to steer away from teaching and into research?



Judy Robertson
“The Importance of
Quest to Learn”

<http://cacm.acm.org/blogs/blog-cacm/99121>

A thought-provoking article, “Learning by Playing: Video Games in the Classroom,” appeared in *The New York Times* recently on games in education. Author of the indigestible yet indispensable academic game design bible *Rules of Play*, Katie Salen has a project in a New York city school called Quest to Learn in which children’s learning is based around game playing and design. It sounds like a big budget, integrated version of what teachers in the U.K. have been doing in their own practice: the teachers playing games on a big screen in front of the class while soliciting advice from a super-engaged, drooling set of kids; and learners using game-authoring software to explore concepts with game design. What’s different is that this project has been systematically devised and rolled out across a school, rather than relying on the individual innovative practices of some teachers. Particularly telling is the fact that there are three game design-

ers working with 11 teachers to devise games around interdisciplinary curricular content. That’s a fantastic resource.

The *Times* article acknowledges that sixth-graders who took part in Quest to Learn did no better on standardized tests than learners who had not had the privilege. This makes me uneasy. Sure, you can argue the Quest to Learn values different educational goals that are more relevant to 21st century society than ordinary curricula and that therefore different assessment tools are required. But is this just special pleading? It may be that we need to win the wider battle of making school assessment more appropriate to the skills required in current society before we worry about installing Wiis in the classroom.

One of the most interesting aspects of the article was related to the metaphor used to organize school life. The author writes: “What if we blurred the line between academic subjects and remained the typical American classroom so that, at least in theory, it came to resemble a typical American living room, or child’s bedroom or even a child’s pocket circa 2010 ... what if, instead of seeing school the way we have always seen it, we saw school for what our children dreamed it might be: a big, delicious video game?”

Guy Claxton argues that schools are currently organized around a factory metaphor, full of grim tasks that must be completed and overseen by a supervisor figure, and time strictly marshaled by bells. He proposes a metaphor which he considers to be more fruitful: learning as gymnasium, where learners strive to improve their own performance by exercising their brains in an enjoyable fashion. This is consistent with the learning as game metaphor, but perhaps more general purpose. In any case, it is about time we reconsidered how schools work as learning spaces. The industrial revolution has had its day, and we are beginning to realize it is counterproductive to coop children up in learning factories. That’s why we need experiments like Quest to Learn. ■

Daniel Reed is vice president of Technology Strategy & Policy and the eXtreme Computing Group at Microsoft. **Mark Guzdial** is a professor at the Georgia Institute of Technology. **Judy Robertson** is a lecturer at Heriot-Watt University.

© 2011 ACM 0001-0782/11/06 \$10.00



DOI:10.1145/1953122.1953127

Scott E. Delman

Say It With Video

If you're one of the more than 1,000,000 scientists, practitioners, educators, or students from around the world who regularly read *Communications of the ACM* in print or online, you may have noticed that a new feature has recently been introduced on the magazine's Web site—*video*. Since January, ACM has been adding author or media-generated videos to the main landing page of the site (also accessible via the Abstracts tab in the ACM Digital Library). Thus far, these videos have highlighted solely the work of the cover stories, but based on the initial success of these videos and the decision of *Communications'* Web Board, over the coming months we will be expanding the number of videos posted on the site each month and creating a video archive to make these videos easily accessible for our readers and viewers. This archive will go live along with the launch of the new *Communications* Web site later this summer.

Based on our own experience and that of other scholarly publishers, publishing article-oriented videos that focus on articles published in the magazine increases interest in the articles themselves, improves readability, and creates an overall new and powerful experience for users that enhances the value they derive from the publication in general. Not surprisingly, the amount of time each

viewer spends with each article tends to increase and the content itself becomes more approachable in the video format. The challenge for our authors is to develop high-quality video content that makes full-length articles appealing to our broad readership. The challenge for ACM is to make sure the *Communications* Web site does not become a digital warehouse for all computing-related videos that tries to compete with the likes of YouTube or Facebook, and

to make sure these videos are viewable on the full range of screens out there from desktops to tablets to smartphones. For our viewers, it is important they know the same level of editorial selection exists for this growing video archive as for full-length published articles, and that this supplemental material will be directly relevant to the content published in the magazine. *Communications'* focus continues to be on quality versus quantity, and this new video archive will be consistent with that philosophy.

So, if you've submitted an article to *Communications'* Editorial Board and your article has been accepted, please contact us as soon as possible to start planning the creation of your video. Doing so is fast becoming an important way to communicate your work to the community and it is a great way to increase your work's exposure to *Communications'* large and global readership.



ACM Member News

SUSAN LANDAU ON PRIVACY VS. SECURITY



Bringing intellectual clarity to the emotional and often-complex topic of security is at the heart of

Susan Landau's mission. A fellow at the Radcliffe Institute for Advanced Study at Harvard University, Landau confronts the topic in her new book, *Surveillance or Security? The Risks Posed by New Wiretapping Technologies*.

"It is a challenging time," Landau acknowledges. "But it's important to recognize that building surveillance into the communications infrastructure creates long-term risk. We might capture bad guys now, but it could be used against us in the future."

Landau notes, for example, that unknown individuals used a cellular phone system designed for law enforcement purposes to illegally wiretap about 100 Greek politicians in 2004 and 2005. Likewise, some 6,000 Italians, including politicians, judges, and celebrities, were victims of illegal wiretapping from 1996 to 2006.

Moreover, as the technology landscape blazes ahead, new risks will emerge. "Twenty years ago telephones didn't move," says Landau. "Now they track you and they are incredibly revealing. It's another tool that can be used or abused." It used to take 42 days for the U.S. Marshals Service to locate and arrest a suspect, says Landau. Now, by monitoring the location of a suspect's mobile phone, it is a matter of two days.

Landau believes we must question whether particular security technologies are effective and, if so, what price we pay for the protection. ACM members, she says, can play a pivotal role in framing key issues and helping society understand them. "Are there other measures we could take that are more effective?" Landau asks. "It's critical to fully understand the risks and whether these risks are acceptable in terms of policy and civil liberties."

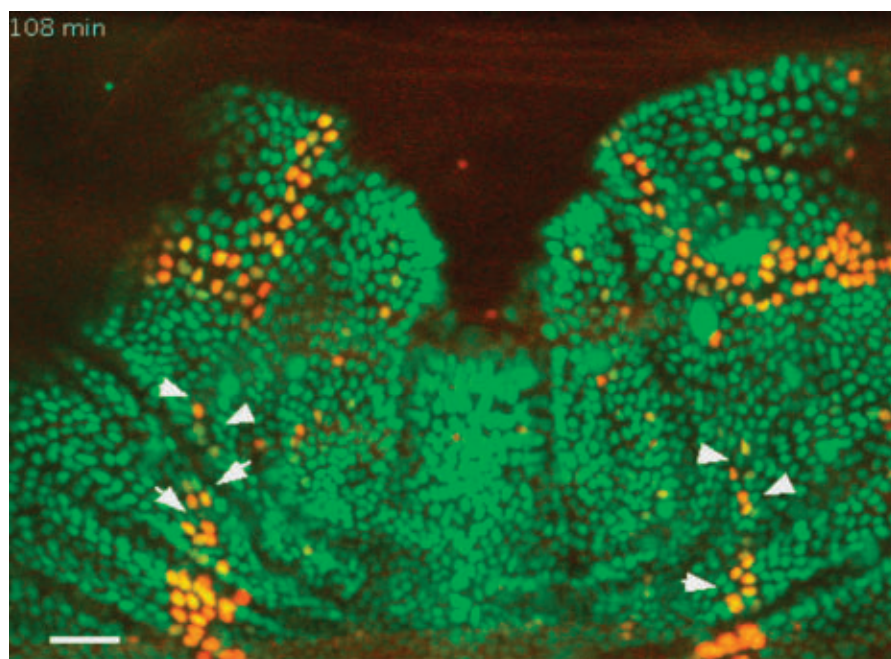
—Samuel Greengard

Biology-Inspired Networking

Researchers have developed a new networking algorithm, modeled after the neurological development of the fruit fly, to help distributed networks self-organize more efficiently.

THE FRUIT FLY'S neurological development is serving as a model for a networking algorithm that researchers say has the potential to obsolesce traditional methods for determining peer relationships in distributed networks. Without prior knowledge of how cells are connected, the fly's developing nervous system allocates certain cells as leaders that provide direct connections with other nerve cells. That development process, the researchers say, is similar to conventional schemes used to manage distributed networks but is much simpler and more robust than anything humans have yet devised.

The lead scientist of the project to develop the new networking algorithm is Ziv Bar-Joseph, an associate professor of machine learning and computational biology at Carnegie Mellon University. Bar-Joseph says he became interested in biology-inspired computing while studying at Massachusetts Institute of Technology during the time of the initial sequencing of the human genome. "I was fascinated by the opportunities this information presented for improving human health and our



In this microscope image of the pupal stage of fruit fly development, nerve cells that become sensory organ precursors (SOPs) are identified by arrows. These cells send chemical signals to neighboring cells, blocking them from becoming SOPs.

understanding of the inner workings of cells in our body," he says. "However, I also realized that the explosion of data coming out of biology would require the development of new computational methods to make sense of this data."

More than a decade later, the need for sophisticated methods for dealing with increasingly large sets of biological data and even new types of information remains a salient characteristic of both computational biology and

biology-inspired computing, with the former applying computational methods to biology and the latter drawing design inspiration from biological processes.

Following earlier work—in neural networking, for example—that focused on creating computing models developed from a high-level understanding of biological systems, researchers working today on biology-inspired computing projects study biological systems at a much lower level, deriving insight from the way molecular systems function. While this new generation of biology-inspired computing is relatively young, researchers in the field are developing computing approaches that already are having an impact on computer science.

But despite the promise of innovative, biological solutions to classic computing problems, the field of biology-inspired computing is filled with many challenges. One of the major issues facing the field is to develop new languages and methodologies so team members, having different backgrounds and expertise, will understand each other more effectively.

Bar-Joseph says the issue of team communication is improving, especially with several new Ph.D. programs in computational biology enabling students to achieve fluency in both biology and computer science. However, despite the rise of dedicated graduate programs, computer scientists

and biologists often take very different approaches to their work. “Finding a common problem that would be of interest to both sides is something that I view as one of the major challenges in this area,” says Bar-Joseph. “You want to work on cutting-edge computer science research while still advancing biology. It’s not trivial to find such problems and good collaborators to work with.”

A Fortunate Visit

The fruit fly research undertaken by Bar-Joseph and his colleagues—Noga Alon and Yehuda Afek of Tel Aviv University and Naama Barkai, Eran Hornstein, and Omer Barad of the Weizmann Institute of Science—emerged from what Bar-Joseph calls an “accidental event” during a visit to a colleague’s lab where a graduate student described research on the molecular mechanisms involved in cell fate determination. Bar-Joseph says the described research seemed to apply to a problem in networking, so the team set about designing the fruit fly project that eventually resulted in what the researchers say is a breakthrough algorithm for distributed networks.

In networking, one method of creating a distributed framework is to allocate a set of leader nodes, sometimes called a maximal independent set (MIS), which can be used to communicate with the network’s other nodes. Every node in such a network is a leader or is connected to a leader, but the leaders are not interconnected. The researchers found that a similar arrangement occurs in the fruit fly, which uses tiny hairs for sensing. Each hair develops from a nerve cell called a sensory organ precursor (SOP), which connects to adjoining nerve cells but does not connect to other SOPs.

Computer scientists have developed several methods to determine how best to elect an MIS in distributed networks. One popular approach is to use a probabilistic method in which some nodes identify themselves as leaders on the basis of how many connections they have with other nodes. Nodes connected to these self-selected leaders take themselves out of the running in subsequent selection rounds. While this selection process is rapid, the network is inundated

with messages because all nodes must continuously monitor how they are connected.

According to the researchers, the fruit fly’s cells have no information about how they are connected and no way to determine how many neighbors they have. As various cells self-select as SOPs, they send out chemical signals that inhibit neighboring cells from becoming SOPs. This process continues until cells are either SOPs or are neighbors of an SOP. The researchers say the probability that any cell will self-select as an SOP during the fly’s development increases not as a function of connections, as in the typical algorithm for distributed networks, but as a function of time.

The algorithm developed by Bar-Joseph and colleagues balances trying to reduce the number of potential collisions—neighboring nodes mistakenly electing themselves as leaders—against trying to quicken the selection of the MIS. To create this trade-off, the new algorithm relies on a time-based method that uses weighted randomization to decide whether a node will nominate itself as a leader node. Determining the probability of the randomization is tricky, the researchers say, because a high probability of self-selection means too many collisions and too many leader nodes. Conversely, a low probability means too few nodes would elect themselves.

Following the pattern of the fruit fly’s development, the algorithm varies that probability over time, starting low and increasing the likelihood that a node will self-select. “It turns out that if you have lots of neighbors, you would be selected early on, or connected to someone selected, whereas if you have few neighbors, it might take a bit longer,” says Bar-Joseph.

The researchers concede that, at this point, the new algorithm is slower than conventional methods for determining an MIS in a distributed network. But that speed shortfall, they say, is outweighed by efficiency, especially in the context of sensor networks where speed in establishing and maintaining an MIS is not the primary objective. The new algorithm is designed to operate efficiently with few assumptions about node location and with a minimal messaging overhead.

Biology-inspired computing is relatively young, but researchers in the field have developed computing approaches that already are having an impact on computer science.

“In sensor networks, the nodes need to initially establish some backbone, which is what our algorithm does,” says Ziv Bar-Joseph. “For such sensors, energy efficiency is an important issue, which is why we believe our method should become the method of choice for these applications.”

Sensor networks, often used to monitor environmental conditions, typically consist of inexpensive devices that can be scattered across a large area. In these networks, there is no prior knowledge of where each sensor will be positioned or even whether all sensors will remain within range of all other sensors. “In sensor networks, the nodes need to initially establish some backbone, which is what our algorithm does,” says Bar-Joseph. “For such sensors, energy efficiency is an important issue, which is why we believe our method should become the method of choice for these applications.”

While the algorithm appears to be well suited to sensor networks, whether its creation will have implications for other areas of distributed computing remains to be seen. For his part, Bar-Joseph says he is hopeful that such biology-inspired ideas will help improve networking frameworks so they can operate more effectively in conditions that challenge today’s best fault-tolerant designs. “Biological systems are often distributed, both within and between cells,” he says. “As such, they face many of the same

challenges faced by modern complex computer systems, so fault tolerance that is a major issue in large computer networks is an issue that we can study in these systems as well.”

Bar-Joseph and his colleagues are working on several new projects in this area, trying to determine how cellular failure can provide clues about which nodes in computer networks are best maintained with backups. He and his team are also working on trying to develop more effective methods for determining the appropriate connectivity of computer networks so they can withstand node failures and message corruption. Indeed, the field of biology-inspired computing appears to be full of possibilities for innovative projects that investigate real-life analogues to computing scenarios.

“As computer science matures, it will start to look outward much more than it does now and play a bigger role in several other areas of science and technology,” says Bar-Joseph. “We are still a rather young discipline, so most of the efforts are inward-looking, but this is slowly changing.” **C**

Further Reading

Afek, Y., Alon, N., Barad, O., Hornstein, E., Barkai, N., and Bar-Joseph, Z.

A biological solution to a fundamental distributed computing problem, *Science* 331, 6014, Jan. 14, 2011.

Ater-Kranov, A., Bryant, R., Orr, G., Wallace, S., and Zhang, M.

Developing a community definition and teaching modules for computational thinking, *Proceedings of the 2010 ACM Conference on Information Technology Education*, Midland, MI, Oct. 7–9, 2010.

Gitter, A., Siegfried, Z., Klutstein, M., Fornes, O., Oliva, B., Simon, I., and Bar-Joseph, Z.

Backup in gene regulatory networks explains differences between binding and knockout results, *Molecular Systems Biology*, June 16, 2009.

Kephart, J.O.

Learning from nature, *Science* 331, 6018, Feb. 11, 2011.

Tero, A., Takagi, S., Saigusa, T., Ito, K., Bebber, D.P., Fricker, M.D., Yumiki, K., Kobayashi, R., and Nakagaki, T.

Rules for biologically inspired adaptive network design, *Science* 327, 5964, Jan. 22, 2010.

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

© 2011 ACM 0001-0782/11/06 \$10.00

In Memoriam

Paul Baran: 1926–2011

Paul Baran, whose insights into distributed communications would form the backbone of the ARPANET and later the Internet, died on March 26 at his home in Palo Alto, CA, at the age of 84. The cause of death was complications from lung cancer.

In the 1960s, Baran worked for RAND Corporation’s computer science division, developing a way to ensure military messages would not be compromised in the event of a nuclear war. Baran created a system that broke signals into small bundles and transmitted them through multiple pathways to be reassembled by a computer at the end. His ideas about breaking messages into discrete bundles to prevent data loss became known as “packet switching,” and they were used by the U.S. Department of Defense to create the ARPANET, which in turn gave birth to the Internet. Packet switching continues to be the underlying technology of today’s Internet.

Baran left RAND in 1968 to cofound the nonprofit Institute for the Future, where he became an authority on emerging digital networks. In the 1970s he started the first of at least 11 companies, five of which went public, according to his son David.

Baran received many awards, including the IEEE Alexander Graham Bell Medal, a Marconi Fellowship, the Bower Award and Prize, the National Inventors’ Hall of Fame, and the National Medal of Technology.

“He was an engineer’s engineer,” according to Vint Cerf, Google’s Chief Internet Evangelist and a longtime colleague and friend of Baran’s. “A technology iconoclast, he went where others feared or refused to go.”

Although he was often lauded as one of “the fathers of the Internet,” Baran gave credit freely when he felt it was due. It was typical of him to say, as he did in a 2001 interview, “The Internet was really the work of a thousand people.”

—Graeme Stemp-Morlock



Beauty and Elegance

Leslie Valiant talks about machine learning, parallel computing, and his quest for simplicity.

DAVID GELERNTER, A computer scientist at Yale University, calls beauty in computing “the happy marriage of simplicity and power.” It would be difficult to find a greater master of those principles than Leslie Valiant, the T. Jefferson Coolidge Professor of Computer Science and Applied Mathematics at Harvard University, and the recipient of the 2010 ACM A.M. Turing Award.

The Turing Award citation notes Valiant’s “transformative contributions to the theory of computation, including the theory of probably approximately correct (PAC) learning, the complexity of enumeration and of algebraic computation, and the theory of parallel and distributed computing.” (An interview with Valiant, “A Lifelong Learner,” appears on p. 128; to view his award citation see <http://bit.ly/jXAH31>.)

“He looks at what many people would see as very complex problems, and he breaks them down into something beautiful and elegant, really getting at the heart of the problem,” says Jennifer Chayes, managing director of Microsoft Research New England. “He thinks with such amazing clarity. Part of it is his solution, but a lot of it is his definition and articulation of these problems.”

Chayes and others point in particular to Valiant’s pioneering 1984 paper, “Theory of the Learnable,” which established the mathematical framework for machine learning with the simple yet profound concept of PAC learning.

“In the 1980s, the field of machine learning was in its infancy and largely empirical and ad hoc,” says Michael Kearns, a professor of computer and information science at the University of Pennsylvania. “With a series of models and results that combined technical depth with broad applicability, Les’ work created a common language and way of thinking about different machine learning problems, an achieve-

“The most critical choice for a scientist is what problems to work on.”

ment that is hard to overestimate.”

“The scientific question I isolated,” says Valiant, “was whether one could specify what it means for a mechanistic process to learn effectively. If we claim that a machine can do this, what exactly is it reasonable to ask of it? My definition of ‘probably approximately correct’ learning is such a specification. It is quantitative, and hence gives a way of comparing different learning algorithms with respect to how much computation they do, how many experiences they need, and how well they generalize. In that way it has encouraged the development of useful learning algorithms.”

In distributed computing, Valiant developed the Bulk Synchronous Parallel model. It is based on “supersteps” consisting of computation, communication, and synchronization, the various parameters of which can define the varieties of parallel computers and specify their throughput. “He articulated the questions of what does it mean to do parallel computation, and what is the interaction between hardware and software,” says Chayes. In so doing, Valiant created an analytic framework for the design of parallel systems.

With conventional computers, we have the von Neumann model, says Valiant, but for parallel processors there is no comparable unifying view.

“Different machines have very different characteristics as far as how many processors they have and how efficiently the internal inter-processor communication is realized,” he says. “This is a very big problem since it means that programs that run efficiently on one machine may be close to useless on another. I have tried to delineate the issues involved here and have worked toward finding an appropriate ‘bridging model’ that might give the sought-after single view and account for the variety of machines that one will inevitably find.”

Kearns observed Valiant up close as a Ph.D. candidate under him at Harvard in the late 1980s. “One strongly feels that his taste in problems is inherently personal,” Kearns says. “He picks very big and fundamental questions and pursues them quietly and rigorously until he has something important to say, and only then will you find him writing a usually landmark paper. In this sense he is immune to fashion, more so than any researcher I have ever encountered.”

About his knack of extracting the simple from the complex, Valiant says, “A basic question about science is whether everything important is really simple when looked at from the right perspective. I would generally answer yes. Of course the path to discovering the right perspective may be tortuous and highly technical. The simplicity and elegance of the solution is apparent only after it has been discovered.

“The most critical choice for a scientist is what problems to work on,” says Valiant, when asked about advice for future Turing Award winners. “The successful ones find problems of significance on which progress is possible using their particular skills.” 

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

© 2011 ACM 0001-0782/11/06 \$10.00

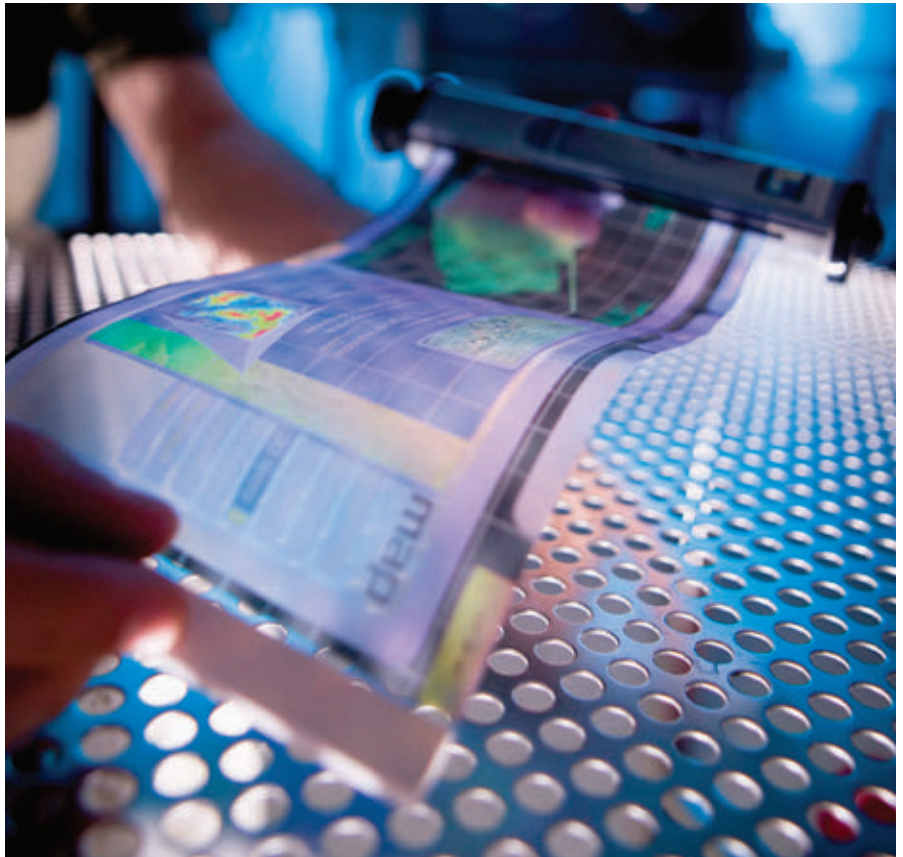
The Promise of Flexible Displays

New screen materials could lead to portable devices that are anything but rectangular, flat, and unbendable.

EVER SINCE SCIENTISTS first hacked oscilloscopes in the 1950s, computer displays have been heavy, fragile things. Even in a super-thin laptop, the quarter-kilogram screen is one of its heaviest parts—and the case required to protect it adds further bulk and weight. The screen also defines the form factor of most electronic devices as it's a single component that's rectangular, flat, and unbendable.

Now several developments portend an industrywide change to displays that are lighter, tougher, and more flexible than the sheet glass of a traditional liquid-crystal display (LCD). These displays are just beginning to appear on the market, and are produced with methods that may also improve energy efficiency, lower production costs, and allow for display shapes and sizes currently impossible to achieve.

Already these displays have started to appear in a variety of applications. The 75th anniversary issue of *Esquire* magazine gave them wide exposure by featuring e-paper that blinked “The future is now” on its front cover. Other uses already in the market include keyboards with auto-changing layouts, low-power shelf tags and point-of-sale ads, and display windows on credit-card-size smart cards. But designers are also rethinking product design around these new screens' possibilities. Carl Taussig, director of Advanced Display Research at Hewlett-Packard, notes the Dutch company Polymer Vision has demonstrated a cell phone with a roll-out display, and suggests other possible form factors. “You might have a display that you keep folded up like a piece of paper,” Taussig says. “You might open it halfway and use it that way, or you might open it all the way. You might have it partitioned with a keyboard on one part and a screen on the other.”



A flexible electronic display is rolled out at Arizona State University's (ASU's) Flexible Display Center. The unbreakable displays were developed by ASU and Hewlett-Packard Labs.

Front Plane, Back Plane

These new display technologies comprise two parts: a front plane that contains the imaging component, and a back plane that controls which pixels are on. For a display to be fully flexible, both parts must have that characteristic. While dozens of companies are competing to provide back planes, two front plane technologies have taken the lead.

One is electronic paper, or e-paper, like that used in the Amazon Kindle, Barnes & Noble Nook, and Sony Reader Digital Book. The producer of the e-paper in all three devices is the Massachusetts-based E Ink Corporation. Those

devices currently pair the flexible display layer with an inflexible glass back plane. E-paper is a reflective display; its brightness comes from ambient light hitting the display's face, as on a sheet of paper. Pixels in E Ink's e-paper are electrically dipolar units colored white on one side and black on the other, floating in thick oil. Their refresh rate is currently quite low, at about four frames per second. E-paper is bistable, retaining its image until it receives the next signal. As a result it consumes energy only when transitioning from one image to the next.

The other front plane technology is organic light-emitting diodes (OLEDs),

which emit rather than reflect light. OLED displays consume more power than e-paper, but are brighter and have faster refresh rates. They have a higher lumen-per-watt rating—that is, they use less power for comparable brightness—than comparable LCDs because the latter cause liquid crystals to block photons coming from a backlight. On the other hand, OLEDs actually emit light themselves.

Nick Colaneri, director of the Flexible Display Center at Arizona State University, believes OLED displays may eventually consume as little as 1/10th the power of LCDs, even as they deliver “a kind of intangibly superior image quality. Put side-by-side, OLED displays are often seen as strikingly better looking.” That makes them a likely challenger in the \$100 billion LCD industry, although OLED displays are extra difficult to ruggedize because they’re far more sensitive to oxygen and moisture. (E-paper is more likely to be used in applications where power and reliability trump brightness and contrast.)

Some manufacturers have proposed trying to capture both markets with hybrid transfective versions based on e-paper, similar to the Pixel Qi LCD screen on One Laptop Per Child’s XO computer. That screen includes both a backlight for low-light situations and a reflective layer for easy reading when the sun is out. Transfective displays with e-paper or OLED front planes are not currently available, but they’re possible because the back plane could be made from reflective or transparent materials.

With their similarity to LCDs, the potential rewards for OLED displays are enormous. At the same time, e-paper is defining new market segments, notably in portable electronics and large-scale signage. According to Jennifer Colegrove, vice president of emerging display technology at DisplaySearch, flexible plastic displays will grow at an annual rate of nearly 60%, surpassing \$8 billion in sales in 2018 from its current level of about \$300 million. Colegrove believes that although flexible displays will have fast growth in the next several years, they won’t become truly mainstream before 2018, both because of technical problems and the task of making them

Flexible displays will have fast growth during the next several years, but won’t become mainstream before 2018, says Jennifer Colegrove.

cost competitive with traditional glass-based displays.

Beyond Rectangular Thinking

Commercial products are available for both e-paper and OLED displays, including e-readers and advertising displays. Today, e-paper claims far greater market pull than OLED, largely because of e-readers. But Sriram Peruvemba, E Ink’s chief marketing officer, points to other uses for e-paper that weren’t possible with previous display technologies. “Most requests I get from design engineers are for rectangular displays, because that’s all they were able to get before,” says Peruvemba. “I tell them, ‘Think outside the rectangle.’ With our flexible display, you can cut it in any shape, so it’s only limited by imagination. Right now the device has to accommodate the display, but in the future the display will accommodate the device.”

Shape isn’t the only feature available to this new breed of displays but unavailable to LCDs. Because an LCD’s transparent electrodes must remain a set distance apart, they can’t be flexible, and are therefore made of thick, rigid glass. Transporting large sheets of glass is difficult and expensive, so LCDs are generally limited to a few square feet in size. But flexible displays can be made on material that’s stored and shipped in a roll, much like a web-fed paper press, enabling display sizes of a few meters wide by hundreds of meters long.

But with that flexibility are some practical problems. “How do you handle flexible substrates for TFT [thin-

In Memoriam

Philippe Flajolet Dies at 62

Soon after Philippe Flajolet passed away from a serious illness on March 22, tributes started appearing in an online book of tribute at INRIA from colleagues, former students, and the legions of computer scientists who were influenced by his contributions to the study of algorithms.

Those who knew Flajolet, an INRIA research director, best remember him as more than just an important theorist, however; he was a proudly independent researcher, as well as a gifted raconteur and free spirit who loved to play practical jokes on friends using complex mathematical formulae.

While working at INRIA, Flajolet earned the nickname “Algorithmix,” a nod to the popular Asterix books. As Richard J. Lipton noted in a blog post after Flajolet’s death, the nickname could hardly have fit more perfectly: “More than his development of particular algorithms, one can credit him much toward the development of *algorithmics* as a professional discipline.”

From his perch at INRIA, Flajolet devoted most of his career to studying the computational complexity of algorithms, the theory of average-case complexity, his transform-based asymptotic analysis, and the symbolic method, a novel approach to deriving the properties of combinatorial objects.

“The contribution of Philippe to the research on algorithms was essentially analytical,” says Micha Hofri, a computer science professor at Worcester Polytechnic Institute, “and even his algorithmic innovations, such as approximate counting of elements in multisets, came as the result of a mathematical insight.”

Hofri recalls meeting Flajolet at INRIA in the 1980s, when Flajolet wagered a bottle of champagne over whether one of Hofri’s analyses had already been completed by another researcher. Hofri lost the bet, but made a lifelong friend. “It was a good bottle,” he recalls.

—Alex Wright

film transistor] manufacturing?” Colegrove of DisplaySearch has asked. “Glass is easy: It’s a rigid sheet, and you just put layers on it. But with flexible displays it can be hard to register for the very accurate positioning you need. Also, some plastics deform when you heat them up, which can affect manufacturing. Therefore, several new processes have been invented.”

To address registration problems, some manufacturers have championed a roll-to-roll process that allows them to print electronics with greater precision than traditional photolithography allows. Photolithography is an “exposure” system: Light affects chemistry that etches electronics, similar to how an old-fashioned photo enlarger affects an image on paper. By comparison, the roll-to-roll process places electronics directly on a substrate and is more similar to a modern inkjet printer.

Hewlett-Packard’s explorations with unusual form factors led it to close a contract with the U.S. Army for wrist-mounted devices that would make critical information more easily available to soldiers in the field. As designed, the devices are only about 1.5mm thick and feature a screen that curves halfway around the wrist, the other half being a set of flexible photovoltaic cells to provide power. Taussig expects to deliver prototypes to the Army by this autumn, and sees viability for such devices well beyond its first customer.

“We imagine a lot of use cases,” he says. “[A flexible plastic screen] could be built into the cuff of a delivery driver’s uniform and include mapping information and the like. Or maybe it’s

for a mechanic or technician who’s assembling jet aircraft, and won’t have to go back to a table to see a schematic. Or a health-care technician or nurse could keep track of medication and patients and such. It’s very hands free. If it gets smashed into a door it’s not going to break. It’s very low power, and it’s light.”

Hewlett-Packard’s devices will be built with E Ink’s e-paper displays on a plastic substrate. But plastic wouldn’t work for OLED displays as its permeability would doom the sensitive OLEDs to a short life. For those applications, Corning Inc. is developing a kind of flexible glass by extending its proprietary fusion process, which it already uses to create thin glass substrates such as the Eagle XG line. While glass is necessarily heavier and more sensitive to certain stresses, Corning flexible glass commercial program manager Jill VanDewoestine believes the industry will respond well to this development because of glass’ distinctive qualities.

“Glass is really the standard,” VanDewoestine says. “Its surface is very smooth; it’s stable; high process temperatures are possible; it’s transparent; and it’s an excellent barrier against oxygen and moisture. So people really want to work with flexible glass—it gives them the handleability of roll-to-roll processes with the characteristics of glass that make high-quality electronics possible.” The company is currently developing flexible glass that’s 1/10th of a millimeter thick and can be wrapped around a core that’s three inches in diameter.

But whether incorporating glass,

plastic, or some other material, flexible display technologies represent the leading edge in changing the shape of computing devices. Consider the laptop computer, which in essence comprises a screen, keyboard, and processor. Flexible rubber keyboards are already available at big-box stores, and several manufacturers make “virtual keyboards” that project laser images of the keys onto any flat surface. Flexible plastic electronics are in active development, with the Russian corporation Rusnano recently announcing development of a \$700 million factory to produce such electronics. The age of the Dick Tracy wrist computer is already upon us. With flexible displays leading the way, there’s no limit to the shapes of computers to come. **C**

Further Reading

Chung, I.-J. and Kang, I. Flexible display technology—opportunity and challenges to new business application, *Molecular Crystals and Liquid Crystals* 507, Sept. 2009.

FlexTech Alliance for Displays & Flexible Printed Electronics
<http://www.flextech.org/>

Forge, S., Blackman, C., and Lindmark, S. Plastic promises: the disruptive potential of OLEDs and e-paper for the European display industry, *Foresight* 11, 3, 2009.

Jeong, J.K. The status and perspectives of metal oxide thin-film transistors for active matrix flexible displays, *Semiconductor Science and Technology* 26, 3, 2011.

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

© 2011 ACM 0001-0782/11/06 \$10.00

Milestones

American Academy Announces the Class of 2011

The American Academy of Arts and Sciences elected 212 new members in its Class of 2011, including nine computer scientists. As members of one of the most prestigious honorary societies and a center for independent policy research, the Class of 2011 includes some of the world’s most accomplished leaders in academia, business, public affairs, the humanities,

and the arts. Members contribute to studies of science and technology policy, global security, social policy and American institutions, the humanities, and education.

Newly elected members in the computer sciences are Edmund M. Clarke, Carnegie Mellon University; Edward W. Felten, Princeton University; Eric Horvitz, Microsoft Research;

Michael I. Jordan, University of California, Berkeley; Shree K. Nayar, Columbia University; Patricia Griffiths Selinger, IBM Almaden Research Center; Peter Williston Shor, Massachusetts Institute of Technology; and Avi Wigderson, Institute for Advanced Study. Leah H. Jamieson, Purdue University, was inducted in the section of engineering sciences

and technologies.

“It is a privilege to honor these men and women for their extraordinary individual accomplishments,” said Leslie Berlowitz, American Academy president. “The knowledge and expertise of our members give the Academy a unique capacity—and responsibility—to provide practical policy solutions to the pressing challenges of the day.”

Unlimited Possibilities

M. Frans Kaashoek discusses systems work, “undo computing,” and what he learned from Andrew S. Tanenbaum.

TO THE GREATER public, the people behind groundbreaking computational advances may be anonymous. However, the results of those advances are easy to show anybody, and within the computer science community, M. Frans Kaashoek believes there is a sense of camaraderie and mutual regard that assures a stunning pace of technological innovation.

“I think it would be fairly simple to explain how important this work is,” says Kaashoek, professor of computer science at the Massachusetts Institute of Technology (MIT), “because it is easy to see how dramatically society is changing due to the computer and innovations in computer science. People see their life change because they have Twitter or the iPhone. It’s quite clear that a lot of thought and innovation went into these technologies, and that this work is the result of a lot of people standing on other peoples’ shoulders.”

Kaashoek’s peers have recognized him as among those possessing the discipline’s brawniest metaphorical shoulders with the 2010 ACM-Infosys Foundation Award in the Computer Sciences. The award, which carries a \$150,000 prize and recognizes work by young computer scientists and systems developers, was awarded to Kaashoek for “his contributions to the structuring, robustness, scalability, and security of software systems underlying many applications.”

Kaashoek, 46, deflects celebratory ruminations about the ACM-Infosys Foundation Award in favor of praising his colleagues and students. However, Kaashoek does express an appreciation for the foundations of computational innovation, a love of programming, and a sense of virtually unlimited possibilities.

“Computers allow you to build almost anything,” he says. “Your imagination is the limit, not a physical prop-



M. Frans Kaashoek, recipient of the 2010 ACM-Infosys Foundation Award in the Computer Sciences.

erty. Of course, there are fundamental limits, but they are not really the ones that currently limit what you can construct out of hardware and software.”

Even where those fundamental limits loom, Kaashoek and colleagues have researched methods to improve the ways computers perform. His work on exokernels, for example, opened up possibilities for applications to access a computer’s physical resources

“Computers allow you to build almost anything. Your imagination is the limit, not a physical property.”

much faster than those that must employ high-level abstractions.

Kaashoek’s current research may have profound ramifications for the future of computational architectures. A recent experiment evaluating system applications on a 48-core Linux architecture revealed relatively simple coding changes, rather than wholesale operating system rewriting, may improve multicore system performance. In the security domain, Kaashoek is conducting research into “undo computing” with MIT colleagues Nickolai Zeldovich and Robert Morris. The basic premise is the exploration of “undoing” any actions caused by malignant code, while preserving any subsequently created, legitimate data.

Kaashoek acknowledges a system that would erase infected code and save good work would be a boon to the vast majority of computer users and administrators, but says his intended audience is the development community. His goal is to create well-functioning systems upon which robust innovation can flourish. And he is trying to pass along this love of systems work to his students as his mentor at Vrije Universiteit, Andrew S. Tanenbaum, did to him.

“I liked programming and building software artifacts, but had no idea how exciting research could be, and that a small team could build complete systems to demonstrate and evaluate an idea,” says Kaashoek. “In terms of advising, I learned a lot from Andy. He worked with a few students, allowing him to work with them in a true apprentice relationship. I have tried to follow that same model with my own students.”

A model that may also be strengthening the next generation of strong shoulders. 

Gregory Goth is an Oakville, CT-based writer who specializes in science and technology.

© 2011 ACM 0001-0782/11/06 \$10.00

All the News That's Fit for You

Personalized news promises to make daily journalism profitable again, but technical and cultural obstacles have slowed the industry's adoption of automated personalization.

IN 2006, a pair of entrepreneurs started a company called Outbrain with a promising premise: sifting through the massive volume of news available online to find what's most interesting to each reader. As Outbrain CEO Yaron Galai puts it, "You might have just read something that was a waste of time, and others feel the same way, so we could get some collaborative filtering going. From a reader's perspective, it's very tempting to solve this problem." But a year into product development, the Outbrain founders decided to change course because of a factor they hadn't appreciated: the sheer velocity of online news. "News goes away too quickly," says Galai, "and there's not enough time to collect deep-enough data to see who likes what and how it relates to other readers."

Delivering personalized news poses much harder problems than delivering personalized recommendations of books and movies as Amazon and Netflix do. Yet, despite the difficulties, personalized news seems all the rage these days. In February alone, *The New York Times*, *The Washington Post*, and Yahoo! all announced some form of automatic personalization, and Google is quietly running its own experiments in personalized news delivery.

These companies, along with several startups, see the revenue potential of giving each user more of what he or she wants. "I don't think it's rocket science to say that people who read the product more frequently and more thoroughly are more likely to subscribe to it than those who do not," says Marc Frons, the chief technology officer at *The New York Times*, which earlier this year started limiting the number of articles non-subscribers can read for free.

Joshua Benton, who directs Har-



The New York Times and other media companies are implementing personalized news delivery.

vard University's Nieman Journalism Lab (mission: "to help journalism figure out its future in an Internet age"), agrees that personalization offers enormous business potential. "*The New York Times* has well over a decade of data about what stories I've read, how many seconds I've spent on each story, and what sections I've read, so you would think they would be able to tailor my experience in a way that would be more pleasing to me," Benton says. "As a result, the page becomes a more valuable piece of property to an advertiser." Regardless of how the *Times'* paywall pans out, more advertising revenue would be particularly welcome in an industry whose sharply declining print circulations have led to decreases in ad sales and, in many cases, the death of entire newspapers.

But despite the promise of algorithmic personalization, the idea is far

simpler in theory than in practice, and newspapers have struggled to figure out how to do it without giving up their traditional role as arbiters of news.

"Computer scientists may think it's nirvana to get what you want to get," says Penelope Abernathy, professor of digital media economics at The University of North Carolina at Chapel Hill. "But a newsperson will say, 'My role is to bring you the world, and it may be news you didn't know you needed to know.'"

Print journalists don't usually view themselves as mere news retailers but as crucial players in democratic society—and news personalization threatens to erode that function. As in the balkanization of culture in general—from niche books and movies to ideologically focused television news channels—personalized news risks doing the opposite of what newspapers have traditionally done: Instead of uniting people through a common discourse, personalization may make readers ever more insular.

A Hybrid Approach

The solution at *The New York Times* has been a hybrid approach. The site is supplementing its home page, with its standard mix of editor-selected content, with its recently introduced Recommendations page, which shows a ranked list of stories each logged-in user might find interesting based on his or her reading history.

Industry analyst Ken Doctor, a veteran journalist and the author of *News-onomics*, says most newspapers' inertia stems in part from a lack of expertise. While editors focus on bringing readers the world, and the business side sells ad space around the articles, no one is left to tackle the considerable problem of delivering customized content.

"Newspaper companies aren't very tech-savvy," says Doctor, noting that

even their upper management usually has an operations background.

The New York Times, with its deeper resources, is showing itself to be an exception; but even so, its foray into automated personalization is still rudimentary. “We spend a lot of time and effort tagging content,” says Frons, referring to the system of semantic tags that tell the recommendations engine what each story is about. The site keeps track of your reading habits at the rather coarse level of these tags—noting, for example, that my most-read topics include “Demonstrations, Protests, and Riots” as well as “Television” and “Search Engines”—and makes recommendations accordingly. Apart from ignoring stories you’ve clicked on inadvertently, the *Times* recommendations engine doesn’t consider how long you’ve spent on each story or how much you liked a piece.

Furthermore, since the *Times* does not use collaborative filtering, it has no way of discovering your latent interests, since these aren’t revealed in the semantic tags. What’s more, because the engine looks only at articles, and not at your search history, it can’t suggest stories on topics you care about that the *Times* hasn’t previously covered. However, Frons says the challenge is less about the underlying technology and more about delivering the kind of user experience that readers expect from the paper of record. “I think *The New York Times* is probably unique among many publishers and content providers online in that people who come to us do so not only for the individual articles but also for our news judgment—for what the *Times* thinks are the most important stories of the day.”

The *Times* is not entirely special in this regard, though. Neil Thurman, senior lecturer in the graduate school of journalism at City University London, heard a version of Frons’ sentiment from many of the editors he surveyed for his recent study about the use of personalization at 11 national news sites in the U.S. and the U.K. “The editors said to me that they felt users had, to an extent, personalized their news experience by choosing what newspaper to buy or which site to visit,” he says. Automated recommendations can, depending upon how they are

implemented, sometimes break down the news organization’s editorial voice, as exemplified by an incident on the right-wing *Telegraph* Web site, when the site linked out from its pages on the environment to several liberal blogs and Web sites. “Those decisions about linking were run algorithmically, so you got some surprising links being generated,” Thurman says.

The most established companies have the most to fear from unwelcome surprises—and even those with state-of-the-art technology, like Yahoo!, are proceeding cautiously. “The footprint of what we do is very broad” [more 100 million daily visitors on the Yahoo! home page alone], “so we want to take careful steps,” says Raghu Ramakrishnan, the company’s chief scientist for search and cloud computing.

The technical obstacles are monumental—from the scalability challenges of combing through terabytes of daily click logs on thousands of servers worldwide to the difficulty of learning from nearly real-time feedback. The Yahoo! CORE algorithm, which is at the heart of the matchmaking process between content and users’ interests, starts with the system’s prior knowledge about users, but then proceeds to learn more about how each user will respond to a particular article by using explore-and-exploit strategies on a slice of the user base.

“What to exploit is easy,” says Ramakrishnan, “but what to explore is where the secret sauce comes in.”

The CORE algorithm must simultaneously optimize for a complex mix of variables, figuring out the right trade-off between click-through rate and engagement, between getting a broad-enough sample of users and not wasting these users’ attentional bandwidth, and between different types of ads. Thanks to CORE, says Ramakrishnan, the click-through rate on articles is already up more than 160% compared to what it was based on human editorial judgment alone. Despite this feat of algorithmic personalization, Ramakrishnan is mindful of the need for human judgment, which can help avert machine-learning problems like overfitting. “If I keep showing you what I think you want to see,” explains Ramakrishnan, “there’s a danger that I might get very narrow in what I show you.”

Yahoo! gives editors “dashboards,” decision-support tools to help them make sense of the ever-changing analytics and select particular stories for particular groups of users. As for individual personalization, Yahoo! has yet to begin testing it on either its home page or its news page, starting instead with lower-profile properties like its sports page. “We’re being cautious because we don’t want to compromise our users’ trust,” says Ramakrishnan. Google, too, is still at the experiment stage, showing some users news customized to their clicking history.

It’s too soon to tell if personalization will save the journalism business, but it will certainly provide a reality check for journalists. Sree Sreenivasan, digital media professor at Columbia University’s Graduate School of Journalism, recalls a time not long ago when, as a reporter, you could say that a million people read your story because your newspaper had a million circulation. “You fooled yourself,” Sreenivasan says. **G**

Further Reading

Agarwal, D., Chen, B.-C., and Elango, P. Explore/exploit schemes for Web content optimization, *Proceedings of the 2009 9th IEEE International Conference on Data Mining*, Washington, D.C., Oct. 6–9, 2009.

Kamba, T., Bharat, K., and Albers, M.C. The Krakatoa chronicle—an interactive, personalized newspaper on the Web, *Proceedings of the 4th International World Wide Web Conference*, New York, NY, Dec. 11–14, 1995.

Li, L., Chu, W., Langford, J., and Schapire, R.E. A contextual-bandit approach to personalized news article recommendation, *Proceedings of the 19th International Conference on World Wide Web*, New York, NY, April 26–30, 2010.

Liu, J., Dolan, P., and Rønby Pedersen, E. Personalized news recommendation based on click behavior, *Proceeding of the 14th International Conference on Intelligent User Interfaces*, New York, NY, Feb. 7–10, 2010.

Thurman, N.J. Making “the daily me”: Technology, economics and habit in the mainstream assimilation of personalized news, *Journalism: Theory, Practice & Criticism* 12, 4, May 2011.

Based in San Francisco, **Marina Krakovsky** is the co-author of *Secrets of the Moneylab: How Behavioral Economics Can Improve Your Business*.

© 2011 ACM 0001-0782/11/06 \$10.00

Privacy and Security Identity Management and Privacy: A Rare Opportunity To Get It Right

The National Strategy for Trusted Identities in Cyberspace represents a shift in the way the U.S. government is approaching identity management, privacy, and the Internet.

SINCE 1976, WHEN Whitfield Diffie and Martin Hellman first surmised the possibilities for the potential uses for digital signatures,¹ there has been ongoing discussion of building an online identity management structure. As use of the Internet has become more central to daily life and our financial and physical security has become intertwined with cyber security, the calls to authenticate and identify individual users have increased. However, we still have not seen a single set of answers to these issues that offer a path to an interoperable identity management system that will achieve the goals of authenticating users at different levels of risk, keeping the Internet as an innovative and growing hub for the world's interactions, and building trust among Internet users. Therefore, it is easy to be doubtful and even cynical that we can build an identity management infrastructure that is

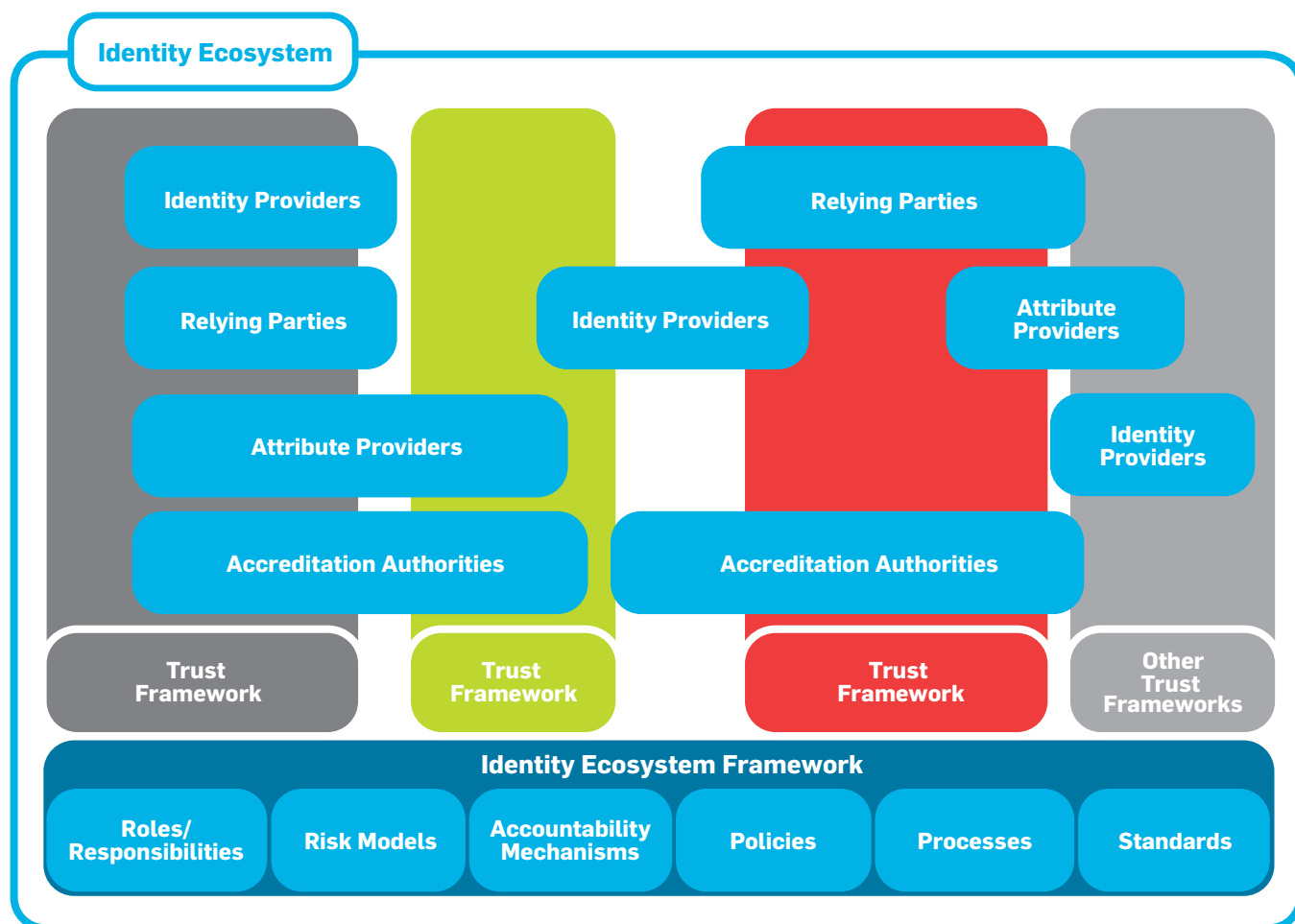
NSTIC represents only one pillar in an overall approach to resilient and effective cyber security, but it is an essential component to overall success.

voluntary, privacy-protective, secure, and interoperable. Over the next few years, we have a rare opportunity to build such a system, and this opportunity may be our last.

Many countries have put forward online identity management strategies tied to centralized databases and national ID cards, but another path was clearly laid out in an important 2004

article entitled "The Accountable Net." The authors, John Palfrey, David Johnson, and Susan Crawford, suggest the current threats on the Internet create an unsustainable situation where we risk losing the benefits of the Internet's decentralized structure. They urge us to find ways of building an Internet governance that makes peers accountable to one another as that risk is lower than the risk of empowering either an existing government or building a centralized global authority. Instead of fearing the change that will take place as the Internet becomes more accountable, those of us that love its current structure must embrace change, but also "keep the fundamental architecture and values of the Internet in mind as we do so."²

President Obama recognized these concerns in the release of the May 2009 Cyberspace Policy Review (CPR), which outlined the steps the public and private sector should take to overcome the risks associated with online transactions. These actions included



Multiple trust frameworks built on the foundation of the Identity Ecosystem Framework.

improving identity solutions, services and privacy-enhancing technologies, and enhancing protection for individuals' online information. In response to the short-term actions laid out in the CPR, the National Strategy for Trusted Identities in Cyberspace (NSTIC) was created and was signed by the President earlier this year; see <http://www.nist.gov/nstic> for details.

As the President and his Cyber Security Coordinator have noted, NSTIC represents only one pillar in an overall approach to resilient and effective cyber security, but it is an essential component to overall success. It seeks to enhance online trust by focusing on establishing identity solutions that improve our ability to identify and authenticate the organizations, individuals, and underlying infrastructure (such as routers, servers, desktops, mobile devices, software, data) involved in online transactions. NSTIC is not only about credentials and authentication. It also seeks to limit the amount of identity-

ing information that is collected and transmitted during the course of on-line transactions. This concept is clearly articulated in the NSTIC vision statement: "individuals and organizations utilize secure, efficient, easy-to-use, and interoperable identity solutions to access online services in a manner that promotes confidence, privacy, choice, and innovation."

The Identity Ecosystem

In order to realize the vision, the strategy outlines a next generation of a trusted identity environment, the Identity Ecosystem, where individuals and organizations can operate with trust and confidence through abiding by standards and policies for identifying and authenticating their digital identities.

Four Guiding Principles establish the framework for participation in the Identity Ecosystem and form the foundation for the strategy:

- Identity solutions will be *voluntary and privacy-enhancing*. Individuals may

choose among multiple identity providers—both private and public—and among multiple digital credentials, precluding the possibility of the creation of a national ID card or single ID database. The Identity Ecosystem will support a range of solutions that will enable limited data collection and use and distribute only relevant and necessary information about users. To accomplish this goal, it is essential to match the level of authentication to the level of risk associated with the particular transaction. The Identity Ecosystem must maintain appropriate safeguards on information and be responsive to individuals' privacy expectations tied to globally recognized Fair Information Practice Principles.^a

^a See a discussion of modern Fair Information Practice Principles in the U.S. Department of Commerce Green Paper entitled "Commercial Data Privacy and Innovation in the Internet Economy: A Dynamic Policy Framework" at http://www.ntia.doc.gov/reports/2010/IPTF_Privacy_GreenPaper_12162010.pdf

► Identity solutions will be *secure and resilient*; they must stand against attack or misuse. In the Identity Ecosystem, solutions will provide reliable methods for electronic authentication that are resistant to theft, tampering, and exploitation. At the same time, credentials must be able to recover from loss or compromise and be adaptable to the dynamic nature of cyberspace itself.

► Identity solutions will be *interoperable*. They will be easily used by a wide variety of service providers, and they will be scalable across various boundaries, including geography, semantics, and policy.

► Identity solutions will be *cost-effective and easy to use*. Future identity solutions should help to reduce the complexity and the risk associated with managing multiple credentials, especially for individuals. Identity solutions should be simple to understand and enabled by technologies that are easy to use and require little training.

In order to reach these goals, not only does the government need the help of the private sector, but also the private sector must lead its implementation. In January 2011, the U.S. Department of Commerce was named the head of a National Program Office for NSTIC. As a non-regulatory agency, Commerce's job as the lead on NSTIC is to promote voluntary private sector cooperation to facilitate the growth of this Identity Ecosystem in a peer governance model similar to that recommended in "The Accountable Net."²

To do so, the U.S. Department of Commerce will promote private-sector involvement and engagement; build consensus on legal and policy frame-

Presidential calls on IT issues, with pilot funds behind them, do not come every day.

works necessary to achieve the vision, including ways to enhance privacy, free expression, and open markets; work with industry to identify where new standards or collaborative efforts may be needed; support inter-agency collaboration and coordinate interagency efforts associated with achieving programmatic goals; and promote important pilot programs and other implementations.

There is no panacea or magic bullet to solve all cyber security issues, but leadership on an identity management infrastructure can build trust, can improve security and, if done properly, can enhance privacy, but it must be led by the same type of innovators that have made the Internet what it is today. There will be an opportunity for anyone interested to participate and, it is essential that those of us that care about the future of the Internet do so if we are to be successful.

Possible Future Scenarios

I realize many will read this call to participate in NSTIC and think they have heard this concern before, but in this case, I urge everyone involved in related areas to think long and hard about the future of identity management, privacy, and the Internet. There are only a few possible future scenarios.

First, we continue on our current path. In other words, we stumble along. The market may have some good ideas on authentication that address some important Internet values: they probably will be voluntary; they may or may not protect privacy; they may be open or may be completely proprietary. In the meantime, we can expect at least a few more decades of inefficiencies, lost opportunity costs, and heavy fraud losses.

Another vision would be that governments will not wait decades and will work together or separately to begin to require identity management solutions. Although these solutions are likely to be more privacy protective, they are also likely to be more prescriptive schemes that may raise costs and make it more difficult to deliver products and services consumers want to use.

Finally, there is the NSTIC, an organized attempt to address as many issues as possible before they arise taking the leadership of the private sector and teaming it with consumer protection input from the government. It also may succeed or fail. Success has clear benefits to those who would like to ensure important aspects of today's Internet and protect privacy. Failure will put us back into one of the other scenarios.

With these options, it is clear which path is better for innovation, better for privacy, and better for openness. Certainly, in the age of identity theft, any project regarding identity bears close watching no matter who is running it or how it is run and NSTIC is no different in that respect.

Presidential calls on IT issues, with pilot funds behind them, do not come every day. There may indeed be other opportunities to develop a similar means to address related security issues while maintaining the critical values of Internet openness and privacy, but that is not a wager I am willing to make. In short, we have a chance to make a difference now. I hope you will join me and work with the U.S. Commerce Department and private sector leaders to take advantage of what may just be the last best chance to get the governance for the future of online authentication right. □

References

1. Diffie, W. and Hellman, M. New directions in cryptography. *IEEE Transactions on Information Theory* IT-22, 6 (Nov. 1976), 644–654.
2. Johnson, D.R., Crawford, S.P., and Palfrey, J.G. The accountable Net: Peer production of Internet governance. Berkman Center for Internet & Society at Harvard Law School *Virginia Journal of Law and Technology* 9, 9 (2004); <http://ssrn.com/abstract=529022> or DOI:10.2139/ssrn.529022.

Ari Schwartz (ari.schwartz@nist.gov) is Senior Internet Policy Advisor at the U.S. National Institute of Standards and Technology in Gaithersburg, MD.

Copyright held by author.

In the age of identity theft, any project regarding identity bears close watching no matter who is running it or how it is run.

The Profession of IT Who Are We—Now?

Considerable progress has been made toward the formation of a computing profession since we started tracking it in this column a decade ago.

IN 2001, WE launched this column because ACM was interested in tracking the evolution of the computing profession and wanted to explore ideas of potential value to the professional. Let us take stock of where we have come in the past decade and what it means for you.

In the late 1990s, ACM leaders realized the computing field had matured to the point where members were becoming interested in ACM support for their professional activities. ACM wanted to assure itself that there is a deep community need and adequate depth in the existing organization. We asked: Is there a need for a comput-

ing profession? What will it be called? What are its components?²

We argued from first principles that professions form when considerable expertise is needed to take care of people's enduring concerns in a domain. This is plainly true of long-standing professions such as health and law. Computing has become such an integral part of business and everyday life that the reliable operation of ever-advancing computing technologies is an enduring concern. Considerable expertise is needed to properly address this concern. Thus, a profession must inevitably form.

ACM already had considerable depth in professional activities in the

various specialties covered by the SIGs. Other professional societies hosted professional groups around other specialties, and corporate universities were offering certifications in some occupational specialties. We counted over 40 organized professional groups and inventoried them in three categories: IT core, IT intensive, and IT occupations (infrastructure). Table 1 is the inventory with a few additions that account for changes since that time. Although the table is not exhaustive, it clearly shows the breadth and depth of the field. Several new specialties that appeared since then are shown in bold type.

At the time, IT (for information technology) seemed to be a broader term than computing, and thus the label "IT profession" seemed to be a good umbrella for all the professional activities listed in Table 1.

Elements of a Profession

In a remarkable 1996 study, Gary Ford and Norman Gibbs laid out a map of the essential elements of a mature profession³, and validated their map against a number of existing professions including health, law, and architecture. Their purpose was to lay out a path to maturity for the software engineering profession. Their map is summarized in Table 2.

Let us define certain terms used in the map and in our subsequent discussions. A *profession* is a community of practice that forms to take care of people's enduring concerns in some area of life or work. A *professional* is a member of that community who renders service to clients of the profes-

Table 1. Professional subdivisions of the computing field.

Computing-Core Disciplines	Computing-Intensive Disciplines	Computing-Infrastructure Disciplines
Artificial intelligence	Aerospace engineering	Computer technician
Cloud computing	Bioinformatics	Database administrator
Computer science	Cognitive science	Help desk technician
Computer engineering	Digital library science	Network technician
Computational science	E-commerce	Professional IT trainer
Database engineering	Genetic engineering	Security specialist
Computer graphics	Information science	System administrator
Cyber security	Information systems	Web identity designer
Human-computer interaction	Public policy and privacy	Web programmer
Network engineering	Instructional design	Web services designer
Operating systems	Knowledge engineering	
Performance engineering	Management info systems	
Robotics	Network science	
Scientific computing	Multimedia design	
Software architecture	Telecommunications	
Software engineering		

sion. *Knowledge* refers to the organized body of knowledge that professionals rely on. *Practice* refers to the skills displayed by professionals at various levels of competence as they render service. Practicing professionals are often called practitioners.

It is important to note that the elements of the map are not the point of the profession; service to clients is. The map elements are the means.

It is useful to define three other terms. A *craft* refers to a set of practices shared by a community of practice, but it has no special social status. Wood crafting and programming are examples. A *trade* refers to an organized group of practitioners, such as a guild or labor union, with restrictions imposed by society in return for freedom to practice for the benefit of society. At this time, there are no computing-specific trades. A *discipline* refers to a field of study and research that provides knowledge. Chemistry and computer science are examples. Professions include crafts and trades and rely on one or more disciplines. Disciplines are concerned intellectual knowledge, professions with performance and practice.

What Has Changed

The term information technology has not achieved the all-encompassing character we once envisioned. Today, IT refers mainly to technology and business applications of computing. Computing is now the preferred umbrella term.

As a sign of the field’s continued vitality, several important new specialties have appeared. These include cyber security, cloud computing, network science, and Web programming. We expect expansion of the computing profession to continue.

Certification is becoming more common, mainly at the sub-profession or specialty level. A growing number of computing jobs require specialized certifications such as Microsoft or Cisco network engineer. Many organizations, including the U.S. Department of Defense, now require that cyber security personnel have the CISSP certification (certified information systems security professional). The American Society for Quality now certifies Software Quality Engineers via their CSQE program, and the IEEE-CS certifies

Software Development Professionals through their CSDP.

Licensing of professionals has proceeded more slowly. Licensing has begun to appear in engineering specialties of computing, where concerns for health and safety are strongest. Computer engineers have been licensed in the U.S. for several years. Software engineers have been licensed in Canada, Australia, and Great Britain for some time. In the U.S., the NCEES (National Council for the Examination of Engineers and Surveyors), with considerable technical help from the IEEE-CS, is developing a licensing examination for software engineers in response to a request from 10 states. ACM has generally not supported licensing and has taken no official position on certification.

Starting in 2002 there was a sharp drop worldwide in the numbers of students enrolling as majors in computing disciplines. This was a great puzzle to computing leaders because demand for computing professionals in the job market was (and continues to be) very high. That concern led government agencies, such as the National Science Foundation in the U.S., to support programs to generate more student interest in computing in colleges and universities. Much of the concern over the educational pipeline has shifted to the K–12 arena, because most high school students have no access to any computing course, serious problems with the unpopular Advanced Placement curriculum have led to its cancellation, and budget shortfalls have blocked computing from being added as a new program.

ACM has built an impressive array of programs that support computing professionals. It spent a lot of money to scan the entire ACM literature, back

to its founding in 1947, into the Digital Library. It has entered extensive licensing agreements with businesses and consortia so that almost all employed professionals have access to the Digital Library through their employers. It has enhanced the Digital Library with individual favorite folders and technical reading packs. It has sponsored more conferences for practicing professionals. It created the Profession Board to watch over the interests of professionals in ACM. It created more publications for practicing professions, mostly notably *Queue* and its new linkage with *Communications*. It created the online publication *Ubiquity* to explore the future of computing and the people who are inventing it. It has cooperated with IEEE Computer Society and others in joint programs for professionals, notably curriculum and accreditation guidelines.

Open Issues

We noted previously that most of the certification and licensing activity has been focused on sub-professions of computing. So too, most of the growth in bodies of knowledge, codes of ethics, education, and training programs has been in specialties. This is similar to what has happened in other professions. The health profession is a good example. That profession has established conventions, standards, core knowledge, and common terminology for the whole profession. Most health certifications and training programs focus on specialties. Nurses, doctors, and technicians get accredited degrees in their fields, and then obtain certifications and training for the specific specialties in which they practice.

What can we learn from analogies to medicine, architecture, and other

Table 2. Elements of a mature profession.	
Practitioner Level	Infrastructure Level
Professionals	Initial professional education
Knowledge	Accreditation
Professional practice	Skills development
	Certification
	Licensing
	Professional Development
	Code of ethics
	Professional societies

professions? It is striking that each profession has its own unique model. The medical profession, the engineering profession (which overlaps with the computing profession), and the architecture profession have very different solutions to the elements in Table 2. Computing has not developed good solutions to all the elements in Table 2 because it has not settled on a model or paradigm for itself. If history is a guide, this will not be an easy path. The interdisciplinary character of computing and its overlap with and use by so many other professions will require a degree of nimbleness that may be unique among professions. As we work to build our profession, we can set our sights on some of the characteristics of the most admired professions, which will surely include: well defined and widely accepted terminology and core knowledge; an infrastructure that keeps the various parts of the profession aligned and reasonably consistent with each other; and, ultimately, a structure that the public—whom we serve—will grow to respect, admire, and count on.

Neville Holmes in his *Computer* magazine column, the counterpart of this column, has raised a number of issues worth noting.⁴ He is concerned that we do not seem to have standard definitions of key terms in computing such as data, information, computation, programming. Further, he postulates, with some grounding, that many practitioners do not care about being identified with a profession. He is concerned about practitioners who believe their programs are the end products of their work; as professionals, they should be concerned with their clients—how the program affects clients goes on long after the program is delivered. Are computing practitioners non-professional if they have such an attitude? Should we be instilling a sense of professionalism in our students, regardless of their specialties?

What This Means for You

All the above looks like it would be interesting to organizational leaders looking to build useful professional society programs. But what about the “ordinary practicing person?” Why should you pay attention to this?

Writing in the *Chronicle of Higher Education*, Kevin Carey, the director

of education for a Washington, D.C., think tank, gave a beautiful answer to this question.¹ He himself started out to join the computing profession and became a good programmer. But his true interest turned out to be bringing computing to the education of young people. The practice of programming gave him a discipline of precision and conciseness that has helped him in his current profession. He says, “I left computer science when I was 17 years old. Thankfully, it never left me.”

A particularly insightful benefit of Carey’s computing knowledge was how it enabled him, as an employee of the Indiana state-budget office, to help rewrite Indiana’s school financing law to make it substantially simpler and easier to administer. Seeing it as similar to a large program that had grown too hoary with age, he “sat down, mostly as an intellectual exercise, to rewrite the formula from first principles.” Many other laws and government structures could benefit from such an approach.

Even if you do not have a professional certification, you are probably operating in an environment where you have customers with various concerns around computing. You have the expertise to help them and you want to help them. You want to conduct yourself ethically and provide benefits to society as well as your clients. In short, you want to be a professional. We hope the map we have outlined here can help you find the services to assist you doing these things. 

References

1. Carey, K. Decoding the value of computer science. *Chronicle of Higher Education* (Nov. 7, 2010); <http://chronicle.com/article/Decoding-the-Value-of-Computer/125266/>
2. Denning, P.J. Who Are We? *Commun. ACM* 44, 2 (Feb. 2001), 15–19.
3. Ford, G., and Gibbs, N. A mature profession of software engineering. Software Engineering Institute, Carnegie-Mellon University, 1996. Available for download at <http://www.sei.cmu.edu/library/abstracts/reports/96tr004.cfm>
4. Holmes, N. The future of the computing profession. *IEEE Computer* (Oct. 2010), 98–100.

Peter J. Denning (pjd@nps.edu) is Distinguished Professor of Computer Science and Director of the Cebrowski Institute for information innovation at the Naval Postgraduate School in Monterey, CA, and is a past president of ACM.

Dennis J. Frailey (frailey@lyle.smu.edu) recently retired from Raytheon Company as a principal fellow. He is an adjunct professor at Southern Methodist University, a member of the IEEE-Computer Society Board of Governors, and a past vice-president of ACM.

Copyright held by author.

Calendar of Events

June 15–17

16th ACM Symposium on Access Control Models and Technologies, Innsbruck, Austria, Sponsored: SIGSAC, Contact: Dr. Ruth Breu, Email: ruth.breu@iuiibk.ac.at

June 15–17

Web Science 2011, Koblenz, Germany, Sponsored: SIGWEB, Contact: Ruth Ehrenstein, Email: ruth.goetten@uni-koblenz.de

June 20–23

The 10th International Conference on Interaction Design and Children, Ann Arbor, MI, Contact: Tom Moher, Email: moher@uic.edu

June 20–24

16th International Conference on Reliable Software Technologies, Edinburgh, UK, Contact: Steve Riddle, Email: steve.riddle@ncl.ac.uk

June 20–21

International Workshop on Enterprise & Organizational Modeling and Simulation, London, United Kingdom, Contact: Joseph Barjis, Phone: 347-410-9121

June 20–22

The 16th International Symposium on Web3D Technology, Paris, France, Sponsored: SIGGRAPH, Contact: Dr. Jerome Royan, Email: Jerome.royan@orange-ftgroup.com

June 22–24

Shape Modeling International 2011, Herzliya, Israel, Contact: Ayellet Tal, Email: ayellet@ee.technion.ac.il

June 22–25

Designing Pleasurable Products and Interfaces, Milano, Italy, Contact: Francesco Zurlo, Email: Francesco.zurlo@polimi.it



The Business of Software

Practical Application of Theoretical Estimation

Estimation models at the extreme.

ONE OF THE most popular and successful approaches to estimating software projects is the Putnam model. Developed by Larry Putnam, Sr., in the 1970s as the Software Lifecycle Model (SLIM®)³, this model shares with other models a reciprocal exponent relationship of effort/cost to development time. That is, effort/cost is given by a formula of the

type: **Effort or Cost** $\sim$ **Time**ⁿ (see Figure 1). The value of **n** varies between models. In the SLIM model it is 4. This reciprocal fourth-power relationship predicts an enormous increase in cost and effort (and the associated project headcount) when a project's development duration is compressed substantially. It also shows that there are great savings to be gained if a project's delivery schedule can be relaxed.

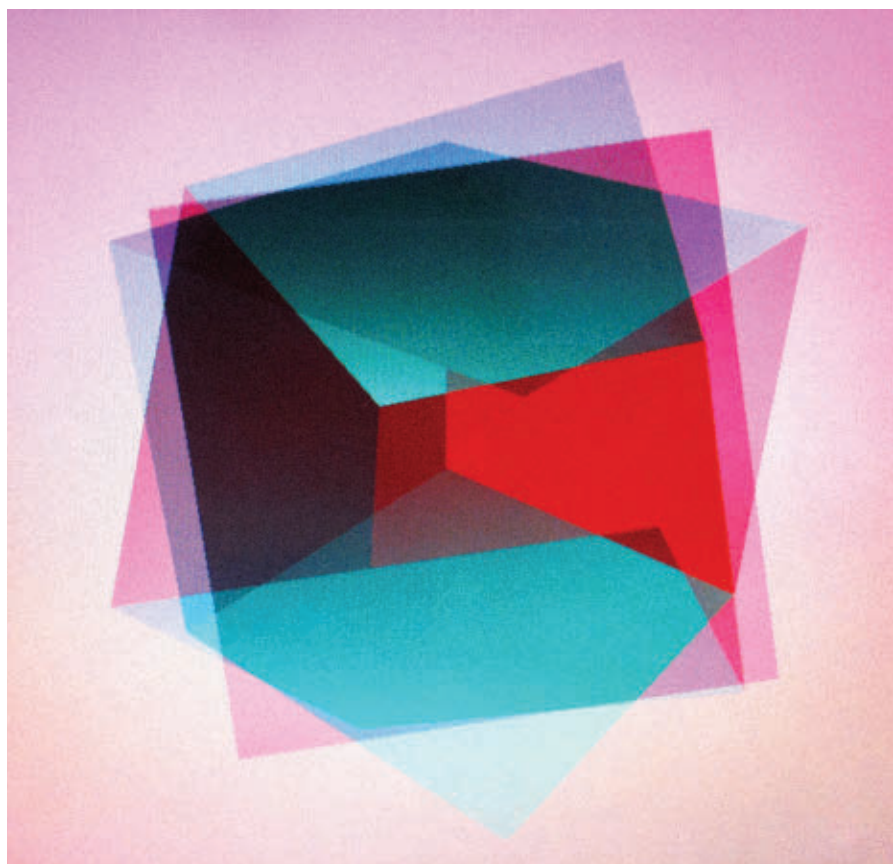
I suggested a qualitative explanation for this in a previous column¹—the enormous increase in effort with very short schedules is due to the distribution of much of the work on the project into a wasteful activity I called “optional chaos.” When projects have relaxed schedules, they don't have much of this valueless overhead so a much higher percentage of their work goes into actually creating value in the product.

Theory Goes on Forever

The theoretical curve could extend out to infinite time. In practice, the SLIM-Estimate tool (see <http://www.qsm.com>) that employs this mathematics has constraints built into it and will not allow infeasible solutions that are too short/high cost or too long/low cost. Furthermore, the application of any theory must always be subject to real-world considerations: if a short-cycle solution requires more staff than the company can get in that time it is de facto infeasible irrespective of any model or calculation. And if a long duration project exceeds its market window, it might be cheap, but no one will buy the result. In between these unattractive options, the time-effort curve very usefully demonstrates the trade-off between the cost of a project and how long it will take to complete the project.

Regression Models

Other estimation models that have been created using different concep-



tual approaches or from empirical data support this reciprocal exponent model. However, several of them, past a certain point in project duration, show either a flattening out (no further decrease in cost) or predict a modest *increase* in cost and effort as the project timeline extends further.^a At the other, compressed, end of the timeline, increases in project headcount and effort past a certain point have been anecdotally thought to *increase* the project schedule. To paraphrase Fred Brooks²: adding people to a project can make it take longer (Brooks was talking about *late* projects, but we can extend to projects that will be late). It seems as if, beyond certain points at each end, the observed project behavior curls away from the conceptual curve, as illustrated in Figure 2. Along the middle of the curve (which is where most projects will be and should be committed), the theoretical and practical curves are congruent and projects tend to behave as predicted. And this is what QSM has found in over three decades of tracking such projects. At the extreme ends, projects may not behave as they should.

Outliers: Compressed and Relaxed

In any multivariable stochastic system, such as a collection of software projects, there are always outliers. Projects that we try to do very quickly or very slowly are by their nature, different from the norm. At the compressed end, it is likely the curve curls over simply due to the overwhelming preponderance of the “optional chaos” noise. In this case, all of the additional work on the project is being absorbed by the effort involved in managing the sheer numbers of people—indeed additional work and time is needed just to manage the noise. It is analogous to a multi-tasking processor being overloaded where all the machine cycles are going into switching between tasks and none into the tasks themselves. Adding more people to try to further reduce the schedule simply pushes the

schedule out further (as well as costing a lot more).

At the “relaxed” end of the timeline, the project cost rises the further out we try to position the product delivery. The reasons for this are usually in the marketplace and external to the project, though an extended timeframe might increase the likelihood of turn-

over of staff or change in technology, which can add to cost. As the delivery time moves further out, there is a higher probability that the underlying reasons for the project will change. When the project tries to play catch-up to the changing requirements, they find they have to rework more of the product thereby increasing costs.

Figure 1. Relationship of effort/cost to development time.

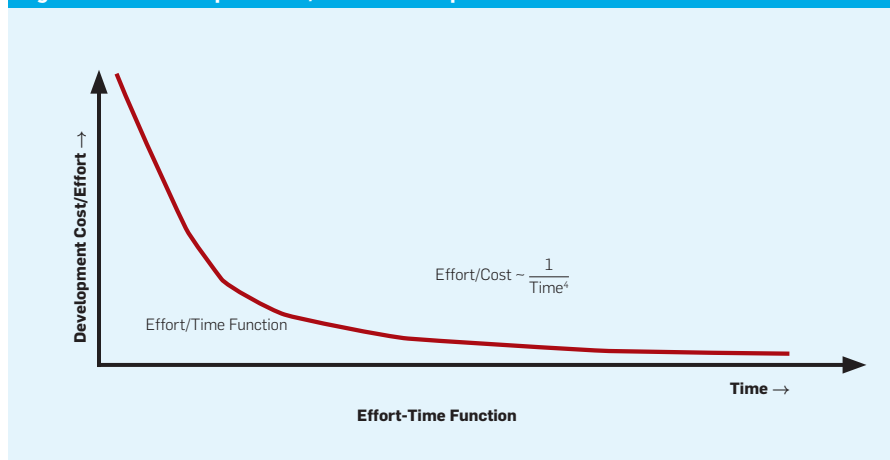


Figure 2. Project behavior curling away from the conceptual curve.

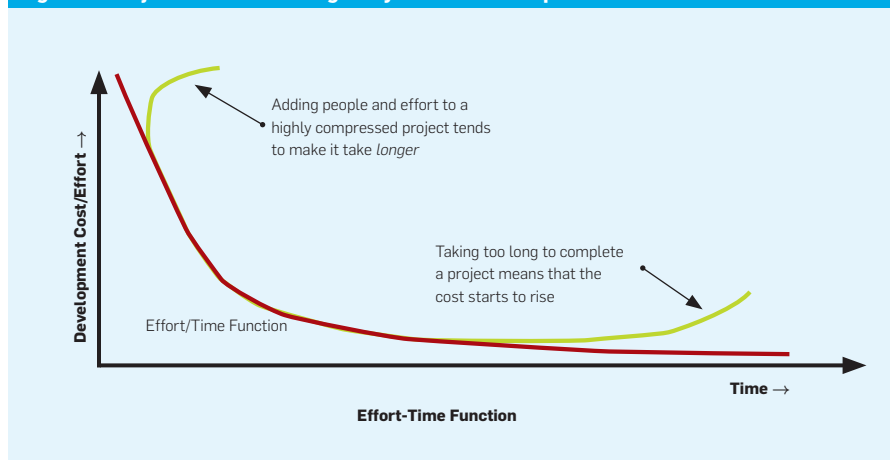
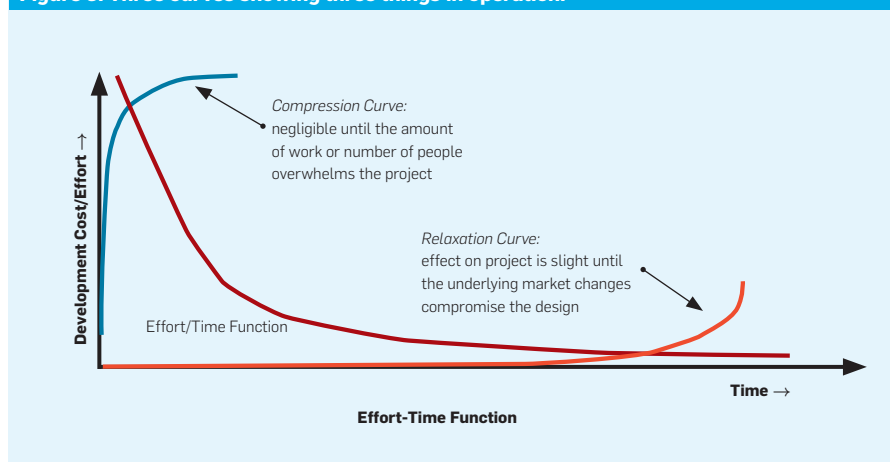
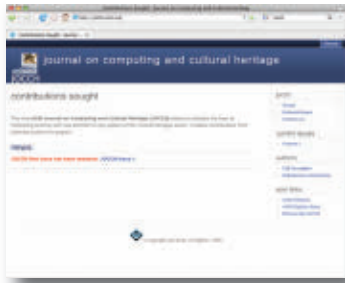


Figure 3. Three curves showing three things in operation.



^a This was brought to my attention by Charles Symons in correspondence on estimation models. Specifically the COCOMO I and II models, and the SEER and Price-S models show a flattening or increase of project cost past a certain point in time.

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

Three Curves

A typical approach of any theory, when confronted by non-standard behavior, is to try to subdivide the observation into discrete components that are more understandable and predictable. In this case there seems to be three things in operation (see Figure 3):

► **Normal:** The “normal” behavior of a project—this is well-predicted by the Putnam model as years of experience and thousands of projects have shown.

► **Compression:** The “Compression Curve” (shown in blue in Figure 3) is probably a log function of time, possibly something like: $\text{Effort} \sim \log(\text{Time})/\text{Time}$. Given the probable underlying organizational causes of this behavior, it is likely to be related to the stress boundaries beyond which a project team cannot function “normally.”

► **Relaxation:** This curve (shown in orange in Figure 3) is generated by underlying changes in the market requirements and feasibility of the product being built as time goes by. It could be viewed as a mirror image of the Compression Curve, but it probably has some different behaviors. It is shown here as an exponential curve and might have a function something like: $\text{Effort} \sim \text{Time} \cdot \exp(\text{Time})$.

The overall time-effort behavior curve would then be the sum of these three curves.

Practicing the Theory

Systems development is an entirely human activity and a cooperative one at that. Therefore it tends to be somewhat erratic; people themselves not being entirely predictable. While the relaxation curve might be more predictable, some research has tried to measure it, and a few estimation models have attempted to characterize it, I suspect the behavior is quite situational. That is, when and how quickly the costs start rising depends very much on the specific product and the underlying market forces that determine the product's value. I am not sure how generalizable and predictable this behavior might be.

The compression curve is probably even more unpredictable. It is related to the stress experienced on projects as they get to their limits of capability; it is probably also related to the limits of cooperative learning,

but that is a subject for another day. The more predictable aspects of this behavior are already captured in the Putnam Model—indeed this is why the effort and cost rise so precipitately as schedules get shorter. But at some point the system breaks down, the noise overwhelms the signal and the chaos overwhelms the work. One of the characteristics of systems that approach their operational limits is that their behavior becomes locally unpredictable—we know the system will break, but we may not be able to foretell where or when.

On the relaxation side, normal schedule constraints should take care of most of the challenges. Any reasonable feasibility proposal for a system development should incorporate a market window limit, beyond which we expect the value of the system will start degrading and the cost of rework will start going up. So we should not aim to commit to development that goes past this point anyway.

We probably shouldn't worry too much about this unpredictability on compression too. Massively over-staffing is really expensive, it requires a lot of people, it generates a lot of defects, and it doesn't even work to reduce schedule like we want it to. Simply knowing this means we should never attempt to reduce a development schedule past the point where the system breaks down. Trying to predict the project's behavior beyond that point is not very useful and we've had plenty of warning that it's going to be way too expensive long before we leave the “theoretical” part of the curve.

So to summarize, we should never allow a project to plan too long a delivery schedule, and we should definitely never try to push a project to completion too soon, since both extremes are ineffective and really expensive.

The problem is, that is what we *should* do. In theory. □

References

1. Armour, P.G. Real work, necessary friction, optional chaos. *Commun. ACM* 47, 6 (June 2004), 15–18.
2. Brooks, F.P., Jr. *The Mythical Man Month*. Addison-Wesley, NY, 1975, 25.
3. Putnam, L.H. *Measures for Excellence*. Prentice-Hall, NY, 1992.

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

Copyright held by author.

Inside Risks

The Risks of Stopping Too Soon

Good software design is never easy, but stopping too soon makes the job more difficult.

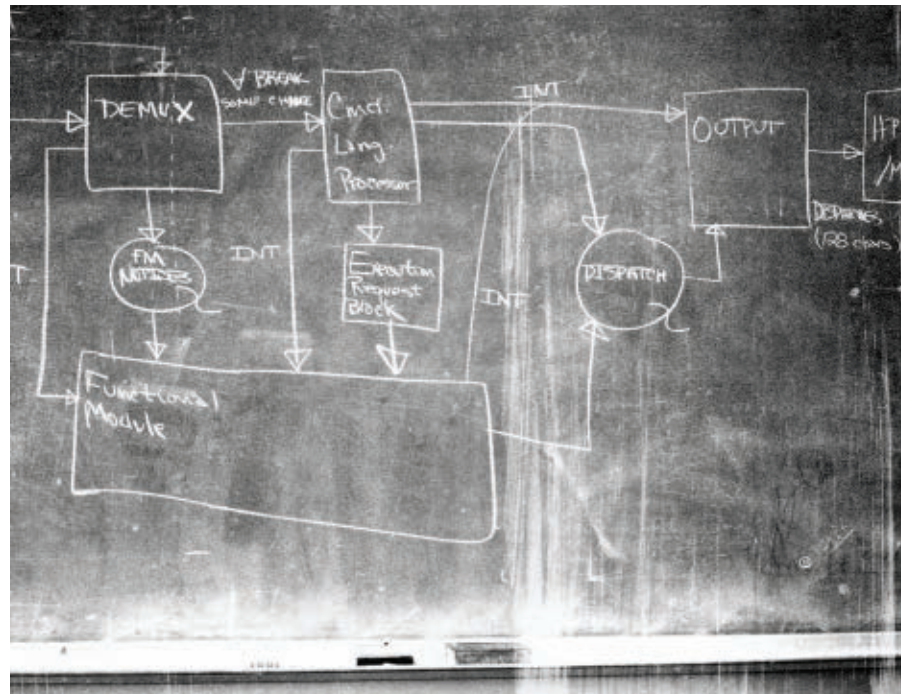
SOFTWARE DEVELOPMENT SUFFERS from an infirmity best called *Premature Termination*; the symptoms are that developers begin to do something useful but stop too soon. The result is something that is not only not very useful, but often harmful. There are four obvious causes for this illness:

- ▶ The work that is done before termination is the easy part of the task; what remains undone would require tedious, detailed work.
- ▶ Those doing the work have not been taught how to do the job correctly or how to determine when it is complete.
- ▶ Those who review the work, or purchase the product, do not insist on proper completion of the key tasks.
- ▶ There is pressure to meet deadlines and get a product “out the door.”

Premature termination can be observed throughout the development, deployment, and post-deployment improvement of software products. The problem is also present when experts advocate and describe a software development process. In this column, I describe some of the disease’s manifestations in requirements documentation, diagrammatic description of software, interface documentation, and quality control.

Requirements Elicitation and Documentation

Nowhere is premature termination more evident than in the field sometimes called “Requirements Engineer-



Primitive buzz-diagram sighted in the wild (the meaning of the diagram eluded capture).

ing” (RE). In RE, developers identify properties they want a system to have and assemble a list of these wishes. Often, some wishes conflict; they are almost always too vague to tell the programming team what to build. Requirements lists include statements such as:

- ▶ The system must be easy for clerks to use.
- ▶ The system must contain an UNDO command.
- ▶ The product shall allow users to work in their native languages.

Each of these examples leaves many

questions unanswered. Among them:

▶ What will be the training of the clerks that will use the system? What information will they have at hand? What characteristics of the interface would make the system easy for them to use?

▶ How many past actions should one be able to undo? Does the requirement apply to all commands or only to a proper subset of those commands? If a subset, which subset? Should you be able to undo the most recent command if the file was closed after it was executed and then reopened? Should

you be able to undo a command without undoing the subsequent commands first?

► What languages (and variants) should be offered? Should it be easy for a user to add languages? Should the keyboard interpretation change when a user switches languages? Should hyphenation rules change with the language?

If the requirements document does not answer such questions, it will be the programmers who determine what the system does. Programmers are chosen for their ability to design good algorithms and data structures as well as their knowledge of specific programming languages and support environments. They should not be expected to understand the needs and characteristics of the future users. Often, a programmer makes incorrect guesses about the detailed requirements and, consequently, extensive revisions are required (either before deployment or after users complain).

A list of wishes like the examples here is not adequate as a requirements document. Even if the aforementioned questions are answered, there is no way to check that there are no other relevant questions. The completeness and consistency of such a list cannot be verified. There are always things left unstated. Such lists are a good start but considerable work must be done before one has resolved the conflicts and ambiguity in such a list to produce a complete and precise software requirements document, one that tells the programmers what they must build to satisfy the agreed wishes. It isn't wrong to produce a wish list but it is wrong to hand it over to the programmers as a requirements document.

Drawing Pictures of Programs

The debate over whether pictures are a useful way to document programs is an old one that never seems to get resolved. The issue came up again in a recent column by Grady Booch,¹ in which he observes that most of the architecture pictures shown to him do not communicate well; he then goes on to describe a process of improvement through discussion. One is left to wonder exactly what is wrong with those pictures and what would make a good one.

Booch's plea for better pictures reminded me of a presentation by the late Edsger Wybe Dijkstra around 1975. At a meeting of IFIP W.G.2.3, a working group on programming methods, he advised against drawing pictures saying, "Every time someone draws a picture to explain a program, it is a sign that something is not understood." I found this surprising; during my education in electrical engineering, we were often shown how to use a diagram when designing or when analyzing a proposed design. Those diagrams were sufficiently meaningful that one could derive equations from them. On the other hand, I could not find a diagram that was intended to be a program description that did not raise more questions than it answered. All were so vague that it was very likely that two people would look at a diagram and interpret it differently. Most raise more questions in my mind than they answer.

Dijkstra's observation struck me as so thought provoking that, when I returned to my group in Germany, I repeated it to them. One of my associates, Wolfram Bartussek, responded immediately with a German version. However, in his "translation" he changed the statement's emphasis. "Yes," he said, "drawing a picture is what you do when you are trying to understand a program or trying to help someone else understand it."^a Reflecting on this, I found it was true. A picture is often very helpful when trying to understand a complex problem. Bartussek had not contradicted Dijkstra's observation but he had explained why Dijkstra's advice (not to draw diagrams) was wrong. Diagrams can be a good starting point.

Subsequent experience with software documents deepened my understanding of the problem with diagrams of software systems. A few years later, I was asked to review a project that had repeatedly missed deadlines. In a series of meetings with key designers, I began by asking each one to draw a diagram that explained the workings of the system. I used a Polaroid camera to take a picture of each drawing. When each meeting began, the new expert studied the diagram on

the board from the previous meeting and said, "That's an interesting picture but it's not our system." I showed each one the pictures I had accumulated but they found those no better. Each asked to erase some or all of the existing diagram so they could draw another one.

Those photos are faded now, but it is still clear that the diagrams are all different, all are vague, and none of them contains enough information to allow someone to understand what its creator meant. In a few cases, an expert reused part of the previous picture but the discussion revealed he was interpreting it differently from the person who had originally drawn it.

When I am presented with "box and line" diagrams that are open to many interpretations, I ask that the picture be completed by adding a legend, that is, an explanation of what property something must have to be represented by a box and what relation must exist between two boxes if there is a line connecting them. When the pictures come back, they have been altered to have several distinct box shapes and types of arrows. When the authors were trying to create the legend, they realized that they were using one symbol to represent several different kinds of objects or relations. The original picture had been a "buzz-diagram";^b the new one is always better than the first but usually not good enough. It takes many iterations before a reviewer can understand what the author is trying to convey; only then can we begin to discuss the design. When we do discuss it, more changes are made to the diagram. Some of those changes are design changes but others are adding components and connections that were part of the design that had been overlooked when the diagram was prepared.

Often, when the necessary information is added, the diagram becomes so cluttered and complex that it no longer helps people to understand the system. In such situations, the authors convert the diagram to some

a This is an informal translation, not Bartussek's exact words.

b A buzz-diagram, like a buzzword, has the property that most people think they understand it but are unable explain it clearly when asked to do so.

other representation of the information (often tables). The resulting document becomes an essential basis for subsequent work on the product.

The lesson should be clear; a picture is a good way to begin to understand something but most software developers stop too soon. They work on the picture until it means something to *them* but then stop; they stop even if the diagram does not contain all of the needed information or does not communicate clearly to others. If they do add information to a diagram, another representation of its content may eventually replace the picture.

Talking About “the Architecture”

Related to the problem of pictures is talking about “the architecture” of the system. That phrase suggests there is a single structure that can be so identified. In fact, as has been discussed in Klein and Weiss² and Parnas³ there are many distinct, separately designable, structures. For example, the “uses” structure identified in those papers can be quite different from the module or component structures. Keeping the two structures separate eliminates many apparent conflicts. Booch calls for clearer descriptions of a system’s architecture; if each of the structures mentioned in Klein and Weiss² and Parnas³ is diagrammed separately, the results would probably be much more to Booch’s liking.

Software Documentation

Problems caused by premature termination are found in all kinds of software documentation. For example, Java library documentation is often cited as an example of good documentation. However, if you read the descriptions of methods in a Java library, you can usually identify questions that can be answered only by experimenting with an implementation or reading the code. So-called “odd cases” are either not covered or ambiguously covered. One often finds a list of exceptions a class will “throw” but it is not clear if cases such as zero length string arguments are exceptions or what happens when two of the listed exceptions could be “thrown.” Generally, it appears the documenter started to write down a description but quit when they got to the tedious details.

The disease of premature termination seems to be immune to itself; it never stops.

Testing and Inspection

We can also recognize the problem of premature termination in testing and inspection. The constant stream of “updates” (usually a euphemism for corrections) that we are asked to install is evidence that the developers stopped their testing and inspecting too soon. Inspections are often considered finished when the participants are able to estimate the number of remaining errors and the estimate is small. One honest tester answered my question, “How do you know when to stop testing?” with “When the schedule says there is no more time.” Another replied, “When my boss says to stop.” These events can occur long before the software is trustworthy enough to merit release.

Incomplete Advice

It is particularly vexing that the problem of premature termination is also found in the articles and talks by people who claim to be telling practitioners how to do their job better. Many books and papers describe development processes by giving good advice such as “determine the characteristics of the users,” but they stop without explaining how to do it, how to document the information, or how to know when you have completed the task. Most of the books and papers use buzz-diagrams to describe a process and give examples of work products that are themselves incomplete and unclear. Many of these “gurus” exhort people to do better, and assure them they can do better, but eschew anything that looks like tedious “dog work” or has even a hint of mathematics.¹

“Stopping Too Soon” Never Stops

The disease of premature termination seems to be immune to itself; it never

stops. There are some obvious reasons for this.

► Physicist Wolfgang Pauli is said to have commented on papers by saying, “This is so bad it’s not even wrong!” It is often difficult to detect premature termination because the work is not wrong; it is just not enough.

► Many managers don’t demand disciplined, careful, complete work. Cost and schedule are their primary concern. Short-term cost is easy to measure; long-term cost is unknown and won’t affect their next pay raise.

► When selling methods, gurus are so eager to gain converts (and customers) that they try to make their approach seem easy and fun. The work that is not described has neither of those characteristics.

► Educators in universities are so eager to give a complete survey of the available methods that they are (unavoidably) shallow. Rather than pick a strong method and teach it thoroughly, they teach a little about each of many methods. Some also avoid anything that looks like “theory.” Others make a point of teaching students how things are currently done in industry and avoid teaching methods that would be improvements.

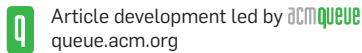
There are methods for doing each of the tasks mentioned (requirements elicitation and documentation, architecture documentation, testing, and inspection) that make it possible to know when the job is complete (but not necessarily correct). All require the use of basic mathematics and discipline and are a sharp departure from current practice. Good software design is never easy; stopping too soon, while easy in the short run, makes the job harder in the long run. C

References

1. Booch, G. Draw me a picture. *IEEE Software* 28, 1 (Jan./Feb. 2011).
2. Klein, J. and Weiss, D. What is architecture?, Chapter 1 of *Beautiful Architecture*, Spinellis & Gousios, Eds., O’Reilly, 2009.
3. Parnas, D.L., On a ‘buzzword’: Hierarchical structure. *IFIP Congress ’74*, North Holland Publishing Company, 1974, 336–339. Reprinted as Chapter 8 in Hoffman, D.M. and Weiss, D.M., Eds., *Software Fundamentals: Collected Papers by David L. Parnas*, Addison-Wesley, 2001. Reprinted in M. Broy and E. Denert, Eds., *Software Pioneers: Contributions to Software Engineering*, Springer Verlag, Berlin-Heidelberg, 2002.

David Lorge Parnas is President of Middle Road Software as well as Professor Emeritus at McMaster University and the University of Limerick.

Copyright held by author.



Kode Vicious

Think Before You Fork

KV's thoughts on forking, config files, and using internal wikis.

Dear KV,

Until recently I was working on a small open source project that had only three or four maintainers. I sent in a large and extensive patch, and got little or no comment. When I asked for commit access to the project repository I was refused, and was asked to go back and split up my patches into smaller sections for review. All of this back and forth happened over two months! Since the point of open source development and source-control systems is to commit and update early and often, you can imagine that I'm frustrated by this foot dragging on the part of the project's official maintainers. I have considered just forking the project, but I'm not sure if I want to go it alone. I really just want my patches integrated and my new features to be available to the community.

Frustrated and Forking

Dear Forking,

Most people who work on open source projects—especially the smaller ones that are not supported by some corporate entity—are, as you probably know, volunteers. Volunteers don't have a great deal of time on their hands, unless they're very wealthy volunteers. Have you met a lot of wealthy volunteers? I thought not.

Submitting a large patch to an open source project with only a small number of possible reviewers is actually pretty rude. You are asking people who don't know you to trust you with their



hard-won code base. Breaking down a large patch into small patches is just common courtesy to the maintainers, so they can understand your code in their few free moments and then integrate it without upsetting the quality of the larger code base. Having a patch rebuffed and then asking for commit access to their repo would be seen by most project maintainers as annoying, if not insulting, and a wise project maintainer would stay well away from a prospective committer who acted in this way.

Open source projects are as much social arrangements as they are technical undertakings. The people on the project have to trust each other, and

trust is not earned by sending in a single large patch; it is earned over time and via repeated exposure to a person via mailing lists, forums, and other nontechnical communication channels associated with the project.

A good project maintainer will lay out these kinds of guidelines in documentation on the project site, as well as communicating them in personal and group email messages. Certainly you should have taken the first rebuff not as an insult or as some sort of foot dragging but as an invitation to work with the project maintainer to get your code into the system. Breaking down your large patch into smaller, more easily reviewable chunks would simply have been common courtesy.

I have worked on too many projects where someone comes along saying he or she has solved a ton of problems with one big patch, and rarely have I seen the integration of such a patch go well. Often through a lack of testing, or simple shortsightedness as to the knock-on effects of the change, the time it takes to integrate the large patch far outstrips the amount of time it would take to integrate a larger number of smaller, and easier-to-understand, patches.

As to whether or not one should fork a project, I can think of only one clear case where that is usually called for. When a project has gotten to the point of being unmaintained, it is easy to see why you would create a new one if you could not get the previous maintainers simply to hand over the old project to you. Forking a project

in other circumstances often makes you look like a petulant and spoiled child who wants to take your toys and go home. Of course, I've worked with enough programmers to know that often that is exactly the case.

KV

Dear KV,

I have finally tired of adding command-line switches to a program I have written and now I am going to switch it to using a configuration file. I think a config file will allow me to provide a richer environment to my users, and, I'm running out of both upper- and lowercase letters for command-line switches. One of the problems is there doesn't seem to be a standard for how to write a configuration file parser, and of the several I've seen, each has its warts. Do you have any advice on configuration file parsers?

Configed and Confused

Dear Configed,

Name three letters that are not arguments to the `ls` command! Yes, it's an old joke, but it's funny because it's true.

Depending on which language you're programming in, there may already be a library to help you parse a configuration file. If there is a library, then I strongly suggest you use it. Configuration file parsers are another in a long list of technologies that programmers seem to delight in reimplementing, and, usually, they get it wrong.

My favorite example of getting it wrong is when white space is made too significant. A configuration file parser that requires spaces in one place but forbids them in another related place is an example of a piece of

Breaking down a large patch into small patches is just common courtesy to the maintainers.

code that should be taken out, hanged, shot, burned, reconstituted, and then burned again (just for good measure). Although you may think this is a trivial complaint, I can assure you that people continue to write code in this way, because I just dealt last week with a system that does exactly the wrong thing. Alas, I did not select the product in question and was not allowed actually to rid myself of it, so I am now dealing with it. By *dealing with it*, I mean that I have made my displeasure quite clear. Of course, the person who did select the product sits quite close so he has been subjected to occasional outbursts of cursing every time some bit of white space goes awry. I notice he recently brought in headphones. I think it's time to make sure I can send some live audio data to his computer as well.

KV

Dear KV,

Every time I optimize something at work I put the results of my latest benchmarks on our internal wiki. The problem is that people seem to leap onto these measurements to show whatever it is they want to prove, or claim to have proven before, usually without understanding the preliminary nature of some of these measurements. I don't want to hide my work, but it seems that no matter what kind of disclaimer I add, people just ignore it and read into the numbers things that aren't really there.

Disclaimed and Dejected

Dear Disclaimed,

Announcing the results of your work, as you have now seen, is actually a lot trickier than you might have thought at first. The problem is people. I know, the problem is always people. While the scientific world has a basic way of handling this problem—which is to get colleagues to review work marked preliminary as, well, preliminary—this kind of peer-review process is not common inside of most companies or work groups. People in companies want results, and if they see something that looks like a result, they are going to jump on it, like a fox on a rabbit, often without checking to see if the rabbit is, in fact, alive.

Depending on which language you're programming in, there may already be a library to help you parse a configuration file.

While it is a fine idea to post preliminary results in an attempt to get some feedback, it is probably best to send these to a small number of people instead of placing them in an easy-to-access place that is open to any peeping Tom, Dick, or Mary. When people write scientific papers they send the first draft off to a small group of trusted friends for feedback. Note that I said *trusted*, so you might want to make sure these are people you can really trust. If someone passes your results on without asking you, you should not send them preliminary results again. Although you may not be writing a paper on your work, the same advice still applies: email a few people you trust to check your work before announcing to your group or company as a whole what you've come up with.

KV

Q Related articles on queue.acm.org

A Plea to Software Vendors from Sysadmins—10 Do's and Don'ts

Thomas A. Limoncelli

<http://queue.acm.org/detail.cfm?id=1921361>

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

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

There's No Such Thing as a Free (Software) Lunch

Jay Michaelson

<http://queue.acm.org/detail.cfm?id=1005066>

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

Copyright held by author.

Viewpoint

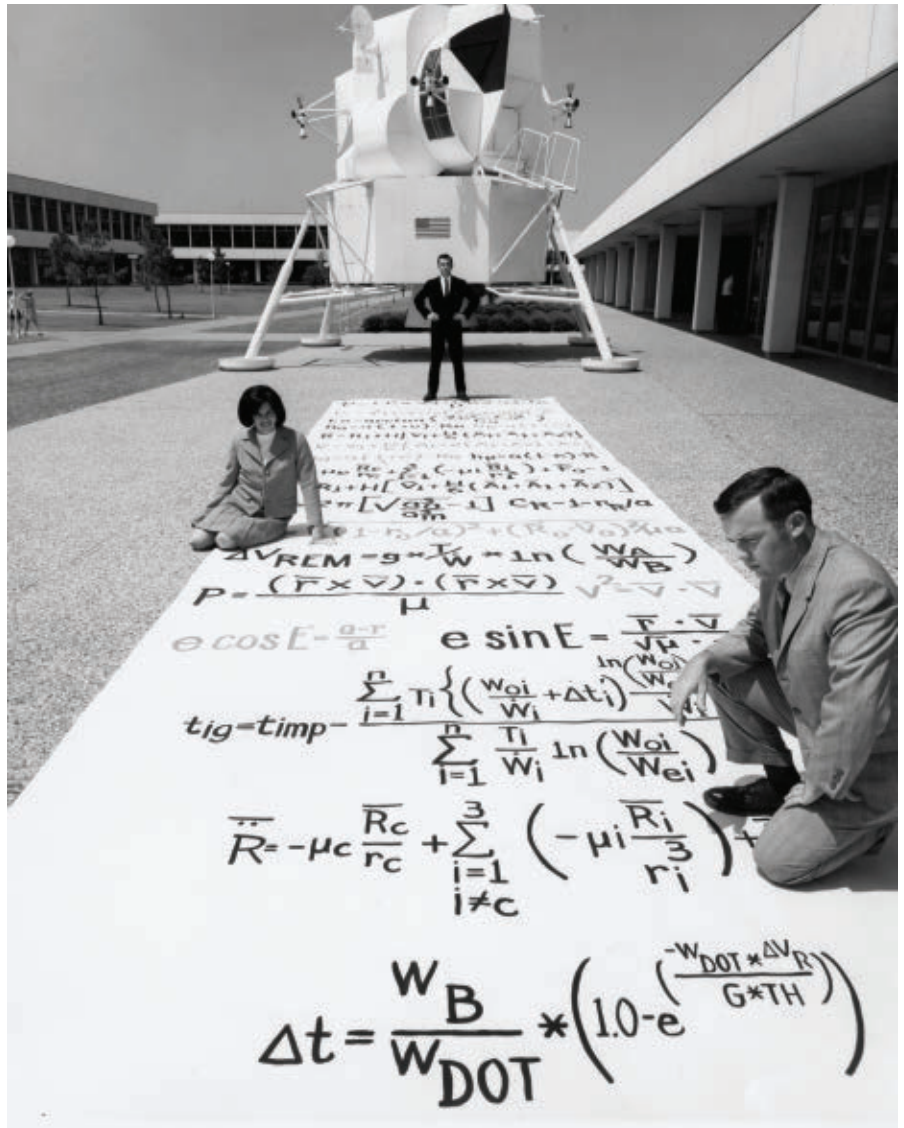
Computer Science

Can Use More Science

Software developers should use empirical methods to analyze their designs to predict how working systems will behave.

THE SO-CALLED LAMP stack (Linux OS/Apache Internet server/MySQL DBMS/Perl PL) consists of 10 million lines of code,⁵ interacting in myriad ways to achieve impressive functionality and performance. This approaches the intellectual complexity of the Saturn V rocket (with three million parts) that took humans to the moon. The similarities between these two enormous engineering feats are important: good engineering design practices have been followed; requirements were defined and met; rigorous testing and debugging has taken place. Yet, to date, the LAMP stack is much less well understood than the Saturn V rocket. It is much more difficult to predict how the LAMP stack will perform under varying conditions and where things might go wrong than it is to consider how the Saturn V may behave in different operating environments.

What the engineers at NASA have that the developers and users of the LAMP stack do not is an understanding of how configurations of system components (in NASA's case, physical materials) will behave in a variety of (physical) contexts and an idea of where the boundary conditions of those behaviors lie. This understanding is the product of physical theories about universal laws of nature—laws that have been identified through a tradition of model construction and empirical testing to produce general principles.



The Saturn V rocket that launched humans to the moon relied on an IBM System/360 for data processing.

Developers of the LAMP stack do not approach this level of understanding.

If computer science can achieve this level, that is, uncover the kinds of scientific theories and laws that physics provides, software developers building artifacts could do what NASA engineers do now: analyze their designs according to underlying theories and predict how a working system will behave. Extracting such laws and principles is the goal of science.

Whether and how computer science is a science has been a topic of discussion since the beginnings of the field. Thinkers including Herbert Simon⁶ have offered deep insight into the special nature of computation and how it exists as a phenomenon—in part, it's artificial, something we create. However, the nature of what computer science is studying, or how best to study it, is by no means a settled topic. Just because we design and build computational systems does not mean we understand them; special care and much more work is needed to correctly characterize computation as a scientific endeavor.

The early challenges of the field were engineering in nature, leading to a de-emphasis of empirical methods. In recent years, with increasing complexity of computational systems, empirical methods are now needed to discover system limits and predict future behavior.

The Methodology of Debugging

Let's start with the parts of the scientific method we do well. Programmers engage naturally in one of the purest forms of empirical investigation: debugging. Whenever a program exhibits a fault, in which a run over particular input data yields an exception or incorrect output, the programmer generally follows a series of steps. First, the programmer develops a *hypothesis* of where the fault lies (whether in a particular statement or more generally with the logic of the design) and considers how to correct it. This hypothesis is based on the programmer's mental model of how the programming environment, referenced libraries, operating system, and environment (such as input data, system events, and so forth) function and interact. This model is generated and refined through addi-

Why, then, do we not have the same depth of understanding of the LAMP stack as the Saturn V rocket?

tional runs of the program with print statements or through interaction with a debugger in which the program state can be examined as the program is single-stepped. The programmer then *tests this hypothesis* by making changes believed to fix the problem and rerunning the program, predicting that the change will result in the correct output for that particular test input. If the program still faults, the programmer refines the hypothesis, updating their mental model of how they believe the underlying system is behaving, and further investigates possible causes. Programmers frequently demonstrate they are highly skilled at understanding complex interactions by forming models and testing and revising them.

Toward General Predictive Models

Why, then, do we not have the same depth of understanding of the LAMP stack as the Saturn V rocket? Formulating hypotheses, devising tests and carrying them out, and then revising hypotheses based on test outcomes are critical components of the basic methodology of empirical science. But these activities are not all that science does. What is missing from the debugging picture is searching for generalizations: attempting to extract general patterns and identify the factors that govern their behavior. This requires looking past individual systems, past simply getting the program to run in a particular context, to characterizing general principles of system behavior. The predictive model underlying the hypothesis testing undertaken by our programmer is specific to the extreme.

Empirical generalization³ progresses from description of particular instances to prediction (via models) of classes of systems. This progression is advanced

to an explanation that connects what is going on in particular observed cases to a *general class*, in this case of certain kinds of software systems in certain kinds of configurations. The models and the kinds of systems they describe should both be broadly construed to render a true understanding of these systems. Are such general predictive models even possible for software systems? Experience indicates that when such systems are examined experimentally, with an explicit goal of discovering causal models, such models can in fact be found and highlight deep structure.⁷

The articulation of scientific theories as predictive causal models, the methodology of empirical generalization, and the evaluation of such theories via hypothesis testing is prevalent in isolated sub-disciplines of computer science, including human-computer interaction, empirical software engineering, Web science, and data mining. It is, however, rarely used in other sub-disciplines of computer science, especially those concerned with software systems artifacts, such as compilers, databases, networks, operating systems, and programming languages.

Examples of Computational Models

One example of a computational model that is the product of empirical generalization is the *Theory of Locality*.⁴ This model arose from a study of the cost of managing page transfers between main memory and a much slower disk drive, within the general area of operating systems. What Denning found was that data relevant to the current context of a running program tended to be grouped in space and time to local chunks, and this was the product of the way programs are written by humans, rather than due to any underlying constraints of the computing system. When our memory management algorithms respect this general pattern, performance improves dramatically. The resulting theory—that human-constructed information processing systems will exhibit locality—is inherently predictive and has been tested many, many times. This theory has subsequently been generalized to apply to computational systems of all kinds: “in virtual memory to organize caches for address translation and to design the replace-

ment algorithms, ...in buffers between computers and networks,...in Web browsers to hold recent Web pages,... in spread spectrum video streaming that bypasses network congestion.”⁴

Note what Denning did: He recognized a kind of behavior that spans a class of systems. He searched for the factors responsible for the behavior he and others had observed. And, *how* he searched is of utmost importance: he conducted experiments, adjusting the candidate factors responsible for producing the behavior, collected data and carefully analyzed it and used the results to update his hypotheses. Denning wasn’t debugging and he wasn’t deriving proofs; he was revising his model based on experimental data while expanding the model to characterize a larger class of computational systems that exhibit locality. In doing so, Denning was able to achieve greater clarity about this phenomenon and how and why a computational system will exhibit this phenomenon.

We don’t yet know whether the laws of computational systems will cohere into explanatory frameworks like other natural sciences, or as elegantly as the theory of locality. For example, many physical systems are continuous in nature while most computer science domains are discrete. Discrete systems tend to exhibit chaotic behavior, in which small changes in inputs or initial conditions lead to dramatic differences in outcomes. It is not known what portions of computer science phenomena are chaotic. But even here, empirical and analytical tools are available to help, such as dynamic systems theory (Bradley provides an example of applying dynamic systems theory in a computer science context¹). Discrete systems are analytically very challenging, but incorporating empirical tools will help expand our modes of analysis.

Another notable episode of empirical investigation in computer science, still unfolding, is in the study of where the hard problems lie within the parameter space of computationally hard (NP-complete) problems. Many of the problem instances turn out to be relatively easy to solve, while others are insurmountably difficult. Early models proposed that NP-complete problems would transition from easy to hard then back to easy around

Good scientific models will generate new questions to be answered and will drive our field to yet deeper understanding.

a threshold value of certain parameters. Subsequent studies have since systematically explored this phenomenon, and have discovered that where these transitions occur may depend on a variety of additional factors, including the kind of problem-solving method used. Different problem-solving methods provide us with different tools with which to study the complexity of these problems; an apt analogy has been made to astronomers using telescopes that operate on different light wavelengths to provide different perspectives on the structure of the cosmos.² The jury is still out on how to best characterize the transition behavior, but the key point for our purposes is to again focus on the methodology employed: these researchers conduct experiments, take measurements, and refine their models, while striving for generality.

Realizing the Benefits of More Science in CS

These examples and other extant computer science theories emphasize that by embracing the methodology of developing and evaluating predictive models through experimentation over multiple members of a class of software systems, a more complete understanding of such artifacts will emerge. In addition, this observational and experimental scientific perspective encourages the computer scientist to actively look for relevant phenomena that may have been missed because they aren’t currently described by existing, closed-form analytical descriptions.⁸ The developed explanatory model provides better prediction and ultimately

more dependable products built on those principles. Finally, good scientific models will generate new questions to be answered and will drive our field to yet deeper understanding.

How can these benefits be realized? How might we change what we do? We can adapt our already very skilled hypothesis testing in debugging and broaden it by asking more general questions, identifying the classes of system properties that contribute to behavior, identifying their boundary conditions, and working to fit them into a unified picture. We can also more broadly adopt additional methodological tools, such as from statistics and dynamic systems; a number of computer science subfields already profitably do so.

The pristine presentations of scientific reasoning and the tremendous successes of such reasoning in other fields may appear to the practicing computer scientist as out of reach. But many of our colleagues have started down this path, the tools are accessible, and the promise is great.

Acknowledgments

We thank the reviewers and editors for an enjoyable and vigorous conversation around the topic of this column. The reviewers will not agree with every detail, but they have certainly helped us refine the presentation. 

References

1. Bradley, E. Time-series analysis. In M. Berthold and D. Hand, Eds., *Intelligent Data Analysis: An Introduction*. Springer Verlag, 1999.
2. Coarfa, C. et al. Random 3-SAT: The plot thickens. In *Proceedings of the 6th International Conference on Principles and Practice of Constraint Programming*. Springer LNCS, 2000, 143–159.
3. Cohen, P. *Empirical Methods for Artificial Intelligence*. MIT Press, 1995.
4. Denning, P.J. The locality principle. *Commun. ACM* 48, 7 (July 2005), 19–24.
5. Neville-Neil, G.V. Code spelunking redux. *Commun. ACM* 51, 10 (Oct. 2008), 36–41.
6. Simon, H.A. *The Sciences of the Artificial*, Third Edition, MIT Press, 1996.
7. Snodgrass, R.T. Ergalics: A Natural Science of Computation, February 2010; <http://www.cs.arizona.edu/projects/focal/ergalics/whatis.html>
8. Tichy, W.F. Should computer scientists experiment more? *IEEE Computer* 31, 5 (May 1998), 32–40.

Clayton T. Morrison (clayton@cs.arizona.edu) is a research assistant professor of computer science at the University of Arizona. He has more than a decade of experience building artificial intelligence systems for DARPA, AFOSR and NSF projects.

Richard T. Snodgrass (rts@cs.arizona.edu) is a professor of computer science at the University of Arizona. He is an ACM Fellow, has chaired the ACM Publications Board and SIGMOD, and has served as the editor-in-chief of *ACM Transactions on Database Systems*.

Copyright held by author.

ADVANCE YOUR CAREER WITH ACM TECH PACKS...

For Serious Computing Professionals.



Searching through technology books, magazines, and websites to find the authoritative information you need takes time.

That's why ACM created "Tech Packs."

- Compiled by **subject matter experts** to help serious computing practitioners, managers, academics, and students understand today's vital computing topics.
- Comprehensive **annotated bibliographies**: from ACM Digital Library articles, conference proceedings, and videos to ACM Learning Center Online Books and Courses to non-ACM books, tutorials, websites, and blogs.
- **Peer-reviewed** for relevance and usability by computing professionals, and **updated** periodically to ensure currency.

Access to ACM resources in Tech Packs is available to ACM members at <http://techpack.acm.org> or <http://learning.acm.org>.

Current topics include **Cloud Computing** and **Parallel Computing**. In development are Gaming, Globalization/Localization, Mobility, Security, and Software as a Service (SaaS).

Suggestions for future Tech Packs? Contact:

Yan Timanovsky
ACM Education Manager
timanovsky@hq.acm.org



Association for
Computing Machinery

Advancing Computing as a Science & Profession

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

In today's humongous database systems, clarity may be relaxed, but business needs can still be met.

BY PAT HELLAND

If You Have Too Much Data, then 'Good Enough' Is Good Enough

CLASSIC DATABASE SYSTEMS offer crisp answers for a relatively small amount of data. These systems hold their data in one or a relatively small number of computers. With a tightly defined schema and transactional consistency, the results returned from queries are crisp and accurate.

New systems have humongous amounts of data content, change rates, and querying rates and take lots of computers to hold and process. The data quality

and meaning are fuzzy. The schema, if present, is likely to vary across the data. The origin of the data may be suspect, and its staleness may vary.

Today's data systems coalesce data from many sources. The Internet, B2B, and enterprise application integration (EAI) combine data from different places. No computer is an island. This large amount of interconnectivity and interdependency has led to a relaxation of many database principles. Let's consider the ways in which today's answers differ from what we used to expect.

The Erosion of Principles

Many different principles of classic SQL databases are being eroded by combining too much data:

- ▶ *Unlocked data.* Classic SQL semantics depend on the data in the database being locked. Unlocking the data changes the semantics from the classic database.

- ▶ *Inconsistent schema.* Different documents come from different sources. You need to deal with extensibility, different semantics, and unknown semantics.

- ▶ *Extract, transform, and load.* Data may come from many sources, and you are attempting to shoehorn it into something resembling commonality.

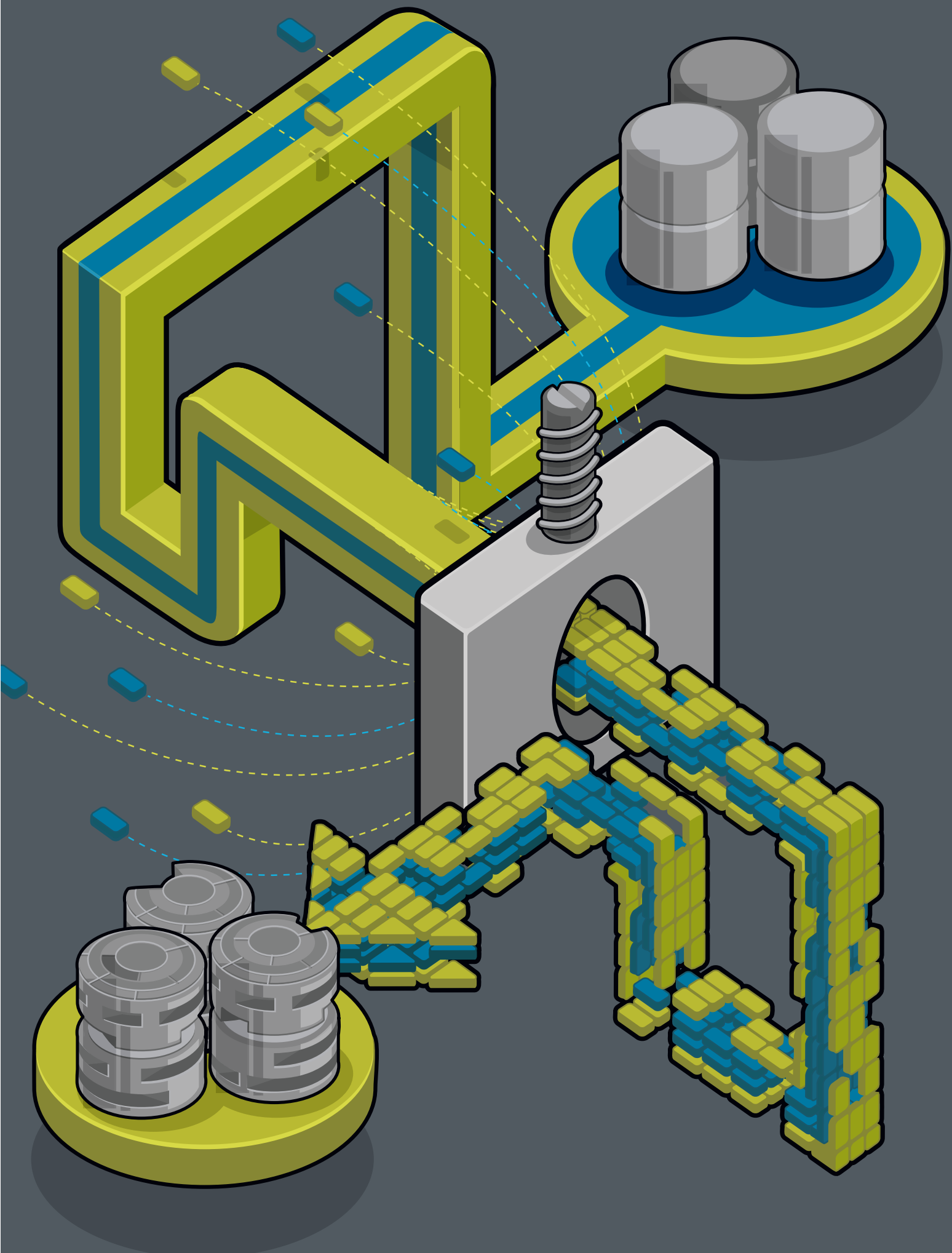
- ▶ *Patterns by inference.* Where are the connections that you didn't think of? What can you infer by continuously examining the data?

- ▶ *Too much to be accurate.* By the time you do the calculations, the answer may have changed. Too much, too fast—you need to approximate.

Business needs lead to lossy answers. While we would like to be perfect, business needs answers even when the answers are not perfect.

- ▶ *Sometimes, the data causes challenges.* There may be a huge volume of data; it may be from many different sources or unclear sources; it may arrive over a period of time.

- ▶ *Sometimes, the processing causes challenges.* You may need to convert, transform, or interpret the data in a way that loses information (see Figure



1). You may need to make inferences and assumptions about the data.

We can no longer pretend to live in a clean world. SQL and its Data Definition Language (DDL) assume a crisp and clear definition of the data, but that is a subset of the business examples we see in the world around us. It's OK if we have lossy answers—that's frequently what business needs.

Einstein Was on to Something

Huge scale has implications on the consistency semantics of the data. It turns out that most large systems in the NoSQL (not only SQL) genre cannot support transactions across all of their data. This carries implications on how data can be used together. Here, I examine some of the issues with data

from distant systems.

Transactions, classic databases, and “now.” Transactions make you feel alone. The purpose of a database transaction is to give the appearance that nothing else is changing the data while you are doing your work.

As a transaction geek, I've spent decades working on systems that provide a guaranteed form of consistency. One special form of this consistency is called *serializability*, or the appearance of a serial order. The individual transactions do not have to execute in serial order, but there must be at least one serial order that corresponds to the observed behavior.

Each transaction shown in Figure 2 seems to be executing in a crisp and clear “now.” Some stuff happened in the

past; other stuff happens in the future; the transaction sees “now.” The definition of “now” is the boundary within which a transaction can be applied.

SQL databases depend on living in the “now” to provide their semantics. Though many SQL systems offer relaxed consistency semantics to increase concurrency, transaction serializability is still the clearest way to think about the general SQL mechanisms.

Unlocked data is not “now.” Messages and documents contain unlocked data that has been sent out into the wild, cruel world. When messages are sent, they usually cross boundaries that can support distributed transactions; hence, the data is not transaction protected. Instead, the data is typically given an identity and a version. The data from the originating system may change shortly after the message is sent. Unlocking it allows change. Messages and documents are always from the past.

Simultaneity does not exist at a distance. Knowledge travels at the speed of light. By the time you see a distant object in the night sky, it may have changed. Similarly, by the time you receive a message from a distant computer, the data contained in that system may have changed.

Transactions and application/business boundaries define simultaneity. Inside a single system (either one computer or a small number of computers) you can have a transactional database with transaction-protected application operations. Many single applications are growing so large that the implementation of these as a single transactional domain is impractical. Hence, we are now seeing applications using NoSQL-style data in which different portions of an application's data live within different transactional boundaries. Some recent scalable systems offer a small collection of data within which you have transactional consistency. Across these boundaries, looser consistency is provided and the application is responsible for application-specific consistency. These include Google's Megastore¹ and Microsoft's SQL Azure.⁴

Unlocked data: A blast from the past. All data on the Internet is from the “past.” By the time you see it, the truthful state of any changing values may be different. Each independent database,

Figure 1. Data transformation is frequently a lossy operation.

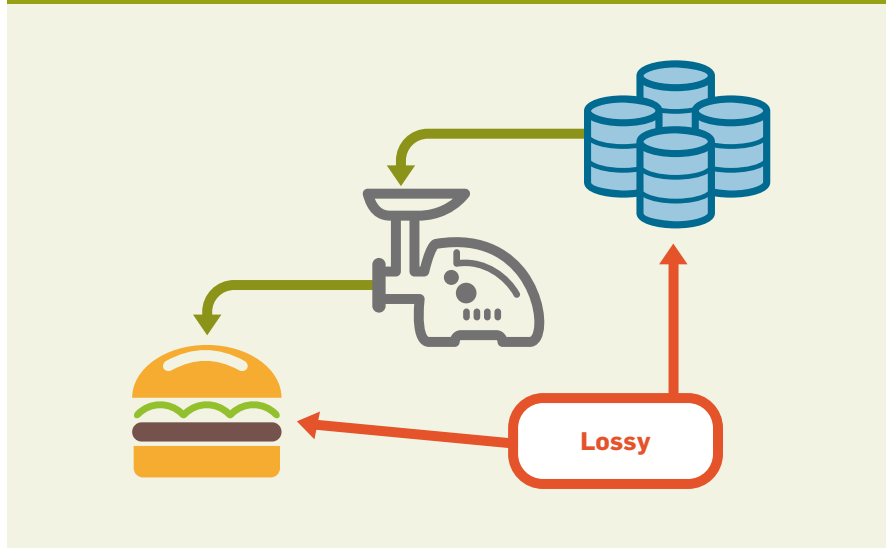
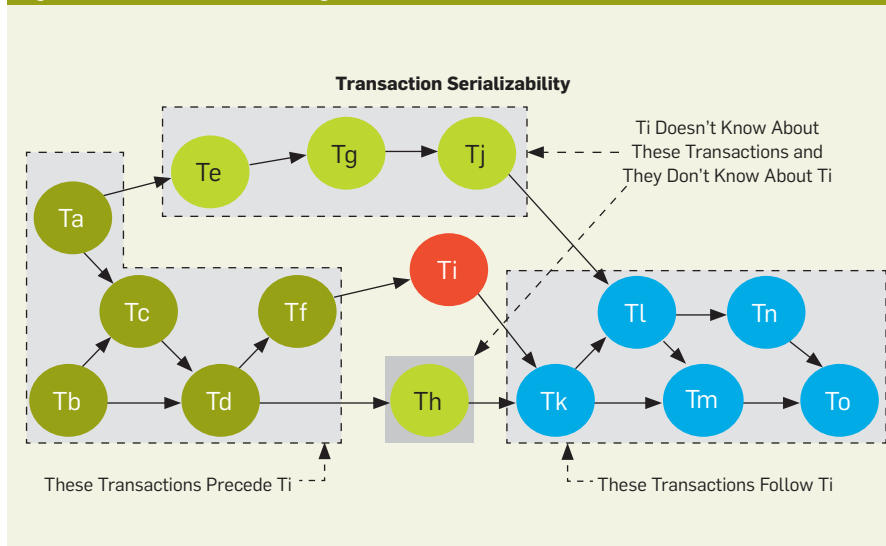


Figure 2. Transactions executing in the “now.”



system, or application will cultivate its own data, which will change over time. In loosely coupled systems, each system has a “now” inside and a “past” arriving in messages. By the time you’ve seen some unlocked message or document, the truth may have changed.

If you depend on a set of data being cohesive, it must be written by its source as a single identity within a single atomic transaction. This cohesive data unit must have a unique identity and version. In today’s large, loosely coupled systems, there will be many of these sets of data. Within each, the temporal nature can be cohesive. Across them, the application must reason about their different histories.

Application semantics must deal with “now” and “then.” Each application running in this large and loosely connected world must have its own subjective sense of “now,” and it must understand that information from other systems is not in the same “now.” Stuff may have changed by the time you see it. It is essential that business concepts accept these truths. This is no different from what took place in the years before telephones when people used handwritten messages and couriers to coordinate business.

By necessity, the solution to this temporary ambiguity (the uncertainty about the truth) must be handled in an application-specific way. Different business problems have different business mechanisms for coping with uncertainty.

One of my favorite notifications is: “Usually ships in 24 hours.” This tells you absolutely nothing about any guarantees, but it is incredibly useful—I am always adding or removing things from my shopping cart based on this ill-defined nonguarantee.

In large, loosely coupled distributed systems, we must also recognize that there’s more than one computer. Each of these computers will have its own time domain and its own sequence of transactional history. We must build our applications to cope with the ambiguity of these multiple histories.

What the Heck Are You Talking About?

When data is published in a document or sent in a message, there is usually an attempt to describe it so the reader can



Huge scale has implications on the consistency semantics of the data. It turns out that most large systems in the NoSQL (not only SQL) genre cannot support transactions across all of their data.



understand it. The nature of this schema for the data is intrinsically different in a very large and loosely connected distributed world. Here, I discuss some of the challenges in communicating what is supposed to be communicated.

Messages and schema. When a message is written in a distributed and heterogeneous system, the writing system (ideally) understands its semantics. At the time it is written, there is a context and intent. Hopefully, a schema *describing* the intention of the message accompanies the content explicitly or by reference. The goal of this schema would be to facilitate the understanding of the intent of the message.

Messages are immutable. They are written, sent, and possibly retried. It is considered buggy to have a message retry semantically different from the first attempt. Thus, when describing the message, the schema information should be immutable, too.

You know what you mean when you say it. When a message or document is written, the writing system knows its meaning. Frequently, some description or schema is attached to the document to facilitate the reader in figuring out what the sender meant. This description may be in any of a number of representations including XML (and its many schema variants). One nice thing about XML is that essentially all systems can parse it well enough to get confused about the semantics of the message rather than the syntax. This is a huge advancement.

I’m trying to understand you but your accent is thick. Unless the reader of a message or document is specifically programmed for it, there will likely be confusion. The meaning of the message, the interpretation of its fields, and much more will be subject to approximation and a loss of clarity. Different companies, different countries, and even different regions within a country have different understandings of data. Sometimes it is harder to communicate with a different department inside the same company than with some external partners. (Of course, there are always contextual issues that are notoriously difficult to capture. The schema author will likely do his or her very best to express the meaning of the data. But when modern readers try to decipher a medieval work, there are

cultural assumptions that may be as inscrutable as those posed to the parents of teenagers. Similar challenges pervade message and document schema.)

Who's the dog and who's the tail?

When two organizations try to communicate, there is always an economic dog and an economic tail. The dog wags the tail and the tail moves. In messaging and/or document definition, it is the economic dog that defines the semantics. If there is any ambiguity, the onus remains on the economic tail to work it out.

Walmart, for example, is a dominant force in retailing and dictates many things to manufacturers that want to sell through it. In addition to packaging and labeling standards, Walmart imposes messaging standards for communication with the manufacturers. Walmart prescribes the meaning, and the manufacturers adapt.

Schema versus name/value. Increasingly, schema definition is captured in the “name” of a name/value pair. This can be seen in the move from a SQL DDL (which is intended to be a tight and prescriptive statement of the meaning of the data) to XML (which is much more intended as the author's description of what was written in the message or document). Name/value pairs (and their hierarchical cousins in XML, JSON, and the like) are becoming the standards for data interchange.

We are *devolving* from schema to name/value pairs. The transition away from strict and formal typing is causing a loss of correctness. Bugs that would have been caught by a stricter description can now squeeze through.

On the other hand, we are *evolving* from tightly defined prescriptive schema to the more adaptable, flexible, and extensible name/value pairs. In very large, loosely coupled systems, adaptability and flexibility seem to offer more value than crispness and clarity.

Extensibility: Scribbling in the margins. Extensibility is the addition of stuff that was not specified in the schema. By definition, it is data the reader did not expect but the sender wanted to add, anyway. This is much like scribbling additional instructions in the margins of a paper form that was not designed for such additions. Sometimes the person reading the form will



When data is contained inside a database, it may be normalized and subjected to DDL schema transformations. When data is unlocked, it must be immutable.



notice these additional instructions, but sometimes not.

Stereotypes are in the eye of the beholder. A person dresses in a style usually intended to provide information to strangers. When people began to transition from living in small villages (where everyone knew you and knew what to expect from you) to living in large cities (where most of the folks you encounter are strangers), it became important to signal some information to others by dressing in a particular way.

People dynamically adapt and evolve their dress to identify their stereotype and community. Some groups change quickly to maintain elitism (for example, grunge); others change slowly to encourage conformity (for example, bankers).

Dynamic and loose typing allows for adaptability. Schema-less interoperability is *not* as crisp and correct as tightly defined schema. It presents more opportunities for confusion.

When interpreting a message or document, you must look for patterns and infer the role of the data. This works for humans when they examine a stranger's stereotype and style. It allows for flexibility for data sharing (which includes a cost for making mistakes).

Sure and certain knowledge of a person (or schema) has advantages. Scaling to an infinite number of friends (or schemas) isn't possible. The emerging adaptive schemas for data like stereotypes in people. While you can learn a lot quickly (but not perfectly), it scales to very large numbers of interactions.

Descriptive, not prescriptive schema. In very large and loosely coupled systems, we see descriptive, not prescriptive schema:

- *Descriptive schema.* At the time the data is written, the author describes what is intended.

- *Prescriptive schema.* The data is forced into a fixed format that is consistently shared by all authors.

The larger and more disconnected the system, the more impractical it is to maintain a prescriptive schema. Over time, the attempt at consistency becomes a fragility that breaks. Natural selection drives extremely large systems away from consistency and prescription.

Would You Like Some Mulligan Stew?

In the early 1900s, hobos living on the rails would frequently eat “Mulligan stew.”⁵ Members of the community would contribute whatever ingredients were available, and everything would be tossed into a community pot. Based on circumstances, there could be many different meats, fishes, vegetables, and other edibles all combined into the same stew. Leftovers would form the basis of tomorrow’s stew. While this dish can be delicious, it’s difficult to define its contents precisely.

Many large systems extract data from a tremendous number of sources. This disparate data is processed, crunched, shoehorned, and pounded into a mush—a Mulligan stew of sorts. By combining data and reasoning about its relationship to other data, astonishing businesses have sprouted such as Google and Amazon. This section examines some of the mechanisms by which this is accomplished and the compromises needed in the process.

Extract, transform, and load. Many data systems perform extract, transform, and load ETL: collecting data from one or more sources, converting it into a new form with a transformation, and then loading it into a data store for further operation (Figure 3). This is a valuable process growing in its use and scale. The transformation of the data is frequently a lossy operation, meaning that there is less information after the transformation than before. You can’t take the transformed data and go back to the original infor-

mation. It is much like creating a hamburger from some chunks of beef—you can’t go back to the original form (see Figure 3).

Lest you think this is meant as a criticism, I want to point out that I am very fond of hamburgers. Also, I am very fond of the gigantic data sets produced by these ETL operations, including the search indices of Google and Bing, as well as the product catalog from Amazon. While the result cannot support backward processing, which returns us to the source data, the resulting stew is really tasty.

Building a heterogeneous product catalog. Shopping sites such as Amazon, Google Shopping, and Bing Shopping gather data from literally millions of sources through a combination of data feeds from partners and crawling the Web. Rarely is this data in a standard format, and, if it is, there are always challenges with the semantics. (This same discussion can apply to creating search information for Internet searches.)

Typically, the first step involves cleanup. How can you unify the data so it can be compared and contrasted? For example:

► *Who’s the manufacturer?* HP, Hewlett-Packard, Hewlett-Packard, Compaq, Digital, DEC, H-P?

► *What’s the color?* Green, emerald, asparagus, chartreuse, olive, pear, shamrock?

Mapping into a single value increases the ability to see the data as equivalent, but you may lose knowledge.

Identity and ambiguity. There are

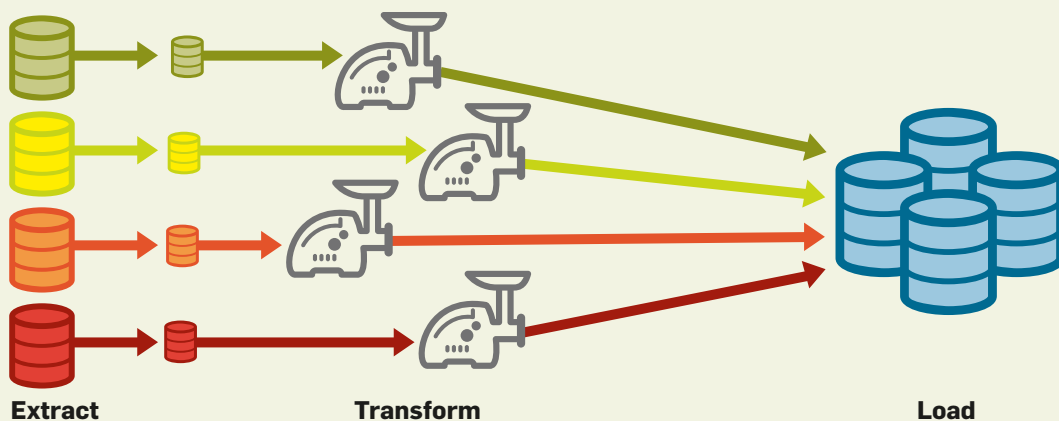
many competing and unifying mechanisms for assigning a unique number for a type of product—for example, GTIN (Global Trade Identification Number), UPC (Universal Product Code), and EAN (European Article Number). Amazon internally uses its ASIN (Amazon Standard Identification Number), which includes the book-centric standard ISBN (International Standard Book Number).

As an example of the challenges in standardizing identity, a different ISBN is assigned to each edition and variation (except reprinting) for each book. This means that there is a separate identity for the paperback and hardback of each book. Reviews and commentary on the literary value need to be connected separately across the identities. Similarly, some products have the same UPC even when they have different colors. For some reason, shoes carry no standard identification number but, instead, are identified by manufacturer and brand name (and some manufacturers recycle the brand names).

Based upon these tenuous identities, a product catalog must now try to discern which merchants’ offerings match other offerings. Can we determine an industry-standard identification? Can we accurately correlate this product from merchant A with the same product from merchant B so we can offer the customer a comparison?

Transformation and consolidation. Once multiple products from a set of merchants are recognized as the same and mapped to a unique identity, it is time to create a unified product de-

Figure 3. ETL operations can produce gigantic data sets.



scription. The best information from multiple merchants (and sometimes the manufacturer) is processed to create a catalog entry. Based on the identity, you want to get the best product description possible.

Observing patterns by inference. Yet another form of knowledge is gathered by patterns and inference. By studying the contents of data, it is sometimes possible to infer that two seemingly disparate identities are really the same. This happens when looking at a catalog of shoes from different merchants, for example. The lack of an industrywide identifier such as a UPC for shoes means they are particularly subject to erroneously interpreting two shoe descriptions as independent items.

Inference engines look at the relationships between identities (and attributes of the identities) to recognize that two ostensibly different items are the same. Once you realize two identities are the same, new relationships are recognized by merging the two identities into one. Inference engines are constantly appending new information to the data gleaned from the Web and partner companies.

Serendipity when you least expect it. The recognition gathered by inference engines is incredibly valuable. Some important areas of application include:

► *Fraud analysis.* The recognition of fraudulent patterns of credit card usage is vital in online credit card usage. It is only because of these fraud-detection mechanisms that online pay-

ments are economically viable.

► *Homeland Security.* There is tremendous traction in tracking surprising patterns of suspicious people. There is also a recent growth in *anonymizing* identities in a pattern that shares relationships without divulging identities and violating privacy.

► *Item matching in marketplace catalogs.* Are those two SKUs the same product for sale?

Inevitably, this “enhancement” of the source data results in new insight and increasing distance from the source data.

What kind of stew is that mulligan stew? Identifying the provenance of data is increasingly difficult. Where did this result come from? Who is responsible for the correctness of the data? When data from multiple sources is smashed together with data from other derived sources, what is the provenance? What is to be done when legal changes mean that some of your input data should no longer be used? The loss of licensing rights and/or the divestiture of a portion of your company can mean the input data is no longer yours to use. If that input data has contributed to the Mulligan stew of your intellectual property, you have a problem. (There is some irony to be found in complaining about provenance in an article that uses Wikipedia for some of its references. Hmmm...)

Through the looking glass. When considering the large and scalable applications that are gravitating to the NoSQL model, we must take into ac-

count the semantics of the derived data that inevitably accompanies them. The application probably wants a simplified and distilled view of the aggregated data. This view is inevitably a lossy prism that looks at a subset of the knowledge. But only by accepting the inevitability of the loss can we look for ways to gain business value from the resulting tasty stew.

Heisenberg was an Optimist

Werner Heisenberg⁶ showed that as things get small, you cannot know everything precisely. Well, it turns out, as things get big, we can get confused, too.

How certain are you of those search results? Web search results are very challenging. Web crawlers are, well, crawlers. Ensuring that search results are relevant to the person issuing the request is notoriously difficult because of the ambiguity of the request and the sheer magnitude of the Web. The demographics of the audience affect the desired results. Teenagers frequently want different results than their parents, for example. The requester’s location, interests, and recent searches dramatically affect the expected answers.

The U.S. census is hard. Any census of a large collection of people is very hard to achieve. You cannot accurately count people when you can’t be at every house at precisely the same moment. People move; people lie; people live with their girlfriends and don’t tell Mom and Dad. Should you count by address, Social Security number, name, or something else? What if someone dies after you have counted him or her? What if someone is born after the family’s house is counted but before the census is complete?

Chads and the election results. In the U.S. presidential election of 2000, the results ended up depending on the State of Florida. The vote in Florida was nearly tied, and each recount yielded a different answer. Much of the ambiguity came from the automatic voting machines and the presence of chads (illustrated in Figure 4) that occasionally made it difficult (and/or subjective) to interpret the voters’ intentions.

The vote in Florida was very close by any measure, with the first result showing a difference of about 0.01%.

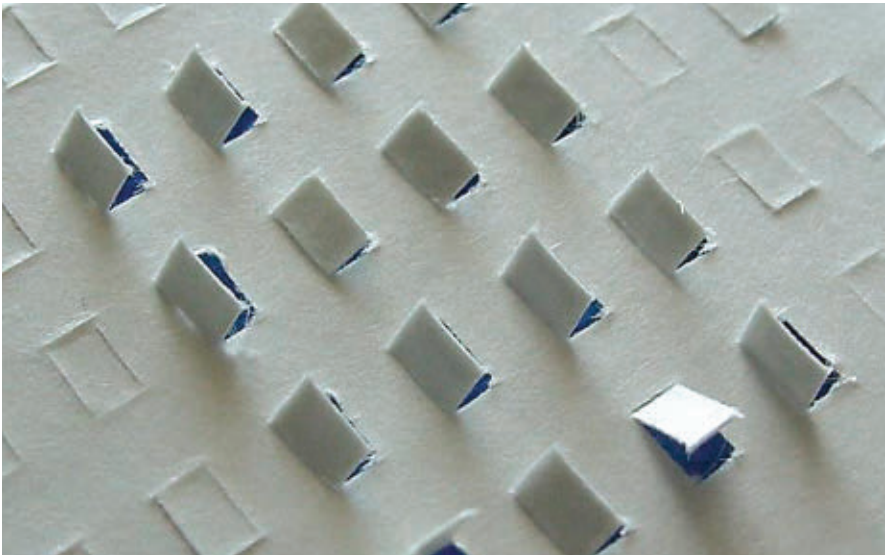


Figure 4. Chads created ambiguity in a large-scale data set—Florida’s vote count.

In any large system, the inaccuracies don't matter much until precision is required. If the winner has a 10% lead, there is no need for a recount.

Under scale, we lose precision. Big is hard! Time, meaning, mutual understanding, dependencies, staleness, and derivation all become a challenge. Heisenberg pointed out that at a small scale, uncertainties are a fact of life. In computing at a large scale, uncertainty is also a fact of life.

Conclusion

NoSQL systems are emerging because the world of data is changing. The size and heterogeneity of data means that the old guarantees simply cannot be met. Fortunately, we are learning how to meet the needs of business in ways outside of the old and classic database.

Data on the outside versus data on the inside. In a paper presented at the 2005 Conference on Innovative Data System Research (CIDR), I observed that data that is locked (and inside a database) is seminally different from data that is unlocked.² Unlocked data comes in clumps that have identity and versioning. When data is contained inside a database, it may be normalized and subjected to DDL schema transformations. When data is unlocked, it must be immutable (or have immutable versions).

Database normalization is designed to help avoid update anomalies. In large-scale systems, you don't update data, you add new data or create a new version.³ The treatment of data in large-scale systems is seminally different.

A new theory for data needed. The database industry has benefited immensely from the seminal work on data theory started in the 1970s. This work changed the world and continues to be very relevant, but it is apparent now that it captures only part of the problem.

We need a new theory and taxonomy of data that must include:

- *Identity and versions.* Unlocked data comes with identity and optional versions.

- *Derivation.* Which versions of which objects contributed to this knowledge? How was their schema interpreted? Changes to the source would drive a recalculation just as in Excel. If a legal reason means the source data may not be used, you

Another form of knowledge is gathered by patterns and inference. By studying the contents of data, it is sometimes possible to infer that two seemingly disparate identities are really the same.

should forget about using the knowledge derived from it.

- *Lossiness of the derivation.* Can we invent a bounding that describes the inaccuracies introduced by derived data? Is this a multidimensional inaccuracy? Can we differentiate loss from the inaccuracies caused by sheer size?

- *Attribution by pattern.* Just like a Mulligan stew, patterns can be derived from attributes that are derived from patterns (and so on). How can we bound taint from knowledge that we are not legally or ethically supposed to have?

- *Classic locked database data.* Let's not forget that any new theory and taxonomy of data should include the classic database as a piece of the larger puzzle.

It's a great time to be working with data. Lots of new and exciting things are happening! 

Related articles on queue.acm.org

A Call to Arms

Jim Gray, Mark Compton

<http://queue.acm.org/detail.cfm?id=1059805>

Exposing the ORM Cache

Michael Keith, Randy Stafford

<http://queue.acm.org/detail.cfm?id=1394141>

The Case Against Data Lock-in

Brian W Fitzpatrick, JJ Lueck

<http://queue.acm.org/detail.cfm?id=1868432>

References

1. Baker, J. Bond, C., Corbett, J. C., Furman, J.J., Khortin, A., Larson, J., Léon, J. M., Li, Y., Lloyd, A., Yushprakh, V. Megastore: providing scalable, highly available storage for interactive services. In *Proceedings of Conference on Innovative Data Systems Research* (Asilomar, CA, 2011); www.cidrdb.org/cidr2011/Papers/CIDR11_Paper32.pdf.
2. Helland, P. Data on the outside versus data on the inside. *CIDR 2005*; www.cidrdb.org/cidr2005/papers/P12.pdf.
3. Helland, P. Normalization is for sissies. Conference on Innovative Data Systems Research (2009); www-db.cs.wisc.edu/cidr/cidr2009/gong/20Helland.ppt.
4. Microsoft Developer Network. SQL Azure (2011); <http://msdn.microsoft.com/en-us/windowsazure/sqlazure/default.aspx>.
5. Wikipedia. Mulligan stew; [http://en.wikipedia.org/wiki/Mulligan_stew_\(food\)](http://en.wikipedia.org/wiki/Mulligan_stew_(food)).
6. Wikipedia. Werner Heisenberg; http://en.wikipedia.org/wiki/Werner_Heisenberg.

Pat Helland has been working in distributed systems, transaction processing, databases, and similar areas since 1978. For most of the 1980s, he was the chief architect of Tandem Computers' Transaction Monitoring Facility (TMF), which provided distributed transactions for the NonStop System. With the exception of a two-year stint at Amazon, Helland has worked at Microsoft Corporation since 1994 where he was the architect for Microsoft Transaction Server and SQL Service Broker. He is currently working on Cosmos, a distributed computation and storage system that provides back-end support for Bing.

© 2011 ACM 0001-0782/11/06 \$10.00

How do large-scale sites and applications remain SQL-based?

BY MICHAEL RYS

Scalable SQL

ONE OF THE leading motivators for NoSQL innovation is the desire to achieve very high scalability to handle the vagaries of Internet-size workloads. Yet many big social Web sites (such as Facebook, MySpace, and Twitter) and many other Web sites and distributed tier 1 applications that require high scalability (such as e-commerce and banking) reportedly remain SQL-based^a for their core data stores and services.

The question is, how do they do it?

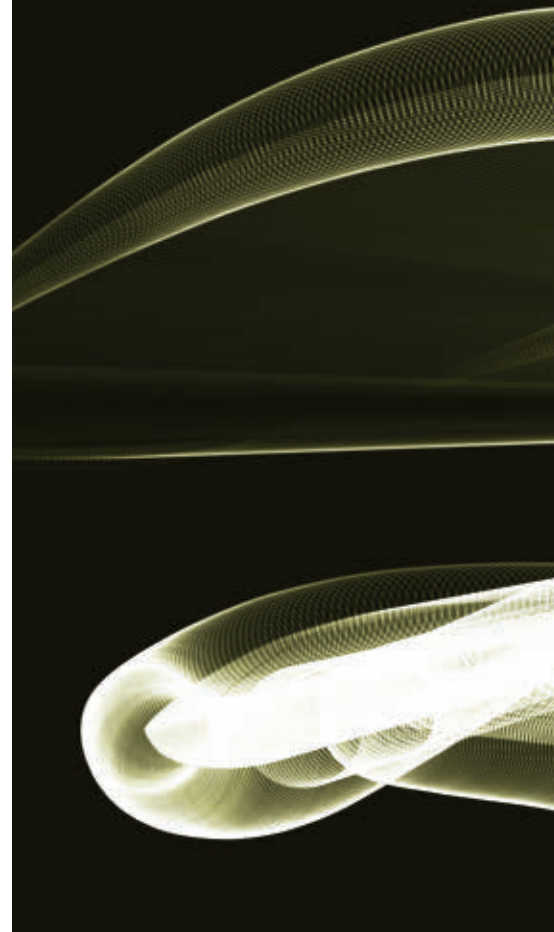
The main goal of the NoSQL/big data movement is to achieve agility. Among the variety of agility dimensions—such as *model* agility (ease and speed of changing data models), *operational* agility (ease and speed of changing operational aspects), and

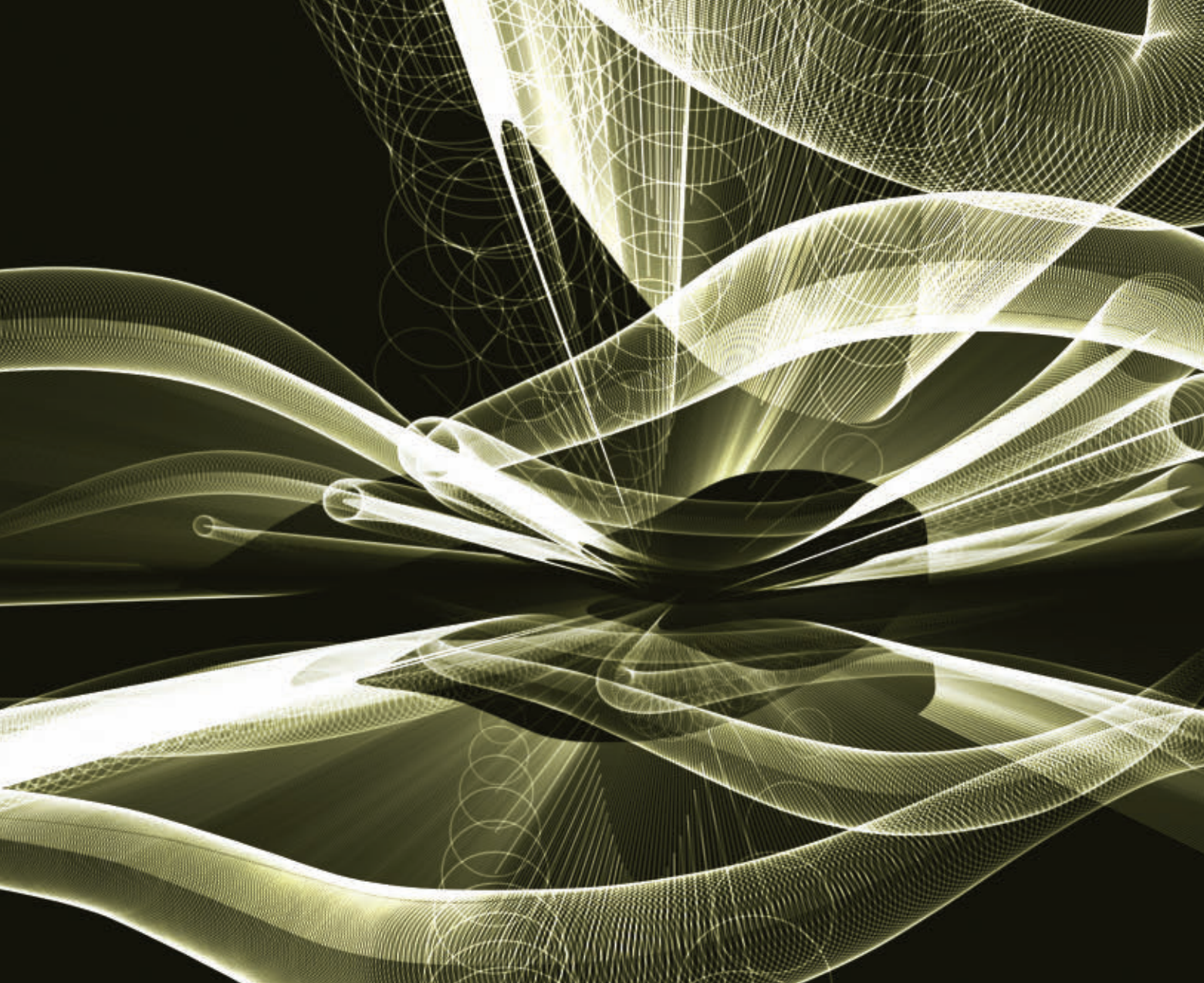
^a SQL is actually the name of a declarative query language, while more precisely this article concerns traditional relational database systems. Since it is common to talk about NoSQL as the opposite of relational database systems, we have taken the editorial liberty to use SQL as a synonym for relational database systems.

programming agility (ease and speed of application development)—one of the most important is the ability to quickly and seamlessly scale an application to accommodate large amounts of data, users, and connections. Scalable architectures are especially important for large distributed applications such as social networking sites, e-commerce Web sites, and point-of-sale/branch infrastructures for more traditional stores and enterprises where the scalability of the application is directly tied to the scalability and success of the business.

These applications have several scalability requirements:

- ▶ *Scalability in terms of user load.* The application needs to be able to scale to a large number of users, potentially in the millions.
- ▶ *Scalability in terms of data load.*





The application must be able to scale to a large amount of data, potentially in petabytes, either produced by a few or produced as the aggregate of many users.

► *Computational scalability.* Operations on the data should be able to scale for both an increasing number of users and increasing data sizes.

► *Scale agility.* In order to scale to increasing or decreasing application load, the architecture and operational environment should provide the ability to add or remove resources quickly, without application changes or impact on the availability of the application.

Scalable Architectures

Several major architectural approaches achieve high-level scalability. Most of them provide scale-out based on some form of functional and/or data

partitioning and distributing the work across many processing nodes.

Functional partitioning often follows the service-oriented paradigm of building the application with several independent services each performing a specific task (see Figure 1). This allows the application to scale out by assigning separate resources to these services as needed. Functional scale-out partitioning alone, however, often does not provide enough scalability since the number of tasks is limited and not in direct relationship to the big drivers of scalability requirements: the number of users and size of data. So functional partitioning is often combined with data partitioning.

Data partitioning distributes the application's processing over a set of data partitions (Figure 2). Different forms of data partitioning are de-

ployed based on the topology of the processing nodes and the characteristics of the data. For example, if the user base is geographically dispersed and there is a locality requirement for scalability and performance reasons, such as in worldwide social networking sites, then data often is partitioned according to those geographic boundaries. On the other hand, data may be more randomly partitioned—for example, based on customer IDs—if the scale-out requirements are more constrained by the cost of running data-analysis algorithms over the data. In this case, equal partition sizes are more important.

Once an application is built using a distributed model to achieve scale, it will have to deal with a set of requirements above and beyond simple centralized application structures:

► Because of the distribution of both data and processing, the database that in a centralized application model would provide a *consistent* view of the data and transactional execution is now distributed among many databases. Thus, the application (or a middle tier) has to provide an additional transactional/consistency layer to provide consistency across the partitions.

► In addition, changes to the applications have to be rolled out to all the partitions in a way that will not interfere with the consistency guarantees and requirements of the application. For example, if the application issues distributed queries against a set of tables that are partitioned across several nodes, and the application is updating the schema of some of these distributed tables, then either the schema change needs to be backward-compatible so it can be rolled out locally without affecting the ongoing queries, or the schema must be updated globally, thus impacting the application's *availability* during the rollout phase.

► Finally, there is an increased probability of partition node failures and network *partitioning*. Therefore, nodes

need to be made redundant and applications have to be resilient to network partitioning.

Furthermore, all three of these requirements have to be fulfilled without negatively impacting the availability of the application's services, the main reason why the application probably was scaled out in the first place.

In 2000, Eric Brewer made the conjecture that it is impossible for a distributed Web service to provide all three guarantees—consistency, availability, partition tolerance—at the same time.¹ This conjecture is now commonly known as the CAP theorem² and is one of the main arguments why traditional relational database techniques that provide strong ACID guarantees (atomic transactions, transactional consistency and isolation, and data durability) cannot provide both the partition tolerance and availability required by large-scale distributed applications.

So why are many of the leading social networking sites (Facebook, MySpace, Twitter), e-commerce Web sites (hotel reservation systems and shopping sites), and large banking

applications still implemented using traditional database systems such as MySQL (Facebook,³ Twitter⁹) or SQL Server (MySpace,⁷ Choice Hotels International,⁶ Bank Itau⁵) instead of using the new NoSQL systems?

How Do You Scale Out with SQL?

The high-level answer is that the application architecture is still weighing the same trade-offs required by the CAP theorem. Given that the availability of the application has to be guaranteed for business reasons, and that partition and node failures cannot be excluded, the application architecture has to make some compromises around the level of provided consistency. Note this does not mean that relational databases cannot be used per se; it means the strong consistency guarantee of a single partition node cannot be made across all nodes and that the application architecture cannot use “traditional” database technologies such as distributed querying, full ACID transactions, and synchronous processing of requests without running into availability and scalability issues.

For example, traditional distributed query engines such as Microsoft SQL Server's linked servers assume close coupling of the data sources and are not able to adjust to quickly changing topologies—whether because of nodes being added or because of node failures. They operate synchronously and will wait for nodes to reply or fail the query in case of a node failure, thus impacting availability of the service.

What are some of the ways to build scalable applications using relational database systems as their underlying data stores? Basically the application architectures follow the same service-oriented, functional- and data-partitioning schemes outlined previously. Each leaf partition will be using a relational database, providing local consistency and query processing. To guarantee node availability, each node will be mirrored and made highly available. Depending on the service-level guarantee around failover and read versus update frequency, each mirror will be managed either synchronously or asynchronously.

Global consistency across the many locally consistent nodes will be provided to the level that the application

Figure 1. Functional partitioning of a commerce system.

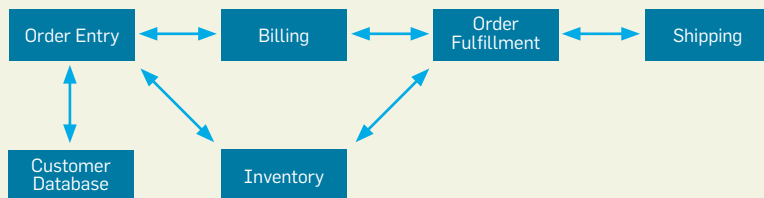
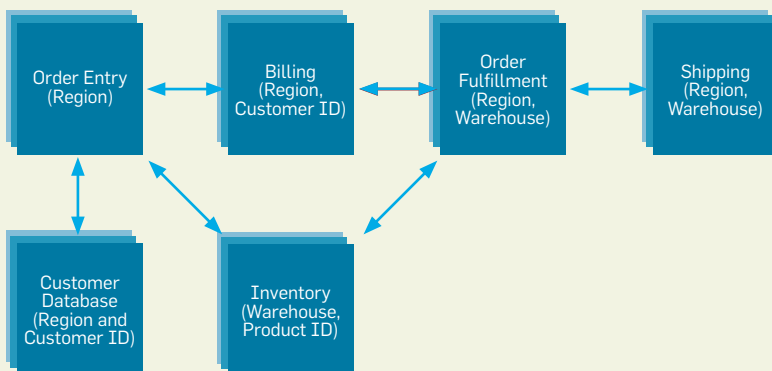


Figure 2. Data partitioning of a commerce system with partitioning keys.



requires, most often relaxing the atomicity, strong consistency, and/or isolation of the global operation. Many techniques exist, such as open nested transaction systems (Saga,⁴ multilevel concurrency control¹⁰) and optimistic concurrency control approaches, and specific partitioning and application logics to reduce the risk of inconsistencies. For example, open nested multilevel transactions relax transactional isolation by allowing certain local changes to become globally visible before the global transaction commits. This increases transactional throughput at the risk of potentially costly compensation work when a global transaction and its impact have to be undone. Thus, the openness often is restricted to specific operations that are commutative and have a clearly defined compensating action. In practice, such advanced transaction models have not yet been widely used, even though some transaction managers provide them.

More frequently, the application partitions data in a first step to avoid local conflicts and then uses optimistic approaches that assume that conflicts rarely occur. This approach takes into account the idea that most people are in fact fine with eventually consistent states of the global data.

Accepting short-term “incorrect” global states and results is actually pretty common in our day-to-day lives. Even bank transactions are often “eventually consistent.” For example, redeeming a check or settling an investment transaction will not be fully consistent at a global level at the time the transaction is executed. The money will potentially go into the seller’s account before it gets deducted from the buyer’s account, but there is a guarantee that the money will eventually be deducted and the global state will become consistent.

Using eventual consistency is a more complex application design paradigm than assuming a globally consistent state at all times. The programmer has to determine the acceptable level of inconsistencies—how long the data can be kept in an inconsistent state. The platform provider has to design the system in a way that programmers can easily understand the possible inconsistencies and provide mechanisms



Besides providing a scalable architecture, Service Broker provides a communication fabric guaranteeing that messages to a service are delivered reliably, in order, and exactly once.



to handle them when they appear. Often the agility and scalability gains are worth the additional complexity of the application architecture.

Using eventual consistency as an acceptable global consistency guarantee also allows the application to provide availability during network failures and thus achieve higher scalability. On the one hand, updating a node that has become unavailable will no longer block or fail the global transaction, as long as the system can guarantee that it will eventually be updated. On the other hand, eventual consistency allows the application to operate on older data and still provide useful results; sometimes it even allows partial results if a node cannot be queried (although this is a decision the application has to make). It also means that the architecture can be built using asynchronous services that will provide for higher scalability because the functional services and individual data partitions can do their work without blocking the application.

An Example of How to Scale with SQL

As we already mentioned, several applications with high scalability requirements are being built on top of traditional relational database systems. For example, Twitter uses the NoSQL database Cassandra for some of its needs, but its core database system that manages tweets is still using the MySQL relational database system.⁹

The following example presents a high-level overview of how MySpace achieves scalability of its architecture using Microsoft SQL Server. MySpace is still one of the largest social networking sites. In 2009 it used 440 SQL Server instances to manage 130 million users and one petabyte of data with 4.4 million concurrently active users at peak time.⁷

As outlined earlier, MySpace has chosen to use both functional and data partitioning. Data partitions are geographically distributed to be closer to the users in an area, as well as becoming further partitioned by user IDs for scale. This makes sense since most users will want to access their own data most frequently. Obviously, since MySpace is a social networking site where individual users connect and

leave messages and comments, operations not only target a single partition, but also need to update data across partitions. Given the large demands on availability and scalability, MySpace needs to achieve a balance between scale and correctness.

The basic approach is to perform most of the work in an asynchronous fashion. The asynchronous processing of the change events and interactions with the application provides high availability, and by having the partitions operate on the queued requests in a uniform fashion, the system is able to scale out easily. Using a reliable message infrastructure provides the guarantee that the changes eventually become visible, thus delivering eventual correctness.

Figure 3 provides a high-level abstraction of MySpace's service dispatcher architecture. It consists of a few dozen request routers that dispatch incoming requests to perform a certain user or system action—for example, posting a comment on a friend's picture, submitting a blog entry, or a system request such as deploying a new schema object. During steady state, the request routers are exact copies of each other, including a routing table mapping services to partitions.

The requests are asynchronously distributed across the routers and get dispatched to the individual account partitions (around 440 in the case of MySpace) and the requested service endpoint. Note that the account partitions provide all the same services and schemata at steady state, thus guaranteeing that every service can be provided by every node without being dependent on any other node.

Each of the routers and each of the partitions and services are implemented using SQL Server and SQL Server Service Broker. Service Broker is the key ingredient that enables this architecture to work reliably and efficiently. It provides the asynchronous messaging capabilities that allow the requests to flow at a high rate between the services. Each service exposes a queue to accept requests and the ability to dispatch workers on each item in the queue. Service Broker, like other service-bus and asynchronous messaging components, allows scaling out by simply adding multiple instances of the same service across different partitions. Requests are load balanced across these service instances without having to change the application logic. An interesting difference to some of the other message buses such as MQ-

Series, RabbitMQ, NServiceBus, and Microsoft Message Queuing (MSMQ) is that Service Broker is deeply built into the database engine.

Besides providing a scalable architecture, Service Broker provides a communication fabric guaranteeing messages to a service are delivered reliably, in order, and exactly once. This guarantees that even in case of a network partition or a node failure, a message is not lost but will eventually be delivered once the node has been reconnected. Since every service will be performed by the database server, local consistency is provided at the level specified for the specific transaction. The use of Service Broker to build and scale the services will provide global eventual consistency.

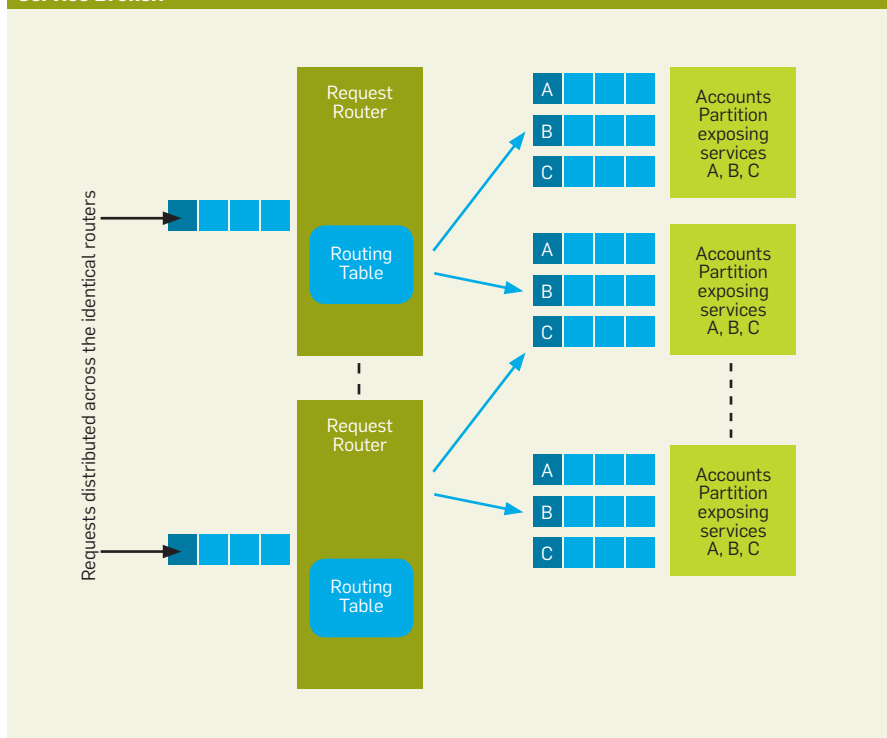
The availability of each partition can be improved by providing a failover copy using database mirroring. If a failover occurs, the Service Broker connection also automatically and transparently fails over.

The application scale-out architecture as described avoids a single point of failure and contention by replicating all the routing information across all the request routers and providing the services on all partitions. The asynchronous processing using Service Broker provides scalability, as well as eventual consistency. The architecture, services, and partitioning, however, will evolve over time. Therefore, the changes to the routing information when data gets repartitioned and the updates to services and schemas also need to be maintained in a scalable way. It would not be good if a global lock is being taken across all the request routers when adding a new partition to the routing table.

To address this, the current architecture uses the same Service Broker-based approach to roll out changes to the services and schemas. A repartition of the account services will be updated asynchronously. To detect a change in the partition by a router before its routing table has been updated, the partitions will fail a request if the partition assumption is invalid and will provide updated information back to the router, which then retries the request based on the new routing information.

A similar architecture is also being used for several e-commerce Web sites

Figure 3. High-level diagram of a scalable service dispatch architecture using SQL Server Service Broker.



that build on relational databases. For example, Bank Itau provides a scalable branch banking system⁵ and Choice Hotels International has a highly scalable online hotel reservation system⁶ using asynchronous messaging.

Summary and Outlook

Building scalable database applications is not necessarily a question of whether one should use a relational database system or a NoSQL system. It is more a question of choosing the right application architecture that is agile enough to scale. For example, combining asynchronous messaging with a relational database system provides the powerful infrastructure that enables developers to build highly scalable, agile applications that provide partition tolerance and availability while providing a high level of eventual consistency.

Scale-out applications with SQL are being built using similar architectural principles as scale-out applications using NoSQL while providing more mature infrastructure for declarative query processing, optimizations, indexing, and data storage/high availability. In addition, scaling out an existing SQL application without having to replace the data tier with a different database system that has different configuration, management, and troubleshooting requirements is very appealing.

Other aspects such as data models, agility requirements, query optimization, data-processing logic, existing infrastructures, and individual capabilities, strengths, and weaknesses will have to be considered as well when deciding between a SQL and NoSQL database system. Discussing these aspects are unfortunately outside the scope of this article.

All database systems, however, whether relational or NoSQL, still need to provide additional services that make it easier for the developer to build massively scalable applications. For example, relational database systems should add integrated support for data-partitioning scale-out such as sharding.⁸ NoSQL databases are working on providing more of the traditional database capabilities such as secondary indices, declarative query languages, among others.

Until the database systems provide simple-to-use scale-out services, de-

All database systems, be they relational or NoSQL, still need to provide additional services that make it easier for the developer to build massively scalable applications.

velopers will have to design their applications with scale-out in mind and use more generic application patterns such as asynchronous messaging, functional and data partitioning, and fault tolerance to build fault-resilient systems that provide high availability and scalability.

Acknowledgments. I would like to express my gratitude to Erik Meijer, Luis Vargas, and Terry Coatta for their reviews and insightful comments that greatly improved this article. 

Related articles on queue.acm.org

Data in Flight

Julian Hyde

<http://queue.acm.org/detail.cfm?id=1667562>

Cybercrime 2.0: When the Cloud Turns Dark

Niels Provos, Moheeb Abu Rajab,

Panayiotis Mavrommatis

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

Beyond Relational Databases

Margo Seltzer

<http://queue.acm.org/detail.cfm?id=1059807>

References

1. Brewer, E. A. Towards robust distributed systems. (Invited talk) Principles of Distributed Computing (Portland, OR, July 2000).
2. CAP; http://en.wikipedia.org/wiki/CAP_theorem.
3. Facebook; <http://blog.facebook.com/blog.php?post=7899307130>.
4. Garcia-Molina, H., Salem, K. Sagas. *Proceedings of the 1987 ACM SIGMOD International Conference on Management of Data*. 249–259; <http://www.informatik.uni-trier.de/~ley/db/conf/sigmod/sigmod87.html#Garcia-MolinaS87>.
5. Helland, P., Stelzmueller, C. SQLCAT: SQL service broker: high-performance distributed applications in real-world deployments. PASS Summit (2009); <http://www.softconference.com/pass/sessionDetail.asp?SID=174551>.
6. Microsoft Case Studies. Global hotel company delivers reservations in milliseconds with highly reliable system (2011); <http://www.microsoft.com/casestudies/Microsoft-SQL-Server-Service-Broker/Choice-Hotels-International/Global-Hotel-Company-Delivers-Reservations-in-Milliseconds-with-Highly-Reliable-System/4000009199>.
7. Microsoft Case Studies. MySpace uses SQL Server Service Broker to protect integrity of 1 petabyte of data (2009); http://www.microsoft.com/Casestudies/Case_Study_Detail.aspx?casestudyid=4000004532.
8. MSDN Blogs; <http://blogs.msdn.com/b/cbiyikoglu/archive/tags/federations/>.
9. Twitter; <http://engineering.twitter.com/2010/07/cassandra-at-twitter-today.html>.
10. Weikum, G. A theoretical foundation of multilevel concurrency control. *Proceedings of the 5th ACM SIGACT-SIGMOD Symposium on Principles of Database Systems* (1986), 31–43.

Michael Rys (mrys@microsoft.com) is principal program manager on the SQL Server RDBMS team at Microsoft. He is responsible for the Beyond Relational Data and Services scenario that includes unstructured and semi-structured data management, search, Spatial, XML, and others. He represents Microsoft Corp. in the W3C XML Query working group and the ANSI SQL standardization effort. For more information, see http://sqlblog.com/blogs/michael_rys/default.aspx or follow Rys at @SQLServerMike (when he finds time to tweet).

© 2011 ACM 0001-0782/11/06 \$10.00

Methods for evaluating and effectively managing the security behavior of employees.

BY QING HU, ZHENGCHUAN XU, TAMARA DINEV, AND HONG LING

Does Deterrence Work

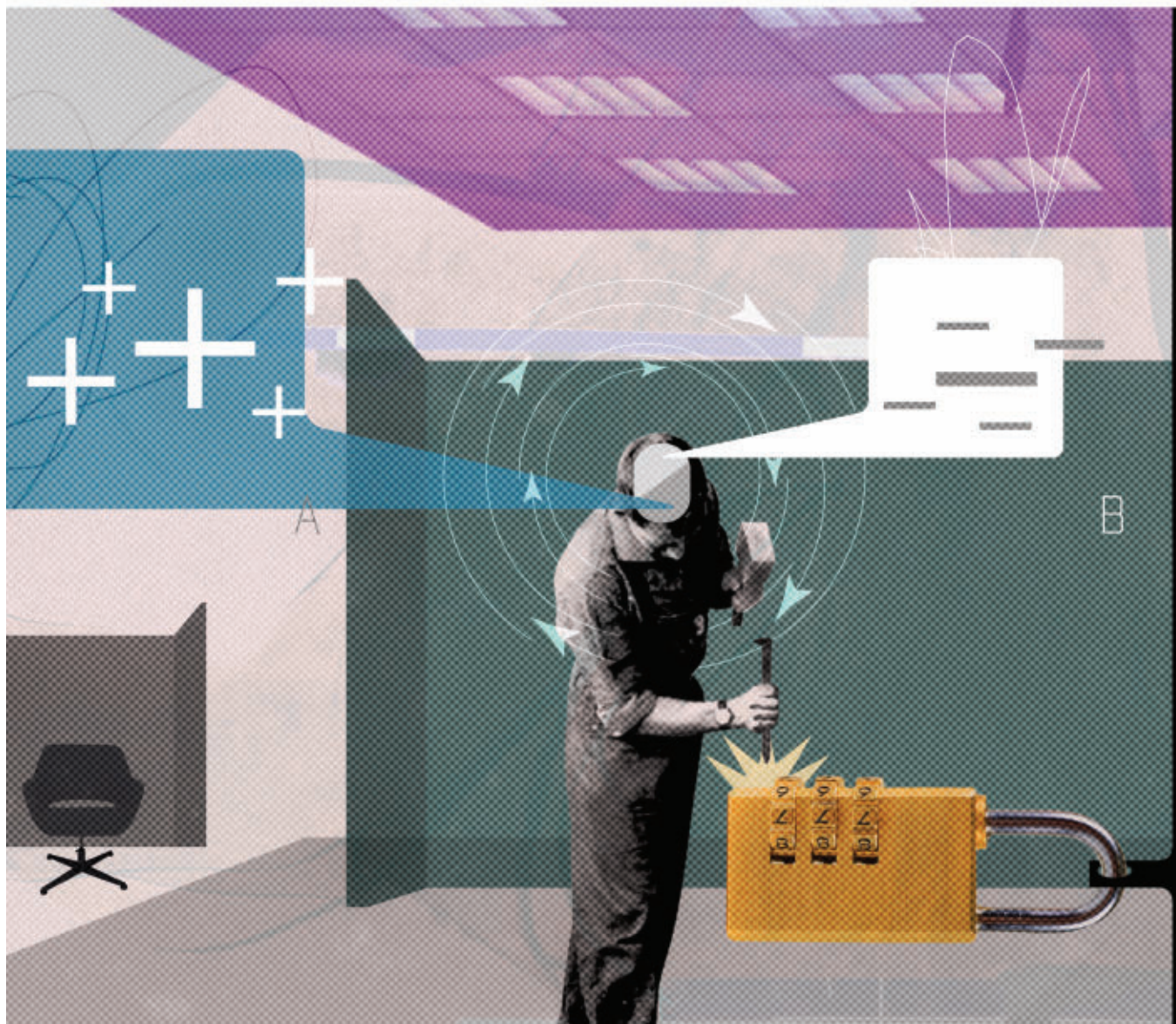
in Reducing Information Security Policy Abuse by Employees?

HACKING INTO CORPORATE IT systems and individuals' computers is no longer a sport for bragging rights, but a major organized economic activity aiming for significant profits controlled largely by underground networks of criminals and organized crime on a global scale.² The financial impact of the computer crimes and related activities is estimated at over one trillion dollars each year worldwide.¹⁷ Unfortunately, despite significant advances in hardware and software technologies against computer and network offenses, in the digital security ecosystem around any organization, human agents are still the weakest link in the defense against outside attacks and the most dangerous to the organizations from within. Indeed, the effectiveness of other elements in the

security system, such as security technology, organizational policies and procedures, as well as government regulations, are largely dependent on the effort of the human agents, especially those who work within the organizations.

While the media headlines tend to focus on the spectacular events perpetrated by external hackers, employees inside an organization often pose silent but more dangerous threats than those outside the organization,³⁰ due to their intimate knowledge about the organizational systems and the permissions they receive either properly or improperly for their work activities. In a recent survey of IT managers of global companies, 60% of the respondents said employee misconduct involving information systems (IS) is a top concern about information security.¹¹ The 2008 CSI Computer Crime and Security Survey shows that 44% of respondents reported insider abuse of computer systems, making it the second most frequent form of security breach, only slightly behind virus incidents (49%), but well above the 29% of respondents who reported unauthorized access from external sources.²²

In this study, we focus on information security policy violations by employees in organizational settings. Employee information security policy violations vary widely in motives, forms, targets, and consequences. We define information security policy violation as any act by an employee using computers that is against the established rules and policies of an organization for personal gains. By this definition, information security policy violations include but are not limited to unauthorized access to data and systems, unauthorized copying or transferring of confidential data, or selling confidential data to a third party for personal gains, and so forth. With this focus, two questions have been central to research and practice in information security in the last two decades: Why do employees go rogue and commit policy violations, and what could



employers do to minimize the threat and the damage? Recent research^{8,10,25} based on human behavioral theories has made two primary recommendations: implement comprehensive information security policies and procedures and conduct thorough awareness training for employees, and establish clear and swift sanctions against security misconduct to deter and reduce future violations by employees. Surveys^{11,22} indicate these recommendations have been widely adopted and implemented by companies, yet the threat of policy violations remains a top concern of information security management. In this article, we highlight a study conducted by the authors in an effort to understand the mystery and provide better insights into security management practices. We set out to extend the current re-

search on the behavioral aspects of information security and attempt to accomplish three objectives: to integrate multiple theories to develop a comprehensive model about security misconduct behavior in corporate settings; to test the causal relationships among rational choice, deterrence, individual propensity, and security misconduct behavior of individuals; and to provide prescriptive guidance to information security management based on the results of this research.

Criminological Theories of Human Behavior

The significant role of human agents in organizational information security has long been recognized by scholars, along with the effort in developing and deploying more advanced protective technologies and establishing and en-

forcing effective security policies and procedures. Early studies of information security by IS scholars were largely based on surveys of managers and employees in organizations using ad hoc theoretical or empirical frameworks. Only recently have IS scholars started to use more established theories in their analyses of the information security issues (for example, D'Arcy,⁸ Dinev,¹⁰ Siponen²⁵). Given the similarity between information security policy violations in organizational settings and criminal behavior in social settings, the theories and theoretical perspectives developed in criminology literature have been widely adopted as the foundations for information security research, including but not limited to deterrence theory, rational choice theory, and social control theory. Table 1 summarizes the main

thesis of the most relevant theories for this study.

Although rational frameworks have been popular in criminology literature, they are often used in conjunction with other theoretical frame-

works. Since human beings are only capable of bounded rationality,²⁴ it is reasonable to assume the rational decision-making process will be subject to a variety of individual and situational factors. Paternoster and Simp-

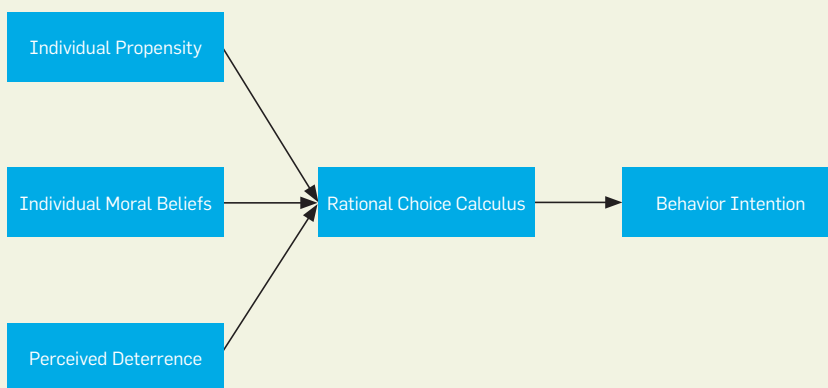
son¹⁹ emphasized that the rational choice theory of crime is essentially a subjective expected utility theory that has two assumptions: decisions by an individual to offend are based on the evaluation of cost and benefit of offending; and the critical elements in this calculus are the perceived or subjective expectations of cost and benefit by the individual. Therefore, the cognitive characteristics of the individual and the specific characteristics of the situation will affect how the benefit and cost are perceived.⁷ Tibbetts and Gibson²⁸ found that studies with both the rational choice measures and the estimates of individual propensities show that variables from these two perspectives significantly influence the deviant behavior, even when factors from other theoretical perspectives are controlled.

The literature review led us to propose a research model with the rational choice at its core and other theoretical frameworks at its periphery to form a theoretical model of information security policy violation behavior. The treatment of individual propensity, moral beliefs, and perceived deterrence as antecedents to the rational choice calculus may be debatable, but is primarily based on the findings of criminological studies (for example, Nagin,¹⁸ Piquero²¹) that use rational choice theory as the foundation for human behavior, as well as our preliminary testing that showed insignificant moderating effects by these three constructs on the rational choice theory model. We submit that when an individual is presented with an opportunity to commit policy violations, his or her behavior depends on the rational calculus of the costs and the benefits. However, the mechanisms of the cost-benefit evaluation are affected by at least three independent forces: two internal and one external to the individual. The two internal forces are the individual propensity (defined as the degree of low self-control), and the individual's moral beliefs (defined as the individual's judgment about right and wrong). The external force is the perceived deterrence related to the misconduct, defined as the perceived certainty, severity, and celerity of sanctions against the behavior. We argue that low self-control, moral beliefs,

Table 1. Highlights of relevant criminological theories.

Theory	Type	Description	Primary Sources
Rational Choice	Behavioral Theory	- Individuals are sensitive to the consequences of their actions and make reasoned judgments based on the cost-benefit analysis of the intended acts. - The decision to engage in criminal behavior is a function of the perceived costs and benefits of the criminal act.	Becker ⁵ Cornish ⁷ Paternoster ¹⁹
Self-Control	Behavioral Theory	- All human beings have the potential of committing crimes, but not everyone does because of differences in the ability of self-control. - Self-control is established early and remains relatively stable throughout an individual's lifetime. - Individuals with low self-control have a tendency to respond to tangible stimuli in the immediate environment, and are more likely to be seduced by the thrill and excitement of criminal acts.	Gettfredson ¹²
Deterrence	Behavioral Theory	- Human beings are fundamentally rational in their behavior and choose crime only when it pays. - Individuals are less likely to commit criminal acts if the perceived certainty, severity, and celerity of sanction against the acts are greater.	Gibbs ¹³ Tittle ²⁷
Shame	Behavioral Modifier	- Shame is the experience of temporary loss of self-esteem in an internal evaluation of the self by an individual. - Shame has a deterrence effect on various criminal acts through three mechanisms: self-imposed shame (or shame), socially imposed embarrassment, and state-imposed legal sanctions.	Lewis ¹⁶ Piquero ²¹
Moral Beliefs	Behavioral Modifier	- Moral beliefs are individual judgment of right and wrong about specific behavior. - Moral beliefs against committing criminal acts increase the degree of perceived sanctions and the shame associated with the acts. - Moral beliefs also have a negative effect on the perceived pleasure from the deviant behavior.	Bachman ⁴ Piquero ²¹

Figure 1. Conceptual model.



and deterrence are antecedents of the rational choice calculus rather than moderators as seen in some studies (for example, D'Arcy,⁸ Paternoster¹⁹). This is because we believe these factors are more likely to change how the benefits and costs are assessed rather than how the benefits and costs are weighed in the decision calculus by an individual. For instance, an individual with strong moral beliefs is likely to underestimate the benefits while over-estimating the risks of the misconduct. Similarly, an individual with low self-control seeking immediate thrill and excitement is likely to overestimate the benefits while underestimating the costs of committing the act. This thesis leads to the formulation of the integrated model of information security policy violation by internal employees, as shown in Figure 1.

Testing the Theory

Due to the secrecy involved in criminal or deviant behavior, it is natural that individuals will be unwilling to report their actual behavior and actions related to policy violations or crimes. In criminology and IS security research, faced with this difficulty, scholars have often resorted to the use of scenarios of criminal or abusive activities to elicit input from survey subjects. In this study, we adopted the scenario-based survey methodology to collect data from employees in organizations who may or may not have committed policy violations toward corporate computer systems. Using scenarios to elicit individual responses has been a common technique in criminology research (for example, Bachman,⁴ Paternoster,¹⁹ Piquero²¹), and it has been increasingly used by IS scholars in information security research (for example, D'Arcy,⁸ Siponen²⁵).

The constructs shown in the conceptual model were operationalized into latent variables that can be measured using a survey instrument, as shown in Table 2. The survey instrument was developed based on the proposed conceptual model after the constructs were operationalized into measurable items with a 7-point Likert scale. All of the items were adapted from the extant literature in order to maximize the validity and reliability of

the measurement model. Three carefully designed information security policy violation scenarios were presented to each respondent at the beginning of the survey to elicit his or her intention to do the same as the actor did in each of the scenarios. The scenarios included unauthorized access to payroll data, unauthorized access to and transfer of product designs, and stealing and selling confidential price and cost data to competitors. A validation question that asks the respondents about how likely the scenarios could occur in their companies was included after each scenario.

The survey was pilot tested using EMBA students enrolled in a major Chinese university. Oral comments were collected and diagnostic statistics were conducted. Minor revisions were made based on the results of the pilot test. The final survey was distributed to employees in five large organizations in China. These organizations were selected largely due to their willingness to cooperate with this

research after the authors contacted a number of organizations. Primarily because of the supportive arrangement of the managers of these five organizations, the response rate of the survey was nearly 100% from about 50 randomly selected employees in each organization. The surveys were distributed and collected by the research team directly in a meeting room setting at each location. The employees were assured the management would not have access to the individual surveys. In the end, 227 surveys were received, 207 were deemed as complete and usable, 20 were discarded due to incomplete answers. More than 79% of the respondents were younger than 35 years old, and more than 71% had a college or higher level education, reflecting a young and well-educated work force in these organizations. Over 75% of the respondents classified themselves as "employee" as opposed to having supervisory responsibilities. Of the respondents, 58% were male, and 42% were female, reflecting a typi-

Table 2. Operationalization of theoretical constructs.

Theoretical Construct	Latent Variables	Description
Individual Propensity	Low Self-Control (LSC)	Measured as a combination of the impulsivity, risking seeking, and self-centered characteristics of an individual.
Individual Moral Beliefs	Moral Beliefs (MRB)	Measured in terms of how morally wrong and morally unacceptable the intended act is to the respondent.
Perceived Deterrence	Perceived Certainty of Sanctions (CRT)	The perceived probability of being punished for the intended act.
	Perceived Severity of Sanctions (SVR)	The perceived degree of punishment for the intended act.
	Perceived Celerity of Sanctions (CEL)	The perceived swiftness of being punished for the intended act.
Rational Choice Calculus	Perceived Extrinsic Benefits (PEB)	The perceived material benefits of committing the intended act.
	Perceived Intrinsic Benefits (PIB)	The perceived mental pleasure of committing the intended act.
	Perceived Formal Risk (PFR)	Measured as the product of how likely and how much a problem are the formal sanctions—loss of job and face legal action—to the respondent.
	Perceived Informal Risk (PIR)	Measured as the product of how likely and how much a problem are the informal sanctions—loss of respect from family, relatives, friends, and colleagues—to the respondent.
	Perceived Risk of Shame (SHM)	Measured as the product of how likely and how much a problem are they are—losing self-esteem and feeling of guilt about the intended act—to the respondent.
Behavioral Intention	Intention to Commit Violation (INT)	Measured in terms of how likely and how willing the respondent would do what the fictitious characters did in the specific scenarios.

cal gender composition in these organizations.

To analyze the measurement quality as well as the path model for hypothesis testing, we used SmartPLS²³ as the primary statistical tool. Following the widely adopted two-step approach to structural equation modeling,^{3,15} we first assessed the quality of the measurement model to ensure the validity and reliability of the construct measurements. This was followed by structural modeling to test the research

hypotheses and the overall quality of the proposed model. The primary indicators for quality of the structural model are the R^2 values of the endogenous variables,¹⁵ which measure how much of the variances in the endogenous constructs are explained by the exogenous constructs specified in the model. The R^2 value for the dependent variable, INT, is 0.337, indicating that the variables in the model explained about 34% of the variance in the dependent variable, which, by the stan-

dard of structural equation modeling, is moderately high when compared to published studies (for example, D'Arcy⁸). Figure 2 presents the results of the structural analysis.

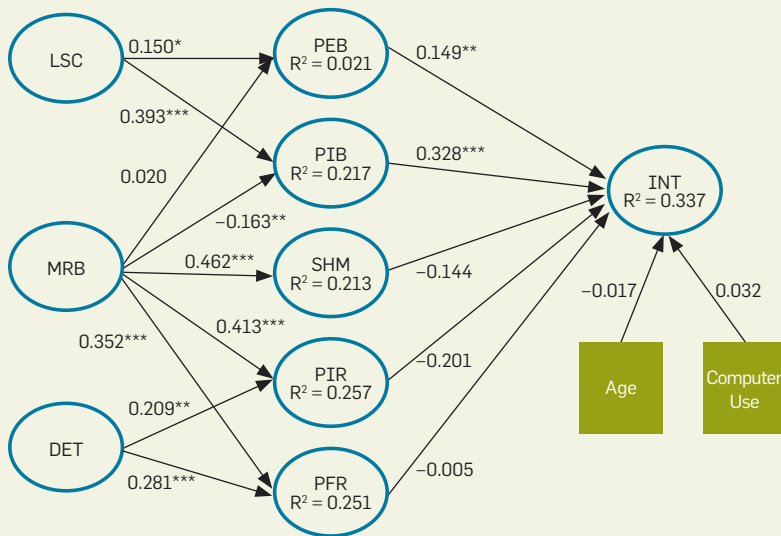
Does Deterrence Really Work?

The interesting results from this structural model are about the individual calculus that leads to the intention to commit policy violations toward computer systems in organizational settings. The hypotheses related to the impact of perceived benefits (PEB $\rightarrow$ INT, $\beta=0.149$, $p < 0.05$ and PIB $\rightarrow$ INT, $\beta=0.328$, $p < 0.01$) are supported, while the hypotheses related to the perceived risks (PIR $\rightarrow$ INT, $\beta=-0.201$, $p > 0.1$ and PFR $\rightarrow$ INT, $\beta=-0.005$, $p > 0.1$) are not supported. This result suggests that when an individual is contemplating whether to commit policy violations, the perceived benefits dominate the perceived risks in the rational decision-making process. The significant path from PIB to INT suggests the individual's intrinsic satisfactions that would be gained from the misconduct, such as thrill and happiness, are even more influential than the extrinsic material gains, such as the possession of money and goods, on the behavioral choice of the individual.

Perhaps equally interesting is the insignificance of the hypotheses related to deterrence, which is modeled as a second order formative construct of certainty, severity, and celerity of sanctions against the policy violations. Our results show the deterrence has no significant impact on individual intention to commit policy violations. While the deterrence does impact the perceived informal risk ($\beta=0.209$, $p < 0.05$) and perceived formal risk ($\beta=0.281$, $p < 0.01$), such impacts apparently do not have a significant influence on the individual's intention to commit the violations against the established information security policies.

These results seem to contradict, at least partially, the prevalent view about deterrence in the information security literature and practice. To be sure, we also tested a straightforward deterrence model in which only the three deterrence constructs (certainty, severity, and celerity) were directly linked to the INT construct. The result is shown in Figure 3. All path coeffi-

Figure 2. Results of structural analysis.



(***, **, and * indicate statistically significant at $p < 0.01$, 0.05 , and 0.1 levels)
 (LSC-low self control, MRB-moral beliefs, DET-deterrence, PEB-perceived extrinsic benefits, PIB-perceived intrinsic benefits, SHM-perceived risk of shame, PIR-perceived informal risks, PFR-perceived formal risks, INT-intention to commit computer misconduct)

Figure 3. Testing of deterrence effect.



(NS indicates not statistically significant)
 (CRT – certainty of sanction, SVR – severity of sanction, CEL – celerity of sanction, INT – intention to commit policy violations)

cients were found to be insignificant and the total variance explained is less than 6%. This result clearly suggests deterrence alone will not be effective, at least in the sample population of this study, in terms of reducing employee information security policy violations.

These results are consistent with the findings of Tunnell²⁹ in a criminology study that criminal offenders primarily think about positive consequences and little about negative consequences—approximately 60% of the criminals interviewed said they simply did not think about the possible legal consequences of their criminal actions before committing the crimes. Interestingly, studies on user behavior at online social networking sites^{1,9} found similar results that the benefits of social networking outweigh the risks of privacy concerns and users seem to ascribe the risks more to others than to themselves. A recent study by Siponen and Vance²⁵ showed deterrence mechanisms are not effective in the presence of neutralization mechanism in reducing security policy violations. This raises some critical questions: Why do benefits dominate risks in the rational calculus of individual behavior? How can the risks be accentuated? And why are the risks often ignored? We don't have answers to these questions and they will be the focus of our future research on information security.

While our model explains why deterrence may not be effective, it also shows what does work in reducing the policy violation intentions of employees: self-control and moral beliefs. Although moral beliefs have a strong impact on the perceived intrinsic benefits ($\beta=-0.163$, $p<0.05$), shame ($\beta=0.462$, $p<0.01$), perceived informal risk ($\beta=0.413$, $p<0.01$), and perceived formal risk ($\beta=0.352$, $p<0.01$), their impact on the policy violation intention is dampened somewhat because only the perceived benefits have significant impacts on the policy violation intentions. Nonetheless, the strong negative causal chain from moral beliefs to perceived informal risk to violation intention makes moral beliefs an important factor in the rational calculus of employees for committing policy violations.



Individuals with low self-control are more likely to be tempted by the appeal of the violations in terms of perceived benefits, and thus more likely to commit the acts.



The role of low self-control in shaping employee policy violation intentions is explained by our research model and confirmed by the data. Individuals with low self-control are more likely to be tempted by the appeal of the violations in terms of perceived benefits, and thus more likely to commit the acts. This is because low self-control leads to higher levels of perceived extrinsic benefits ($\beta=0.150$, $p<0.1$) and perceived intrinsic benefits ($\beta=0.393$, $p<0.01$), which in turn strongly influence the intention to commit the policy violations ($\beta=0.149$, $p<0.05$ and $\beta=0.328$, $p<0.01$).

What Can Employers Do?

While the results suggest deterrence may not work as effectively as many scholars and practitioners have believed, our findings suggest at least two directions for effective management of employee information security behavior: lowering the perceived benefits of committing the violations, and screening applicants with a high level of self-control and strong moral beliefs for sensitive positions. To lower the perceived benefits of violations against corporate information security policy, companies can take a number of proactive actions to reduce the perceived value of the data assets in corporate information systems. Data assets with high perceived value, such as customer records, credit card numbers, and confidential product designs, attract attention from internal and external entities and thus increase the chances of being misappropriated. Making data assets less attractive to potential offenders is difficult but not impossible. For example, a company could choose not to store customer credit card numbers on its servers (in most cases such storage is not necessary for completing financial transactions), protect confidential data with strong encryption systems, and distribute sensitive data to multiple secure servers (physical or virtual) with different layers of protection and access mechanisms, such as using different operating systems and security software and hardware from different vendors.

For positions where access to highly valued data assets must be authorized, companies should consider screening

applicants using psychometric instruments to ensure that only those who are strong on self-control and have compatible moral beliefs are assigned to these positions in addition to their technical qualifications and job responsibilities. Employees with low self-control could well be productive, innovative, and valuable in appropriate positions, but should not be assigned as custodians of sensitive data assets. On the other hand, the moral beliefs of employees can be strengthened by adopting high standards of organizational ethics and corporate citizenship. This is because organizations with strong ethics attract and retain employees with high moral standards.⁶ By recruiting and retaining employees with high moral standards, companies may reduce the chances for information security policy violations committed by employees.

Conclusion

In this article, we described a comprehensive model of information security policy violation by employees in organizational settings based on multiple criminological theories. We presented the results of empirical tests of this model and the implications of the findings for information security management theory and practice. While we cannot claim we have unlocked the mystery of why deterrence is not effective in reducing policy violations, we shed some light on this complex and significant issue. We found that while the rational choice framework of deviant behavior is largely supported, the perceived benefits of violations often dominate the perceived risks in individual decision calculus. As a result, the deterrence antecedents are less effective than the self-control and moral belief antecedents in shaping individual behavior.

With this understanding of employee information security behavior, our results reveal what might help companies effectively reduce the policy violations by employees: lowering the perceived value of the data assets in corporate information systems, and screening for individuals with high self-control and high moral standards. We find that individuals with low self-control, those who are more concerned about themselves

than others, more interested in what happens now than in the future, and more risk taking than risk averse, are more likely to overestimate the benefits of the misconduct. On the other hand, employees who have strong moral beliefs about right and wrong are less inclined to commit the violations even when the opportunities are presented. We believe this type of prescription for information security management has not been presented before and could have a significant impact on information security management practices.

We must caution, however, that this study has several caveats that may limit its generalizability. First, our findings are based on the data sample from a population with a unique Eastern culture where the concept of following rules and policies may be quite different from Western cultures. Culture has been found to significantly influence individual behavior in numerous studies.¹⁴ Second, the neutral wording used in scenarios of the survey could introduce unintended bias against deterrence in some respondents. Finally, the small number of organizations where the surveys were administered could bias the data as well. An interesting follow-up would be to replicate this study in multiple countries with various cultures using random samples that cover a large number organizations and industries. **C**

References

- Acquisti, A. and Gross, R. Imagined communities: Awareness, information sharing, and privacy on the Facebook. In *Proceedings of the 6th Workshop on Privacy Enhancing Technologies* (Cambridge, U.K., June 28–30, 2006).
- Anderson, R., Böhme, R., Clayton, R., and Moore, T. Security economics and European policy. In *Proceedings of the Workshop on Economics of Information Security* (New Haven, CT, 2008).
- Anderson, J.C., and Gerbing, S.W. Structural equation modeling in practice: A review and recommended two-step approach. *Psychological Bulletin* 103, 3 (1988), 411–423.
- Bachman, R., Paternoster, R., and Ward, S. The rationality of sexual offending: Testing a deterrence/rational choice conception of sexual assault. *Law & Society Review* 26, 2 (1992), 343–372.
- Becker, G. Crime and punishment: An economic approach. *Journal of Political Economy* 76, (1968), 169–217.
- Cable, D.M. and Judge, T.A. Person–organization fit, job choice decisions, and organizational entry. *Organizational Behavior and Human Decision Processes* 67, 3 (1996), 294–311.
- Cornish, D.B. and Clarke, R.V. *The Reasoning Criminal: Rational Choice Perspectives on Offending*. Springer-Verlag, New York, NY, 1986.
- D'Arcy, J., Havav, A., and Galletta, D. User awareness of security countermeasures and its impact on information systems misuse: A deterrence approach. *Information Systems Research* 20, 1 (2009), 79–98.
- Debatin, B., Lovejoy, J.P., Horn, A.K., and Hughes, B.N. Facebook and online privacy: Attitudes, behaviors, and unintended consequences. *Journal of Computer-Mediated Communication* 15, 1 (2009), 83–108.
- Dinev, T. and Hu, Q. The centrality of awareness in the formation of user behavioral intentions towards preventive technologies in the context of voluntary use. *Journal of the Association for Information Systems* 8, 7 (2007), 386–408.
- Ernst & Young. *Global Information Security Survey* (2008); <http://www.ey.com>.
- Gettfredson, M. and Hirschi, T. *A General Theory of Crime*. Stanford University Press, Stanford, CA, 1990.
- Gibbs, J. P. Crime, Punishment, and Deterrence. Elsevier, New York, NY, 1975.
- Hofstede, G. *Cultures and Organizations: Software of the Mind*. McGraw-Hill, New York, NY, 1991.
- Hulland, J. Use of partial least squares (PLS) in strategic management research: A review of four recent studies. *Strategic Management Journal* 20 (1999), 195–204.
- Lewis, M. *Shame: The Exposed Self*. Macmillan, New York, NY, 1992.
- Mercuri, R.T. Analyzing security costs. *Commun. ACM* 46, 6 (June 2003), 15–18.
- Nagin, D.S. and Paternoster, R. Enduring individual differences and rational choice theories of crime. *Law & Society Review* 27, 3 (1993), 467–496.
- Paternoster, R. and Simpson, S. Sanction threats and appeals to morality: Testing a rational choice model of corporate crime. *Law & Society Review* 30, 3 (1996) 549–583.
- Paternoster, R., Saltzman, L.E., Waldo, G.P., and Chiricos, T.G. Perceived risk and social control: Do sanctions really deter? *Law & Society Review* 17, 3 (1983), 457–480.
- Piquero, A. and Tibbetts, S. Specifying the direct and indirect effects of low self-control and situational factors in offenders' decision making: Toward a more complete model of rational offending. *Justice Quarterly* 13, 3 (1996), 481–510.
- Richardson, R. CSI Computer Crime & Security Survey (2008); <http://www.cse.msstate.edu/~cse6243/readings/CSISurvey2008.pdf>
- Ringle, C.M., Wende, S., and Will, A. SmartPLS, 2.0 (beta), University of Hamburg, Hamburg, Germany, 2005; <http://www.smartpls.de/>
- Simon, H. Bounded rationality in social science: Today and tomorrow. *Mind & Society* 1, 1 (2000), 25–39.
- Siponen, M. and Vance, A. Neutralization: New insights into the problem of employee information systems security policy violations. *MIS Quarterly* 34, 2 (2010).
- Straub, D.W. and Welke, R.J. Coping with systems risk: Security planning models for management decision making. *MIS Quarterly* 22, 4 (1998), 441–469.
- Tittle, C. R. *Sanctions and Social Deviance: The Question of Deterrence*. Praeger, New York, NY, 1980.
- Tibbetts, S.G. and Gibson, C.L. Individual propensities and rational decision-making: Recent findings and promising approaches. In *Rational Choice and Criminal Behavior: Recent Research and Future Challenges*. A. R. Piquero and S. G. Tibbetts, eds. Routledge, New York, NY, 3–24.
- Tunnell, K. Choosing crime: Close your eyes and take your choices. *Justice Quarterly* 7, 4 (1990), 673–690.
- Warkentin, M. and Willison, R. Behavioral and policy issues in information systems security: The insider threat. *European Journal of Information Systems* 18 (2009), 101–105.

Qing Hu (qinghu@iastate.edu) is a professor and chair of the Department of Supply Chain and Information Systems at Iowa State University's College of Business, Ames, IA.

Zhengchuan Xu (zcxu@fudan.edu.cn) is an associate professor in the Department of Information Management and Information Systems at Fudan University's School of Management, Shanghai, China.

Tamara Dinev (tdinev@fau.edu) is an associate professor in the Department of Information Technology and Operations Management at Florida Atlantic University's College of Business, Boca Raton, FL.

Hong Ling (hling@fudan.edu.cn) is a professor and chair of the Department of Information Management and Information Systems at Fudan University's School of Management, Shanghai, China.

ACM TechNews Goes Mobile

iPhone & iPad Apps Now Available in the iTunes Store

ACM TechNews—ACM's popular thrice-weekly news briefing service—is now available as an easy to use mobile apps downloadable from the Apple iTunes Store.

These new apps allow nearly 100,000 ACM members to keep current with news, trends, and timely information impacting the global IT and Computing communities each day.



TechNews mobile app users will enjoy:

- **Latest News:** Concise summaries of the most relevant news impacting the computing world
- **Original Sources:** Links to the full-length articles published in over 3,000 news sources
- **Archive access:** Access to the complete archive of TechNews issues dating back to the first issue published in December 1999
- **Article Sharing:** The ability to share news with friends and colleagues via email, text messaging, and popular social networking sites
- **Touch Screen Navigation:** Find news articles quickly and easily with a streamlined, fingertip scroll bar
- **Search:** Simple search the entire TechNews archive by keyword, author, or title
- **Save:** One-click saving of latest news or archived summaries in a personal binder for easy access
- **Automatic Updates:** By entering and saving your ACM Web Account login information, the apps will automatically update with the latest issues of TechNews published every Monday, Wednesday, and Friday

The Apps are freely available to download from the Apple iTunes Store, but users must be registered individual members of ACM with valid Web Accounts to receive regularly updated content.

<http://www.apple.com/iphone/apps-for-iphone/>

<http://www.apple.com/ipad/apps-for-ipad/>

ACM TechNews



DOI:10.1145/1953122.1953143

Before building the network or its components, first understand the home and the behavior of its human inhabitants.

**BY W. KEITH EDWARDS, REBECCA E. GRINTER,
RATUL MAHAJAN, AND DAVID WETHERALL**

Advancing the State of Home Networking

THE CREATORS OF the original Internet architecture imagined a network of networks linked by protocols connecting the world's diverse and disparate networks. That vision has been profoundly successful, with the Internet reaching beyond the government and corporate contexts in which it was born into settings ranging from public hotspots to places of worship, to rural sub-Saharan Africa, even to space.

In the Western industrialized world, networking is rapidly being adopted in the home; for example, in the U.S. in 2009, approximately 63% of homes had a broadband connection, and over 50% had a “home network,” defined as multiple computers sharing a broadband connection via either a wired or wireless network within the home.¹⁷ These networks are part of the global Internet, participating in it just like corporate, government, and other networks and representing an increasingly large percentage of nodes on the global Internet.

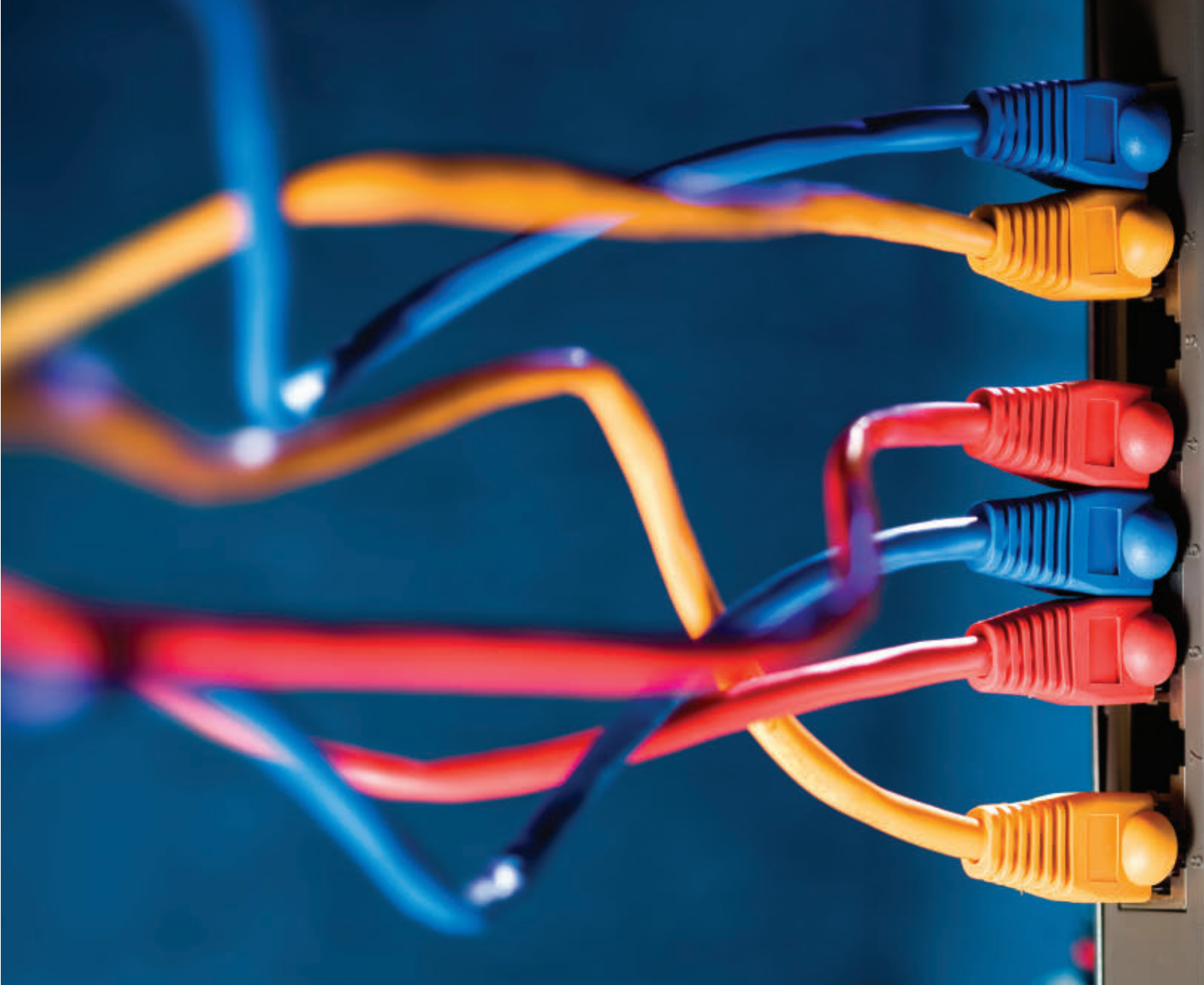
By such measures, home networking is a success. The potential benefit of this deployment is huge because it opens the home to new commercial services in entertainment, education, health care, and communication. These applications have positive implications for sustainability (such as through telework), public access to governmental services, and better care of elders in their own homes. All these applications depend on secure, manageable, cost-effective deployment in the home.

However, home networks are fraught with problems, particularly in terms of the difficulties end users face managing and securing their networks.^{8,9} Many of these problems arise because Internet-style networks, which were developed for managed environments, have been transplanted relatively unchanged to the home. In the best cases, the problems result in mere hassle and frustration for householders; in the worst cases, they pose a threat to individual privacy and lost opportunity to adopt next-generation applications. For example, while advanced home-automation technologies (such as those by Crestron and Control4) have been available for more than 20 years, adoption is limited, with fewer than 159,000 units shipping worldwide in 2009.¹

As a starting point for fixing problems with home networks, we articulate

» key insights

- **Usability of home-networking technologies is a key impediment to adoption of new applications in the home.**
- **Network usability problems run deep because the technology was originally developed for research labs and enterprise networks and does not account for the unique characteristics of the home: lack of professional administrators, deep heterogeneity, and expectations of privacy.**
- **Addressing the challenges of home networking is not simply a matter of designing better user interfaces but a concerted, interdisciplinary effort by networking, HCI, and systems researchers.**



the diverse underlying factors responsible for these problems, rooted in technical aspects of the Internet architecture and protocols, in human-oriented aspects of householders and the home setting itself, and in the economic and commercial factors hindering effective solutions to the problem, even though the market is lucrative.

This diversity of factors makes clear that the problems in home networking today are not due to core technical issues nor exclusively to issues that can be solved through better user education or user interfaces (UIs). Dramatic improvement is unlikely to be brought about by addressing the two sets of issues independently, whether by painting a UI veneer on top of existing technologies or by designing new protocols and architectures without accommodating human practices,

needs, and routines. There is a deep connection between the user experience and the network's underlying capabilities, leading us to suggest the two sides of the problem must be tackled together through cross-disciplinary research approaches.

Additionally, such approaches must be developed bearing in mind the uniqueness of the home environment—lack of trained administrators, extreme heterogeneity across homes, and strong privacy considerations. A large body of work addresses how to simplify network management and diagnosis in enterprise networks and Internet service providers (ISPs), but the considerations at play in the home mean it cannot be applied directly or easily adapted there.

We conclude by sketching example approaches combining technical capa-

bilities and sensitivity to the user experience while being compatible with the home environment. We do not claim these approaches are the only fruitful directions but hope they will stimulate the research community to try fresh cross-disciplinary approaches.

Strains on the Home Network

To understand the home network, we should appreciate where it comes from:

The Internet went home. Internet use in the home has followed a path similar to many other advanced home technologies. Tech-savvy “early adopters” were the first to bring networks into their homes, often to support telework.²⁶ They had both the technical sophistication and the hobbyist's motivation to persevere. In the 1980s and earlier, installing a home network

meant being committed to installing and managing commercial-grade routers and switches, running structured wiring throughout the home, and explaining to the local telephone carrier that one needed, say, a T-1 or ISDN connection.

Internet adoption in the home went mainstream following the advent of the Web, with the standard Internet Protocol suite deployed in consumer operating systems like Windows 95. The growing number of Web sites attracted consumers who had previously participated only in closed, paid online services like AOL and CompuServe. This period brought about the consumer ISP, as well as a commercial market for consumer-grade networking hardware. The role of home networks during this period was generally limited to sharing the Internet connection with multiple computers (such as one in the home office and another for children) and potentially providing access to a shared printer.

The next big shift in home networks occurred around 2001, as inexpensive wireless connectivity in the form of IEEE 802.11b made it possible to provide high-speed network connectivity throughout a home without structured wiring. Home networks today have shifted beyond simple Internet and printer sharing to include new devices and applications (such as media streaming, game consoles, and

WiFi-enabled phones), as well as new connectivity options (such as faster versions of WiFi and Ethernet over powerlines), helping spread the network throughout the home.

Figure 1 is from a Web site dedicated to helping hobbyist users show off their network topologies, depicting an advanced home network. Note it includes a mix of wireless and wired connectivity, along with a range of devices, including PCs, printers, smartphones, cameras, and game consoles.

The broad strokes of Internet adoption in the home are well known; we lay them out to highlight that when network adoption in the home shifted from a trickle to a flood, the network brought home was the same Internet adopted in the corporate world and designed for environments radically different from the home.

The home is different. The benefits of adopting the “one true Internet” at home are profound, as well as obvious, benefitting consumers through standardization on widely adopted, open protocols; applications and devices built for the Internet protocol suite can be run at home as easily as they can in the enterprise; and economies of scale allow ever more inexpensive networked devices to find their way into homes, as in the figure.

At the same time, adoption of the Internet in the home brings problems due to the misfit of the technology with

the home’s dynamics and context. The Internet architecture was developed for a world of technical sophistication (where experts handle network connectivity and management) and shared trust (where network operators share responsibility for protecting the network’s integrity). These properties do not carry over to home networks.^{4,23}

Home networks differ from other Internet-style networks (such as enterprise networks, data centers, and ISPs) in three significant ways:

No professional administrator. Home networks do not have professional, trained “administrators” in the same sense as other networks,^a making difficult their management, security, and diagnosis;

Deep heterogeneity. The home network is a place of deep heterogeneity, even experimentation, as new applications and devices are installed, and householders push the technological boundaries of their networks. This heterogeneity means applications and services in the home may be deployed onto new and potentially unforeseen home-network infrastructure and configurations, and conversely that the network infrastructure in the home must be able to support new types of applications and services introduced by users. This complexity means isolating any resulting problems may be difficult; and

Expectation of privacy. There is an expectation of privacy in the home that is different from what is expected in the workplace. Whereas users in most companies have no expectation of privacy on their networks (due to policy or the needs of the IT department), home users have a strong expectation of privacy, extending even to restricting certain access to those who help manage them (such as ISPs and professional troubleshooters). Also, unlike corporate networks, homes are unlikely to have centralized, managed-access controls for their devices and information.

a We deliberately do not say home users “are unsophisticated.” Rather, home users are deeply sophisticated about their individual domestic practices and the routines of their own homes, along with how home networking should be meshed with these practices and routines, about which the purveyors of home networking solutions are conversely generally unsophisticated.

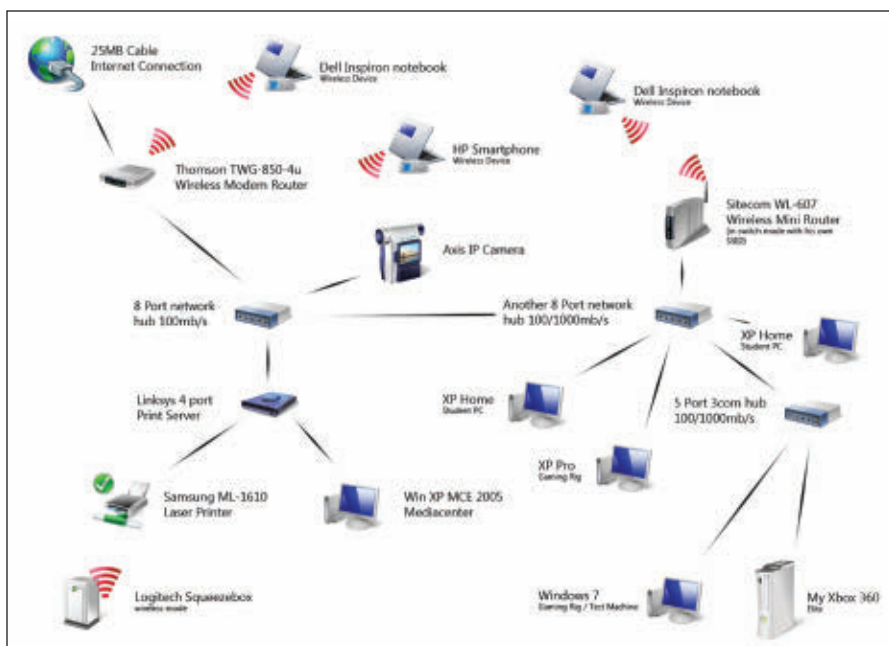


Figure 1. Schematic of real home network.

Rising tide of problems. These factors diminish the ease of using the home network for householders. Heterogeneity may increase the number of network faults and difficulty of isolating problems. Users often have neither the technical training nor the motivation to deal with them, raising frustration and support costs for service providers and vendors. Beyond these factors, the need for privacy complicates solutions that may be workable in the enterprise (such as remote administration by paid professionals).

It is an exaggeration to say that home networks are broken, since they have proliferated, and many people are able to watch Internet movies at home and back up their photos. However, there is also a significant tax on user goodwill due to the time and effort spent (wasted) dealing with home-networking issues; for example, in 2008, approximately half of all home technology users in the U.S. needed help from others to set up or install new devices.¹⁸

The state of home networking clearly results in user frustration; it also translates into significant monetary costs for companies. For example, in 2002 when home networking was relatively new, Parks and Associates²² found that home network devices were the single most returned category of items at “big box” consumer electronics stores in the U.S. As recently as 2010, over 25% of all wireless access points were returned, despite most presumably functioning as intended.¹⁹ Data indicates only 5% of consumer-electronics product returns worldwide are due to actual technical failure.²

Furthermore, the current state of home networks functions as a barrier to adoption of new technology (such as home-automation systems). Indeed, consumers in the U.S. regularly cite technical complexity as the primary disincentive to adding new networked devices in the home.¹³

The silver lining, if there is one, is that home networks also represent an opportunity for research with benefits beyond merely improving the user experience, increasing security, or decreasing support costs. If computer scientists can advance the state of home networking, then they may enable new applications to move out of the lab into

practice. Many of the applications envisioned by the research community—from entertainment and social media to telework and remote collaboration to health and wellness—require functioning, manageable, correctly configured, secure home networks. Removing the barriers to such networks has the potential to unleash a wave of innovation in the networked home.

Fault Lines

Given the problems around home networks, the natural question for computer scientists to ask is whether they are easily fixed; for example, does home networking technology simply suffer from a lack of UI polish? Indeed, this is not the case, and the underlying causes are deeper and structural. Some causes stem from technical aspects of the Internet architecture and protocols, others from human-oriented aspects of householders and the home, and still others from economic and commercial aspects. Together, the picture they paint is problematic and unlikely to improve greatly without sustained involvement from multiple computing disciplines.

Technical. The technical causes are, by and large, an inherent aspect of the design of the Internet architecture:

Necessity of configuration. A core precept of the end-to-end argument espoused as a key architectural principle of the Internet is that the network itself is an application-neutral carrier of bits; on the other hand, end nodes must be able to speak a wide (and potentially open-ended) array of application-layer protocols. End-user devices are often programmable and general-purpose and must be configured correctly to work on the network. In the home, end users generally perform this configuration, possibly involving settings at the link layer (such as network name and encryption keys), network layer (such as default gateway, Domain Name Service, or DNS, settings, and IP addresses), and application layer (such as Web proxies and print servers). While some of these settings may be handled by network services (such as Dynamic Host Configuration Protocol, or DHCP), many more require direct user intervention to be set correctly.

Note that the need for configuration is not an inherent characteristic

of communication networks. For example, in the public switched telephone network (PSTN), a different set of design decisions manifest in a radically different user experience, where infrastructure devices and end-user device settings are entirely removed from the purview of end users. A landline phone plugged into an RJ-11 connector “knows” its phone number and is immediately integrated into the global network without the user having to be configured in any way. On the flip side, such a device does not support the radical extensibility to new applications or even the entirely new protocols many Internet devices have today.

But where configuration is a necessity there is also the possibility for misconfiguration, possibly exposing the network to privacy and security risks or even to the catastrophic severing of a device’s connectivity. Exposing settings to users and generally requiring that they interact with them presents a distinctly unfriendly user experience in which users must grapple with technical jargon and low-level settings far removed from what they want to do. This configuration task also tends to be highly complex due to several factors:

Interaction of applications and network. Application developers tend to think in terms of an ideal world where devices or applications abstractly “sit on top of the network,” viewing it as a more-or-less opaque infrastructure that carries bits. Unfortunately, this abstraction does not match reality, and the mismatch is sometimes exposed in application failures difficult for users to understand and fix.

Consider that homes today provide great topological flexibility, possibly including multiple wireless access points, powerline Ethernet bridges based on the HomePlug standard, structured wiring, and Ethernet-over-coaxial solutions (such as the Multimedia Over Coax, or MOCA, standard), each with its own adapters and configuration utilities. This infrastructure does not always work with applications unless configured in highly specific, technical ways; for example, adding a new access point may introduce a second DHCP server and subnet, resulting in some clients getting IP addresses incompatible with the rest of the home

network. Multicast discovery protocols may then be unable to cross subnet boundaries. The result is that some applications may stop working properly when new and seemingly unrelated devices are added.

Applications may also require the infrastructure be configured in specific ways; for example, playing an Internet-connected game may require tweaking firewall or Network Address Translation (NAT) settings so the game communicates with outside servers. While relatively new protocols (such as the NAT Port Mapping Protocol, or NAT-PMP) alleviate some of this manual tweaking by enabling programmatic configuration of NAT port forwarding, many manual tasks persist; for example, sharing a folder of pictures with a distant relative may require users set up Dynamic DNS (so the relatives do not need to know the potentially ever-changing IP address of the home router), understand static-versus-dynamic addressing on the home network, or change network and host security settings. Such tasks require users to step outside the application to modify the network; these tasks are unlikely to be easily automated in the near future because they crosscut multiple layers of the networking stack, as well as multiple applications and services.

Today's home networks do not provide good abstractions to applications. End users must manually configure the network to implement their goals. To do so, they are essentially required to be network managers for a small subnet within the Internet, managing their devices, network infrastructure, and applications.

Conflation of policy with mechanism. Further complicating the task of managing home networks is the fact that the controls available today are generally low-level and divorced from any high-level policies users may wish to establish for their networks. These controls (such as to tweak routing and translating and forwarding packets on the home network) and parameters (such as the size of the maximum transmission unit, or MTU, and wireless security primitives) are difficult to translate into actionable, high-level policies at the user level.

This tension is perhaps most apparent in network security, in which

security policies are deeply intertwined with low-level network mechanisms and topology. The situation today is characterized by devices on the “inside” of the network having an implicit trust relationship with one another, along with often weak host defenses. This means network misconfiguration can potentially expose poorly defended hosts to a range of attacks. Further, the conflation of security with relatively coarse-grain configuration controls makes it difficult or impossible to express certain security goals using the set of available “knobs.”

As an example, consider how one might securely allow a visitor to access one's home network. This secure access is handled in most home networks in one of two ways: turn off wireless security completely (exposing all hosts on the network to public access) or give the wireless key to the visitor. Even giving the key to a visitor—undeniably better than turning off security entirely—may not map well to the user's intended policies. In most home networks, having access to the network itself includes not only the shared Internet connection but all the hosts on the network. While this may be acceptable for some family members and close friends it may be completely unacceptable for visiting technicians and friends of teenagers' children. Some newer wireless routers offer a separate guest wireless network, allowing guests to access the Internet but not the other devices connected to the home network. While this feature represents a step in the right direction, it is too coarse-grain; for example, it does not support users wanting to let their guests use the printers in the home but not access media files.

Human-centric. Not all problematic aspects of home networking are directly or solely technical in nature; many interact with human needs, behavior, and expectations. Here, we outline some of the causes of home-network problems rooted in these human concerns.

Paucity of conceptual models. Home networks do not provide solid “hooks” for users to form reliable, predictable conceptual models of how the network functions. The importance of such models has been long known (Norman¹⁵ is a classic example), allowing

users to observe the behavior of a potentially complex system and formulate a plausible model for how it works based on their observation, using it to control the system. Such models need not correspond completely with “reality” to be useful. Kempton's work on “folk theories” of home-heating controls¹² illustrates how even incorrect models allow people to work with complex systems.

However, current networks do not expose the kind of information necessary for users to form such models; for instance, much of the behavior and state of the network is unobservable, as are many effects of changes on the network. For most users, the only readily observable states are whether the network is working and whether it is dysfunctional. Wireless technologies exacerbate the problem, as there is not even a cable to show how data might flow. The effects of a lack of conceptual models show up almost immediately in user-oriented studies of networking,^{8,9} whereby ordinary consumers are often baffled by network behavior, unable to devise a troubleshooting strategy that is more sophisticated than hopeful rebooting and often unaware of even what capabilities are provided by their own network. Figure 2 (from Grinter et al.⁹) is in stark contrast to Figure 1, showing an example of a household's sketch of a home network and illustrating the lack of an actionable conceptual model; note the absence of any aspect of network topology, link type, or router.

Broken expectations. Desire for communication, entertainment, and other applications is the key reason households adopt home networking in the first place. Indeed, desire for relatively advanced functionality (such as in-home media sharing among home devices and media sharing across homes) factor prominently in home-network adoption.⁶ However, achieving such functionality in a way that fits the ecosystem of the home is beyond the grasp of many, betraying the expectations and desires that lead users to adopt home networking.

Part of the problem is that households' nontechnical requirements are often not addressed by home-network technologies. Requirements like device aesthetics, flexibility of device

placement, and a device's ability to be used by multiple members of the household are often highly relevant in the domestic setting but rarely determined solely by a device's technical properties. Even though there may be nothing technically wrong with a device, it may still not fit well in the domestic ecology of the home due to social, cultural, aesthetic, or other constraints.

A second aspect of the problem is the expertise required to install and use advanced home-network devices. Device- and network-management chores are often divided among householders, meaning not only do new devices bring new domestic labor into the home but the expertise needed to fix problems may be spread among multiple members of the household or even among family and friends outside the household. Thus, the technical burden of keeping the network running becomes a social burden, whereby householders must leverage family and friends for support.

Personalized home network. Any effort to make networks more usable is immediately complicated by the fact that the home network, as with any domestic technology, is deeply intertwined in the everyday social routines and realities of home life. Much prior research has demonstrated the degree to which home networks reflect household domestic routines; see, for example, Tolmie et al.²⁵ The demands of a particular household may govern how and when the network is used, the placement of machines (to allow parents to keep an eye on children's Internet use or for aesthetic reasons), what applications are present, and what media or content are considered "shared" by the family versus owned by a particular individual. These human considerations, in turn, drive technical heterogeneity in the network; for example, a desired device placement could motivate installation of such new infrastructure devices as wireless and HomePlug; such desired applications as media sharing could lead to introduction of new devices onto the home network; and the patterns of sharing and privacy in a given home may dictate aspects of network configuration, device configuration, and content placement. Moreover, there may be as

many variations along each of these dimensions as there are homes.

The deep personalization in today's networks is unlikely to go away over time; studies of other domestic technologies illustrate the degree to which the domestic order varies across households.¹⁶ Such personalization also complicates naïve approaches to "solving" the problems of home networking; many of us have called tech support only to find the "expert" at the other end of the line has no knowledge of the particulars of the home network and is simply following a scripted troubleshooting routine. One core issue with such approaches is that the remote troubleshooter is unaware of any local customizations or idiosyncratic uses of the home network; such a one-size-fits-all approach is likely to remain problematic for the home, given its deep personalization.²¹

Privacy and the home network. Home networks are not only deeply personalized but also deeply personal. Users have information on the network they do not want to share with others outside the home and in some cases even with those within the same home. The expectation of privacy within the home runs deep, and is different from

any such expectations that may exist in most corporate networks. These differences can also complicate the task of seeking help when the network does not function as desired. Many home users might, for instance, be uncomfortable giving unfettered access to their entire network and the information on it to an outside technician, even if the technician is there to help repair the network.

Economic. One might argue that, given the size of the market for networked devices and services in the home, market pressure would drive commercial interest to fix the problems of home networking. Commercial players are certainly tackling problems; for example, Cisco's Valet is a recent attempt to simplify the setup for wireless routers,¹⁹ and Apple's Bonjour enables easier discovery of devices on home networks.²⁴ Such solutions, while beneficial, are still too limited to truly advance the state of the art. Several economic factors tend to damp more sweeping but potentially more effective long-term change, either unilaterally on the part of companies or as part of a broader effort.

Lack of data. Solving a problem requires reliable data, but data on prob-

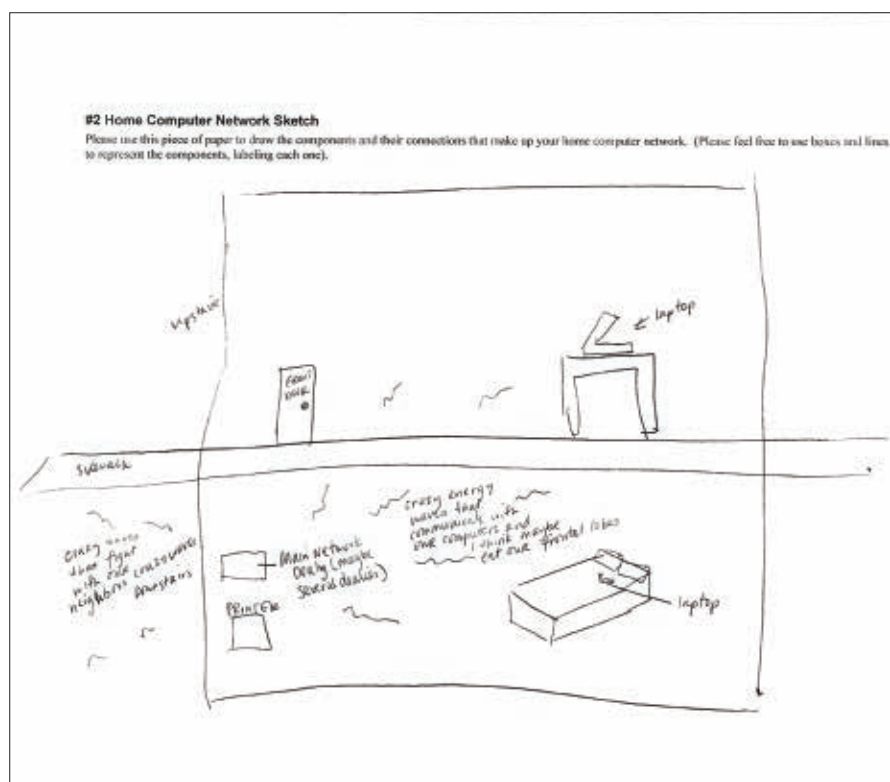


Figure 2. Sketch of a home network by a householder, who, though a regular user of the network (set up by someone else), was incapable of troubleshooting or extending it.

lems inside home networks is difficult to come by. Since each home is different, one needs data from numerous homes to address the problem broadly. Ethnographic studies (such as those cited earlier) provide rich accounts of how individuals understand and work with a network within the social setting of the home. Yet they cannot provide detailed technical data about what is actually happening on home networks, nor can they be widely scaled to understand how network-level behaviors may vary across households.

Automated problem reporting would be valuable, in much the same way the introduction of Microsoft Windows's error reporting led to significant improvement in bug fixing.⁷ However, databases of user reports of home-network problems are much less useful today, spread across multiple organizations, including ISPs, operating system vendors, and device vendors. This is because users tend to report problems to different organizations, depending on which they perceive (correctly or incorrectly) as responsible. Thus, each organization has a limited window into the problems while also being reluctant to share for competitive reasons, and no one party has a holistic view that makes it easier to develop effective manageability solutions for the home. Further, user reports of network problems may not correspond well to the actual state of the network; a call to a technical support line saying "The network doesn't work," followed by instructions to reboot the router and all attached devices, does not reveal the actual source of the error. While self-reports, call logs, and trouble tickets may describe a user's perception of a problem, they do not represent ground truth about the network itself.

Cost of heterogeneity. A complication that makes it costly to develop services for the home is the high degree of diversity across homes, with deep personalization leading to this heterogeneity. Homes today have different sets of devices, topologies, and user preferences that will likely always be the case. Diversity leads to many possible network configurations, as well as failure symptoms and root causes. For ease of use, network products and services must tackle this diversity, despite increasing the complexity of use, along

with the costs of development, testing, and support.

Evidence of the difficulty of handling diversity across home networks is the increase in vertical integration of hardware and software; for instance, security-products manufacturer Schlage (<http://www.schlage.com/>) offers home-automation devices (such as door locks and lamp modules). Not only do they not integrate well with other vendors' devices, to enable remote access to them, users must acquire an additional Schlage device (called LiNk). While this architecture makes the solution easier to develop and more reliable by constraining the problem, it ultimately increases costs for providers and users alike, and the lack of device composition limits functionality.

Standard interfaces for cross-device interactions may eventually mitigate the problem. However, progress has been slow at best due to the lack of clarity on the specific problems and the tussle in the marketplace as companies seek to differentiate their products. It is not in the business interest of a particular vendor to make a significant engineering investment that might lift the market as a whole.

Burden of support. Even if home networks become much easier to use, some faults will still occur. When something goes wrong that users cannot remedy easily, their recourse is to call the service provider for technical support or return the product. Diagnosing and repairing these faults is expensive for vendors in today's cost structure. Monthly service charges are small relative to the cost of qualified technical personnel; even a single service call can wipe out the profit due to a customer. Returns often represent the loss of a customer and all associated revenue rather than an actual equipment fault that can be repaired. Both factors make home networking less lucrative than it might otherwise be.

A more perverse aspect of support costs is that they are not always borne by the responsible party. From the householders' point of view, the network has "gone down" if they are unable to communicate through a networked application, easily leading to the wrong party receiving a demand for technical assistance. A classic example is when computer problems are

misconstrued as network problems, as when a computer virus severs network connectivity. Users mistaking the problem are hardly to blame; some studies have shown that users deal with from three to seven service providers, including ISPs, cellular data plans, email providers, and others, on average in the U.S. to "make the network work," in addition to multiple hardware and software vendors.⁹ Faults may lie with the computer, network infrastructure in the home, ISP, or remote servers, and there are no well-established attribution mechanisms. The result is to dump costs in ways that complicate advances in home networking.

Research Agenda

These fault lines help explain why home networks are difficult to use and secure and expensive to support but can be reinterpreted as a research agenda to advance home networks and the ways they are used. Pursuing it is not simply about reducing the costs of the status quo but about enabling significant innovation. After all, if users are unwilling to install a piece of equipment for fear of breaking the network, how will computer scientists enable novel applications in the home for health care, entertainment, sustainability, or other worthwhile needs?

Transforming the fault lines into a research agenda calls for action to address each factor we've identified. To succeed, the research requires two properties: The first is it must specifically target the unique characteristics of home networks: lack of trained administrators, high level of heterogeneity across home networks, and user expectations of privacy. The home is a drastically different setting from the enterprises and government labs that developed and first adopted the Internet.

This difference requires researchers to reexamine aspects of our accepted wisdom to truly support this growing portion of the global network, and means approaches for network management and diagnosis developed for other settings (such as enterprise networks¹¹) are unlikely to apply directly to the home network. For example, some tasks (such as strict reachability limits for machines that hold payroll data) are not critical in the home, while others (such as managing consumer-

grade devices that lack support for enforcing policies) are indeed unique to the home. It is likely that some of the underlying information-processing techniques of enterprise tools (such as those for failure correlation and localization) are usable in the home. But even in these cases, the focus of the techniques is often on scale. In the home, the focus must be on making results accessible to end users. The results of today's diagnostic systems are often difficult to interpret even for trained professionals.¹⁴

The starting point for effective management research is often data that provides insight into real problems. However, the uniqueness of the home setting means existing data sets—collected in other networking contexts—may not be applicable. Even though computer scientists may have access to great volumes of data about traffic on the backbone Internet or routing behavior internal to enterprises, they have shockingly little insight into what happens in home networks. Ethnographic studies have begun to reveal the human side of the equation, but they must be coupled with data about the network itself. This goal also represents a technological challenge, and one might imagine different approaches for collecting and correlating such data; for example, the combined data could be supported by configuration-reporting mechanisms that would let users send a summary of their network state, along with a problem report. However, developing such facilities is difficult because the configuration data must be extracted from a distributed, heterogeneous set of devices at a time when the network may not be working. Whatever method for collecting the data is considered ideal, there is value in acknowledging that the uniqueness of the home makes it a necessary (and worthy) site for further data collection.

Second, research that rethinks the management of the home network must involve co-design of the networking and human-computer interface (HCI) aspects of any solution. Many of the problems in today's home networking straddle both human and technical challenges, so are neither solvable by computer systems and networking researchers focusing in



Exposing settings to users and generally requiring that they interact with them presents a distinctly unfriendly user experience far removed from what they want to do.



isolation on new technology nor by HCI researchers focusing in isolation on user-experience approaches. These two sides of the problem must be tackled hand-in-hand because there is a deep connection between the desired user experience and the network's underlying capabilities. Rethinking the home network requires more than HCI researchers painting a veneer of UI on top of existing, unworkable technologies; likewise, it requires more than networking researchers designing new protocols and architectures in the absence of knowledge about human practices, needs, and routines in the home. For example, how data is collected might first appear to be a purely technical challenge but touches on user perception of privacy. Similarly, providing more realistic conceptual models of the network (perhaps through end-user-centric visualization tools) may drive new technical requirements for specific forms of instrumentation and monitoring in the home network; developing automated approaches to troubleshooting must be done with understanding and respect for the variance across households in terms of domestic routines.

Here, we offer four examples of potential projects that cross the disciplinary divide between networking and HCI, intending not so much to chart a complete trajectory for interdisciplinary research in home networking but to illustrate the kind of interaction that may result from such collaboration:

Privacy-respecting remote diagnosis. Remote management systems (such as Intel's Remote PC Assist Technology, or RPAT) are emerging,¹⁰ with related tools and services allowing a remote operator to directly access and reconfigure a computer. This remote technology can lead to significant improvement when users act as agents of change while on a support call. However, when used in the home, the technology raises privacy concerns due to the private data on home computers and the traditional situation of an administrator having unfettered access; technology from the corporate world may not be the best fit with the home context.

How might ISPs provide remote-diagnosis tools that support the home's unique privacy constraints? Clearly,

it is not necessary for every remote-support technician to be able to view any and all contents to diagnose and repair network connectivity problems nor retain configuration data that permits future access. Creating remote-diagnosis tools for the home involves a synthesis of technical- and human-oriented work involving understanding users' orientation toward privacy (what information can be revealed and when) and accountability, allowing users to see what remote users have accessed and changed, potentially even rolling back these changes if unhelpful. It also involves creating new mechanisms to allow data to be collected from disparate elements of the distributed system that is the home network, security techniques to ensure that protected user information is not revealed, and frameworks that allow reversal of changes to networks or hosts as needed.

Leveraging social networks. Most of us are familiar with at least one effective technique for coping with the complexities of home networks: go to family and friends rather than wait on the help line for expert guidance. Householders use it not just for cost or availability reasons but because family and friends are much more likely to understand the complex, personalized, situated nature of our home networks and home routines.²⁰ Home networks could be much more effective at supporting such collaborative troubleshooting; they provide no support for it today. Imagine tools that extract a summary of the home network configuration and search a householder's social network for friends with similar setups or even with shared problems. These are the friends most likely to be able to help. Some research systems, notably the NetPrints system,³ come close to exploring this approach by collecting collaborative databases of network problems to diagnose problems and reveal possible solutions. As with remote diagnosis, privacy is a chief challenge in exposing enough of the network internals to facilitate troubleshooting without leaking information users might consider sensitive.

Leveraging the cloud. Professionally managed cloud-based services may play a key role in home networks of the future. Indeed, even today the



Application developers tend to think in terms of an ideal world where devices or applications abstractly “sit on top of the network,” viewing it as a more-or-less opaque infrastructure that carries bits.



cloud provides an opportunity to circumvent tasks that would be exceedingly complex in home networks; for example, uploading a folder of photos to a service (such as Flickr) to share with family and friends is far easier than configuring the home network to share the folder directly. In effect, the cloud provides an opportunity to outsource certain services to a professional (often for-profit) provider, removing the management burden from householders. The movement of users' personal data—email, calendars, photos, music libraries, even financial data—into the cloud is increasing. However, as with other forms of outsourcing, privacy risks might be associated with this movement.

What about future research and commercial opportunities? The cloud may enable a shift in other sorts of management tasks; for example, lightweight, potentially simpler client devices may depend more on the cloud for services (such as management and data backup). Ubiquitous, low-cost, wide-area wireless may even make it possible for the home network as a separate, discrete entity to vanish, replaced by direct connections from each device to the cloud. In each case, however, the challenges are not just how to ensure adequate connectivity and robust security. They also touch on humans being able to understand the implications of the shift and manage their relationship (and their data's relationship) to the services being offered and that they pay for.

High-level attribution tools. Standard tools that can attribute a problem with the operation of a networked device in the home to a responsible party would lead to a more direct and efficient path to problem resolution. Simply being confident that a problem is with the wireless access point rather than the ISP is a major step forward. However, to be effective, such tools should produce results that map to the conceptual models of users, especially since so much problem solving requires shared agency between tools and the people using them, with both working together to identify and correct problems. This style of interaction is very different from what exists today. Testing tools (such as ping and traceroute) pro-

vide low-level results, even if they are covered with a UI veneer as part of a troubleshooting “wizard.” These results may not be helpful for attribution; for example, if a computer fails to connect to an access point, the user may not know whether the problem is with the computer or with the access point. New tools that identify not just symptoms but also potential causes of network problems must be integrated with interfaces that allow users to assess the network, test hypotheses, and supply information that may be unavailable to the tools themselves.

Balancing self-configuration and user control. Methods for engineering networks that configure themselves would be valuable for reducing the user’s burden, as well as the possibility of misconfigurations. This might be accomplished in a number of ways, each with its own trade-offs; for example, it might be possible to impose a fixed topology (such as requiring all devices in the home connect to a single centralized gateway) to limit the range of faults that might occur and impose a single point at which policy is enforced.⁵ Other approaches might leverage the cloud as an intermediary by, say, serving as an external rendezvous point or by outsourcing services (such as storage) to a place where they are more easily configured by technical professionals.

However, a fully self-configuring network, removing all customizability from user control, is unlikely to be workable. Given the level of personalization and desire for integrating the network into domestic practices, users will always need some form of control. The challenge is finding and striking the right balance between self-configuration and user control. What things can be removed from users’ purview, and over what things must users retain control? Computer scientists do not yet know enough about the right balance to begin to inform technical solutions to self-configuration.

Conclusion

While adoption of home networks has steadily increased since the late 1990s and early 2000s, this growth also reflects deep problems and limitations. The problems are a result of the colli-

sion between technical design choices and fundamental aspects of the human condition. The problems millions of users now face with their networks go beyond inconvenience and nuisance. Threatening the privacy of home users, they also pose significant barriers to the adoption of next-generation applications in the home in areas as diverse as health care, sustainability, and education. Addressing them is inherently cross-disciplinary and will involve researchers from networking, systems, and HCI, as well as industry.

Acknowledgments

This article grew out of the 2009 University of Washington/Microsoft Research Summer Institute (<http://www.cs.washington.edu/mssi/2009/index.html>) where researchers from academia and industry explored opportunities in home networking. We thank all participants, as well as our colleagues and collaborators at our respective institutions. The Georgia Tech authors are supported by National Science Foundation awards #CNS-0626281 and #IIS-0904431. 

References

1. ABI Research. *Home Automation and Security Technical Report*. New York, 2009; <http://www.abiresearch.com/>
2. Accenture. *Big Trouble with 'Trouble Found' Returns*. Technical Report. Dublin, Ireland 2008; http://www.accenture.com/SiteCollectionDocuments/PDF/22701_ReturnsRepairsRvn_v04lr.pdf
3. Aggarwal, B., Bhagwan, R., Das, T., Eswaran, S., Padmanabhan, V., and Voelker, G. NetPrints: Diagnosing home network misconfigurations using shared knowledge. In *Proceedings of the Sixth USENIX Symposium on Networked Systems Design and Implementation* (Boston, Apr. 22–24). USENIX Association, Boston, 2009, 349–364.
4. Blumenthal, M. and Clark, D.D. Rethinking the design of the Internet: The end-to-end argument versus the brave new world. *ACM Transactions on Internet Technology* 1, 1 (Aug. 2001), 70–109.
5. Calvert, K.C., Edwards, W.K., and Grinter, R.E. Moving towards the middle: The case against the end-to-end argument in home networking. In *Proceedings of ACM Workshop on Hot Topics in Networks* (Atlanta, Nov. 14–15). ACM Press, New York, 2007.
6. Edwards, W.K. and Grinter, R.E. At home with ubiquitous computing: Seven challenges. In *Proceedings of the Third International Conference on Ubiquitous Computing* (Atlanta, Sept. 30–Oct. 2). Springer-Verlag, Heidelberg, 2001, 256–272.
7. Glerum, K., Kinshumann, K., Greenberg, S., Aul, G., Orgovan, V., Nichols, G., Grant, D., Loihle, G., and Hunt, G. Debugging in the (very) large: Ten years of implementation and experience. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles* (Big Sky, MT, Oct. 11–14). ACM Press, New York, 103–116.
8. Grinter, R.E., Edwards, W.K., Chetty, M., Poole, E.S., Sung, J.-Y., Yang, J., Crabtree, A., Tolmie, P., Rodden, T., Greenhalgh, C., and Benford, S. The ins and outs of home networking: The case for useful and usable domestic networking. *ACM Transactions on Computer-Human Interaction* 16, 2 (June 2009), 1–28.
9. Grinter, R.E., Edwards, W.K., Newman, M., and

- Ducheneaut, N. The work to make a home network work. In *Proceedings of Ninth European Conference on Computer-Supported Cooperative Work* (Paris, Sept. 18–22). Springer-Verlag, Heidelberg, 2005, 469–488.
10. Intel. *Intel Remote PC Assist Technology (Intel RPAT) Software*; <http://www.intel.com/support/services/rpat/>
11. Kandula, S., Mahajan, R., Verkaik, P., Agarwal, S., Padhye, J., and Bahl, P. Detailed diagnosis in enterprise networks. In *Proceedings of the ACM SIGCOMM Conference on Data Communication* (Barcelona, Aug. 17–21). ACM Press, New York, 2009, 243–254.
12. Kempton, W. Two theories of home heat control. *Cultural models in language and thought. Cognitive Science* 10, 1 (1986), 75–90.
13. Laszlo, J. *Home Networking: Seizing Near-Term Opportunities to Extend Connectivity to Every Room Technical Report*. Jupiter Research, 2002.
14. Liu, Z., Lee, B., Kandula, S., and Mahajan, R. NetClinic: Interactive visualization to enhance automated fault diagnosis in enterprise networks. In *Proceedings of the IEEE Conference on Visual Analytics Science and Technology* (Salt Lake City, Oct. 24–29). IEEE, New York, 2010, 131–138.
15. Norman, D.A. *The Design of Everyday Things*. Doubleday, New York, 1990.
16. O'Brien, J., Rodden, T., Rouncefield, M., and Hughes, J. At home with the technology: An ethnographic study of a set-top-box trial. *ACM Transactions on Computer-Human Interaction* 6, 3 (Sept. 1999), 282–308.
17. Pew Internet and American Life Project. *Home Broadband Adoption*. Washington, D.C., 2009; <http://www.pewinternet.org/Reports/2009/10-Home-Broadband-Adoption-2009.aspx>
18. Pew Internet and American Life Project. *When Technology Fails*. Washington, D.C., 2008; <http://www.pewinternet.org/Reports/2008/When-Technology-Fails.aspx>
19. Pogue, D. Hot-spot shortcut in the weeds. *New York Times* (Apr. 7, 2010); <http://www.nytimes.com/2010/04/08/technology/personaltech/08pogue.html>
20. Poole, E.S., Chetty, M., Morgan, T., Grinter, R.E., and Edwards, W.K. Computer help at home: Methods and motivations for informal technical support. In *Proceedings of the 27th International Conference on Human Factors in Computing Systems* (Boston, Apr. 4–9). ACM Press, New York, 2009, 739–748.
21. Poole, E.S., Edwards, W.K., and Jarvis, L. The home network as a socio-technical system: Understanding the challenges of remote home network help. *Journal of Computer-Supported Cooperative Work* 18, 2–3 (2009), 277–299.
22. Scherf, K. *Parks Associates Panel on Home Networking* at the Consumer Electronics Association Conference (San Francisco, Oct. 14–16, 2002).
23. Shehan, E. and Edwards, W.K. Home networking and HCI: What hath God wrought? In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems* (San Jose, CA, Apr. 28–May 3). ACM Press, New York, 2007, 547–556.
24. Steinberg, D. and Cheshire, S. *Zero Configuration Networking: The Definitive Guide*. O'Reilly Media, Inc., Sebastopol, CA, 2005.
25. Tolmie, P., Pycock, J., Diggins, T., MacLean, A., and Karsenty, A. Unremarkable computing. In *Proceedings of the ACM Conference on Human Factors in Computing Systems* (Minneapolis, Apr. 20–25). ACM Press, New York, 399–406.
26. Venkatesh, A. Computers and other interactive technologies for the home. *Commun. ACM* 39, 12 (Dec.1996), 47–54.

W. Keith Edwards (keith@cc.gatech.edu) is an associate professor in the School of Interactive Computing of the Georgia Institute of Technology, Atlanta.

Rebecca E. Grinter (beki@cc.gatech.edu) is an associate professor in the School of Interactive Computing of the Georgia Institute of Technology, Atlanta.

Ratul Mahajan (ratul@microsoft.com) is a researcher in Microsoft Research, Redmond, WA, and an affiliate professor in the Department of Computer Science & Engineering of the University of Washington, Seattle.

David Wetherall (djw@uw.edu) is an associate professor in the Department of Computer Science & Engineering of the University of Washington, Seattle.

© 2011 ACM 0001-0782/11/06 \$10.00

DOI:10.1145/1953122.1953144

Partition data and operations, keep administration simple, do not assume one size fits all.

BY MICHAEL STONEBRAKER AND RICK CATTELL

10 Rules for Scalable Performance in ‘Simple Operation’ Datastores

THE RELATIONAL MODEL of data was proposed in 1970 by Ted Codd⁵ as the best solution for the DBMS problems of the day—business data processing. Early relational systems included System R² and Ingres,⁹ and almost all commercial relational DBMS (RDBMS) implementations today trace their roots to these two systems.

As such, unless you squint, the dominant commercial vendors—Oracle, IBM, and Microsoft—as well as the major open source systems—MySQL and PostgreSQL—all look about the same today; we term these systems general-purpose traditional row stores, or GPTRS,

sharing the following features:

- ▶ Disk-oriented storage;
- ▶ Tables stored row-by-row on disk, hence, a row store;
- ▶ B-trees as the indexing mechanism;
- ▶ Dynamic locking as the concurrency-control mechanism;
- ▶ A write-ahead log, or WAL, for crash recovery;
- ▶ SQL as the access language; and
- ▶ A “row-oriented” query optimizer and executor, pioneered in System R.⁷

The 1970s and 1980s were characterized by a single major DBMS market—business data processing—today called online transaction processing, or OLTP. Since then, DBMSs have come to be used in a variety of new markets, including data warehouses, scientific databases, social-networking sites, and gaming sites; the modern-day DBMS market is characterized in the figure here.

The figure includes two axes: horizontal, indicating whether an application is read-focused or write-focused, and vertical, indicating whether an application performs simple operations (read or write a few items) or complex operations (read or write thousands of items); for example, the traditional OLTP market is write-focused with simple operations, while the data warehouse market is read-focused with complex operations. Many appli-

» key insights

- Many scalable SQL and NoSQL datastores have been introduced over the past five years, designed for Web 2.0 and other applications that exceed the capacity of single-server RDBMSs.
- Major differences characterize these new datastores as to their consistency guarantees, per-server performance, scalability for read versus write loads, automatic recovery from failure of a server, programming convenience, and administrative simplicity.
- Applications must be designed for scalability, partitioning application data into “shards,” avoiding operations that span partitions, designing for parallelism, and weighing requirements for consistency guarantees.

1
Look for
shared-nothing
scalability.

High-level
languages are
GOOD
and need not hurt
performance.

Plan to carefully
LEVERAGE main
memory databases.

4
High availability and automatic
recovery are essential for
SO scalability.

5
**ONLINE
EVERYTHING.**

Avoid
MULTI-NODE
operations.

Don't try to
build ACID
consistency
yourself.

Look for
administrative
SIMPLICITY.

Pay attention
to **NODE**
performance.

OPEN SOURCE
gives you more
CONTROL over your future.

cations are, of course, in between; for example, social-networking applications involve mostly simple operations but also a balance of reads and writes. Hence, the figure should be viewed as a continuum in both directions, with any given application somewhere in between.

The major commercial engines and open source implementations of the relational model are positioned as “one-size-fits-all” systems; that is, their implementations are claimed to be appropriate for all locations in the figure.

However, there is also some dissatisfaction with one-size-fits-all. Witness, for example, the commercial success of the so-called column stores in the data-warehouse market. With these products, only those columns needed in the query are retrieved from disk, eliminating overhead for unused data. In addition, superior compression and indexing is obtained, since only one kind of object exists on each storage block, rather than several-to-many. Finally, main-memory bandwidth is economized through a query executor that operates on compressed data. For these reasons, column stores are remarkably faster than row stores on typical data-warehouse workloads, and we expect them to dominate the data-warehouse market over time.

Our focus here is on simple-operation (SO) applications, the lower portion of the figure. Quite a few new, non-GPTRS systems have been designed to provide scalability for this market. Loosely speaking, we classify them into the following four categories:

Key-value stores. Includes Dynamo, Voldemort, Membase, Membrain, Scalaris, and Riak. These systems have the simplest data model: a collection of

objects, each with a key and a payload, providing little or no ability to interpret the payload as a multi-attribute object, with no query mechanism for non-primary attributes;

Document stores. Includes CouchDB, MongoDB, SimpleDB, and Terastore in which the data model consists of objects with a variable number of attributes, some allowing nested objects. Collections of objects are searched via constraints on multiple attributes through a (non-SQL) query language or procedural mechanism;

Extensible record stores. Includes BigTable, Cassandra, HBase, HyperTable, and PNUTS providing variable-width record sets that can be partitioned vertically and horizontally across multiple nodes. They are generally not accessed through SQL; and

SQL DBMSs. Focus on SO application scalability, including MySQL Cluster, other MySQL derivatives, VoltDB, NimbusDB, and Clustrix. They retain SQL and ACID (Atomicity, Consistency, Isolation, and Durability)^a transactions, but their implementations are often very different from those of GPTRS systems.

We do not claim this classification is precise or exhaustive, though it does cover the major classes of newcomer. Moreover, the market is changing rapidly, so the reader is advised to check other sources for the latest. For a more thorough discussion and references for these systems, see Cattell⁴ and the table here.

The NoSQL movement is driven largely by the systems in the first three categories, restricting the traditional notion of ACID transactions by allowing only single-record operations to be

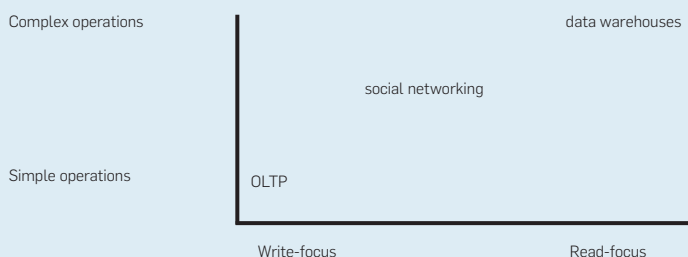
transactions and/or by relaxing ACID semantics, by, say, supporting only “eventual consistency” on multiple versions of data.

These systems are driven by a variety of motivations. For some, it is dissatisfaction with the relational model or the “heaviness” of RDBMSs. For others, it is the needs of large Web properties with some of the most demanding SO problems around. Large Web properties were frequently start-ups lucky enough to experience explosive growth, the so-called hockey-stick effect. They typically use an open source DBMS, because it is free or already understood by the staff. A single-node DBMS solution might be built for version 1, which quickly exhibits scalability problems. The conventional wisdom is then to “shard,” or partitioning the application data over multiple nodes that share the load. A table can be partitioned this way; for example, employee names can be partitioned onto 26 nodes by putting all the “A”s on node 1 and so forth. It is now up to application logic to direct each query and update to the correct node. However, such sharding in application logic has a number of severe drawbacks:

- ▶ If a cross-shard filter or join must be performed, then it must be coded in the application;
- ▶ If updates are required within a transaction to multiple shards, then the application is responsible for somehow guaranteeing data consistency across nodes;
- ▶ Node failures are more common as the system scales. A difficult problem is how to maintain consistent replicas, detect failures, fail over to replicas, and replace failed nodes in a running system;
- ▶ Making schema changes without taking shards “offline” is a challenge; and
- ▶ Reprovisioning the hardware to add additional nodes or change the configuration is extremely tedious and, likewise, much more difficult if the shards cannot be taken offline.

Many developers of sharded Web applications experience severe pain because they must perform these functions in application-level logic; much of the NoSQL movement targets this pain point. However, with the large number of new systems and the wide

A characterization of DBMS applications.



^a ACID; see <http://en.wikipedia.org/wiki/ACID>

range of approaches they take, customers^b might have difficulty understanding and choosing a system to meet their application requirements.

Here, we present 10 rules we advise any customer to consider with an SO application and in examining non-GP-TRS systems. They are a mix of DBMS requirements and guidelines concerning good SO application design. We state them in the context of customers running software in their own environment, though most also apply to software-as-a-service environments.

We lay out each rule, then indicate why it is necessary:

Rule 1. Look for shared-nothing scalability. A DBMS can run on three hardware architectures: The oldest—shared-memory multiprocessing (SMP)—means the DBMS runs on a single node consisting of a collection of cores sharing a common main memory and disk system. SMP is limited by main memory bandwidth to a relatively small number of cores. Clearly, the number of cores will increase in future systems, but it remains to be seen if main memory bandwidth will increase commensurately. Hence, multicore systems face performance limitations with DBMS software. Customers choosing an SMP system were forced to perform sharding themselves to obtain scalability across SMP nodes and face the painful problems noted earlier. Popular systems running on SMP configurations are MySQL, PostgreSQL, and Microsoft SQL Server.

A second option is to choose a DBMS that runs on disk clusters, where a collection of CPUs with private main memories share a common disk system. This architecture was popularized in the 1980s and 1990s by DEC, HP, and Sun but involves serious scalability problems in the context of a DBMS. Due to the private buffer pool in each node's main memory, the same disk block can be in multiple buffer pools. Hence, careful synchronization of these buffer-pool blocks is required. Similarly, a private lock table is included in each node's main memory. Careful synchronization is again required

but limits the scalability of a shared disk configuration to a small number of nodes, typically fewer than 10.

Oracle RAC is a popular example of a DBMS running shared disk, and it is difficult to find RAC configurations with a double-digit number of nodes. Oracle recently announced Exadata and Exadata 2 running shared disk at the top level of a two-tier hierarchy while running shared-nothing at the bottom level.

The final architecture is a shared-nothing configuration, where each node shares neither main memory nor disk; rather, the nodes in a collection of self-contained nodes are connected to one another through networking. Essentially, all DBMSs oriented toward the data warehouse market since 1995 run shared-nothing, including Greenplum, Vertica, Asterdata, Parac-

cel, Netezza, and Teradata. Moreover, DB2 can run shared-nothing, as do many NoSQL engines. Shared-nothing engines normally perform automatic sharding (partitioning) of data to achieve parallelism. Shared-nothing systems scale only if data objects are partitioned across the system's nodes in a manner that balances the load. If there is data skew or "hot spots," then a shared-nothing system degrades in performance to the speed of the overloaded node. The application must also make the overwhelming majority of transactions "single-sharded," a point covered further in Rule 6.

Unless limited by application data/operation skew, well-designed, shared-nothing systems should continue to scale until networking bandwidth is exhausted or until the needs of the application are met. Many NoSQL

System information sources.

Systems	Link
Asterdata	http://asterdata.com
BigTable	http://labs.google.com/papers/bigtable.html
Clustrix	http://clustrix.com
CouchDB	http://couchdb.apache.org
DB2	http://ibm.com/software/data/db2
Dynamo	http://portal.acm.org/citation.cfm?id=1294281
Exadata	http://oracle.com/exadata
Greenplum	http://greenplum.com
Hadoop	http://hadoop.apache.org
HBase	http://hbase.apache.org
HyperTable	http://hypertable.org
MongoDB	http://mongodb.org
MySQL	http://mysql.com/products/enterprise
MySQL Cluster	http://mysql.com/products/database/cluster
Netezza	http://netezza.com
NimbusDB	http://nimbusdb.com
Oracle	http://oracle.com
Oracle RAC	http://oracle.com/rac
Paracel	http://paracel.com
PNUTs	http://research.yahoo.com/pub/2304
PostgreSQL	http://postgresql.org
Riak	http://basho.com/Riak.html
Scalaris	http://code.google.com/p/scalaris
SimpleDB	http://amazon.com/simpledb
SQL Server	http://microsoft.com/sqlserver
Teradata	http://teradata.com
Terrastore	http://code.google.com/p/terrastore
Tokyo Cabinet	http://1978th.net/tokyocabinet
Vertica	http://vertica.com
Voldemort	http://project-voldemort.com
VoltDB	http://voldb.com

^b We use the term "customer" to refer to any organization evaluating or using one of these systems, even though some of them are open source, with no vendor.

systems reportedly run 100 nodes or more, and BigTable reportedly runs on thousands of nodes.

The DBMS needs of Web applications can drive DBMS scalability upward in a hurry; for example, Facebook was recently sharding 4,000 MySQL instances in application logic. If it chose a DBMS, it would have to scale at least to this number of nodes. An SMP or shared-disk DBMS has no chance at this level of scalability. Shared-nothing DBMSs are the only game in town.

Rule 2. High-level languages are good and need not hurt performance. Work in a SQL transaction can include the following components:

- ▶ Overhead resulting from the optimizer choosing an inferior execution plan;
- ▶ Overhead of communicating with the DBMS;
- ▶ Overhead inherent in coding in a high-level language;
- ▶ Overhead for services (such as concurrency control, crash recovery, and data integrity); and
- ▶ Truly useful work to be performed, no matter what.

Here, we cover the first three, leaving the last two for Rule 3. Hierarchical and network systems were the dominant DBMS solutions in the 1960s and 1970s, offering a low-level procedural interface to data. The high-level language of RDBMSs was instrumental in displacing these DBMSs for three reasons:

- ▶ A high-level language system requires the programmer write less code that is easier to understand;
- ▶ Users state what they want instead of writing a disk-oriented algorithm on how to access the data they need; a programmer need not understand complex storage optimizations; and
- ▶ A high-level language system has a better chance of allowing a program to survive a change in the schema without maintenance or recoding; as such, low-level systems require far more maintenance.

One charge leveled at RDBMSs in the 1970s and 1980s was they could not be as efficient as low-level systems. The claim was that automatic query optimizers could not do as good a job as smart programmers. Though early optimizers were primitive, they were quickly as good as all but the best hu-

man programmers. Moreover, most organizations could never attract and retain this level of talent. Hence, this source of overhead has largely disappeared and today is only an issue on very complex queries rarely found in SO applications.

The second source of overhead is communicating with the DBMS. For security reasons, RDBMSs insist on the application being run in a separate address space, using ODBC or JDBC for DBMS interaction. The overhead of these communication protocols is high; running a SQL transaction requires several back-and-forth messages over TCP/IP. Consequently, any programmer seriously interested in performance runs transactions using a stored-procedure interface, rather than SQL commands over ODBC/JDBC. In the case of stored procedures, a transaction is a single over-and-back message. The DBMS further reduces communication overhead by batching multiple transactions in one call. The communication cost is a function of the interface selected, can be minimized, and has nothing to do with the language level of the interaction.

The third source of overhead is coding in SQL rather than in a low-level procedural language. Since most serious SQL engines compile to machine code or at least to a Java-style intermediate representation, this overhead is not large; that is, standard language compilation converts a high-level specification into a very efficient low-level runtime executable.

Hence, one of the key lessons in the DBMS field over the past 25 years is that high-level languages are good and do not hurt performance. Some new systems provide SQL or a more limited higher-level language; others provide only a “database assembly language,” or individual index and object operations. This low-level interface may be adequate for very simple applications, but, in all other cases, high-level languages provide compelling advantages.

Rule 3. Plan to carefully leverage main memory databases. Consider a cluster of 16 nodes, each with 64GB of main memory. Any shared-nothing DBMS thereby has access to about 1TB of main memory. Such a hardware configuration would have been considered

extreme a few years ago but is commonplace today. Moreover, memory per node will increase in the future, and the number of nodes in a cluster is also likely to increase. Hence, typical future clusters will have increasing terabytes of main memory.

As a result, if a database is a couple of terabytes or less (a very large SO database), customers should consider main-memory deployment. If a database is larger, customers should consider main-memory deployment when practical. In addition, flash memory has become a promising storage medium, as prices have decreased.

Given the random-access speed of RAM versus disk, a DBMS can potentially run thousands of times faster. However, the DBMS must be architected properly to utilize main memory efficiently; only modest improvements are achievable by simply running a DBMS on a machine with more memory.

To understand why, consider the CPU overhead in DBMSs. In 2008, Harizopoulos et al.⁶ measured performance using part of a major SO benchmark, TPC-C, on the Shore open-source DBMS. This DBMS was chosen because the source code was available for instrumentation and because it was a typical GPTRS implementation. Based on simple measures of other GPTRS systems, the Shore results are representative of those systems as well.

Harizopoulos et al.⁶ used a database size that allowed all data to fit in main memory, as it was consistent with most SO applications. Since Shore, like other GPTRS systems, is disk-based, it meant all data would reside in the main memory buffer pool. Their goal was to categorize DBMS overhead on TPC-C; they ran the DBMS in the same address space as the application driver, avoiding any TCP/IP cost. They then looked at the components of CPU usage that perform useful work or deliver DBMS services.

Following is the CPU cycle usage for various tasks in the new-order transaction of TPC-C; since Harizopoulos et al.⁶ noted some shortcomings that are fixed in most commercial GPTRSs, these results have been scaled to assume removal of these sources of overhead:

Useful work (13%). This is the CPU

cost for actually finding relevant records and performing retrieval or update of relevant attributes;

Locking (20%). This is the CPU cost of setting and releasing locks, detecting deadlock, and managing the lock table;

Logging (23%). When a record is updated, the before and after images of the change are written to a log. Shore then groups transactions together in a “group commit,” forcing the relevant portions of the log to disk;

Buffer pool overhead (33%). Since all data resides in the buffer pool, any retrievals or updates require finding the relevant block in the buffer pool. The appropriate record(s) must then be located and the relevant attributes in the record found. Blocks on which there is an open database cursor must be “pinned” in main memory. Moreover, Least-Recently-Used or another replacement algorithm is utilized, requiring the recording of additional information; and

Multithreading overhead (11%). Since most DBMSs are multithreaded, multiple operations are going on in parallel. Unfortunately, the lock table is a shared data structure that must be “latched” to serialize access by the various parallel threads. In addition, B-tree indexes and resource-management information must be similarly protected. Latches (mutexes) must be set and released when shared data structures are accessed. See Harizopoulos et al.⁶ for a more detailed discussion, including on why the latching overhead may be understated.

A conventional disk-based DBMS clearly spends the overwhelming majority of its cycles on overhead activity. To go a lot faster, the DBMS must avoid all the overhead components discussed here; for example, a main memory DBMS with conventional multithreading, locking, and recovery is only marginally faster than its disk-based counterpart. A NoSQL or other database engine will not dramatically outperform a GPTRS implementation, unless all these overhead components are addressed or the GPTRS solution has not been properly architected (by, say, using conversational SQL rather than a compiled stored procedure interface).

We look at single-machine perfor-

Shared-nothing DBMSs are the only game in town.

mance in our analysis, but this performance has a direct effect on the multi-machine scalability discussed in Rule 1, as well as in our other rules.

Rule 4. High availability and automatic recovery are essential for SO scalability. In 1990, a typical DBMS application would run on what we would now consider very expensive hardware. If the hardware failed, the customer would restore working hardware, reload the operating system and DBMS, then recover the database to the state of the last completed transaction by performing an undo of incomplete transactions and a redo of completed transactions using a DBMS log. This process could take time (several minutes to an hour or more) during which the application would be unavailable.

Few customers today are willing to accept any downtime in their SO applications, and most want to run redundant hardware and use data replication to maintain a second copy of all objects. On a hardware failure, the system switches over to the backup and continues operation. Effectively, customers want “nonstop” operation, as pioneered in the 1980s by Tandem Computers.

Furthermore, many large Web properties run large numbers of shared-nothing nodes in their configurations, where the probability of failure rises as the number of “moving parts” increases. This failure rate renders human intervention impractical in the recovery process; instead, shared-nothing DBMS software must automatically detect and repair failed nodes.

Any DBMS acquired for SO applications should have built-in high availability, supporting nonstop operation. Three high-availability caveats should be addressed. The first is that there is a multitude of kinds of failure, including:

- ▶ Application, where the application corrupts the database;
- ▶ DBMS, where the bug can be recreated (so-called Bohr bugs);
- ▶ DBMS, where the bug cannot be recreated (so-called Heisenbugs);
- ▶ Hardware, of all kinds;
- ▶ Lost network packets;
- ▶ Denial-of-service attacks; and
- ▶ Network partitions.

Any DBMS will continue operation for some but not for all these failure

modes. The cost of recovering from all possible failure modes is very high. Hence, high availability is a statistical effort, or how much availability is desired against particular classes of failures.

The second caveat is the so-called CAP, or consistency, availability, and partition-tolerance, theorem.³ In the presence of certain failures, it states that a distributed system can have only two out of these three characteristics: consistency, availability, and partition-tolerance. Hence, there are theoretical limits on what is possible in the high-availability arena.

Moreover, many site administrators want to guard against disasters (such as earthquakes and floods). Though rare, recovery from disasters is important and should be viewed as an extension of high availability, supported by replication over a wide-area network.

Rule 5. Online everything. An SO DBMS should have a single state: “up.” From the user’s point of view, it should never fail and never have to be taken offline. In addition to failure recovery, we need to consider operations that require the database be taken offline in many current implementations:

Schema changes. Attributes must be added to an existing database without interruption in service;

Index changes. Indexes should be added or dropped without interruption in service;

Reprovisioning. It should be possible to increase the number of nodes used to process transactions, without interruption in service; for example, a configuration might go from 10 nodes to 15 nodes to accommodate an increase in load; and

Software upgrade. It should be possible to move from version N of a DBMS to version $N + 1$ without interruption of service.

Though supporting these operations is a challenge, 100% uptime should be the goal. As an SO system scales to dozens of nodes and/or millions of users on the Internet, downtime and manual intervention are not practical.

Rule 6. Avoid multi-node operations. Two characteristics are necessary for achieving SO scalability over a cluster of servers:

Even split. The database and ap-



Avoid multi-shard operations to the greatest extent possible, including queries that must go to multiple shards, as well as multi-shard updates requiring ACID properties.



plication load must be split evenly over the servers. Read-scalability can be achieved by replicating data, but general read/write scalability requires sharding (partitioning) the data over nodes according to a primary key; and

Scalability advantage. Applications rarely perform operations spanning more than one server or shard. If a large number of servers is involved in processing an operation, the scalability advantage may be lost because of redundant work, cross-server communication, or required operation synchronization.

Suppose a customer has an employee table and partitions it based on employee age. If it wants to know the salary of a specific employee, it must then send the query to all nodes, requiring a slew of messages. Only one node will find the desired data; the others will run a redundant query that finds nothing. Furthermore, if an application performs an update that crosses shards, giving, say, a raise to all employees in the shoe department, then the system must pay all of the synchronization overhead of ensuring the transaction is performed on every node.

Hence, a database administrator (DBA) should choose a sharding key to make as many operations single-shard as possible. Fortunately, most applications naturally involve single-shard transactions, if the data is partitioned properly; for example, if purchase orders (POs) and their details are both sharded on PO number, then the vast majority of transactions (such as new PO and update a specific PO) go to a single node.

The percentage of single-node transactions can be increased further by replicating read-only data; for example, a list of customers and their addresses can be replicated at all sites. In many business-to-business environments, customers are added or deleted or change their addresses infrequently. Hence, complete replication allows inserting the address of a customer into a new PO as a single-node operation. Therefore, selective replication of read-mostly data can be advantageous.

In summary, programmers should avoid multi-shard operations to the greatest extent possible, including queries that must go to multiple shards, as well as multi-shard updates

requiring ACID properties. Customers should carefully think through their application and database design to accomplish this goal. If that goal is unachievable with the current application design, they should consider a redesign that achieves higher “single-shardedness.”

Rule 7. Don’t try to build ACID consistency yourself. In general, the key-value stores, document stores, and extensible record stores we mentioned have abandoned transactional ACID semantics for a weaker form of atomicity, isolation, and consistency, providing one or more of the following alternative mechanisms:

- ▶ Creating new versions of objects on every write that result in parallel versions when there are multiple asynchronous writes; it is up to application logic to resolve the resulting conflict;
- ▶ Providing an “update-if-current” operation that changes an object only if it matches a specified value; this way, an application can read an object it plans to later update and then make changes only if the value is still current;
- ▶ Providing ACID semantics but only for read and write operations of a single object, attribute, or shard; and
- ▶ Providing “quorum” read-and-write operations that guarantee the latest version among “eventually consistent” replicas.

It is possible to build your own ACID semantics on any of these systems, given enough additional code. However, the task is so difficult, we wouldn’t wish it on our worst enemy. If you need ACID semantics, you want to use a DBMS that provides them; it is much easier to deal with this at the DBMS level than at the application level.

Any operation requiring coordinated updates to two objects is likely to need ACID guarantees. Consider a transaction that moves \$10 between two user accounts; with an ACID system, a programmer can simply write:

```
Begin transaction
Decrement account A
Increment account B
Commit transaction
```

Without an ACID system, there is no easy way to perform this coordinated action. Other cases requiring

ACID semantics include charging customers’ accounts only if their orders ship and synchronously updating bilateral “friend” references. Standard ACID semantics give the programmer the all-or-nothing guarantee needed to maintain data integrity in them. Although some applications do not need such coordination, a commitment to a non-ACID system precludes extending such applications in the future in a way that requires coordination. DBMS applications often live a long time and are subject to unknown future requirements.

We understand the NoSQL movement’s motivation for abandoning transactions, given its belief that transactional updates are expensive in traditional GPTRS systems. However, newer SQL engines can offer both ACID and high performance by carefully eliminating all overhead in Rule 3, at least for applications that obey Rule 6 (avoid multinode operations). If you need ACID transactions and cannot follow Rule 6, then you will likely incur substantial overhead, no matter whether you code the ACID yourself or let the DBMS do it. Letting the DBMS do it is a no-brainer.

We have heard the argument for abandoning ACID transactions based on the CAP theorem,³ stating you can have only two of three characteristics: C consistency, A availability, and P partition-tolerance. The argument is that partitions happen, hence one must abandon consistency to achieve high availability. We take issue for three reasons: First, some applications really do need consistency and cannot give it up. Second, the CAP theorem deals with only a subset of possible failures, as noted in Rule 4, and one is left with how to cope with the rest. And third, we are not convinced that partitions are a substantial issue for data sharded on a LAN, particularly with redundant LANs and applications on the same site; in this case, partitions may be rare, and one is better off choosing consistency (all the time) over availability during a very rare event.

Though true that WAN partitions are much more likely than LAN partitions, WAN replication is normally used for read-only copies or disaster recovery (such as when an entire data center goes offline); WAN latency is

too high for synchronous replication or sharding. Few users expect to recover from major disasters without short availability hiccups, so the CAP theorem may be less relevant in this situation.

We advise customers who need ACID to seek a DBMS that provides it, rather than code it themselves, minimizing the overhead of distributed transactions through good database and application design.

Rule 8. Look for administrative simplicity. One of our favorite complaints about relational DBMSs is their poor out-of-the-box behavior. Most products include many tuning knobs that allow adjustment of DBMS behavior; moreover, our experience is that a DBA skilled in a particular vendor’s product, can make it go a factor of two or more faster than one unskilled in the given product.

As such, it is a daunting task to bring in a new DBMS, especially one distributed over many nodes; it requires installation, schema construction, application design, data distribution, tuning, and monitoring. Even getting a high-performance version of TPC-C running on a new engine takes weeks, though code and schema are readily available. Moreover, once an application is in production, it still requires substantial DBA resources to keep it running.

When considering a new DBMS, one should carefully consider the out-of-the-box experience. Never let the vendor do a proof-of-concept exercise for you. Do the proof of concept yourself, so you see the out-of-the-box situation up close. Also, carefully consider application-monitoring tools in your decision.

Lastly, pay particular attention to Rule 5. Some of the most difficult administrative issues (such as schema changes and reprovisioning) in most systems require human intervention.

Rule 9. Pay attention to node performance. A common refrain heard these days is “Go for linear scalability; that way you can always provision to meet your application needs, while node performance is less important.” Though true that linear scalability is important, ignoring node performance is a big mistake. One should always remember that linear scalability

means overall performance is a multiple of the number of nodes times node performance. The faster the node performance, the fewer nodes one needs.

It is common for solutions to differ in node performance by an order of magnitude or more; for example, in DBMS-style queries, parallel DBMSs outperform Hadoop by more than an order of magnitude,¹ and, similarly, H-store (the prototype predecessor to VoltDB) has been shown to have even higher throughput on TPC-C compared to the products from major vendors.⁸ For example, consider a customer choosing between two database solutions, each offering linear scalability. If solution A offers node performance a factor of 20 better than solution B, the customer might require 50 hardware nodes with solution A versus 1,000 nodes with solution B.

Such a wide difference in hardware cost, rack space, cooling, and power consumption is obviously non-trivial between the two solutions. More important, if each node fails on average every three years, then solution B will see a failure every day, while solution A will see a failure less than once a month. This dramatic difference will heavily influence how much redundancy is installed and how much administrative time is required to deal with reliability. Node performance makes everything else easier.

Rule 10. Open source gives you more control over your future. This final rule is not a technical point but still important to mention, and, hence, perhaps, should be a suggestion rather than a rule. The landscape is littered with situations where a company acquired a vendor's product, only to face expensive upgrades in the following years, large maintenance bills for often-inferior technical support, and the inability to avoid these fees because the cost of switching to a different product would require extensive recoding. The best way to avoid "vendor malpractice" is to use an open source product. Open source eliminates expensive licenses and upgrades and often provides multiple alternatives for support, new features, and bug fixes, including the option of doing them in-house.

For these reasons, many newer Web-oriented shops are adamant about using only open source systems. Also,

The best way to avoid "vendor malpractice" is to use an open source product.

several vendors have proved it possible to make a viable business with an open source model. We expect it to be more popular over time, and customers are well advised to consider its advantages.

Conclusion

The 10 rules we've presented specify the desirable properties of any SO datastore. Customers looking at distributed data-storage solutions are well advised to view the systems they are considering in the context of this rule set, as well as in that of their own unique application requirements. The large number of systems available today range considerably in capabilities and limitations.

Acknowledgments

We would like to thank Andy Pavlo, Rick Hillegas, Rune Humberstad, Stavros Harizopoulos, Dan DeMaggio, Dan Weinreb, Daniel Abadi, Evan Jones, Greg Luck, and Bobbi Heath for their valuable input on this article. 

References

1. Abadi, D. et al. A comparison of approaches to large-scale data analysis. In *Proceedings of the 2009 SIGMOD Conference on Management of Data* (Providence, RI, June 29). ACM Press, New York, 2009, 165–178.
2. Astrahan, M.M. et al. System R: A relational approach to data management. *ACM Transactions on Database Systems* 1, 2 (June 1976), 97–137.
3. Brewer, E. Towards Robust Distributed Systems. Keynote at Conference on Principles of Distributed Computing (Portland, OR, July 2000); <http://www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf>
4. Cattell, R. Scalable SQL and NoSQL datastores. *ACM SIGMOD Record* 40, 2 (June 2011).
5. Codd, E.F. A relational model of data for large shared databanks. *Commun. ACM* 13, 6 (June 1970), 377–387.
6. Harizopoulos, S. et al. OLTP: Through the looking glass and what we found there. In *Proceedings of the 2008 SIGMOD Conference on Management of Data* (Vancouver, B.C., June 10). ACM Press, New York, 2008, 981–992.
7. Selinger, P. Access path selection in a relational data management system. In *Proceedings of the 1979 ACM SIGMOD Conference on Management of Data* (Boston, May 30). ACM Press, New York, 1979, 23–24.
8. Stonebraker, M. et al. The end of an architectural era (It's time for a complete rewrite). In *Proceedings of the 2007 Very Large Databases Conference* (Vienna, Austria, Sept. 23). ACM Press, New York, 2007, 399–410.
9. Stonebraker, M. et al. The design and implementation of Ingres. *ACM Transactions on Database Systems* 1, 3 (Sept. 1976), 189–222.

Michael Stonebraker (stonebraker@csail.mit.edu) is an adjunct professor in the Computer Science and Artificial Intelligence Laboratory at the Massachusetts Institute of Technology, consultant and founder, Paradigm4, Inc., consultant and founder, Goby, Inc., and consultant and founder, VoltDB, Inc.

Rick Cattell (rick@cattell.net) is a database technology consultant at Cattell.Net and on the technical advisory board of Schooner Information Technologies.

Can a programming language really help programmers write better programs?

BY MIKE BARNETT, MANUEL FÄHNDRICH, K. RUSTAN M. LEINO, PETER MÜLLER, WOLFRAM SCHULTE, AND HERMAN VENTER

Specification and Verification: The Spec# Experience

A MAIN REASON for the difficulty of programming is the inability of programmers to ensure their programs behave as intended. Needed is both a way to record that intent and the tools to enforce it. Spec# (pronounced “speck sharp”)^a is a project from Microsoft Research aimed at addressing the

problem in the context of modern object-oriented languages, using the well-known approach of contracts, or specification constructs, to document behavior.²² Contracts standardize the common practice of writing assertions within code through two main constructs:

Method pre- and postconditions. Method pre- and postconditions are part of the application- programming interface (API) for methods. Preconditions describe what is to be true at method entry, callers establish them,

^a <http://specsharp.codeplex.com>

» key insights

- Programmers would provide more information about their code than existing programming languages allow, if only they received some benefit from doing so.
- Object-oriented languages should use a type system that distinguishes references that can never be null; programmers spend far too much of their time having to compensate for this limitation.
- Program verification systems receive a tremendous benefit by leveraging a shared infrastructure and a higher-level way of communicating with automatic theorem provers.

and implementers can assume them. Postconditions describe what is to be true at method exit; implementers establish them, and callers can assume them upon method return.

Object invariants. Object invariants are a way to specify the steady-state properties that all “good” instances of a class should maintain. A crucial feature setting Spec# apart from previous programming systems is a sound technique for reasoning about when object

invariants hold; why this is such a difficult problem and how Spec# solves it are covered in the section on invariants.

Spec# enforces both kinds of contracts with instrumentation for run-time checking and with an automatic program verifier for static, compile-time checking.

Programmers interact with Spec# just as they do with any other programming system: Type in the program,

and respond to errors. The difference is that in Spec#, one writes specifications, as well as code. In return, the system analyzes the program as it is being written and detects many errors traditional approaches would reveal only during testing (or deployed execution).

Figure 1 is a glimpse at what Spec# has to offer, with a Spec# project edited in the Visual Studio integrated development environment (IDE); shown is the definition and implementation of an interface used in a parsing framework. The method `ParseBinding` is used to pull apart a string of the form “a = b”. Like many programming languages, Spec# does not allow interface methods to contain code though does allow them to have contracts, in this case a precondition (keyword **requires**) saying the argument provided to the method must contain the character ‘=’.

Contracts are a native part of the Spec# language in two ways: Method contracts are part of the signature of a method, and the expressions contained in the contracts are written in the programming language itself, not in a secondary logical metalanguage (see Figure 2). `ParseBinding`’s precondition is written using a call to a method in the standard .NET library, `Contains`. The Spec# programming system comes with a set of contracts for the .NET Framework, providing a (partial) semantics for such commonly used methods.

All implementations of `ParseBinding` written in Spec# inherit the interface method’s contract and so do not have to perform error checking or defensive programming. This behavior is illustrated in the implementation that calls the library method `IndexOf` with the assurance that the return value is a valid index into the receiver string, as guaranteed by the postcondition (not shown) of the `Contains` method. Thus, programmers can use the return value as an argument to the method `Substring`, the precondition of which requires the argument to be a valid index.

Spec# enforces `ParseBinding`’s precondition on any client making a call to the interface method. A (particularly stupid) client is shown in Figure 2, where the Spec# system has noticed

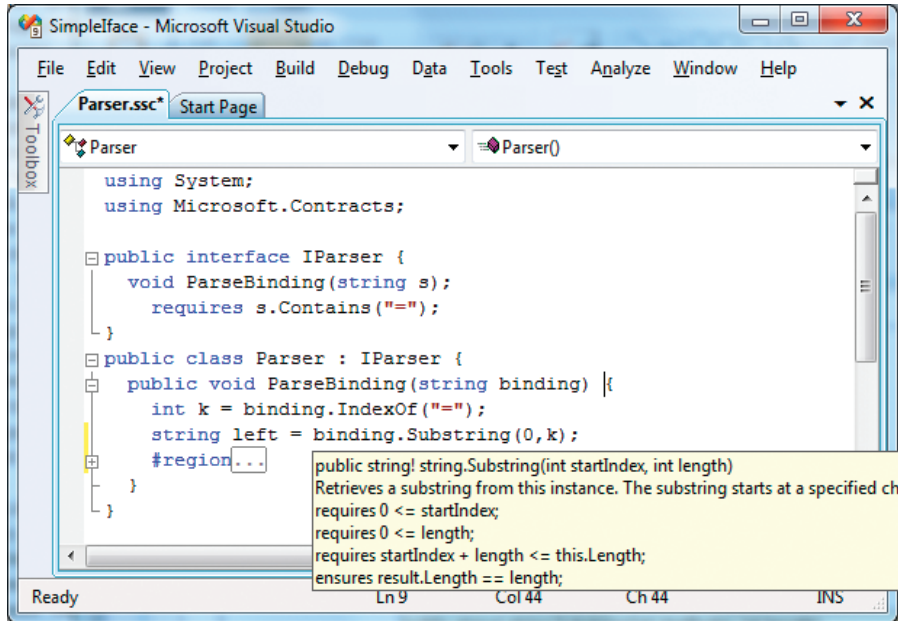


Figure 1. A (partial) Spec# interface. The yellow box is a tooltip that appears when the mouse hovers over the call to `Substring`, showing the signature of the method and a short programmer-written summary and its contract. Just like the barking dog (as in *Silver Blaze*, Arthur Conan Doyle), the important thing to notice is the *absence* of warnings on the call to `Substring`.

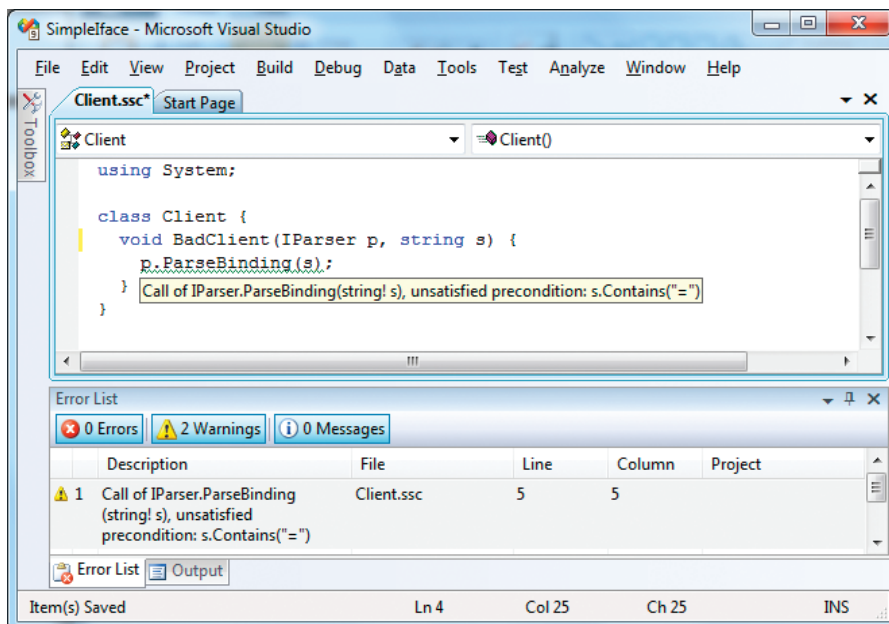


Figure 2. A client using the `IParser` interface incorrectly; notice that verification errors are presented in the same format as compiler errors.

the error as the programmer was typing the code. The resulting *squiggly*, the visual underlining in the editor, alerts the programmer that the code is violating a contract. The tooltip window shown pops up in response to the programmer hovering over the squiggled text with the mouse. The error list in the IDE is also populated with a warning about the contract error.

Note that Figures 1 and 2 do not include warnings about object references possibly being null because Spec# distinguishes between non-null types and possibly-null types. In the examples, `IParser` and `string` are both non-null types.

The Spec# Language

Spec# is an object-oriented language, a superset of C# v2.0 (released in 2005), compiling to the Microsoft Intermediate Language bytecode (MSIL) and running on the .NET virtual machine and integrated into Visual Studio's IDE, which provides language services (such as syntax highlighting and the ability to run the program verifier in the background as the code is being written).

The extensions to C# consist chiefly of the standard design-by-contract features²² (method contracts and object invariants), as well as a non-null type system. For a full introduction to the language, see the Spec# tutorial²⁰; Figure 3 outlines the most commonly used features.

The first postcondition (keyword **ensures**) of `CrankItUp` uses the expression `old(Volume())` to refer to the value of `Volume()` on entry to the method, promising the value of `Volume()` is increased by `amount`. The second postcondition expresses that the method returns the final value of `Volume()`; the Spec# keyword **result** refers to the return value of the method.

Since contracts must not cause state changes, methods may be used in contracts only if they are side-effect free,⁴ as indicated by the `[Pure]` custom attribute^b, as in the definition of `Volume()`.

Reasoning about a method call is in terms of the method's contract.

Because method contracts are inherited in subclasses, this reasoning applies even in the presence of dynamic method dispatch where the particular method implementation invoked may not be known until runtime; that is, contract inheritance enforces the well-known concept of behavioral subtyping.^{9,21}

The class `Stereo` declares three object invariants to specify what it means for an object of this class to be consistent. Whereas the first two invariants constrain the values of the fields of a `Stereo` object, the third invariant relates the states of two sub-objects. The first assignment statement in the body of the method `CrankItUp` might break that invariant before the subsequent assignment reestablishes it. To indicate that an object invariant might be temporarily violated, the two assignment statements must appear within an **expose** statement, described in more detail in the section on invariants. Note that no **expose** statement is needed in `ChangeCD`, because the single assignment maintains the invariants.

Null-dereference problems are the

bane of object-oriented programming. We and others have found the single most common specification is the exclusion of the null value from the possible values of a field, method parameter, or result. Spec# refines C#'s type system by distinguishing non-null types (such as the type `Speaker` of the fields `left` and `right` in Figure 3) and possibly null types (written with a postfix question mark, as in `Speaker?`)^c. References of non-null types can be dereferenced safely without requiring runtime checks or proof obligations to prevent errors.

Each object of type `Stereo` is an aggregate object containing references to other objects that make up its internal representation. In Spec#, the aggregate/sub-object relation is expressed using the `[Rep]` custom attribute in the declaration of the field pointing to the sub-object. In our example, the speakers are sub-objects of a `Stereo` object; we say the `Stereo` object

c For traditionalists, Spec# also offers the complementary mode where `Speaker` represents the possibly null type and the non-null type is written as `Speaker!`.

Figure 3. A (partial) Spec# program demonstrating the language's basic features, including method contracts that describe (part of) the method behavior, as well as object invariants that describe the consistent state of each instance of the class.

```
public class Stereo {
    int currentCDSlot;
    [Rep] Speaker left = new Speaker();
    [Rep] Speaker right = new Speaker();

    invariant 0 <= currentCDSlot;
    invariant left != right;
    invariant left.Gain == right.Gain;

    public int CrankItUp(int amount)
        requires 0 <= amount;
        ensures Volume() == old(Volume()) + amount;
        ensures result == Volume();
    {
        expose (this) {
            left.Adjust(amount);
            right.Adjust(amount);
        }
        ...
    }

    [Pure] public int Volume()
    { return left.Gain; }

    public void ChangeCD(int newSlot)
        requires 0 <= newSlot;
    { currentCDSlot = newSlot; }
}
```

b A .NET feature that allows associating meta-data with program elements.

owns its speakers. Due to this ownership relationship, Spec# enforces that two Stereo objects do not share their speakers and that, in general, a speaker can be modified only through its owning Stereo object. This lets a Stereo object maintain object invariants over the state of its speakers (such as the third object invariant).

Enforcing Spec# Contracts

A spectrum of possibilities is available for checking Spec# contracts. One extreme would be to verify them all statically; another would be to check them all dynamically. Either extreme is impractically expensive. The former involves a prohibitive specification and verification effort; the latter involves prohibitive runtime overhead. Instead, Spec# makes some checks mandatory; splitting them between dataflow analyses performed during compilation and runtime checks performed during execution; the rest are optionally enforced by a static program verifier.

The runtime checker is straightforward: each contract indicates some particular program points at which it

must hold; the Spec# compiler generates a runtime assertion for each, and any failure causes an exception to be thrown.

The dataflow analysis part of the Spec# compiler primarily checks three properties: The first, and most important, is enforcing the non-null type system, which can be used independently without the other kinds of contracts in the Spec# system.

In general, a type system guarantees the static type of an expression accurately describes the possible values to which the expression can evaluate at runtime. In Spec#, an expression with a non-null type can never be observed to have a value of `null`. Guaranteeing this property requires controlling both assignments and initialization.¹⁰ In particular, the type system must guarantee that a fresh object doesn't escape from its constructor before the constructor initializes all non-null fields (such as `left` and `right` in Figure 3) with non-null values. Spec# offers two solutions to this problem. One is based on a flexible placement of the `base` constructor call within a constructor body; the other caters to

legacy code by a more sophisticated dataflow analysis.¹¹

The second property is to enforce the purity of, or side-effect free, contracts; that is, they are side-effect free. Purity ensures dynamic contract checking does not interfere with the execution of the rest of the program and that contracts have a simple semantics that can be encoded in the static verifier.

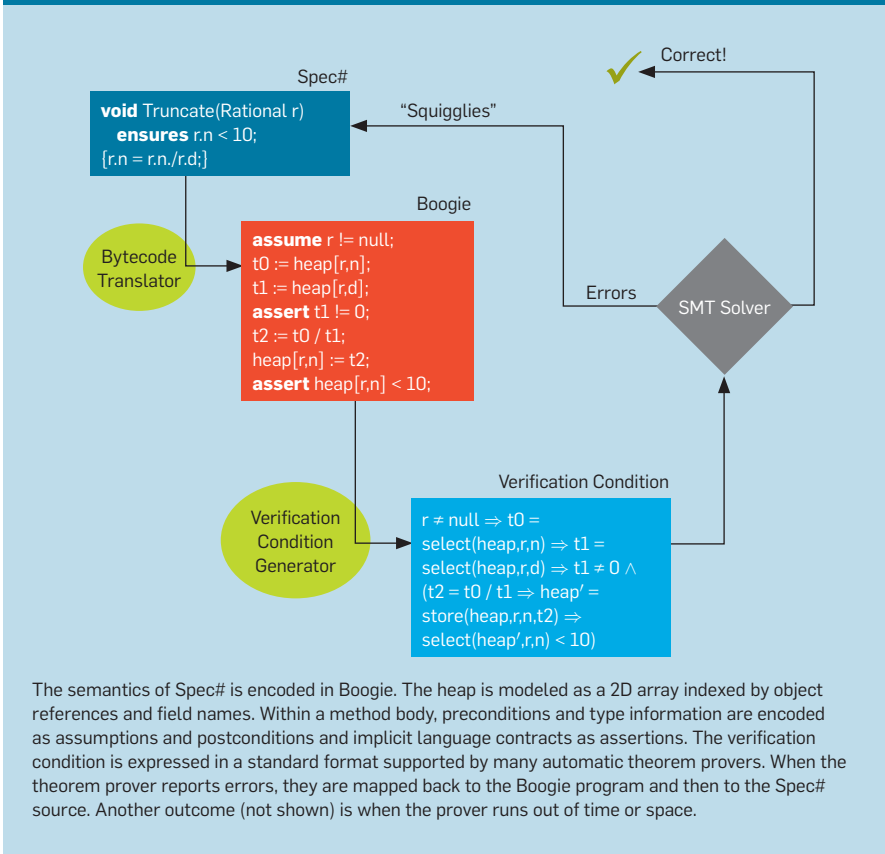
Purity could easily be enforced by forbidding all side-effecting operations (such as field updates), but doing so would be too restrictive; for instance, a method called in a specification might want to iterate over a collection. Creating and advancing an iterator are side effects; however, they are not observable when the method returns. Following JML,¹⁸ Spec# thus enforces weak purity, forbidding pure methods from changing the state of existing objects but allowing updates to objects created within the (dynamic) scope of the method's lifetime.

The final property the compiler enforces is to limit what can be mentioned in an object invariant and what things a pure method is allowed to read. These admissibility checks are crucial for sound static verification.

The static program verifier flags violations of both the explicit contracts and the implicit contracts set forth by the language semantics (such as null dereference and array index out of bounds). It checks one method at a time. If the verification fails, Spec# displays an error message, the location of the error, the trace through the method containing the error, and possibly a counterexample; Figure 2 illustrates how the IDE reports verification errors to the programmer.

The Spec# static verifier is sound but not complete; that is, it finds all errors in a program but might also warn about methods that are actually correct. Such spurious warnings are often fixed by providing more comprehensive specifications. In some cases, it may be necessary for the programmer to add an assumption to the program using the program statement `assume`. The condition `e` is blindly assumed by the static verifier but is checked at runtime. Assumptions require special attention during testing and code reviews.

Figure 4. Verification pipeline.



Verification proceeds via a series of transformations starting with the Spec# program and ending with a mathematical formula that is then input to an automated first-order theorem prover. The formula, called a verification condition, is valid if and only if there are no violations of implicit or explicit contracts (see Figure 4). The gap between the program and the formula is bridged by translating the Spec# program into a much simpler program, much as a compiler bridges the gap between source program and machine code by translating into an intermediate representation. For this purpose, we defined the intermediate verification language Boogie²; we also created a way to derive verification conditions for Boogie programs by computing their weakest preconditions. In essence, Boogie has only assignment statements, assertions, assumptions, and branches. A method call is modeled by asserting all of the method's preconditions, assigning arbitrary values to anything the method might modify (things within its frame) and assuming the method's postconditions. Boogie was designed to support more front ends than Spec#.

Note that static verification does not fully replace testing. Tests are still necessary to ensure the requirements are captured correctly, to check the properties that are not expressed by contracts, and to check properties ignored by the Spec# verifier (such as stack overflows).

Invariants

Spec# performs modular reasoning, which is to a verifier what separate compilation is to a compiler; each module is verified separately and does not need to be re-verified when the modules are combined into a whole program (see Figure 3). To reason soundly in the presence of object invariants and mutable state, Spec# uses a methodology that restricts programs and guides their use of specifications. Here, we outline this important contribution of sound modular reasoning; a full explanation is in the Spec# tutorial.²⁰

A first problem with object invariants is deciding on the program points at which they should hold. An invariant cannot always hold; it is generally



If verification ever makes it into the daily rhythm of mainstream programming, it will be through a design-time interface providing online verification.



necessary to temporarily violate an invariant with later state changes reestablishing it, as outlined in Figure 3 by method CrankItUp. It is also not possible to say an object invariant holds on method boundaries; method calls made within a method could make the object accessible outside the class while in an inconsistent state.

A second related problem is that an object invariant often depends on the state of other objects; for instance, the invariant of an aggregate object typically depends on the state of its sub-objects, as illustrated by Stereo in Figure 3. Consequently, modification of these sub-objects potentially violates the invariant of the aggregate. This situation is inescapable for any system with reusable components, so the methodology must allow it. But the verifier must ensure the aggregate object's invariants are reestablished before the aggregate relies on them again.

Spec# solves the first problem with the **expose** statement, which is similar to a lock statement in concurrent programming. The **expose** statement indicates a non-reentrant lexical region within which an object's state is vulnerable and within which the invariant may be temporarily violated. The object invariant must hold in order for the block to be entered or exited.

The second problem is solved by introducing an ownership system in which the objects of the heap are organized into a collection of tree structures. The edges of the trees indicate ownership, or an aggregate/sub-object relationship. An object invariant can depend roughly only on state contained in the subtree of which it is the root. Within an **expose** block, a method can call down in the ownership tree but not up, preventing method calls on inconsistent objects.

This is the basic approach to specifying aggregate objects in Spec#. However, many object-oriented programs not only involve hierarchical data structures but also consist of mutually referring objects (such as the subject-observer pattern or doubly linked lists). To deal with such peer relationships, the Spec# methodology uses the notion of peer consistency, or that an object and all its peers are consistent;

The Spec# Ecosystem

Spec# also relies on other projects developed within Microsoft Research:

CCI. The Microsoft Research Common Compiler Infrastructure is a set of base classes that implement common functionality needed by compilers, taking care of intermediate code generation and helping with symbol table management, metadata importing, name resolution, overload resolution, and error reporting. It also includes functionality for language integration into the Visual Studio development environment. First developed within Microsoft as part of the implementation of Omega, it has since been used for other compilers, including the Spec# compiler. A redesigned version of the core parts of CCI was released in 2009 as an open source project (<http://ccimetadate.codeplex.com>).

Boogie. Boogie¹⁹ is a verification platform consisting of an intermediate verification language and a tool that generates logical verification conditions. The Boogie language offers a level of abstraction suitable for modeling the behavior and proof obligations of a source language; for example, it supports procedures with contracts, local and global variables, structured and unstructured control flow, a polymorphic type system, and first-order mathematical definitions. These features make it convenient for verification systems to encode imperative and object-oriented programs. Verifiers built on top of Boogie translate source programs into Boogie programs and invoke the Boogie tool. Boogie computes efficient verification conditions for its input program, sends them to a theorem prover (such as the SMT solver Z3), and makes the results available to programmers and upstream tools. We initially developed Boogie within the Spec# project, but, as noted, Boogie is now employed by many program verifiers. Boogie has been an open source project (<http://boogie.codeplex.com>) since 2009.

Z3. Z3⁷ is a state-of-the-art SMT solver combining decision procedures for functions, arithmetic, and logical quantifiers. Due to its high performance, it is the default SMT solver used by the Boogie verification engine. When a proof attempt fails, Z3 returns information from which Boogie extracts the failed assertion, the trace through the method to be verified leading to the failure, and a counterexample with possible values for local variables and heap locations. This information gives programmers precise and helpful error messages; see Z3's Web site <http://research.microsoft.com/projects/z3/>.

see the Spec# tutorial²⁰ for details.

A third problem of modular reasoning is framing, which deals with what “frame” can be put around a method call to limit the effects the call might have; for instance, in method `CrankItUp`, what is the program state after the call to `left.Adjust`? A pessimistic approach is to treat the call as modifying everything in the heap, since potentially all objects in the heap are reachable from every method (such as through static fields). A better solution would be to know exactly which parts of the heap a method changes, but, in the presence of subclassing and information hiding, a method contract cannot name these parts directly. Instead, some form of abstraction is needed but one that is precise enough for the program verifier. Spec# solves this problem by again utilizing its ownership system; without an explicit specification stating otherwise (keyword `modifies`), a method may modify only the fields of the receiver and those of objects within the subtree of which the receiver is the root. Using ownership to abstract over the modifications of sub-objects is justified, be-

cause clients of an object should not be concerned with its sub-objects; for instance, clients of `Stereo` objects need to know only about the result of `Volume()`, not how the volume is stored in the sub-objects of `Stereo`.

Songs of Innocence

We began the Spec# project in 2003 as an attempt to build a comprehensive program-verification system,³ hoping to build a real system real programmers could use on real programs to perform real verification, a system the “programming masses” could use in their everyday work. Along the way, we wanted to explore and push the boundaries of specification and verification technology to get closer to realizing these aspirations.

At the time, program verification was already decades old, starting with formal underpinnings of program semantics and techniques for proving program correctness.¹⁴ Supported by mechanical-proof assistants, early program verifiers included the GYPSY system and the Stanford Pascal Verifier. Later systems, still used today, include full-featured proof assistants

like PVS and Isabelle/HOL.

Another approach to improving program quality via verification technology is extended static checking, which included checkers like ESC/Modula-3⁸ and ESC/Java.¹³ These tools have been more closely integrated into existing programming languages and value automation over expressivity or soundness. The automation is enabled by a breed of combined decision procedures that today is known as Satisfiability Modulo Theories, or SMT, solvers. To make their use easier and more cost-effective, extended static checkers were intentionally designed to be unsound; that is, they could miss certain errors.

Dynamic checking of specifications has always been done by the Eiffel programming language,²² which pioneered inclusion of contracts in object-oriented languages. The tool suite for the Java Modeling Language (JML) also included a facility for dynamic contract checking.⁴ The strong influence of both Eiffel and JML on Spec# is evident.

Our plan in 2003 targeting real programs was no doubt our single most important decision and has permeated every aspect of the Spec# design. Targeting real programs meant not designing a tool for a toy language with idealized features. In addition to learning how to handle difficult or otherwise uncomfortable language features, a benefit of this decision is the large body of programs and libraries that can be used as starting points for specification and verification. It also implied a connection with an existing language, so we built our language extensions around C# and the .NET platform. Other well-known real languages with specifications were Eiffel, Java+JML, and SPARK Ada¹; as in GYPSY and Eiffel, our extensions made specifications part of the language itself.

Our plan to build a system for real programmers immediately ruled out the possibility of exposing programmers to an interactive proof assistant. We felt that while programmers must know how to specify a program, they should not need to understand proof theory, the logical encoding of a program's semantics, or how to issue tactics to guide the proof search. Instead,

we turned to an automatic SMT solver. This is not to say that verification is fully automatic; Spec# programmers must still supply specifications, but all interaction between them and Spec#'s tools takes place in the context of the program and its specifications. The major contender here was ESC/Java and similar tools using JML specifications in Java programs.

Another important consequence of building a system for real programmers was the need for something to attract real programmers. It is a long journey indeed for programmers to arrive at the point of writing specifications that lead to effective verification. To give them immediate benefit for any specification they write, however partial, we included in Spec# dynamic checking of specifications. Throughout the project, we also worked on providing good defaults so programmers would not be unduly burdened in the most common cases.

Finally, our plan for real verification meant not compromising on soundness while also aligning with fully featured proof assistants. Sound verification of object-oriented programs does not come easily. Of the unsound features in ESC/Java, many were known to have sound solutions. But two open key areas were how to verify object invariants in the presence of subclassing and dynamically dispatched methods (giving rise to the possibility of callbacks, or situations where the caller of a method is reentered during the execution of the method it called), as well as method framing. To ensure our verifier would scale to large programs and could be applied to libraries, we also wanted to support modular verification. We began the project with an idea for a methodology that addresses these problems, providing a glimmer of hope for building a sound and modular verifier.

Though we aimed for a broad design, we initially left out several things so we could provide simpler specifications; for example, we provided no support for writing specifications for unsafe (non-type-safe) code, concurrency, higher-level aspects of closure objects, and some functional correctness concerns of algorithmic verification. The specifications focused instead on partial properties, of the kind

every programmer could write down and for which might be willing to accept the runtime overhead of dynamic checking. We subsequently added other features we left out of our initial design (such as generics).

We set out to build a programming system where both the programming language and integrated tooling support specifications. The system was intended to blend into existing practices, provide a range of assurance levels, from dynamic checking to static verification, and deliver static verification that was sound and automatic. Its success depended on answering a number of scientific questions, as well as solving non-trivial engineering concerns.

Influence

Here, we explore Spec#'s influence on researchers and language designers in academia and industry.

Scientific results. The Spec# project's main research focus has been on improving verification methodology by identifying common programming idioms and developing techniques and notations for their specification and verification. We built a state-of-the-art system and advanced the state of verification. First, the Spec# methodology supports sound modular verification of object invariants in the presence of multi-object invariants, subclassing, and reentrancy. We also worked out some of the difficulties with abstraction features (such as pure methods). Spec#'s dynamic ownership model allows programmers to express heap topologies and use them for verification. The Spec# project gained practical experience through a design of non-null types and incorporated flexible object initialization schemes. It advanced the foundations of program verification by, for instance, providing a verification-condition generator for unstructured programs. And finally, by providing IDE support and continuously running the program verifier in the background, Spec# broke new ground in how programmers work with a verifier. The scientific contributions of the Spec# project have been published in more than 30 articles.

Impact on academic research and teaching. A number of research projects build directly on the Spec# infra-

structure; for example, SpecLeuven¹⁶ is an extension of the Spec# methodology and tools to handle concurrency, using Spec#'s ownership system to enforce locking strategies. Several research groups use the Boogie verification engine developed as part of the Spec# project²; for instance, various Java/JML, bytecode/BML, and Eiffel projects use Boogie as a target for their verifiers. At the other end, Boogie's output is now also fed to interactive theorem provers. In addition, we've seen researchers encode and verify new logics, as well as verify challenging examples (such as garbage collectors).

Other projects do not use the Spec# infrastructure but seem to be influenced and inspired by the project. For example, Eiffel supports attached types, a variation of a non-null type system. JML does not include a non-null type system but offers non-null annotations, which are the default for all reference types. The idea to run a program verifier within an IDE and report verification errors just like compiler errors has been picked up by ESC/Java2, which comes with Eclipse integration. Likewise, the Rodin tool provides Eclipse integration for the Event-B tools.

Spec# has also been used to teach program verification at universities, mostly in graduate seminars. We and others have also taught Spec# in a number of summer schools, as well as at major conferences.

Impact within industry. In 2003, we hoped to convince one of the programming language teams at Microsoft to add Spec#-like features. However, influencing such a team is itself a difficult proposition, and even if we had succeeded, our single-language story did not address the fact that .NET is a multi-language platform. We also lacked support for unsafe code and for concurrency while battling a perception that verification is relevant only for safety-critical software. Even so, Spec# has influenced other projects in Microsoft Research and several Microsoft product groups. This influence can be grouped into two main categories: Boogie and Code Contracts for .NET:

Boogie. The verification engine originally developed for Spec#, called

Boogie, has become an independent project (see the sidebar “The Spec# Ecosystem”) used in other projects inside and outside Microsoft. Here are some of them:

The HAVOC tool uses Boogie to verify low-level sequential systems code written in C⁵ and has been applied to verify properties of device drivers and critical components in the Windows kernel. A version of HAVOC has also been targeted in Microsoft at finding specific errors in a very large code base of systems code.

The VCC⁶ tool built at Microsoft Research adopts Spec#’s tool chain and methodology for C code, addressing Spec#’s limits in two dimensions: full functional verification and verifying concurrent operating system code. For the latter, VCC allows two-state invariants spanning multiple objects without sacrificing thread or data modularity. VCC is being used to verify the kernel of Microsoft Hyper-V (an industrial virtualization platform), the PikeOS embedded operating system, and the LEDA data-structure library.

The type safety of the Verve operating system built at Microsoft Research has been verified; the lowest level of that verification concerns assembly code written for verification and compilation in a stylized form of the Boogie intermediate verification language.

Also at Microsoft Research, SymDiff is a project built on Boogie providing an infrastructure for building tools and techniques for statically providing feedback about program changes. It is being used to ensure “app-compat,” whereby evolving programs are checked for compatibility, or relative correctness as opposed to absolute correctness.

Code Contracts for .NET. In 2009, Spec# inspired a new project: Code Contracts for .NET. To avoid having to get programmers to adopt (and support) a new language, we introduced a library-based approach where specifications are written as method calls to the library within the actual code. Calls to the contract library, including methods like `Contract.Requires` for preconditions can be called from any .NET program. Both method contracts and object invariants are supported, though we intentionally do not (yet) offer a sound treatment for invari-

ants. Non-null types and purity checking are not supported.

Standard compilers generate the normal MSIL code for calls to contract methods, whereas the Code Contracts tools use post-build steps to extract the contracts and use them for both dynamic and static checking. Starting with .NET 4.0 in 2010, the contract library is now a part of `mscorlib`, .NET’s standard library. The associated tools are distributed through DevLabs, a Visual Studio Web site^d where early technology is made available for collecting community feedback.

Songs of Experience

Here, we reflect on our initial aspirations and design decisions:

Not a toy language. The fact that we built Spec# as a full-scale .NET language and developed a mode for it within the Visual Studio IDE has had far-reaching consequences; for example, it made the scope of the project large enough to include a wealth of scientific and engineering challenges. The project shows it is possible to build a practical verifier at this scale; given the availability of SMT solvers and verification engines like Boogie and Why,¹² the task of building a verifier is now more straightforward than it was a decade earlier.

Most important, being a full language that compiles to a common platform has increased the credibility of Spec# research, letting us approach programmers and managers, especially at Microsoft, who might not have been impressed by a one-off system. Integration into Visual Studio allowed us to perform background verification at design time, immediately indicating errors in the program text by design-time squiggles. It also allowed us to populate tool tips with contracts that boost programmer understanding of the code. A crucial consequence of the IDE integration is that it has allowed us, through live demos, to communicate the Spec# vision. Demos aside, if verification ever makes it into the daily rhythm of mainstream programming, it will be through a design-time interface providing online verification.

Having access to existing programs

has two important advantages: let research teams try out ideas and brutally reveal problems that still need solutions. It thus both validates research and guides the way to more research problems to be tackled. Seeing the results of our Spec#-related experiments over the years often forced us to support previously ignored features and alter and expand our specification methodology.

Dealing with a full language also has disadvantages. Building the prototype system takes effort but is helped by the initial enthusiasm that goes with creating a new research project. However, with most of the system in place, adding or modifying features becomes a larger effort than one would wish. We often felt we could not move as quickly as we wished; for example, adding a new syntactic feature required changes not just to the parser but often also to the rest of the compiler, the admissibility checker, the part of the system that persists and recalls contracts in compiled libraries, and the verifier. We also put a lot of effort into producing contracts for the existing .NET libraries, using a heuristic tool that mined the binaries, manually adding contracts as needed, and adding system support for them.

Our IDE integration did just enough to communicate our vision. However, our implementation is far inferior to product-quality integration, and the Spec# mode in Visual Studio is downright clunky compared to the whiz-bang C# mode. Using our own integration, we gave programmers feedback as the program is keyed in but unable to take advantage of all IDE advances (such as refactoring support) without a prohibitively large engineering investment.

If we were to do the Spec# research project again, it is not clear that extending an existing language (here C#) would be the best strategy. Not only does extending C# mean having to deal with constructs that are difficult to reason about but also presents a maintenance problem as the base language evolves; for example, migrating Spec# to extend C# versions 3 and 4 would require more development resources than our small research team has, all for the purpose of supporting features of marginal research return.

^d <http://msdn.microsoft.com/en-us/devlabs/dd491992.aspx>

In contrast, SPARK Ada was built as a subset of Ada and more readily lets a language designer pick features that mesh well with verification and trivially solve the problem of what to do as the base language evolves. Subsetting makes it more difficult to apply the verifier to legacy code.

A final point about developing a verifier in a multi-language platform is the question of where in the compilation chain to apply verification. The Spec# verifier actually starts with the MSIL bytecode the compiler produces, letting the verifier ignore syntactic variations offered by the source language (such as `for` loops versus `while` loops) and allowing the building of cross-language verifiers. But for some features, like the syntactic support for iterators in C#, it would be much easier to start with the source constructs than either verify or first reverse engineer the auxiliary classes and chopped-up method bodies the compiler emits into the bytecode. While it may be tempting to start at the source, the right thing is to start with MSIL; otherwise, every language would have to write its own verifier. Compilers should annotate the bytecode to make it easy to recover higher-level information.

Non-null types. Non-null types have proven useful and easy to use. Like other successful type features, they provide an enforceable discipline that hits a sweet spot of ruling out most programs with certain kinds of errors while allowing most programs without such errors. In our experience, programmers almost universally like the non-null types, with the exception of converting legacy code.

We began the Spec# project with reference types being possibly null by default (as in C# and Java) and requiring the type modifier `!` to express a non-null type, except for local variables where the type checker automatically inferred the non-null mode. We found that in our own code, the non-null-by-default option led to less clutter. Consequently, our suggestion to future language designers is to let possibly-null types be the option.

Our non-null type system has some holes resulting from the engineering compromises we had to make to integrate the types into an existing plat-



In our experience, programmers almost universally like the non-null types, with the exception of converting legacy code.



form. A complete system needs support from the .NET virtual machine to, say, ensure that element assignments to arrays respect the covariant arrays of .NET. Another example is that handling non-null static fields requires more control during class initialization than the .NET machine provides. We designed (sometimes complicated) workarounds for the lack of virtual-machine support but did not implement them all. We hope future execution platforms will be designed with non-null types in mind.

Our attempt at technology-transfer of non-null types into Microsoft languages also yielded a surprise. We had felt that the fruits of our non-null research were ready for prime time, but, as just described, non-null types do not reap their full benefits in a single language, instead needing platform support. Also as we described, we had better luck with technology transfer of contracts.

Dynamic and static checking. Another major goal of our initial design was support for both dynamic and static checking of specifications. Such support has several advantages; most important, it immediately rewards programmers for writing specifications, because they turn into useful runtime assertions. Each specification added in that way helps reduce the additional cost of writing provable specifications at a time when the program might be verified. Another advantage of using specifications for both dynamic and static checking is that programmers only need to learn one specification language.

Whenever dynamic checking would be too expensive (such as in the enforcement of method frames) our principle was to drop the dynamic check. If dynamic checks are a subset of the static checks, then such a design has the nice property that any program that is statically verified will run without dynamic violations of specifications. An important exception to this subsetting is the `assume` statement, the sole purpose of which is to trade a dynamic check for an assumption by the static verifier.

A point that often comes up in discussions with colleagues is the prospect of omitting the dynamic checks of those specifications that have been

statically verified. We never got around to trying such an optimization for Spec#. However, we offer a word of caution to others who might consider doing so: Preserving the soundness of such optimizations is difficult unless the whole program is forced to undergo verification. Such a requirement on the whole program is practically impossible in the .NET environment, where the interoperating languages have neither static nor dynamic checking.

Verifying loops. A familiar issue with static verification is the need for loop invariants, which are analogous to the inductive hypotheses used to prove theorems inductively in mathematics. Whereas a tool like ESC/Java avoids this issue by checking only a bounded number of iterations of each loop, Spec# verifies all iterations, but comes at a cost. We have found this cost to be moderately low, explaining it as follows: The effective loop invariant draws from three sources: One is that the Spec# program verifier automatically infers simple loop invariants. A second is loop-modification inference,⁸ whereby Spec# enforces the method frame on every heap update, so the **modifies** clause of the enclosing method can be incorporated as an automatic part of the loop invariant. This is the “biggest” part of the effective loop invariant and also the part no user would want to supply explicitly. The third source is user-supplied loop invariants. Due to the first two sources, many loops require no user-supplied loop invariants at all, especially for methods with no postconditions. As programmers and tools take steps toward functional-correctness verification, the need for user-supplied loop invariants is likely to increase.

Methodology. The Spec# methodology led us to the first implementation of a sound modular approach to specifying and verifying object invariants and method frames. Compared to earlier solutions,²³ the Spec# methodology is better suited for automatic verification using SMT solvers. Since 2003, other researchers have designed alternative methodologies.^{17,24} The Spec# methodology has been streamlined for some common object-oriented patterns (such as aggregate objects), lending itself to concise specifications of



We set out to build a programming system where both the programming language and integrated tooling support specifications.



programs that fall within those common patterns. However, for programs using more complicated patterns, the methodology can be too restrictive; for example, it caused us to lose the interest of one programmer at Microsoft who was avidly trying to verify a large body of code.

The learning curve of the methodology has been steeper for programmers than we would have liked, and non-experts of the methodology sometimes have problems knowing what to do in response to certain error messages. The fact that we constantly changed the methodology to improve it, and along with it the terminology we used, complicated programmers' understanding. We hope the 2010 tutorial²⁰ has mitigated this problem.

More. We'd like to make a few more remarks about our Spec# experience: First, we set out to build a sound verification system. While we found sound solutions to fundamental problems of modular verification of object-oriented programs, our implementation is not perfect, including several unimplemented features and other unfound errors and semantic encoding.

Second, the best and most far-reaching single design decision we made in implementing the Spec# verifier was to introduce the intermediate language Boogie between the Spec# program and the formulas sent to the theorem prover. Boogie permits manual authoring, as well as automatic translation.² This extra layer of indirection allowed us to investigate many alternative design decisions in a lightweight fashion by hand-modifying the translated Spec# program. Another benefit of having this important separation of concerns is the use of Boogie as the backend for other programming languages and verification systems and as the frontend for different theorem provers.

Third, a conclusion we draw from watching programmers beginning to use Spec# is that they really do appreciate contracts, which are not just an esoteric feature prescribed by “Dijkstra clones.” Unfortunately, we have also seen them develop unreasonable expectations for how easy it will be to statically verify their programs using contracts. Post-installation depression might then set in as they have

difficulty trying to verify their own programs.

Fourth, from a research standpoint, having source syntax for specification constructs lets program text be concise and usefully descriptive, making clear the important concepts. However, from an adoption standpoint, this approach involves two problems: One is that the engineering overhead associated with being a superset language is a high price for the developers of the verification system to pay; the other is that we want specifications to be adopted in the platform, not just in a single language. Therefore, we gradually steered toward a language-independent solution by providing the specification constructs via the Code Contracts library, making it possible for individual languages to consider adding convenient source syntax.

Conclusion

Following the Spec# project, the Verified Software Initiative¹⁵ was formed in 2005 to encourage the verification community to work toward larger projects, addressing larger risk, and taking a long-term view of program verification. All our work on Spec# fits this initiative.

Spec# has evolved from a single-language vision into four ongoing technical themes:

One is the Spec# project itself, which entered a new phase in 2009, hoping, through a new open-source release^e, to see continued improvement in the Spec# programming system from a larger community and also more use of Spec# in teaching, especially in light of the 2010 comprehensive tutorial.²⁰

A second theme is the Boogie language and verification engine, providing an infrastructure used as an abstraction layer in several verification projects.^f

A third theme is a new strand of research attempting automatic functional-correctness verification. Using a variation of the Spec# methodology and the Boogie intermediate verification language, VCC is an example of such a project, and Hyper-V's experience with it has been positive.

A fourth theme aims at the mass adoption of specifications in programming. The language-independent Code Contracts project bypasses the problem of trying to get a single new language to be accepted as the standard programming language for all programmers by putting specification facilities into the underlying platform. We hope that since all .NET programmers can make use of the associated tools for runtime checking, static checking, automatic documentation generation, and intelligent programmer assistants, this will lead to an improved general software-engineering process.

These themes continue to push frontiers in the quest for verified software.

Acknowledgments

Spec# owes a great deal to the research interns who helped bring it into existence and to the feedback from programmers who have used it. We also express our thanks to the early reviewers and anonymous referees of this article. C

References

1. Barnes, J. *High-Integrity Software: The SPARK Approach to Safety and Security*. Addison-Wesley, 2003.
2. Barnett, M., Chang, B.-Y. E., DeLine, R., Jacobs, B., and Leino, K.R.M. Boogie: A modular reusable verifier for object-oriented programs. In *Proceedings of Formal Methods for Components and Objects*, Volume 4111 LNCS (Amsterdam, The Netherlands, Nov. 1–4, 2005). Springer, 2006, 364–387.
3. Barnett, M., Leino, K.R.M., and Schulte, W. The Spec# programming system: An overview. In *Proceedings of Construction and Analysis of Safe, Secure, and Interoperable Smart Devices*, Volume 3362 LNCS (Nice, France, Mar. 8–11). Springer, 2005, 49–69.
4. Burdy, L., Cheon, Y., Cok, D. R., Ernst, M. D., Kiniry, J. R., Leavens, G. T., Leino, K. R. M., and Poll, E. An overview of JML tools and applications. *Electronic Notes in Theoretical Computer Science* 80 (2003), 212–232.
5. Chatterjee, S., Lahiri, S. K., Qadeer, S., and Rakamaric, Z. A reachability predicate for analyzing low-level software. In *Proceedings of Tools and Algorithms for Construction and Analysis of Systems*, Volume 4424 LNCS (Braga, Portugal, Mar. 24–Apr. 1). Springer, 2007, 19–33.
6. Cohen, E., Dahlweid, M., Hillebrand, M., Leinenbach, D., Moskal, M., Santen, T., Schulte, W., and Tobies, S. VCC: A practical system for verifying concurrent C. In *Proceedings of Theorem Proving in Higher Order Logics*, Volume 5674 LNCS (Munich, Aug. 17–20). Springer, 2009, 23–42.
7. de Moura, L. and Bjørner, N. Z3: An efficient SMT solver. In *Proceedings of Tools and Algorithms for Construction and Analysis of Systems*, Volume 4963 LNCS (Budapest, Mar. 29–Apr. 6). Springer, 2007, 337–340.
8. Detlefs, D. L., Leino, K. R. M., Nelson, G., and Saxe, J. B. *Extended Static Checking Research Report 159*. Compaq Systems Research Center, Palo Alto, CA, 1998.
9. Dhara, K. K. and Leavens, G. T. Forcing behavioral subtyping through specification inheritance. In *Proceedings of the International Conference on Software Engineering* (Berlin, Mar. 25–30). IEEE Computer Society Press, 1996, 258–267.
10. Fähndrich, M. and Leino, K. R. M. Declaring and checking non-null types in an object-oriented language. In *Proceedings of the ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications* (Anaheim, CA, Oct. 26–30). ACM Press, New York, 2003, 302–312.
11. Fähndrich, M. and Xia, S. Establishing object invariants with delayed types. In *Proceedings of the ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications* (Montréal, Oct. 21–25). ACM Press, New York, 2007, 337–350.
12. Filiâtre, J.-C. and Marché, C. The Why/Krakatoa/Caduceus platform for deductive program verification. In *Proceedings of Computer Aided Verification*, Volume 4590 LNCS (Berlin, July 3–7). Springer, 2007, 173–177.
13. Flanagan, C., Leino, K. R. M., Lillibridge, M., Nelson, G., Saxe, J. B., and Stata, R. Extended static checking for Java. In *Proceedings of ACM SIGPLAN Conference on Programming Language Design and Implementation* (Berlin, June 17–19). ACM Press, New York, 2002, 234–245.
14. Floyd, R. W. Assigning meanings to programs. In *Mathematical Aspects of Computer Science*, Volume 19 of *Proceedings of Symposia in Applied Mathematics*. American Mathematical Society, 1967, 19–32.
15. Hoare, C., Misra, J., Leavens, G. T., and Shankar, N. The verified software initiative: A manifesto. *ACM Computing Surveys* 41, 4 (2009), 1–8.
16. Jacobs, B., Leino, K. R. M., Piessens, F., Smans, J., and Schulte, W. A programming model for concurrent object-oriented programs. *ACM Transactions on Programming Languages and Systems* 31, 1 (2008), 1–48.
17. Kassios, I. T. Dynamic frames: Support for framing, dependencies and sharing without restrictions. In *Proceedings of World Congress on Formal Methods*, Volume 4085 LNCS (Hamilton, Canada, Aug. 21–27). Springer, 2006, 268–283.
18. Leavens, G. T., Baker, A. L., and Ruby, C. Preliminary design of JML: A behavioral interface specification language for Java. *ACM SIGSOFT Software Engineering Notes* 31, 3 (2006), 1–38.
19. Leino, K. R. M. *This is Boogie 2*. Microsoft Research Technical Report (June 2008).
20. Leino, K. R. M. and Müller, P. Using the Spec# language, methodology, and tools to write bug-free programs. *Advanced Lectures on Software Engineering: LASER Summer School 2007/2008*, Volume 6029 LNCS. Springer, 2010.
21. Liskov, B. H. and Wing, J. M. A behavioral notion of subtyping. *ACM Transactions on Programming Languages and Systems* 16, 6 (1994), 1811–1841.
22. Meyer, B. *Object-oriented Software Construction*. Prentice-Hall, 1988.
23. Müller, P. *Modular Specification and Verification of Object-Oriented Programs*, Volume 2262 LNCS. Springer, 2002.
24. Parkinson, M. J. and Bierman, G. M. Separation logic and abstraction. In *Proceedings of the Symposium of Principles of Programming Languages* (Long Beach, CA, Jan. 12–14). ACM Press, New York, 2005, 247–258.

Mike Barnett (mbarnett@microsoft.com) is a principal research software design engineer in the RiSE group at Microsoft Research, Redmond, WA.

Manuel Fähndrich (maf@microsoft.com) is a senior researcher in the RiSE group at Microsoft Research, Redmond, WA.

K. Rustan M. Leino (leino@microsoft.com) is a principal researcher in the RiSE group at Microsoft Research, Redmond, WA.

Peter Müller (peter.mueller@inf.ethz.ch) is a professor in the Department of Computer Science at the Swiss Federal Institute of Technology (ETH), Zürich, Switzerland.

Wolfram Schulte (schulte@microsoft.com) is the research area manager leading the RiSE group at Microsoft Research, Redmond, WA.

Herman Venter (hermanv@microsoft.com) is a principal research software design engineer in the RiSE group at Microsoft Research, Redmond, WA.

© 2011 ACM 0001-0782/11/06 \$10.00

e <http://specsharp.codeplex.com>

f <http://boogie.codeplex.com>

DOI:10.1145/1953122.1953146

The roots of Google's PageRank can be traced back to several early, and equally remarkable, ranking techniques.

BY MASSIMO FRANCESCHET

PageRank: Standing on the Shoulders of Giants

PAGERANK³ IS A Web page ranking technique that has been a fundamental ingredient in the development and success of the Google search engine. The method is still one of the many signals Google uses to determine which pages are most important.^a The main idea behind PageRank is to determine the importance of a Web page in terms of the importance assigned to the pages hyperlinking to it. In fact, this thesis is not new, and has been previously successfully exploited in different contexts. We review the PageRank method and link it to some renowned previous techniques that we have found in the fields of Web information retrieval, bibliometrics, sociometry, and econometrics.

In 1945 Vannevar Bush wrote a celebrated article in *The Atlantic Monthly* entitled “As We May Think” describing a futuristic device he called Memex.⁵ Bush wrote:

a <http://www.google.com/corporate/tech.html>

Wholly new forms of encyclopedias will appear, ready made with a mesh of associative trails running through them, ready to be dropped into the Memex and there amplified.

Bush's prediction came true in 1989, when Tim Berners-Lee proposed the *Hypertext Markup Language* (HTML) to keep track of experimental data at the European Organization for Nuclear Research (CERN). In the original far-sighted proposal in which Berners-Lee attempts to persuade CERN management to adopt the new global hypertext system we can read the following paragraph^b:

We should work toward a universal linked information system, in which generality and portability are more important than fancy graphics techniques and complex extra facilities. The aim would be to allow a place to be found for any information or reference which one felt was important, and a way of finding it afterwards. The result should be sufficiently attractive to use that the information contained would grow past a critical threshold.

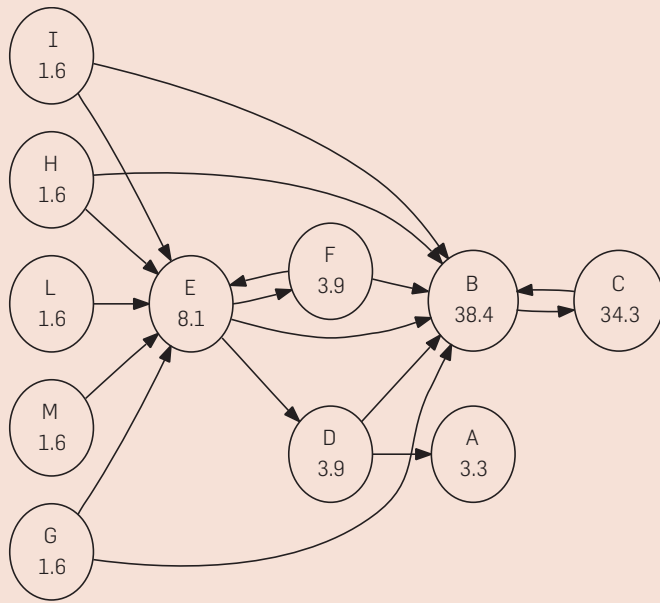
b <http://www.w3.org/History/1989/proposal.html>

» key insights

- Great, pioneering research paved the way for PageRank, the well-known algorithm of Brin and Page that became the basis for the Google search engine and that is still one of the many signals Google uses to determine which pages are most important.
- This article examines the mathematical predecessors to PageRank in ostensibly disparate disciplines such as Web information retrieval, bibliometrics, sociometry, and econometrics.
- In tracing these early efforts we find the fundamental idea behind PageRank—its circular thesis that a Web page is important if it is pointed to by other important pages—was not entirely new, and had basis in work going back decades.



Figure 1. A PageRank instance with solution. Each node is labeled with its PageRank score. Scores have been normalized to sum to 100. We assumed $\alpha = 0.85$.



As we all know, the proposal was accepted and later implemented in a *mesh*—this was the only name that Berners-Lee originally used to describe the Web—of interconnected documents that rapidly grew beyond the CERN threshold, as Berners-Lee anticipated, and became the World Wide Web.

Today, the Web is a huge, dynamic, self-organized, and hyperlinked data source, very different from traditional document collections that are nonlinked, mostly static, centrally collected and organized by specialists. These features make Web information retrieval quite different from traditional information retrieval and call for new search abilities, like automatic crawling and indexing of the Web. Moreover, early search engines ranked responses using only a *content score*, which measures the similarity between the page and the query. One simple example is just a count of the number of times the query words occur on the page, or perhaps a weighted count with more weight on title words. These traditional query-dependent techniques suffered under the gigantic size of the Web and the death grip of spammers.

In 1998, Sergey Brin and Larry Page revolutionized the field of Web information retrieval by introducing the

notion of an *importance score*, which gauges the status of a page, independently from the user query, by analyzing the topology of the Web graph. The method was implemented in the famous PageRank algorithm and both the traditional content score and the new importance score were efficiently combined in a new search engine named Google.

Ranking Web Pages using PageRank

We briefly recall how the PageRank method works keeping the mathematical machinery to the minimum. Interested readers can more thoroughly investigate the topic in a recent book by Langville and Meyer that elegantly describes the science of search engine rankings in a rigorous yet playful style.¹⁶

We start by providing an intuitive interpretation of PageRank in terms of random walks on graphs.²² The Web is viewed as a directed graph of pages connected by hyperlinks. A *random surfer* starts from an arbitrary page and simply keeps clicking on successive links at random, bouncing from page to page. The PageRank value of a page corresponds to the relative frequency the random surfer visits that page, assuming that the surfer goes on infinitely. The more time spent by the random surfer

on a page, the higher the PageRank importance of the page.

A little more formally, the method can be described as follows. Let us denote by q_i the number of distinct outgoing (hyper)links of page i . Let $H = (h_{i,j})$ be a square matrix of size equal to the number n of Web pages such that $h_{i,j} = 1/q_i$ if there exists a link from page i to page j and $h_{i,j} = 0$ otherwise. The value $h_{i,j}$ can be interpreted as the probability that the random surfer moves from page i to page j by clicking on one of the distinct links of page i . The PageRank π_j of page j is *recursively* defined as

$$\pi_j = \sum_i \pi_i h_{i,j}$$

or, in matrix notation, $\pi = \pi H$. Hence, the PageRank of page j is the sum of the PageRank scores of pages i linking to j , weighted by the probability of going from i to j . In words, the PageRank thesis reads as follows:

A Web page is important if it is pointed to by other important pages.

There are in fact three distinct factors that determine the PageRank of a page: the number of links it receives; the link propensity, that is, the number of outgoing links, of the linking pages; and the PageRank of the linking pages. The first factor is not surprising: the more links a page receives, the more important it is perceived. Reasonably, the link value depreciates proportionally to the number of links given out by a page: endorsements coming from parsimonious pages are worthier than those emanated by spendthrift ones. Finally, not all pages are created equal: links from important pages are more valuable than those from obscure ones.

Unfortunately, this ideal model has two problems that prevent the solution of the system. The first one is due to the presence of *dangling nodes*, that are pages with no forward links.^c These pages capture the random surfer indefinitely. Notice that a dangling node corresponds to a row in matrix H with all entries equal to 0. To tackle

^c The term *dangling* refers to the fact that many dangling nodes are in fact pendent Web pages found by the crawling spiders but whose links have not been yet explored.

the problem of dangling nodes, the corresponding rows in H are replaced by the uniform probability vector $u = 1/ne$, where e is a vector of length n with all components equal to 1. Alternatively, one may use any fixed probability vector in place of u . This means the random surfer escapes from the dangling page by jumping to a randomly chosen page. We call S the resulting matrix.

The second problem with the ideal model is that the surfer can get trapped into a *bucket* of the Web graph, which is a reachable strongly connected component without outgoing edges toward the rest of the graph. The solution proposed by Brin and Page is to replace matrix S by the *Google matrix*

$$G = \alpha S + (1 - \alpha) E$$

where E is the *teleportation matrix* with identical rows each equal to the uniform probability vector u , and α is a free parameter of the algorithm often called the *damping factor*. Alternatively, a fixed *personalization* probability vector v can be used in place on u . In particular, the personalization vector can be exploited to bias the result of the method toward certain topics. The interpretation of the new system is that, with probability α the random surfer moves forward by following links, and, with the complementary probability $1 - \alpha$ the surfer gets bored of following links and enters a new destination in the browser's URL line, possibly unrelated to the current page. The surfer is hence teleported, like a Star Trek character, to that page, even if there exists no link connecting the current and the destination pages in the Web universe. The inventors of PageRank propose to set the damping factor $\alpha = 0.85$, meaning that after about five link clicks the random surfer chooses a random page.

The PageRank vector is then defined as the solution of equation:

$$\pi = \pi G \quad (1)$$

An example is provided in Figure 1. Node A is a dangling node, while nodes B and C form a bucket. Notice the dynamics of the method: page C receives just one link but from the most

important page B; its importance is much higher than that of page E, which receives many more links, but from anonymous pages. Pages G, H, I, L, and M do not receive endorsements; their scores correspond to the minimum amount of status of each page.

Typically, the normalization condition $\sum_i \pi_i = 1$ is also added. In this case, Equation 1 becomes $\pi = \alpha \pi S + (1 - \alpha)u$. The latter distinguishes two factors contributing to the PageRank vector: an *endogenous* factor equal to πS , which takes into consideration the real topology of the Web graph, and an *exogenous* factor equal to the uniform probability vector u , which can be interpreted as a minimal amount of status assigned to each page independently of the hyperlink graph. The parameter α balances between these two factors.

Computing the PageRank Vector

Does Equation 1 have a solution? Is the solution unique? Can we efficiently compute it? The success of the PageRank method rests on the answers to these queries. Luckily, all these questions have nice answers.

Thanks to the dangling nodes patch, matrix S is a stochastic matrix,^d and clearly the teleportation matrix E is also stochastic. It follows that G is stochastic as well, since it is defined as a convex combination of stochastic matrices S and E . It is easy to show that, if G is stochastic, Equation 1 has always at least one solution. Hence, we have got at least one PageRank vector. Having two independent PageRank vectors, however, would be already too much: which one should we use to rank Web pages? Here, a fundamental result of algebra comes to the rescue: *Perron–Frobenius theorem*.^{7, 24} It states that, if A is an irreducible^e nonnegative square matrix, then there exists a unique vector x , called the Perron vector, such that $xA = rx$, $x > 0$, and $\sum_i x_i = 1$, where r is the maximum eigenvalue of A in absolute value, that algebraists call the *spectral radius* of A . The Perron vector is the left *dominant eigenvector* of A , that is, the left eigenvector asso-

ciated with the largest eigenvalue in magnitude.

The matrix S is most likely reducible, since experiments have shown that the Web has a bow-tie structure fragmented into four main continents that are not mutually reachable, as first observed by Broder et al.⁴ Thanks to the teleportation trick, however, the graph of matrix G is strongly connected. Hence G is irreducible and Perron–Frobenius theorem applies.^f Therefore, a positive PageRank vector exists and is furthermore unique.

Interestingly, we can arrive at the same result using *Markov theory*.²⁰ The above described random walk on the Web graph, modified with the teleportation jumps, naturally induces a finite-state *Markov chain*, whose transition matrix is the stochastic matrix G . Since G is irreducible, the chain has a unique *stationary distribution* corresponding to the PageRank vector.

A last crucial question remains: can we efficiently compute the PageRank vector? The success of PageRank is largely due to the existence of a fast method to compute its values: the *power method*, a simple iteration method to find the dominant eigenpair

^f Since G is stochastic, its spectral radius is 1.

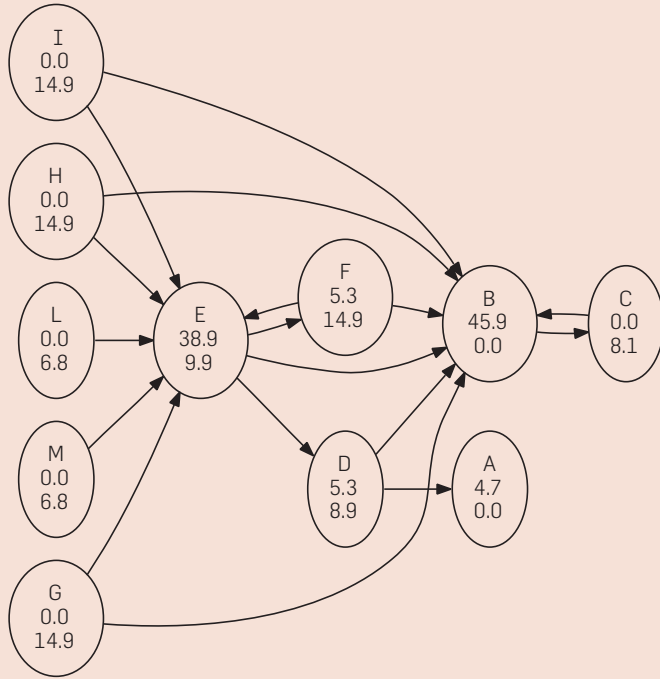
Table 1. PageRank history

Year	Author	Contribution
1906	Markov	Markov theory ²⁰
1907	Perron	Perron theorem ²⁴
1912	Frobenius	Perron–Frobenius theorem ⁷
1929	von Mises and Pollaczek-Geiringer	Power method ³¹
1941	Leontief	Econometric model ¹⁸
1949	Seeley	Sociometric model ²⁹
1952	Wei	Sport ranking model ³²
1953	Katz	Sociometric model ¹¹
1965	Hubbell	Sociometric model ¹⁰
1976	Pinski and Narin	Bibliometric model ²⁶
1998	Kleinberg	HITS ¹⁴
1998	Brin and Page	PageRank ³

^d This simply means that all rows sum up to 1.

^e A matrix is irreducible if and only if the directed graph associated with it is strongly connected, that is, for every pair i and j of graph nodes there are paths leading from i to j and from j to i .

Figure 2. A HITS instance with solution (compare with PageRank scores in Figure 1). Each node is labeled with its authority (top) and hub (bottom) scores. Scores have been normalized to sum to 100. The dominant eigenvalue for both authority and hub matrices is 10.7.



of a matrix developed by von Mises and Pollaczek-Geiringer.³¹ It works as follows on the Google matrix G . Let $\pi^{(0)} = u = 1/ne$. Repeatedly compute $\pi^{(k+1)} = \pi^{(k)} G$ until $\|\pi^{(k+1)} - \pi^{(k)}\| < \epsilon$, where $\|\cdot\|$ measures the distance between the two successive PageRank vectors and ϵ is the desired precision.

The convergence rate of the power method is approximately the rate at which α^k approaches to 0: the closer α to unity, the lower the convergence speed of the power method. If, for instance, $\alpha = 0.85$, as many as 43 iterations are sufficient to gain 3 digits of accuracy, and 142 iterations are enough for 10 digits of accuracy. Notice that the power method applied to matrix G can be easily expressed in terms of matrix H , which, unlike G , is a very sparse matrix that can be stored using a linear amount of memory with respect to the size of the Web.

Standing on the Shoulders of Giants

Dwarfs standing on the shoulders of giants is a Western metaphor meaning “One who develops future intellectual pursuits by understanding the research and works created by notable

thinkers of the past.”⁸ The metaphor was famously uttered by Isaac Newton: “If I have seen a little further it is by standing on the shoulders of Giants.” Moreover, “Stand on the shoulders of giants” is Google Scholar’s motto: “the phrase is our acknowledgement that much of scholarly research involves building on what others have already discovered.”

There are many giants upon whose shoulders PageRank firmly stands: Markov,²⁰ Perron,²⁴ Frobenius,⁷ von Mises and Pollaczek-Geiringer³¹ provided at the beginning of the 1900s the necessary mathematical machinery to investigate and effectively solve the PageRank problem. Moreover, the circular PageRank thesis has been previously exploited in different contexts, including Web information retrieval, bibliometrics, sociometry, and econometrics. In the following, we review these contributions and link them to the PageRank method. Table 1 shows a brief summary of PageRank history. All the ranking techniques surveyed in this paper have been implemented in R²⁷

g From the Wikipedia page for *Standing on the shoulders of giants*.

and the code is freely available at the author’s Web page.

Hubs and Authorities on the Web

Hypertext Induced Topic Search (HITS) is a Web page ranking method proposed by Kleinberg.^{14,15} The connections between HITS and PageRank are striking. Despite the close conceptual, temporal, and even geographical proximity of the two approaches, it appears that HITS and PageRank have been developed independently. In fact, both papers presenting PageRank³ and HITS¹⁵ are today citational blockbusters: the PageRank article collected 6,167 citations, while the HITS paper has been cited 4,617 times.^h

HITS thinks of Web pages as *authorities* and *hubs*. HITS circular thesis reads as follows:

Good authorities are pages that are pointed to by good hubs and good hubs are pages that point to good authorities.

Let $L = (l_{i,j})$ be the adjacency matrix of the Web graph, i.e., $l_{i,j} = 1$ if page i links to page j and $l_{i,j} = 0$ otherwise. We denote with L^T the transpose of L . HITS defines a pair of recursive equations as follows, where x is the authority vector containing the authority scores and y is the hub vector containing the hub scores:

$$\begin{aligned} x^{(k)} &= L^T y^{(k-1)} \\ y^{(k)} &= L x^{(k)} \end{aligned} \quad (2)$$

where $k \geq 1$ and $y^{(0)} = e$, the vector of all ones. The first equation tells us that authoritative pages are those pointed to by good hub pages, while the second equation claims that good hubs are pages that point to authoritative pages. Notice that Equation 2 is equivalent to

$$\begin{aligned} x^{(k)} &= L^T L x^{(k-1)} \\ y^{(k)} &= L L^T y^{(k-1)} \end{aligned} \quad (3)$$

It follows that the authority vector x is the dominant right eigenvector of the authority matrix $A = L^T L$, and the hub vector y is the dominant right eigenvector of the hub matrix $H = L L^T$. This is very similar to the PageRank method, except

h Source: Google Scholar on February 5, 2010.

the use of the authority and hub matrices instead of the Google matrix.

To compute the dominant eigenpair (eigenvector and eigenvalue) of the authority matrix we can again exploit the power method as follows: let $x^{(0)} = e$. Repeatedly compute $x^{(k)} = Ax^{(k-1)}$ and normalize $x^{(k)} = x^{(k)} / m(x^{(k)})$, where $m(x^{(k)})$ is the signed component of maximal magnitude, until the desired precision is achieved. It follows that $x^{(k)}$ converges to the dominant eigenvector x (the authority vector) and $m(x^{(k)})$ converges to the dominant eigenvalue (the spectral radius, which is not necessarily 1). The hub vector y is then given by $y = Lx$. While the convergence of the power method is guaranteed, the computed solution is not necessarily unique, since the authority and hub matrices are not necessarily irreducible. A modification similar to the teleportation trick used for the PageRank method can be applied to HITS to recover the uniqueness of the solution.³⁵

An example of HITS is given in Figure 2. We stress the difference among importance, as computed by PageRank, and authority and hubness, as computed by HITS. Page B is both important and authoritative, but it is not a good hub. Page C is important but by no means authoritative. Pages G, H, and I are neither important nor authoritative, but they are the best hubs of the network, since they point to good authorities only. Notice that the hub score of B is 0 although B has one outgoing edge; unfortunately for B, the only page C linked by B has no authority. Similarly, C has no authority because it is pointed to only by B, whose hub score is zero. This shows the difference between indegree and authority, as well as between outdegree and hubness. Finally, we observe that nodes with null authority scores (respectively, null hub scores) correspond to isolated nodes in the graph whose adjacency matrix is the authority matrix A (respectively, the hub matrix H).

An advantage of HITS with respect to PageRank is that it provides two scores at the price of one. The user is hence provided with two rankings: the most authoritative pages about the research topic, which can be exploited to investigate in depth a research subject, and the most hubby pages, which correspond to portal pages linking to the research



The connections between HITS and PageRank are striking. Despite the close conceptual, temporal, and even geographical proximity of the two approaches, it appears that HITS and PageRank have been developed independently.



topic from which a broad search can be started. A disadvantage of HITS is the higher susceptibility of the method to spamming: while it is difficult to add incoming links to our favorite page, the addition of outgoing links is much easier. This leads to the possibility of purposely inflating the hub score of a page, indirectly influencing also the authority scores of the pointed pages.

HITS is related to a matrix factorization technique known as *singular value decomposition*.⁶ According to this technique, the adjacency matrix L can be written as the matrix product USV^T , where the columns of U , called left-singular vectors, are the orthonormal eigenvectors of the hub matrix LL^T , the columns of V , called right-singular vectors, are the orthonormal eigenvectors of the authority matrix L^TL , and S is a diagonal matrix whose diagonal elements, called singular values, correspond to the square roots of the eigenvalues of the hub matrix (or, equivalently, of the authority matrix). It follows that the HITS authority and hub vectors correspond, respectively, to the right- and left-singular vectors associated with the highest singular value of L .

HITS also has a connection to bibliometrics.⁶ Two typical bibliometric methods to identify similar publications are *co-citation*, in which publications are related when they are cited by the same papers, and *co-reference*, in which papers are related when they cite the same papers. The authority matrix is a co-citation matrix and the hub matrix is a co-reference matrix. Indeed, since $A = L^TL$, the element $a_{i,j}$ of the authority matrix contains the number of times pages i and j are both linked by a third page ($a_{i,j}$ is the number of inlinks of i). Moreover, since $H = LL^T$, the element $h_{i,j}$ of the hub matrix contains the number of times both pages i and j link to a third page ($h_{i,j}$ is the number of outlinks of i). Hence, good authorities are pages that are frequently co-cited with other good authorities, and good hubs are pages that frequently co-reference with other good hubs.

A following algorithm that incorporates ideas from both PageRank and HITS is SALSA¹⁷: like HITS, SALSA computes both authority and hub scores, and like PageRank, these scores are obtained from Markov chains.

Bibliometrics

Bibliometrics, also known as scientometrics, is the quantitative study of the process of scholarly publication of research achievements. The most mundane aspect of this branch of information and library science is the design and application of *bibliometric indicators* to determine the influence of bibliometric units like scholars and academic journals. The Impact Factor is, undoubtedly, the most popular and controversial journal bibliometric indicator available at the moment. It is

defined, for a given journal and a fixed year, as the mean number of citations in the year to papers published in the two previous years. It has been proposed in 1963 by Eugene Garfield, the founder of the Institute for Scientific Information (ISI), working together with Irv Sher.⁸ Journal Impact Factors are currently published in the popular Journal Citation Reports by Thomson-Reuters, the new owner of the ISI.

The Impact Factor does not take into account the importance of the citing journals: citations from highly

reputed journals are weighted as those from obscure journals. In 1976 Gabriel Pinski and Francis Narin developed an innovative journal ranking method.²⁶ The method measures the influence of a journal in terms of the influence of the citing journals. The Pinski and Narin thesis is:

A journal is influential if it is cited by other influential journals.

This is the same circular thesis of the PageRank method. Given a source time window T_1 and a previous target time window T_2 , the journal citation system can be viewed as a *weighted* directed graph in which nodes are journals and there is an edge from journal i to journal j if there is some article published in i during T_1 that cites an article published in j during T_2 . The edge is weighted with the number $c_{i,j}$ of such citations from i to j . Let $c_i = \sum_j c_{i,j}$ be the total number of cited references of journal i .

In the method described by Pinski and Narin, a citation matrix $H = (h_{i,j})$ is constructed such that $h_{i,j} = c_{i,j}/c_j$. The coefficient $h_{i,j}$ is the amount of citations received by journal j from journal i per reference given out by journal j . For each journal an *influence score* is determined which measures the relative journal performance per given reference. The influence score π_j of journal j is defined as

$$\pi_j = \sum_i \pi_i \frac{c_{i,j}}{c_j} = \sum_i \pi_i h_{i,j}$$

or, in matrix notation:

$$\pi = \pi H \quad (4)$$

Hence, journals j with a large *total influence* $\pi_j c_j$ are those that receive significant endorsements from influential journals. Notice that the influence per reference score π_j of a journal j is a size-independent measure, since the formula normalizes by the number of cited references c_j contained in articles of the journal, which is an estimation of the size of the journal. Moreover, the normalization neutralizes the effect of journal self-citations, which are citations between articles in the same journal. These citations are indeed counted both at the numerator and at

Figure 3. An instance with solution of the journal ranking method proposed by Pinski and Narin. Nodes are labeled with influence scores and edges with the citation flow between journals. Scores have been normalized to sum to 100.

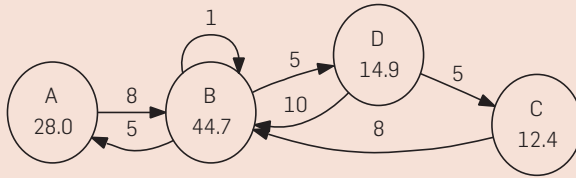


Figure 4. An example of the Katz model using two attenuation factors: $\alpha = 0.9$ and $\alpha = 0.1$ (the spectral radius of the adjacency matrix L is 1). Each node is labeled with the Katz score corresponding to $\alpha = 0.9$ (top) and $\alpha = 0.1$ (bottom). Scores have been normalized to sum to 100.

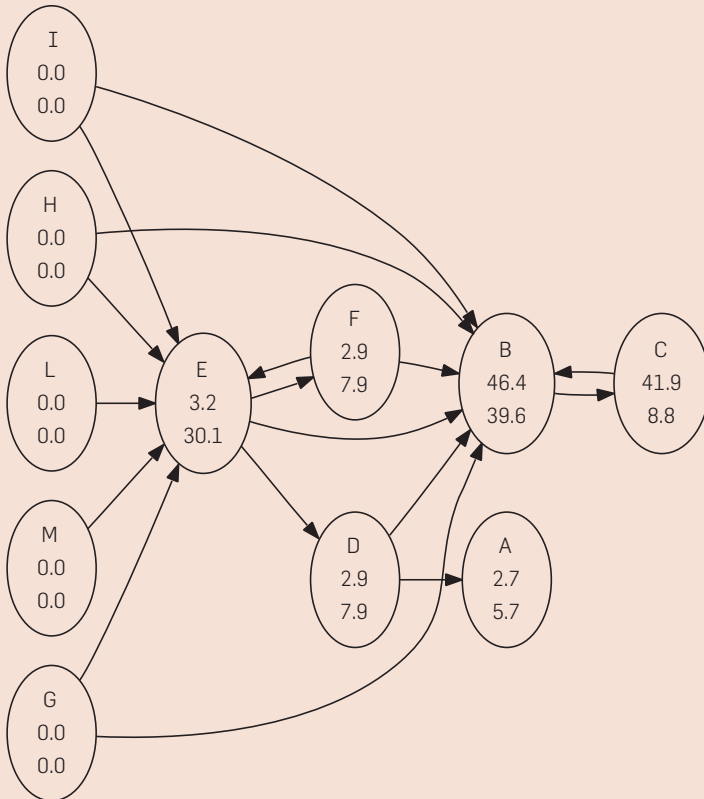
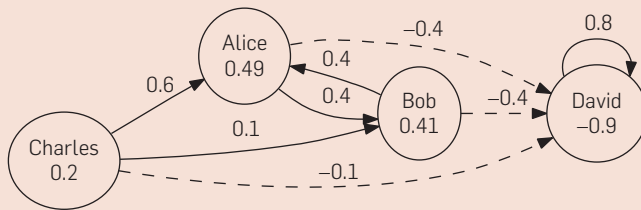


Figure 5. An instance of the Hubbell model with solution: each node is labeled with its prestige score and each edge is labeled with the endorsement strength between the connected members; negative strength is highlighted with dashed edges. The minimal amount of status has been fixed to 0.2 for all members.



the denominator of the influence score formula. This avoids overinflating journals that engage in the practice of opportunistic self-citations.

It can be proved that the spectral radius of matrix H is 1, hence the influence score vector corresponds to the dominant eigenvector of H .⁹ In principle, the uniqueness of the solution and the convergence of the power method to it are not guaranteed. Nevertheless, both properties are not difficult to obtain in real cases. If the citation graph is strongly connected, then the solution is unique. When journals belong to the same research field, this condition is typically satisfied. Moreover, if there exists a self-loop in the graph, that is an article that cites an article in the same journal, then the power method converges.

Figure 3 provides an example of the Pinski and Narin method. Notice that the graph is strongly connected and has a self-loop, hence the solution is unique and can be computed with the power method. Both journals A and C receive the same number of citations and give out the same number of references. Nevertheless, the influence of A is bigger, since it is cited by a more influential journal (B instead of D). Furthermore, A and D receive the same number of citations from the same journals, but D is larger than A, since it contains more references, hence the influence of A is higher.

Similar recursive methods have been independently proposed by Liebowitz and Palmer¹⁹ and Palacios-Huerta and Volij²³ in the context of ranking of economics journals. Recently, various PageRank-inspired bibliometric indicators to evaluate the importance of journals using the academic citation network have been proposed and

extensively tested: journal PageRank,² Eigenfactor,³⁴ and SCImago.²⁸

Sociometry

Sociometry, the quantitative study of social relationships, contains remarkably old PageRank predecessors. Sociologists were the first to use the network approach to investigate the properties of groups of people related in some way. They devised measures like indegree, closeness, betweenness, as well as eigenvector centrality, which are still used today in modern (not necessarily social) network analysis.²¹ In particular, eigenvector centrality uses the same central ingredient of PageRank applied to a social network:

A person is prestigious if he is endorsed by prestigious people.

John R. Seeley in 1949 is probably the first in this context to use the circular argument of PageRank.²⁹ Seeley reasons in terms of social relationships among children: each child chooses other children in a social group with a non-negative strength. The author notices that the total choice strengths received by each children is inadequate as an index of popularity, since it does not consider the popularity of the chooser. Hence, he proposes to define the popularity of a child as a function of the popularity of those children who chose the child, and the popularity of the choosers as a function of the popularity of those who chose them and so in an “indefinitely repeated reflection.” Seeley exposes the problem in terms of linear equations and uses Cramer’s rule to solve the linear system. He does not discuss the issue of uniqueness.

Another model is proposed in 1953 by Leo Katz.¹¹ Katz views a social

network as a directed graph where nodes are people and person i is connected by an edge to person j if i chooses, or endorses, j . The status of member i is defined as the number of weighted paths reaching j in the network, a generalization of the indegree measure. Long paths are weighted less than short ones, since endorsements devalue over long chains. Notice that this method indirectly takes account of who endorses as well as how many endorse an individual: if a node i points to a node j and i is reached by many paths, then the paths leading to i arrive also at j in one additional step.

Katz builds an adjacency matrix $L = (l_{ij})$ such that $l_{ij} = 1$ if person i chooses person j and $l_{ij} = 0$ otherwise. He defines a matrix $W = \sum_{k=1}^{\infty} (aL)^k$, where a is an attenuation constant. Notice that the (i,j) component of L^k is the number of paths of length k from i to j , and this number is attenuated by a^k in the computation of W . Hence, the (i,j) component of the limit matrix W is the weighted number of arbitrary paths from i to j . Finally, the status of member $\pi_j = \sum_i w_{ij}$, that is, the number of weighted paths reaching j . If the attenuation factor $a < 1/\rho(L)$, with $\rho(L)$ the spectral radius of L , then the above series for W converges.

Figure 4 illustrates the method with an example. Notice the important role of the attenuation factor: when it is large (close to $1/\rho(L)$), long paths are devalued smoothly, and Katz scores are strongly correlated with PageRank ones. In the shown example, PageRank and Katz methods provide the same ranking of nodes when the attenuation factor is 0.9. On the other hand, if the attenuation factor is small (close to 0), then the contribution given by paths longer than 1 rapidly declines, and thus Katz scores converge to indegrees, the number of incoming links of nodes. In the example, when the attenuation factor drops to 0.1, nodes C and E switch their positions in the ranking: node E, which receives many short paths, significantly increases its score, while node C, which is the destination of just one short path and many (devalued) long ones, significantly decreases its score.

In 1965 Charles H. Hubbell generalizes the proposal of Katz.¹⁰ Given a set of members of a social context, Hubbell defines a matrix $W = (w_{ij})$

such that $w_{i,j}$ is the strength at which i endorses j . Interestingly, these weights can be arbitrary, and in particular, they can be negative. The prestige of a member is recursively defined in terms of the prestige of the endorser and takes account of the endorsement strengths:

$$\pi = \pi W + \nu \quad (5)$$

The term ν is an *exogenous* vector such that ν_i is a minimal amount of status assigned to i from outside the system.

The original aspects of the method are the presence of an exogenous initial input and the possibility of giving negative endorsements. A consequence of negative endorsements is that the status of an actor can also be negative. An actor who receives a positive (respectively, negative) judgment from a member of positive status increases (respectively, decreases) his prestige. On the other hand, and interestingly, receiving a positive judgment from a member of negative status makes a negative contribution to the prestige of the endorsed member (if you are endorsed by some person affiliated to the Mafia your reputation might drop indeed). Moreover, receiving a negative endorsement from a member of negative status makes a positive contribution to the prestige of the endorsed person (if the same Mafioso opposes you, then your reputation might raise).

Figure 5 shows an example for the Hubbell model. Notice that Charles does not receive any endorsement and hence has the minimal amount of status given by default to each member. David receives only negative judgments; interestingly, the fact that he has a positive self-opinion further decreases his status. A better strategy for him, knowing in advance of his negative status, would be to negatively judge himself,

acknowledging the negative judgment given by the other members.

Equation 5 is equivalent to $\pi(I - W) = \nu$, where I is the identity matrix, that is $\pi = \nu(I - W)^{-1} = \nu \sum_{i=0}^{\infty} W^i$. The series converge if and only if the spectral radius of W is less than 1. It is now clear that the Hubbell model is a generalization of the Katz model to general matrices that adds an initial exogenous input ν . Indeed, the Katz equation for social status is $\pi = e \sum_{i=1}^{\infty} (aL)^i$, where e is a vector of all ones. In an unpublished note Vigna traces the history of the mathematics of spectral ranking and shows there is a reduction from the path summation formulation of Hubbell–Katz to the eigenvector formulation with teleportation of PageRank and vice versa.³⁰ In the mapping the attenuation constant is the counterpart of the PageRank damping factor, and the exogenous vector corresponds to the PageRank personalization vector. The interpretation of PageRank as a sum of weighted paths is also investigated by Baeza-Yates et al.¹

Spectral ranking methods have also been exploited to rank sport teams in competitions that involve teams playing in pairs.^{13,32} The underlying idea is that a team is strong if it won against other strong teams. Much of the art of the sport ranking problem is how to define the matrix entries $a_{i,j}$ expressing how much team i is better than team j (e.g., we could pick $a_{i,j}$ to be 1 if j beats i , 0.5 if the game ended in a tie, and 0 otherwise).¹²

Econometrics

We conclude with a succinct description of the input–output model developed in 1941 by Nobel Prize winner Wassily W. Leontief in the field of econometrics—the quantitative study of economic principles.¹⁸ According to the Leontief input–output model, the

economy of a country may be divided into any desired number of sectors, called industries, each consisting of firms producing a similar product. Each industry requires certain inputs in order to produce a unit of its own product, and sells its products to other industries to meet their ingredient requirements. The aim is to find prices for the unit of product produced by each industry that guarantee the *reproducibility* of the economy, which holds when each sector balances the costs for its inputs with the revenues of its outputs. In 1973, Leontief earned the Nobel Prize in economics for his work on the input–output model. An example is provided in Table 2.

Let $q_{i,j}$ denote the quantity produced by the i th industry and used by the j th industry, and q_i be the total quantity produced by sector i , that is, $q_i = \sum_j q_{i,j}$. Let $A = (a_{i,j})$ be such that $a_{i,j} = q_{i,j}/q_j$; each coefficient $a_{i,j}$ represents the amount of product (produced by industry) i consumed by industry j that is necessary to produce a unit of product j . Let π_j be the price for the unit of product produced by each industry j . The reproducibility of the economy holds when each sector j balances the costs for its inputs with the revenues of its outputs, that is:

$$\begin{aligned} \text{cost}_j &= \sum_i \pi_i q_{i,j} = \\ \text{revenue}_j &= \sum_i \pi_j q_{j,i} = \pi_j \sum_i q_{j,i} = \pi_j q_j \end{aligned}$$

By dividing each balance equation by q_j we have

$$\pi_j = \sum_i \pi_i \frac{q_{i,j}}{q_j} = \sum_i \pi_i a_{i,j}$$

or, in matrix notation,

$$\pi = \pi A \quad (6)$$

Table 2. An input–output table for an economy with three sectors with the balance solution.

	Agriculture	Industry	Family	Total	Price	Revenue
Agriculture	7.5	6	16.5	30	20	600
Industry	14	6	30	50	15	750
Family	80	180	40	300	3	900
Cost	600	750	900			

Each row shows the output of a sector to other sectors of the economy. Each column shows the inputs received by a sector from other sectors. For each sector we also show total quantity produced, equilibrium unitary price, total cost, and total revenue. Notice that each sector balances costs and revenues.

Hence, highly remunerated industries (industries j with high total revenue $\pi_j q_j$) are those that receive substantial inputs from highly remunerated industries, a circularity that closely resembles the PageRank thesis.²⁵ With the same argument used by Geller⁹ for the Pinski and Narin bibliometric model we can show that the spectral radius of matrix A is 1, thus the equilibrium price vector π is the dominant eigenvector of matrix A . Such a solution always exists, although it might not be unique, unless A is irreducible. Notice the striking similarity of the Leontief *closed* model with that proposed by Pinski and Narin. An *open* Leontief model adds an exogenous demand and creates a surplus of revenue (profit). It is described by the equation $\pi = \pi A + v$, where v is the profit vector. Hubbell himself observes the similarity between his model and the Leontief open model.¹⁰

It might seem disputable to juxtapose PageRank and Leontief methods. To be sure, the original motivation of Leontief work was to give a formal method to find equilibrium prices for the reproducibility of the economy and to use the method to estimate the impact on the entire economy of the change in demand in any sectors of the economy. Leontief, to the best of our limited knowledge, was not motivated by an industry *ranking* problem. On the other hand, the motivation underlying the other methods described in this paper is the ranking of a set of homogeneous entities. Despite the original motivations, however, there are more than coincidental similarities between the Leontief open and closed models and the other ranking methods described in this paper. These connections motivated the discussion of the Leontief contribution, which is probably the least known among the surveyed methods within the computing community.

Conclusion

The classic notion of quality of information is related to the judgment given by few field *experts*. PageRank introduced an original notion of quality of information found on the Web: the *collective intelligence* of the Web, formed by the opinions of the millions of people that populate this universe, is exploited to determine the importance,

and ultimately the quality, of that information.

Consider the difference between *expert evaluation* and *collective evaluation*. The former tends to be intrinsic, subjective, deep, slow, and expensive. By contrast, the latter is typically extrinsic, democratic, superficial, fast, and low cost. Interestingly, the dichotomy between these two evaluation methodologies is not peculiar to information found on the Web. In the context of assessment of academic research, peer review—the evaluation of scholarly publications given by peer experts working in the same field of the publication—plays the role of expert evaluation. Collective evaluation consists in gauging the importance of a contribution through the bibliometric practice of counting and analyzing citations received by the publication from the academic community. Citations generally witness the use of information and acknowledge intellectual debt. Eigenfactor,³⁴ a PageRank-inspired bibliometric indicator, is among the most interesting recent proposals to collectively evaluate the status of academic journals. The consequences of a shift from peer review to bibliometric evaluation are currently heartily debated in the academic community.³³

Acknowledgments

The author thanks Enrico Bozzo, Sebastiano Vigna, and the anonymous referees for positive and critical comments on the early drafts of this paper. Sebastiano Vigna was the first to point out the contribution of John R. Seeley. 

References

1. Baeza-Yates, R.A., Boldi, P., Castillo, C. Generic damping functions for propagating importance in link-based ranking. *Internet Math.* 3, 4 (2007), 445–478.
2. Bollen, J., Rodriguez, M.A., de Sompel, H.V. Journal status. *Scientometrics* 69, 3 (2006), 669–687.
3. Brin, S., Page, L. The anatomy of a large-scale hypertextual web search engine. *Comput. Netw. ISDN Syst.* 30, 1–7 (1998), 107–117.
4. Broder, A.Z., Kumar, R., Maghoul, F., Raghavan, P., Rajagopalan, S., Stata, R., Tomkins, A., Wiener, J.L. Graph structure in the Web. *Comput. Netw.* 33, 1–6 (2000), 309–320.
5. Bush, V. As we may think. *Atl. Month.* 176, 1 (1945), 101–108.
6. Ding, C.H.Q., Zha, H., He, X., Hubs, P., Simon, H.D. Link analysis: Hubs and authorities on the World Wide Web. *SIAM Rev.* 46, 2 (2004), 256–268.
7. Frobenius, G. Über Matrizen aus nicht negativen Elementen. In *Sitzungsberichte der Preussischen Akademie der Wissenschaften zu Berlin*, 1912, 456–477.
8. Garfield, E., Sher, H. New factors in the evaluation of scientific literature through citation indexing. *Am. Doc.* 14 (1963), 195–201.

9. Geller, N.L. On the citation influence methodology of Pinski and Narin. *Inf. Process. Manage.* 14, 2 (1978), 93–95.
10. Hubbell, C.H. An input–output approach to clique identification. *Sociometry* 28 (1965), 377–399.
11. Katz, L. A new status index derived from sociometric analysis. *Psychometrika* 18 (1953), 39–43.
12. Keener, J.P. The Perron–Frobenius theorem and the ranking of football teams. *SIAM Rev.* 35, 1 (1993), 80–93.
13. Kendall, M.G. Further contributions to the theory of paired comparisons. *Biometrics* 11, 1 (1955), 43–62.
14. Kleinberg, J.M. Authoritative sources in a hyperlinked environment. In *ACM–SIAM Symposium on Discrete Algorithms*, 1998, 668–677.
15. Kleinberg, J.M. Authoritative sources in a hyperlinked environment. *J. ACM* 46, 5 (1999), 604–632.
16. Langville, A.N., Meyer, C.D. *Google's PageRank and Beyond: The Science of Search Engine Rankings*. Princeton University Press, 2006.
17. Lempel, R., Moran, S. The stochastic approach for link-structure analysis (SALSA) and the TKC effect. *Comput. Netw.* 33, 1–6 (2000), 387–401.
18. Leontief, W.W. *The Structure of American Economy, 1919–1929*. Harvard University Press, Cambridge, 1941.
19. Liebowitz, S.J., Palmer, J.P. Assessing the relative impacts of economics journals. *J. Econ. Lit.* 22 (1984), 77–88.
20. Markov, A. Rasprostranenie zakona bol'shih chisel na velichiny, zavisyaschie drug ot druga. *Izvestiya Fiziko-matematicheskogo obschestva pri Kazanskom universitete*, 2-ya seriya 15, 94 (1906), 135–156.
21. Newman, M.E.J. *Network Analysis: An Introduction*. Oxford University Press, Oxford, U.K., 2010.
22. Page, L., Brin, S., Motwani, R., Winograd, T. The PageRank citation ranking: Bringing order to the Web. Technical Report 1999–66, Stanford InfoLab, November 1999. Retrieved June 1, 2010, from <http://ilpubs.stanford.edu:8090/422/>
23. Palacios-Huerta, I., Volij, O. The measurement of intellectual influence. *Econometrica* 72 (2004), 963–977.
24. Perron, O. Zur theorie der matrices. *Math. Ann.* 64, 2 (1907), 248–263.
25. Pillai, S.U., Suel, T., Cha, S. The Perron–Frobenius theorem: Some of its applications. *IEEE Signal Process. Mag.* 22, 2 (2005), 62–75.
26. Pinski, G., Narin, F. Citation influence for journal aggregates of scientific publications: Theory, with application to the literature of physics. *Inf. Process. Manage.* 12, 5 (1976), 297–312.
27. R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2007. ISBN 3-900051-07-0. Accessed June 1, 2010, at <http://www.R-project.org>
28. SCImago. SJR—SCImago Journal & Country Rank. Accessed June 1, 2010, at <http://www.scimagojr.com>, 2007.
29. Seeley, J.R. The net of reciprocal influence: A problem in treating sociometric data. *Can. J. Psychol.* 3 (1949), 234–240.
30. Vigna, S. Spectral ranking, 2010. Retrieved June 1, 2010 from <http://arxiv.org/abs/0912.0238>
31. von Mises, R., Pollaczek-Geiringer, H. Praktische verfahren der gleichungsaufloesung. *Z. Angew. Math. Mech.* 9, 58–77 (1929), 152–164.
32. Wei, T.H. The algebraic foundations of ranking theory. PhD thesis, Cambridge University, 1952.
33. Weingart, P. Impact of bibliometrics upon the science system: Inadvertent consequences? *Scientometrics* 62, 1 (2005), 117–131.
34. West, J., Althouse, B., Bergstrom, C., Rosvall, M., Bergstrom, T. Eigenfactor.org—Ranking and mapping scientific knowledge. Accessed June 1, 2010, at <http://www.eigenfactor.org>, 2007.
35. Zheng, A.X., Ng, A.Y., Jordan, M.I. Stable algorithms for link analysis. In *International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2001, 258–266.

Massimo Franceschet (massimo.franceschet@udine.it) is a researcher in the Department of Mathematics and Computer Science, University of Udine, Italy.

© 2011 ACM 0001-0782/11/06 \$10.00

research highlights

P. 103

**Technical
Perspective**

The Quest for a Logic for Polynomial-Time Computation

By Phokion G. Kolaitis

P. 104

From Polynomial Time Queries to Graph Structure Theory

By Martin Grohe

P. 113

**Technical
Perspective**

Data Analysis at Astonishing Speed

By Michael J. Franklin

P. 114

Dremel: Interactive Analysis of Web-Scale Datasets

By Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer,
Shiva Shivakumar, Matt Tolton, and Theo Vassilakis

Technical Perspective

The Quest for a Logic for Polynomial-Time Computation

By Phokion G. Kolaitis

THE INTERACTION BETWEEN computation and logic goes back to the early beginnings of computer science with the development of computability theory in the 1930s by A. Turing, K. Gödel, A. Church, S. Kleene, and the other great logicians of that era. In more recent decades, the interaction between computation and logic has spanned the entire spectrum of computer science, from programming languages to artificial intelligence and from computational complexity to database systems. Consider, for example, the P vs. NP problem, which is justly regarded to be the central open problem in theoretical computer science.¹ Formally, the P vs. NP problem asks whether or not the class NP of all algorithmic problems solvable by a non-deterministic Turing machine in polynomial time coincides with the class P of all algorithmic problems solvable by a deterministic Turing machine in polynomial time. Informally, the P vs. NP problem asks whether every algorithmic problem whose solutions can be efficiently verified has the property that its solutions can also be efficiently computed. Logic has played a key role in the investigation of this problem. Indeed, in 1971, S. Cook established the existence of “hardest” problems in NP by showing that *Boolean Satisfiability*, a fundamental problem in propositional logic, is NP-complete. This implies that $P = NP$ if and only if there is an efficient algorithm for telling whether or not a given Boolean formula has a satisfying truth assignment.

Soon after Cook’s seminal discovery, R. Fagin found an even tighter connection between logic and NP by showing that NP coincides with the class of all algorithmic problems that can be specified using a logical formalism known as *existential second-order logic* or, in short, *ESO*. To illustrate this result, consider

3-Colorability, another prominent NP-complete problem, which asks whether or not a given graph G (that is, a set of nodes V and a set of edges E) can be colored by assigning one of three colors, say blue, red, and yellow, to each of its nodes, so that no two nodes joined by an edge have the same color. This problem can be specified by the *ESO*-formula shown at the bottom of the page.

Intuitively, this formula asserts that the set V of nodes can be partitioned into three subsets (color classes) B , R , and Y such that if two nodes x and y are joined by an edge $E(x, y)$, then x and y cannot be in the same color class. Fagin’s result reinforces the unity between computation and logic by providing a logic-based, machine-free characterization of NP that makes no mention of computational resources (time) or bounds (polynomial) on such resources.

Is there a logic for P? In other words, is there a logical formalism L on the class of all graphs such P coincides with the class of all algorithmic problems on graphs that can be specified in L ? This problem was raised by Y. Gurevich in 1984, who actually conjectured that there is *no* logic for P. A related (and essentially equivalent) problem had been raised in 1982 by A. Chandra and D. Harel, who asked: is there is an algorithm that enumerates all polynomial-time computable properties of graphs? Having a logic L for P will make it possible to recast the P vs. NP problem as a problem of pure logic, namely, as the problem of whether or not *ESO* and L have the same expressive power on the collection of all graphs. Furthermore, a logic L for P will serve as a high-level specification language for expressing precisely those algorithmic problems about graphs whose solutions can be efficiently computed.

The quest for a logic for P on the

class of all graphs has been going on for the past 30 years. While the problem remains unresolved to date, the quest has given rise to a fascinating journey that has brought together several different strands of research. The following paper by Martin Grohe contains an account of one of the most sophisticated and beautiful results obtained to date in the quest for a logic for P on the class of all graphs. In a nutshell, Grohe shows there is indeed a logic for P for many large classes of graphs of algorithmic interest and mathematical significance. More precisely, his result asserts that there is a logic for P for *every* class of graphs consisting of all graphs that exclude some fixed graph as a minor, that is, as a graph obtained by edge contractions on a subgraph. This result vastly generalizes earlier work showing that a logic for P exists for large classes of graphs, including the class of all planar graphs.

The logic used in Grohe’s result is fixed-point logic with counting, a natural extension of first-order logic with a recursion and a cardinality-counting mechanism that has found many applications in other areas of computer science and, in particular in databases. From a technical viewpoint, Grohe’s proof is a real tour de force that requires the development of deep graph-theoretic machinery that uncovers new insights into the structure of graphs with excluded minors. Furthermore, the interplay between logic and graph theory yields another bonus, namely, for every class of graphs with excluded minors, the graph isomorphism problem can be solved in polynomial time via a simple combinatorial algorithm. Whether a polynomial-time algorithm exists for the graph isomorphism problem on the class of all graphs is yet another major open problem in computer science. ■

References

1. Fortnow, L. The status of the P versus NP problem. *Commun. ACM* 52, 9 (Sept. 2009), 78–86.

Phokion G. Kolaitis (kolaitis@cs.ucsc.edu) is a professor of computer science at the University of California Santa Cruz and a research staff member at IBM Research—Almaden.

© 2011 ACM 0001-0782/11/06 \$10.00

$$\exists B \exists R \exists Y \forall x ((B(x) \vee R(x) \vee Y(x)) \wedge (\forall y) (E(x, y) \rightarrow \neg (B(x) \wedge B(y)) \wedge \neg (R(x) \wedge R(y)) \wedge \neg (Y(x) \wedge Y(y))))$$

From Polynomial Time Queries to Graph Structure Theory

By Martin Grohe

Abstract

We give a logical characterization of the polynomial-time properties of graphs with excluded minors: For every class $\mathcal{C}$ of graphs such that some graph H is not a minor of any graph in $\mathcal{C}$, a property $\mathcal{P}$ of graphs in $\mathcal{C}$ is decidable in polynomial time if and only if it is definable in fixed-point logic with counting. Furthermore, we prove that for every class $\mathcal{C}$ of graphs with excluded minors there is a k such that a simple combinatorial algorithm, namely “the k -dimensional Weisfeiler–Lehman algorithm,” decides isomorphism of graphs in $\mathcal{C}$ in polynomial time.

1. INTRODUCTION

Problems that are hard to describe are hard to solve and vice versa. This claim may seem naive or just wrong at first sight, but it is the underlying idea of a deep and fruitful connection between the *descriptive complexity* and the *computational complexity* of computational problems. Here descriptive complexity refers to the “language resources” required to express a problem in a formal language, or logic, whereas computational complexity refers to the computational resources needed to solve the problem. The connection goes back to the beginnings of computability theory. For example, the recursively enumerable (or semidecidable) sets of natural numbers are precisely the sets definable by an existential formula of first-order logic in the language of arithmetic.¹⁵ Such a formula consists of a prefix of existential quantifiers followed by a Boolean combination of equations between polynomials in several variables.

In today’s computation theory, we are less interested in the classes of computable and recursively enumerable sets than in much smaller complexity classes such as the class PTIME of all polynomial-time solvable problems or the class NP. Furthermore, we rarely model computational problems as sets of natural numbers. Instead, our basic models are finite graphs and other finite structures such as labeled transition systems, Boolean circuits, and relational databases. (For simplicity, we mostly consider graphs in this article.) The starting point of modern descriptive complexity theory is Fagin’s Theorem,³ stating that a property of finite graphs (or other structures) is expressible in existential second-order logic if and only if it is decidable in nondeterministic polynomial time. More concisely, we say that existential second-order logic *captures* NP. Similar logical characterizations have been obtained for other complexity classes such as the polynomial-time hierarchy PH and PSPACE. However, no logic is known to capture the class PTIME or any natural

complexity class below PTIME. Hence we lack a logical characterization of the problems we are most interested in—those that can be solved efficiently!

The question of whether there is a logic that captures PTIME has first been raised by Chandra and Harel² in 1982 in a fundamental paper on database query languages. Chandra and Harel asked for a query language for relational databases expressing precisely those queries that can be evaluated in polynomial time. Clearly, it would be very nice to have such a language: It would allow the user to ask only queries that can be answered efficiently, while at the same time guaranteeing that all such queries can be asked. Gurevich⁷ rephrased the question in terms of logic. His precise definition of a “logic capturing PTIME” is subtle; we will discuss it below. Gurevich⁷ conjectured that there is no logic capturing PTIME. Observe that this conjecture implies $\text{PTIME} \neq \text{NP}$, because by Fagin’s Theorem there is a logic that captures NP. The question of whether there is a logic capturing PTIME is still open today, and it is viewed as one of the main open problems in finite model theory and database theory. Only partial positive answers are known, most notably the Immerman–Vardi Theorem^{10, 22} stating that least fixed-point logic captures PTIME on ordered structures (see Section 1.2).

1.1. Logics capturing polynomial time

So what exactly constitutes a logic that captures PTIME? A first requirement (L1) is that a logic L should have a *decidable syntax*, that is, a computable set of *sentences*. To define the semantics, we associate a *property* $\mathcal{P}_\phi$ of finite graphs (or other structures) with each sentence ϕ of L . We say that ϕ *defines* $\mathcal{P}_\phi$ and that $\mathcal{P}_\phi$ is *definable* in L . For L to capture PTIME, we require (L2) each graph property decidable in PTIME to be definable in L and, conversely, each property definable in L to be decidable in PTIME. But this is not enough; we are still missing an effective link between sentences and the properties they define. Hence we require (L3) the existence of an algorithm that associates with each sentence ϕ of L an “evaluation algorithm” A_ϕ deciding the property $\mathcal{P}_\phi$ in polynomial time. This completes the definition: we say that a *logic captures* PTIME if it satisfies (L1)–(L3).

The original version of this paper is entitled “Fixed-Point Definability and Polynomial Time on Graphs with Excluded Minors” and was published in *Proceedings of the 25th IEEE Symposium on Logic in Computer Science*, 2010.

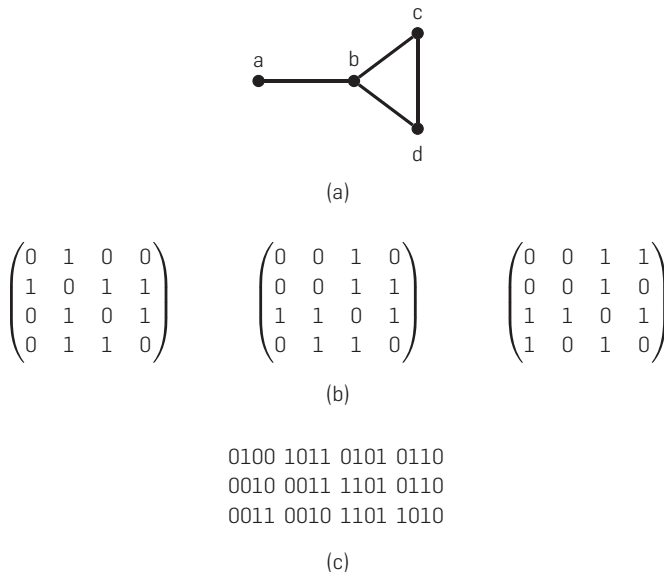
EXAMPLE 1. Consider first-order predicate logic FO. For instance, the FO-sentence $\forall x \exists y E(x, y)$ in the language of graphs defines the graph property of having no isolated vertices. (“Each vertex has a neighbor.”) It can be shown that FO satisfies criteria (L1) and (L3), but not (L2). Hence it is not a logic capturing PTIME.

So why are we interested in a logic capturing PTIME? After all, logic is just another formalism that can be used to describe computation, like Turing machines, the lambda calculus, various grammars and rewriting systems, or “real” programming languages. But there is an important difference between logics capturing complexity classes and the other formalisms: Logics speak directly about graphs and structures, whereas most other formalisms operate on encodings of structures by strings or terms. Hence a logical characterization of a complexity class is *representation-independent*. This may seem like a minor technical issue, but actually is quite significant, mainly because we have no natural *canonical representations* of structures by strings or terms, but have to make arbitrary choices when we construct such representations.

EXAMPLE 2. A standard way of representing graphs is by their adjacency matrices; once we have an adjacency matrix we can obtain a $\{0, 1\}$ -string encoding the graph by concatenating the rows of the matrix.

Consider the graph displayed in Figure 1(a). Figure 1(b) shows three different adjacency matrices of this graph, leading to three different string encodings shown in Figure 1(c). Indeed, the graph has $4! = 24$ adjacency matrices, one for each linear order of the vertices (the three matrices in Figure 1 correspond to the orders $abcd$, $acbd$, $cabd$). Not all 24 matrices are different, but there is no distinguished one that we may declare to be the “right” representation.

Figure 1. Three different adjacency matrices of a graph and the corresponding string encodings (cf. Example 2).



We are used to thinking of graphs as abstract models and of properties of graphs as only depending on the graphs and not on their specific representations. Examples of graph properties are “connectedness,” “acyclicity,” or “Hamiltonicity” (i.e., having a cycle that traverses each vertex exactly once). We would not view a statement like “the first vertex has degree 3” as describing a graph property, because a graph has no distinguished “first” vertex.

This level of abstraction is an integral part of our models. For example, if we use a graph model of a network then we abstract from the physical location of the nodes on purpose (otherwise we should use a different model). Similarly, it is a feature and not a bug of the relational database model to separate the logical from the physical level of a database.

There is a mismatch between these abstract models and the typical algorithms working on them. A graph algorithm does not get an abstract graph as input, but a specific representation of this graph, typically an adjacency matrix or adjacency lists. Nevertheless, its result is expected to be independent of the specific representation. But how can we guarantee that a particular graph algorithm is representation-independent? Unfortunately, it is not decidable whether an algorithm has this property. As a matter of fact, many of the standard graph algorithms we know internally depend on the specific representation of the input and only happen to be representation-independent in their output. A typical example is a connectivity algorithm based on depth-first search. At any vertex of the graph, a depth-first search chooses the “first” neighbor of the vertex to continue the search. But there is no canonical first neighbor; it depends on the specific representation of the input graph which neighbor comes first.

We may ask if there is a programming language that enforces representation independence, in the sense that all syntactically correct programs are representation-independent, and at the same time allows us to implement algorithms for all decidable properties of graphs. It is not too hard to come up with such a language, albeit not a very natural one. But the price to pay appears to be a huge loss in efficiency. The question of whether there is a logic capturing PTIME can be recast as asking for a programming language that enforces representation independence and at the same time allows us to implement polynomial-time algorithms for all properties of graphs that can be decided in polynomial time by conventional algorithms working on specific representations of graphs. (By *polynomial-time algorithms* in our new language we mean programs that can be executed in polynomial time on a conventional machine.)

1.2. Capturing polynomial time on graphs with excluded minors

As mentioned earlier, the question of whether there is a logic capturing PTIME is still open, but there are partial positive results for specific classes of structures. The main theorem of this article is such a result. Informally, a logic L captures PTIME on a class C of structures if precisely the polynomial-time properties of structures in C are definable in

L. (We refrain from giving the formal definition, which is a straightforward adaptation of the general definition of a logic capturing PTIME.)

A *structure* consists of a *universe*, which is simply a finite set, and a finite collection of relations, functions, and constants on this universe. (In general, structures may have infinite universes and infinitely many relations, functions, and constants, but in descriptive complexity we restrict our attention to finite structures.) An *ordered structure* is a structure that has one distinguished binary relation $<$ which is a linear order of the universe. For example, a graph $G = (V, E)$ is a structure with universe V and one binary “edge” relation E . An ordered graph is a triple $(V, E, <)$, where (V, E) is a graph and $<$ is a linear order of V . Ordered structures play a special role in descriptive complexity theory, because they admit a canonical string representation. Recall Example 2. We saw there that for each linear order of the vertices of a graph there is one adjacency matrix, and from this adjacency matrix we can obtain a $\{0, 1\}$ -string representing the graph. Hence for an ordered graph $G = (V, E, <)$ we have a canonical string representation: We represent G by the string derived from the adjacency matrix of (V, E) corresponding to the linear order $<$ of V . This can be generalized from ordered graphs to arbitrary ordered structures. Thus the issue of representation independence disappears when we deal with ordered structures.

In 1982, Immerman¹⁰ and Vardi,²² independently, proved that *least fixed-point logic* FP captures polynomial time on the class of all ordered structures. FP is an extension of first-order predicate logic by a fixed-point operator that allows it to formalize inductive definitions. We will not see many logical formulas in this article and treat logics on a high level that does not require any detailed knowledge of their syntax and semantics. However, since FP will play a central role, it may be worthwhile to briefly look at its main mechanism:

EXAMPLE 3. Suppose we want to define the transitive closure T of the edge relation of a graph $G = (V, E)$. It is well known that this is not possible in first-order logic. But the transitive closure admits the following inductive definition: We let $T_1 := E$, and for all i we let T_{i+1} be the set of all pairs (u, v) such that there is a vertex w with $(v, w) \in T_i$ and $(w, u) \in T_i$. Then T is the union of all the T_i . Equivalently, we may define T as the least fixed point of the (monotone) operator

$$X \mapsto \{(x, y) \leftarrow E(x, y) \vee \exists z (X(x, z) \wedge X(z, y))\}.$$

Here $\vee$ and $\wedge$ denote disjunction and conjunction, respectively. In FP, we can form a formula

$$\text{lfp} (X(x, y) \leftarrow E(x, y) \vee \exists z (X(x, z) \wedge X(z, y))) (v, w)$$

to define the transitive closure. If we call this formula $\text{tc}(v, w)$, then the FP-sentence $\text{conn} := \forall v \forall w (v = w \vee \text{tc}(v, w))$ defines the property of an (undirected) graph being connected.

It can be shown by standard techniques that FP does not capture polynomial time on all structures, the most

blatant reason being that the logic lacks the ability to count. For example, the graph property of having an even number of vertices is not definable in FP. In the late 1980s, Immerman¹¹ conjectured that *fixed-point logic with counting* FP+C, an extension of least-fixed point logic by a mechanism that allows it to define the cardinalities of definable sets, might capture polynomial time. In addition to the vertices of a graph, in FP+C we can also speak about integers in the range from 0 to the size of the graph. Standard arithmetic operations on these integers are available in the logic. The main purpose of the integers is to express cardinalities.

EXAMPLE 4. Recall that a Eulerian cycle in a graph is a closed walk on which every edge occurs exactly once. A graph is Eulerian if it has a Eulerian cycle. It is a well-known fact (going back to Euler) that a graph is Eulerian if and only if it is connected and every vertex has even degree.

The term $\#x' E(x, x')$ counts the number neighbors x' of a vertex x in a graph. Hence the formula

$$\text{even-deg}(x) := \exists y \#x' E(x, x') = y + y,$$

where x, x' are variables ranging over the vertices of a graph and y is a variable ranging over the integers, says that vertex x has even degree. Finally, the FP+C-sentence

$$\text{conn} \wedge \forall x \text{even-deg}(x),$$

where conn is the FP-sentence defined in Example 3, says that a graph is Eulerian.

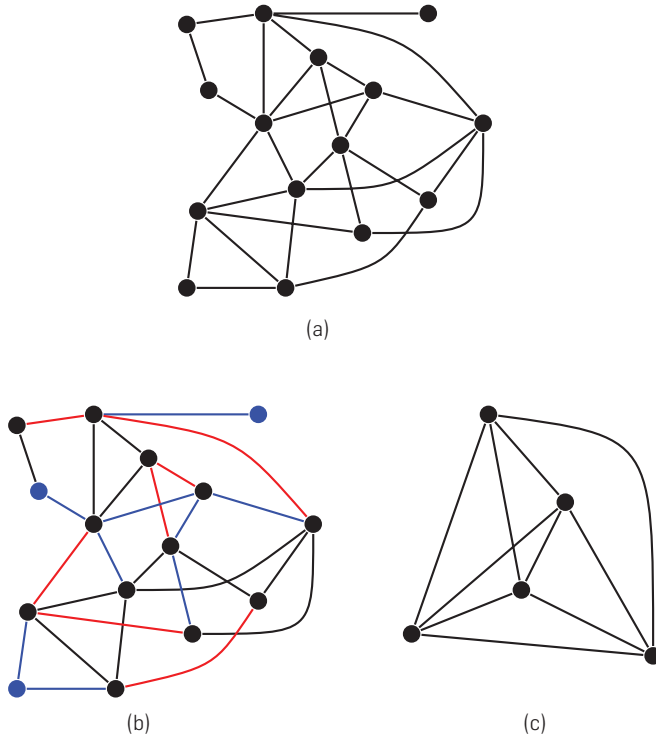
Although in 1992 Cai et al.¹ refuted Immerman’s conjecture that FP+C captures polynomial time, it has turned out since then that FP+C does capture polynomial time on many interesting classes of structures, among them trees,¹² graphs of bounded tree width,⁶ planar graphs,⁵ and interval graphs.¹³ Hella et al.⁸ proved that FP+C captures polynomial time on almost all structures (in a precise technical sense).

Our main theorem generalizes many of these results. To state it, we need to define graph minors. A graph H is a *minor* of a graph G if H is isomorphic to a graph obtained from a subgraph of G by contracting edges (see Figure 2 for an example). We say that H is an *excluded minor* for a class $\mathcal{C}$ of graphs if H is not a minor of any graph in $\mathcal{C}$.

THEOREM 1. Let $\mathcal{C}$ be a class of graphs with at least one excluded minor. Then fixed-point logic with counting (FP+C) captures polynomial time on $\mathcal{C}$.

Examples of classes of graphs with an excluded minor are the class of planar graphs (the complete graph with 5 vertices, K_5 , is an excluded minor), for every surface S the class of all graphs embeddable in S (every complete graph K_k such that $(k-3)(k-4)/6$ is greater than the genus of S is an excluded minor), the class of all graphs that can be embedded into $\mathbb{R}^3$ such that no cycle is nontrivially knotted (K_7 is an excluded minor), for every k the class of all graphs that have

Figure 2. The graph in (c) is a minor of the graph in (a) that is obtained by deleting all blue edges and vertices and contracting all red edges in (b).



a vertex cover of size at most k (K_{k+2} is an excluded minor), for every k the class of all graphs that have a feedback vertex set of size at most k (K_{k+3} is an excluded minor), and for every k the class of all graphs of tree width at most k (K_{k+2} is an excluded minor). To show the limitations of our theorem, let us also mention that the class of all cubic graphs (i.e., graphs in which every vertex has exactly three neighbors) does not exclude any graph as a minor.

A consequence of Theorem 1 is a simple polynomial-time isomorphism test for all classes of graphs with excluded minors: For each such class $\mathcal{C}$ a generic combinatorial “color refinement” algorithm, known as the *k-dimensional Weisfeiler–Lehman algorithm*, decides isomorphism of graphs in $\mathcal{C}$. Remarkably, the algorithm does not build on any of the special graph-theoretic properties the graphs in $\mathcal{C}$ may have, only the parameter k depends on $\mathcal{C}$. Neither does the algorithm use logic in any way; logic is only used to prove that it works. We describe the algorithm in Section 6, which can be read independently of the rest of the article.

To prove Theorem 1, we develop a *definable structure theory* for graphs with excluded minors. It builds on the structure theory for graphs with excluded minors developed by Robertson and Seymour (see Section 2), but ultimately leads to quite different structural decompositions of the graphs. On the graph theoretic side, the main new aspect of our theory is that we need to identify structural properties of graphs with excluded minors that are invariant under automorphisms, which is a prerequisite for making them definable (see the discussion in Section 3.1). On the logical

side, we have to find a notion of definable decomposition that is flexible enough to be widely applicable, yet still carries enough information to enable us to prove Theorem 1. *Definable (ordered) treelike decompositions*, which will be discussed in Section 4, fulfill these requirements. Much of the definable structure theory we develop is fairly general and not tailored toward graphs with excluded minors, and it may find other applications.

The full proof of Theorem 1 is very long, more than 200 pages, and will be published in a monograph, a first draft of which can be found at <http://www2.informatik.hu-berlin.de/~grohe/pub/cap/>.

2. THE STRUCTURE OF GRAPHS WITH EXCLUDED MINORS

The structure of graphs with excluded minors has strong topological aspects. Hence before we describe it, we briefly review a few facts about surface topology. Surfaces are topological spaces that locally look like the plane (“two-manifolds”); in addition, surfaces are required to be connected and compact. They can be classified into two infinite families: The first family of *orientable surfaces* consists of the sphere, the torus (“donut”), the double-torus, the triple-torus (“pretzel”), etc. The surfaces in the second family of *nonorientable surfaces* are obtained by cutting finitely many disjoint disks out of a sphere and then glueing Möbius strips on all the resulting holes (see Figure 3). With one Möbius strip this yields the projective plane and with two Möbius strips the Klein bottle. A convenient parameter to characterize a surface is its *Euler genus*, which for an orientable surface is twice the number of holes (e.g., six for a pretzel) and for a nonorientable surface is the number of Möbius strips used in its construction. A graph is *embeddable* in a surface if it can be drawn on the surface without edges crossing.

In long series of papers, Robertson and Seymour¹⁹ developed a structure theory for graphs with excluded minors. In Robertson and Seymour,²⁰ they proved a structure theorem which says that graphs with excluded minors can be decomposed into pieces that are *almost embeddable* into some surface. The pieces are arranged in a treelike fashion. Intuitively, almost embedding a graph into a surface means first removing a bounded number of vertices from the graph (these vertices are called *apices*) and then drawing the rest of the graph in the surface with no edges crossing except in a bounded number of regions (called *vortices*) in which the surface structure may be violated. The high-level structure is illustrated in Figure 4. Each vortex is

Figure 3. Construction of a projective plane: The boundary cycle of the Möbius strip is identified with the boundary of a hole in the sphere.

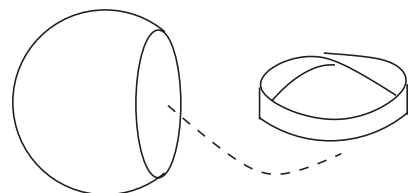
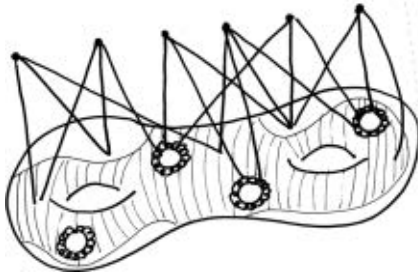


Figure 4. A graph almost embedded in a double torus with four vortices and six apices.



attached to the boundary of a “hole” in the disk. It is beyond the scope of this article to describe the structure of the vortices; the main parameter, called *width* of a vortex, is the maximum number of disjoint paths in a vortex “across the hole.” This width of the vortices is supposed to be bounded in an almost embeddable graph. Thus overall there are four parameters in the definition of almost embeddability: The surface, the number of apices, the number of vortices, and the width of the vortices. These parameters depend on the excluded minor.

The structure theorem for graphs with excluded minors plays a central role in Robertson and Seymour’s proof of their famous Graph Minor Theorem,²¹ stating that every class of graphs that is closed under taking minors can be characterized by finitely many excluded minors. The structure theorem has also found various algorithmic applications.

3. DEFINABLE ORDERS AND DEFINABLE CANONIZATION

Ultimately, all known results about capturing polynomial time on classes of structures are proved by a reduction to ordered structures and the Immerman–Vardi Theorem, and our Theorem 1 is no exception. These reductions are obtained either by defining linear orders on the structures under consideration or by defining ordered copies of the structures (a method known as *definable canonization*). In this section, we will briefly discuss both methods and illustrate them with a few examples. Then in Section 4 we will introduce the key concept of our definable structure theory, *definable ordered treelike decompositions*. Essentially, these are hierarchical decompositions of graphs into pieces on which we can define linear orders. We will see that once we have defined such decompositions in the logic FP on a class $\mathcal{C}$ of graphs, then we can prove that FP+C captures PTIME on $\mathcal{C}$. The reader not interested in the technical details may safely skip the rest of this and the following section and continue with Section 5, where we show that graph classes with excluded minors admit ordered treelike decompositions definable in FP.

3.1. Graphs, isomorphisms, and definable relations

The vertex set of a graph G is denoted by $V(G)$ and the edge set by $E(G)$. An *isomorphism* from a graph G to a graph H is a one-to-one mapping f from $V(G)$ onto $V(H)$ such

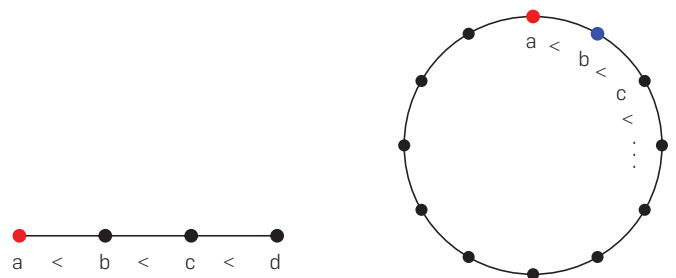
that for all vertices $u, v \in V(G)$ we have $uv \in E(G)$ if and only if $f(u)f(v) \in E(H)$. The representation-independence of graph properties discussed in Section 1.1 can be recast as *invariance under isomorphisms*: If a graph G has a property $\mathcal{P}$ and there is an isomorphism from G to a graph H , then H has property $\mathcal{P}$ as well. An *automorphism* of a graph G is an isomorphism f from G to G ; it is *nontrivial* if $f(v) \neq v$ for at least one v .

All formulas that appear in this article are FP formulas or FP + C formulas in the language of graphs. Free variables are always ranging over the vertices of a graph. By writing $\varphi(x, y)$ we indicate that the free variables of the formula φ are among x, y , and $G \models \varphi[u, v]$ means that G satisfies φ if x is interpreted by u and y is interpreted by v . It is an important fact about definability (not only in FP and FP+C, but any logic) that definable sets and relations are *invariant under automorphisms*. That is, if $\varphi(x_1, \dots, x_k)$ is a formula, G is a graph, and f is an automorphism of G then for all $v_1, \dots, v_k \in V(G)$ we have $G \models \varphi[v_1, \dots, v_k]$ if and only if $G \models \varphi[f(v_1), \dots, f(v_k)]$.

3.2. Definable linear orders

The easiest way to prove that a logic captures PTIME on a class $\mathcal{C}$ of graphs via the Immerman–Vardi Theorem is by defining linear orders on the graphs in $\mathcal{C}$ in the logic. For example, on directed paths we can easily define linear orders in FP: The transitive closure of the edge relation of a directed path happens to be a linear order of the path’s vertices, and we have seen in Example 3 that the transitive closure is definable in FP. On other structures, we cannot define linear orders directly, but need “parameters” in our definition. For example, on an undirected path, it is impossible to define a linear order without parameters, simply because there is no canonical way of defining which endpoint of a path comes first. More generally, if a graph has a nontrivial automorphism then it is impossible to define a linear order on this graph. However, if we fix one endpoint of an undirected path then we can define a linear order in the logic FP. Similarly, if we fix two adjacent vertices in a cycle we can define a linear order on the cycle (see Figure 5). In general, we say that a formula $\varphi(x, y, z_1, \dots, z_k)$ defines a linear order on a graph G with parameters $z_1, \dots, z_k$ if there are vertices $w_1, \dots, w_k$ such that the binary relation $\{(u, v) \in V(G)^2 \mid G \models \varphi[u, v, w_1, \dots, w_k]\}$ is a linear order of $V(G)$. We say that a class $\mathcal{C}$ of graphs admits FP-definable linear orders if there is

Figure 5. Defining an order on a path and cycle; the colored vertices are parameters of the definition.



an FP-formula that defines a linear order on all graphs in $G \in \mathcal{C}$. It follows from the Immerman–Vardi Theorem that if a class $\mathcal{C}$ admits FP-definable linear orders then FP captures PTIME on $\mathcal{C}$.

3.3. Definable canonization

Graphs that admit FP-definable linear orders tend to be rare; even the trivial graphs with no edges do not admit FP-definable linear orders. Fortunately, there is a way to circumvent this problem. Instead of an order, we define an *ordered copy* of a graph. Note the difference: If we define an order on the graph, we know exactly which vertex comes first, second, etc. If we just have an ordered copy (but no mapping associating the vertices of the original graph with the vertices of the copy), it does not give us any additional information about the individual vertices of the original graph. Yet an ordered copy can be very useful. For example, we can use it to obtain a canonical adjacency matrix of our graph. But how can we define an ordered copy of a graph in logic without actually defining a linear order on the vertices? We can do this with a triple $(\varphi_v(x), \varphi_e(x, y), \varphi_<(x, y))$ of formulas: applied to a graph G , we want these formulas to define a set $V' := \{a \mid G \models \varphi_v[a]\}$, a binary relation $E' := \{ab \mid G \models \varphi_e[a, b]\}$, and a binary relation $<' := \{ab \mid G \models \varphi_<[a, b]\}$ such that $G' := (V', E')$ is a graph isomorphic to G and $<'$ is a linear order of V' . The triple $(\varphi_v(x), \varphi_e(x, y), \varphi_<(x, y))$ is called a *transduction* or *syntactical interpretation*. We usually work with transductions consisting of FP+C formulas. Remember that in FP+C we can not only speak about the vertices of a graph, but also about an initial segment of the integers. The trick is to use the integers to represent the vertices of the copy of the input graph and then use the natural order of the integers as an order on these vertices.

EXAMPLE 5. A k -star has $k + 1$ vertices, one center and k rays. Each ray has an edge to the center, and there are no other edges. Exploiting the fact that stars have many automorphisms, it can be shown that the class of all stars does not admit definable linear orders.

We can define an ordered copy $(V', E', <')$ of a k -star S by letting $V' := \{0, 1, \dots, k\}$ and $E' := \{0_i \mid 1 \leq i \leq k\}$ and letting $<'$ be the natural order on V' (see Figure 6). The definition of $(V', E', <')$ can easily be formalized by an FP+C transduction (that does not depend on k). Counting is crucial here, because fixed-point logic without counting cannot determine how many rays a star has.

We say that a class $\mathcal{C}$ of graphs *admits FP+C-definable canonization* if there is an FP+C transduction defining an ordered

copy of each graph in $\mathcal{C}$. It follows from the Immerman–Vardi Theorem that if a class $\mathcal{C}$ admits FP+C-definable canonization then FP+C captures PTIME on $\mathcal{C}$. The idea of using definable canonization to capture polynomial time goes back to Immerman and Lander.¹² It has been developed to great depth by Otto.¹⁷ Example 5 shows that the class of stars admits FP+C-definable canonization. By applying the same idea recursively, Immerman and Lander¹² were able to prove that the class of all trees admits FP+C-definable canonization.

4. ORDERED TREELIKE DECOMPOSITIONS

The key idea in the proof of Theorem 1 is to decompose graphs with excluded minors in a treelike fashion into “simple” pieces on which we can define linear orders in FP and then use an argument similar to the definable tree canonization by Immerman and Lander¹² to show that the graphs admit FP-definable canonization.

The standard graph theoretic decomposition method, which is underlying Robertson and Seymour’s structure theorem for graphs with excluded minors (see Section 2), is based on *tree decompositions*. A *tree decomposition* of a graph G is a pair (T, β) consisting of a tree T and a mapping β associating with every node t of T a set $\beta(t) \subseteq V(G)$, called the *bag* of t , subject to certain technical conditions making sure that the structure of the tree T approximates the connectivity structure of G . To utilize tree decompositions in our context we would have to define them in FP or FP+C. This is impossible because tree decompositions are usually not invariant under automorphisms of the underlying graph. We resolve this problem by introducing a more general notion of decomposition that we call *treelike*. In a treelike decomposition, we replace the tree T underlying a tree decomposition by a directed acyclic graph D . The idea is that certain restrictions of D to subtrees yield tree decompositions of G , and by including many such decompositions we can close the treelike decompositions under automorphisms of a graph.

To get some idea how treelike decompositions look, consider Figure 7, which shows a treelike decomposition of the cycle $C_4 := (\{1, 2, 3, 4\}, \{12, 23, 34, 41\})$. The numbers in the nodes of the decomposition describe the bags. The three red nodes form a tree decomposition of C_4 , and so do the two blue nodes. There are many other tree decompositions of C_4 contained in the treelike decomposition in a similar way. Note the cyclic structure of the whole decomposition, which reflects the structure of the underlying cycle and is the reason for the invariance of the decomposition under automorphisms of the cycle. This cyclic structure is lost in a tree decomposition like the one consisting of the three red nodes.

A treelike decomposition of a graph G is *FP-definable* if there are FP formulas defining the vertex and edge set of the directed acyclic graph D and the bags $\beta(t)$ of the nodes $t \in V(D)$. For example, the treelike decomposition of the cycle C_4 shown in Figure 7 is FP-definable.

An *ordered treelike decomposition* of a graph G consists of a treelike decomposition together with a linear order of each bag. An ordered treelike decomposition is *FP-definable* if its underlying treelike decomposition is FP-definable and

Figure 6. Canonization of a star.

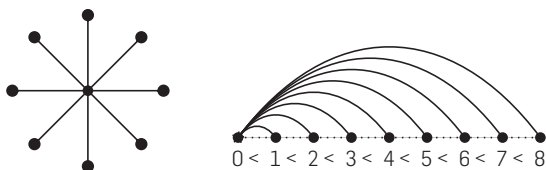
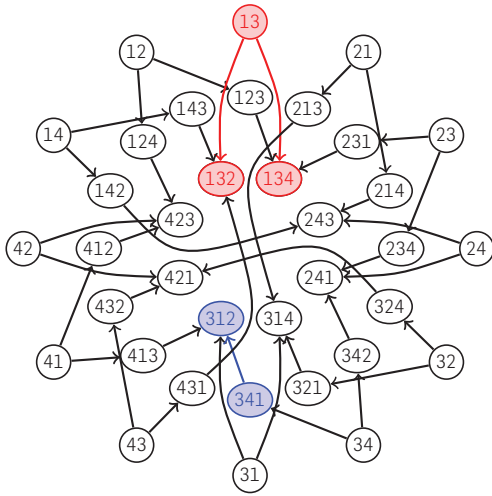


Figure 7. A treelike decomposition of the cycle C_4 .

its bags admit FP-definable linear orders. A class $\mathcal{C}$ of graphs admits FP-definable ordered treelike decompositions if there are formulas defining an ordered treelike decomposition on every graph in $\mathcal{C}$. The following lemma is a far reaching generalization of Immerman and Lander's FP + C-definable canonization of trees.

LEMMA 1. *Let $\mathcal{C}$ be a class of graphs that admits FP + C-definable ordered treelike decompositions. Then $\mathcal{C}$ admits FP + C-definable canonization.*

It turns out that in our application of the lemma we can even define the ordered treelike decompositions in the logic FP.

5. THE DEFINABLE STRUCTURE THEOREM

THEOREM 2 (DEFINABLE STRUCTURE THEOREM). *Every class of graphs with at least one excluded minor admits FP-definable ordered treelike decompositions.*

This theorem is our main technical result. It implies Theorem 1 via Lemma 1 and the Immerman–Vardi Theorem. To prove the Definable Structure Theorem, we built up the decomposition in stages described in the following six steps.

Step 1: Planar graphs. We start by proving that three-connected planar graphs admit FP-definable linear orders. Remember that a graph is *three-connected* if the deletion of any two vertices leaves it connected. The key step in the proof is to define the facial cycles of a three-connected planar graph in FP. We use the fact, going back to Whitney, that the facial cycles (boundaries of the faces) of a three-connected graph embedded in the plane are precisely the chordless and nonseparating cycles. In particular, this means that the facial cycles are the same for every embedding of the graph. Once we have defined the facial cycles, we can use three parameters to fix one facial cycle and then define in FP a linear order by “walking around this cycle in spirals.”

To show that arbitrary planar graphs admit FP-definable ordered treelike decompositions, we prove that the standard decomposition of a graph into its three-connected components yields an FP-definable treelike decomposition.

Step 2: Graphs embeddable in a surface. We exploit the fact that every surface of positive Euler genus has a *noncontractible* cycle. Cutting the surface open along such a cycle and glueing disks on the hole(s) yields one or two surfaces of strictly smaller Euler genus.

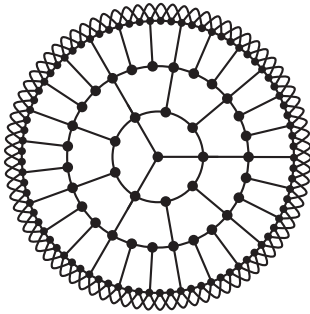
To define ordered treelike decompositions on graphs embeddable in a surface, we proceed by induction on the Euler genus of the surface. Planar graphs are the base case. In the inductive step, we try to define the facial cycles of a graph embedded in a surface. Either we succeed, then we can use the facial cycles to define a linear order in a similar way as for planar graphs. Or we find a noncontractible cycle along the way. Then we can delete this cycle, apply the induction hypothesis to the resulting graph embeddable in one or two surfaces of strictly smaller Euler genus, and lift the decomposition we obtain to the original graph, exploiting the fact that we can define in FP an order on the deleted cycle with just two parameters.

Step 3: Graphs of bounded tree width. A graph has *tree width* at most k if it has a tree decomposition, or equivalently, a treelike decomposition, where every bag contains at most $k+1$ vertices. The *width* of a decomposition is 1 plus the maximum cardinality of the bags. It turns out that we can actually define treelike decompositions of width k on graphs of tree width at most k . We do this by inductively defining partial decompositions in a bottom up fashion. Once we have a treelike decomposition of width at most k we can trivially define linear orders on the bags of size $(k+1)$ using $(k+1)$ parameters.

Step 4: Almost planar graphs. Remember our informal description of almost embeddable graphs in Section 2. Let $\mathcal{A}(p, q, r, s)$ be the class of all graphs almost embeddable in a surface of Euler genus at most r with at most s apices and at most q vortices, each of width at most p . In this and the following step, we want to prove that for all p, q, r, s the class $\mathcal{A}(p, q, r, s)$ admits FP-definable ordered treelike decompositions. We proceed by induction on $q+r$. We already know how to deal with the class $\mathcal{A}(0, 0, 0, 0)$ of planar graphs and more generally the class $\mathcal{A}(0, 0, r, 0)$ of all graphs embeddable in a surface of Euler genus at most r .

In this step, we consider the classes $\mathcal{A}(p, 1, 0, 0)$ of *almost planar graphs*. We think of almost planar graphs as graphs being embedded into a disk with a vortex glued on the boundary of the disk. Figure 8 shows an example. In the key lemma of this step, and actually one of the most difficult lemmas of the whole proof of Theorem 1, we show that the facial cycles of an almost planar graph that are sufficiently far from the vortex do not depend on the specific embedding. That is, no matter how we divide the graph into a vortex and a part embedded in the disk, these cycles will end up in the disk, and they will be facial cycles. Moreover, these cycles are FP-definable. We call the subgraph of the graph induced by these cycles the *center* of the graph. Using the facial cycles, we can define a linear order on each connected component of the center. If we contract each connected component of the center to a single

Figure 8. An almost planar graph (the outer edges connect vertices of distance two on the cycle).



vertex, we obtain a graph of tree width bounded by $O(p^2)$. Using Step 3, we can define an ordered treelike decomposition of this graph, and we can extend the decomposition to an ordered treelike decomposition of the original graph using the linear orders of the components of the center.

Step 5: Almost embeddable graphs. We prove that the classes $\mathcal{A}(p, q, r, s)$ admit FP-definable ordered treelike decompositions by an inductive construction similar to, but far more complicated than the one in Step 2.

Step 6: Graphs with excluded minors. Let $\mathcal{C}$ be a class of graphs with excluded minors. By Robertson and Seymour's structure theorem, there are p, q, r, s such that all graphs in $\mathcal{C}$ have a tree decomposition into pieces from $\mathcal{A}(p, q, r, s)$. If we could define such a decomposition in FP, then we could use the definable decomposition of the graphs in $\mathcal{A}(p, q, r, s)$ obtained in previous step to define ordered treelike decompositions of the graphs in $\mathcal{C}$. But unfortunately I do not know how to define such a decomposition in FP.

Instead, we repeatedly apply the construction of the previous steps to inductively build up an ordered treelike decomposition of a graph in $\mathcal{C}$ from partial decompositions in a bottom up fashion similar to Step 3.

6. APPLICATION TO ISOMORPHISM TESTING

It is a long standing open problem whether there is a polynomial-time algorithm deciding if two graphs are isomorphic. Polynomial-time isomorphism tests are known for many natural classes of graphs including the class of planar graphs,⁹ classes of graphs embeddable in a fixed surface^{4,16} and more generally classes of graphs with excluded minors,¹⁸ and classes of graphs of bounded degree.¹⁴ The isomorphism test for graphs of bounded degree due to Luks¹⁴ involves some nontrivial group theory, and many later isomorphism algorithms built on the group theoretic techniques developed by Babai, Luks, and others in the early 1980s. In particular, Ponomarenko's¹⁸ isomorphism algorithm for graphs with excluded minors heavily builds on these techniques.

Much simpler than the group theoretic algorithms is a combinatorial algorithm known as *color refinement* or *vertex classification*. It computes a coloring of the vertices of a graph by the following iterative procedure: initially, all vertices have the same color. Then in each round, the coloring is refined by assigning different colors to vertices that have a different number of neighbors of at least one color

assigned in the previous round. Thus after the first round, two vertices have the same color if and only if they have the same degree. After the second round, two vertices have the same color if and only if they have the same degree and for each d the same number of neighbors of degree d . The algorithm stops if no further refinement is achieved; this happens after at most n rounds, where n is the number of vertices of the input graph. To use color refinement as an isomorphism test, it is applied to the disjoint union of two graphs. If after the refinement process the colorings of the two graphs differ, that is, for some color c the graphs have a different number of vertices of color c , then we know that they are nonisomorphic. Unfortunately, if both graphs have the same coloring, they may still be nonisomorphic. As an example, consider two nonisomorphic regular graphs with the same number of vertices, such as a cycle of length 6 and the disjoint union of two cycles of length 3. They will get the same coloring even though they are nonisomorphic. We say that color refinement *distinguishes* two graphs if they get different colorings.

The k -dimensional Weisfeiler–Lehman algorithm (for short: k -WL) is a generalization of the color refinement algorithm that colors k -tuples of vertices instead of vertices (see Cai et al.¹ for a history of the algorithm). Let G be the input graph. Initially, two tuples $(v_1, \dots, v_k), (w_1, \dots, w_k) \in V(G)^k$ get the same color if the mapping $v_i \mapsto w_i$ is an isomorphism from the induced subgraph $G[\{v_1, \dots, v_k\}]$ to the induced subgraph $G[\{w_1, \dots, w_k\}]$. For $i \leq k$, we say that a tuple $(v_1, \dots, v_k)$ is an i -neighbor of a tuple $(w_1, \dots, w_k)$ if $v_j = w_j$ for all $j \neq i$. In each round of the algorithm, the coloring is refined by assigning different colors to tuples that for some $i \in [k]$ and some color c have different numbers of i -neighbors of color c . The algorithm stops if no further refinement is achieved; this happens after at most n^k rounds. Again, isomorphic graphs obviously get the same coloring, and we say that k -WL *distinguishes* two nonisomorphic graphs if they get different colorings. It is not so obvious that there are nonisomorphic graphs that are not distinguished by k -WL for any fixed $k \geq 3$, let alone $k = \log n$. By a beautiful construction, which has found several other applications in finite model theory, Cai et al.¹ proved that for each n there are nonisomorphic three-regular n -vertex graphs G_n, H_n that cannot be distinguished by k -WL for $k = o(n)$. They also noted that there is a connection between the Weisfeiler–Lehman algorithm and definability in fixed-point logic with counting.

Based on this connection, we can prove our second main result, which states that a constant dimensional WL suffices to distinguish graphs with excluded minors. Thus the Weisfeiler–Lehman algorithm yields a simple combinatorial polynomial-time isomorphism test for every class $\mathcal{C}$ of graphs with excluded minors. This algorithm completely avoids the sophisticated group theoretic machinery of Ponomarenko's algorithm. Actually, it is a generic algorithm that is not tailored for the specific input graphs in any way except by choosing the parameter k appropriately.

THEOREM 3. *For every class $\mathcal{C}$ of graphs with excluded minors there is a constant k such that k -WL distinguishes any two nonisomorphic graphs in $\mathcal{C}$.*

Theorem 3 follows from the Definable Structure Theorem 2 and Lemma 1 (or directly from Theorem 1 via a result due to Otto¹⁷). Remarkably, Theorem 3 is not referring to logic in any way—the Weisfeiler–Lehman algorithm is a purely combinatorial algorithm—yet our proof of the theorem heavily relies on logic.

Acknowledgment

I'd like to thank Martín Abadi, Dieter van Melkebeek, Moshe Vardi, and Thomas Wilke for very helpful comments on earlier versions of this article. 

References

1. Cai, J., Fürer, M., Immerman, N. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12 (1992), 389–410.
2. Chandra, A., Harel, D. Structure and complexity of relational queries. *J Comput. Syst. Sci.*, 25 (1982), 99–128.
3. Fagin, R. Generalized first-order spectra and polynomial-time recognizable sets. In *Complexity of Computation, SIAM-AMS Proceedings*, vol. 7, R. M. Karp, ed. 1974, 43–73.
4. Filotti, I.S., Mayer, J.N. A polynomial-time algorithm for determining the isomorphism of graphs of fixed genus. In *Proceedings of the 12th ACM Symposium on Theory of Computing*, 1980, 236–243.
5. Grohe, M. Fixed-point logics on planar graphs. In *Proceedings of the 13th IEEE Symposium on Logic in Computer Science*, 1998, 6–15.
6. Grohe, M., Mariño, J. Definability and descriptive complexity on databases of bounded tree-width. In *Proceedings of the 7th International Conference on Database Theory, Lecture Notes in Computer Science*, vol. 1540. C. Beeri and P. Buneman, eds. Springer-Verlag, Jerusalem, Israel, 1999, 70–82.
7. Gurevich, Y. Logic and the challenge of computer science. In *Current Trends in Theoretical Computer Science*. E. Börger, ed. Computer Science Press, 1988, 1–57.
8. Hella, L., Kolaitis, P., Luosto, K. Almost everywhere equivalence of logics in finite model theory. *Bull. Symbol. Logic*, 2 (1996), 422–443.
9. Hopcroft, J., Wong, J. Linear time algorithm for isomorphism of planar graphs. In *Proceedings of the 6th ACM Symposium on Theory of Computing*, 1974, 172–184.
10. Immerman, N. Relational queries computable in polynomial time (extended abstract). In *Proceedings of the 14th ACM Symposium on Theory of Computing*, 1982, 147–152.
11. Immerman, N. Expressibility as a complexity measure: Results and directions. In *Proceedings of the 2nd IEEE Symposium on Structure in Complexity Theory*, 1987, 194–202.
12. Immerman, N., Lander, E. Describing graphs: A first-order approach to graph canonization. In *Complexity Theory Retrospective*. A. Selman, ed. Springer-Verlag, Heidelberg, 1990, 59–81.
13. Laubner, B. Capturing polynomial time on interval graphs. In *Proceedings of the 25th IEEE Symposium on Logic in Computer Science*, 2010, 199–208.
14. Luks, E. Isomorphism of graphs of bounded valence can be tested in polynomial time. *J. Comput. Syst. Sci.*, 25 (1982), 42–65.
15. Matijasevič, J. Enumerable sets are diophantine. *Sov. Math. Dokl.*, 11 (1970), 345–357.
16. Miller, G.L. Isomorphism testing for graphs of bounded genus. In *Proceedings of the 12th ACM Symposium on Theory of Computing*, 1980, 225–235.
17. Otto, M. *Bounded Variable Logics and Counting—A Study in Finite Models*. Lecture Notes in Logic, vol. 9. Springer-Verlag, Berlin, 1997.
18. Ponomarenko, I.N. The isomorphism problem for classes of graphs that are invariant with respect to contraction. *Zap. Nauchn. Sem. Leningrad. Otdel. Mat. Inst. Steklov. (LOMI)*, 174 (Teor. Slozhn. Vychisl. 3):147–177, 182, 1988 (in Russian).
19. Robertson, N., Seymour, P. Graph minors I–XXIII. Appearing in *J. Combin. Theory Ser. B* since 1982.
20. Robertson, N., Seymour, P. Graph minors XVI. Excluding a non-planar graph. *J. Combin. Theory Ser. B*, 77 (1999), 1–27.
21. Robertson, N., Seymour, P. Graph minors XX. Wagner's conjecture. *J. Combin. Theory Ser. B*, 92 (2004), 325–357.
22. Vardi, M. The complexity of relational query languages. In *Proceedings of the 14th ACM Symposium on Theory of Computing*, 1982, 137–146.

Martin Grohe (grohe@informatik.hu-berlin.de), Humboldt-Universität Berlin, Germany.

© 2011 ACM 0001-0782/11/06 \$10.00



Association for
Computing Machinery

Advancing Computing as a Science & Profession



MentorNet®

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.

Technical Perspective

Data Analysis at Astonishing Speed

By Michael J. Franklin

THE IMPORTANCE OF data analysis has never been clearer. Companies on the Web compete on their ability to intuit their users' wants, needs, and goals by analyzing their online behavior in ever-increasing volumes. Globe-spanning scientific collaborations are exploring data-intensive questions at a scale limited only by the power of the computing resources they can harness. The popular press regularly carries stories of analytics-driven applications as varied as timing the purchase of airline tickets, building a cost-efficient major league baseball team, and understanding social phenomena.

While data analytics and business intelligence have long been active areas of research and development, the past decade has seen a dramatic increase in the scale and complexity of the problems being addressed. Since its inception, Google has been aggressively pushing the envelope in terms of data management technology. Over the years they have published a series of highly influential papers describing key aspects of their internal systems.

A recent addition to this series is the following paper on the Dremel system. Dremel is an interactive data analysis tool capable of leveraging thousands of machines to process data at a scale that is simply jaw-dropping given the current state of the art in the field. The system enables Google engineers to analyze billion-record data sets interactively and trillion-record data sets records while they make a macchiatone at a nearby espresso machine.

Large Web properties track billions of events per day, representing tens of terabytes of data. Storing such data is expensive but doable; providing meaningful access to that data, however, is another story. The reality in most organizations today is analysis turnaround times measured in hours or even days. Such batch-oriented delays may have been tolerable for the more leisurely decision cycles of the past, but are severely

limiting for today's fast pace of business and data acquisition. Furthermore, it is not simply a matter of latency. Moving from hours to seconds enables new modes of data exploration providing a qualitative change in the value that can be extracted from complex data sets.

Not surprisingly, Dremel's performance depends on massive parallelism. The authors point out that scanning 1TB in one second would require 10,000 commodity disk drives. But the main take-away from this paper is that simply throwing hardware at the problem is not sufficient. Rather, it is critical to deeply understand the structure of the data to be stored and how that data will be used.

Recent advances in database systems have exploited organization of data vertically along columns, rather than horizontally along rows, to reduce disk accesses through compression and selective retrieval. Dremel follows a similar philosophy but its task is complicated by the need to store irregular, nested data consisting of repeating fields, optional and variable fields, and structured sub-objects—all constructs that violate basic relational database principles.

Dremel describes an efficient, column-oriented structure for storing and sharding (that is, partitioning) such data, as well as a high-performance

Dremel is an interactive data analysis tool capable of leveraging thousands of machines to process data at a scale that is simply jaw-dropping.

mechanism for reconstructing data records stored in such a fashion. A number of other important techniques are addressed, including an execution model based on serving trees, query-result caching, replication, and the willingness to return partial answers to avoid waiting for the "stragglers" (slow tasks) that inevitably arise in a massively parallel system.

The real eye-opener, however, is the experimental section. Prior to reading this paper, I would not have thought it possible to obtain the low latencies over massive data sets demonstrated here, with processing throughputs in the range of 100 billion records per second. The fact that Googlers have been using this technology already for five years was a revelation.

In this regard, the paper raises some serious questions for computing systems researchers. One could reasonably ask if such technology is relevant only to the very few organizations that currently have data at this huge scale. Similarly, some have raised the question of whether those outside of such organizations, without easy access to massive computing resources, can even attempt to participate in such research.

The answer to the first question is obvious. One only needs to look through past issues of this publication and others to see how quickly the "bleeding edge" becomes commonplace in our field. The data volumes described in this paper will clearly become relevant to more organizations over time. The second question is more difficult. Looking at the paper objectively, I see numerous opportunities for optimizations and improvements, such as stream-oriented processing and more aggressive sampling. Arguably, those without easy access to thousands of machines should be particularly motivated to explore such innovative techniques.

In summary, Dremel is a very impressive system that extends our idea of what is possible today and what will be required in the future. As such, this paper makes compelling reading for anyone interested in the future of data management. 

Michael J. Franklin (franklin@cs.berkeley.edu) is a professor of computer science at the University of California, Berkeley and a founder and CTO of Truviso, Inc.

© 2011 ACM 0001-0782/11/06 \$10.00

Dremel: Interactive Analysis of Web-Scale Datasets

By Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, and Theo Vassilakis

Abstract

Dremel is a scalable, interactive ad hoc query system for analysis of read-only nested data. By combining multilevel execution trees and columnar data layout, it is capable of running aggregation queries over trillion-row tables in seconds. The system scales to thousands of CPUs and petabytes of data, and has thousands of users at Google. In this paper, we describe the architecture and implementation of Dremel, and explain how it complements MapReduce-based computing. We present a novel columnar storage representation for nested records and discuss experiments on few-thousand node instances of the system.

1. INTRODUCTION

Large-scale analytical data processing has become widespread in Web companies and across industries, not least due to low-cost storage that enabled collecting vast amounts of business-critical data. Putting this data at the fingertips of analysts and engineers has grown increasingly important; interactive response times often make a qualitative difference in data exploration, monitoring, online customer support, rapid prototyping, debugging of data pipelines, and other tasks.

Performing interactive data analysis at scale demands a high degree of parallelism. For example, reading a terabyte of compressed data from secondary storage in 1s would require more than 10,000 commodity disks. Similarly, CPU-intensive queries may need to run on thousands of cores to complete within seconds. At Google, massively parallel computing is done using shared clusters of commodity machines.⁵ A cluster typically hosts a multitude of distributed applications that share resources, have widely varying workloads, and run on machines with different hardware parameters. An individual worker in a distributed application may take much longer to execute a given task than others or may never complete due to failures or preemption by the cluster management system. Hence, dealing with stragglers and failures is essential for achieving fast execution and fault tolerance.

The data used in Web and scientific computing are often non-relational. Hence, a flexible data model is essential in these domains. Data structures used in programming languages, messages exchanged by distributed systems, structured documents, etc., lend themselves naturally to a *nested* representation. Normalizing and recombining such data at Web scale is usually prohibitive. A nested data model underlies most of the structured data processing at Google²² and reportedly at other major Web companies.

This paper describes a system called Dremel^a that supports interactive analysis of very large datasets over shared clusters of commodity machines. Unlike traditional databases, it is capable of operating on in situ nested data. In situ refers to the ability to access data “in place,” for example, in a distributed file system (like Google File System (GFS)¹⁴) or another storage layer (e.g., Bigtable⁹). Dremel can execute many queries over such data that would ordinarily require a sequence of MapReduce (MR¹²) jobs, but at a fraction of the execution time. Dremel is not intended as a replacement for MR and is often used in conjunction with it to analyze outputs of MR pipelines or rapidly prototype larger computations.

Dremel has been in production since 2006 and has thousands of users within Google. Multiple instances of Dremel are deployed in the company, ranging from tens to thousands of nodes. Examples of system usage include the following:

- Analysis of crawled Web documents
- Tracking install data for applications on Android Market
- Crash reporting for Google products
- OCR results from Google Books
- Spam analysis
- Debugging of map tiles on Google Maps
- Tablet migrations in managed Bigtable instances
- Results of tests run on Google’s distributed build system
- Disk I/O statistics for hundreds of thousands of disks
- Resource monitoring for jobs run in Google’s data centers
- Symbols and dependencies in Google’s codebase

Dremel builds on ideas from Web search and parallel DBMSs. First, its architecture borrows the concept of a serving tree used in distributed search engines.¹¹ Just like a Web search request, a query gets pushed down the tree and is rewritten at each step. The result of the query is assembled by aggregating the replies received from lower levels of the tree. Second, Dremel provides a high-level, SQL-like

^a Dremel is a brand of power tools that primarily rely on their speed as opposed to torque. We use this name for an internal project only.

The original version of this paper was published in VLDB 2010.

language to express ad hoc queries. In contrast to layers such as Pig¹⁹ and Hive,¹⁶ it executes queries natively without translating them into MR jobs.

Lastly, and importantly, Dremel uses a column-striped storage representation, which enables it to read less data from secondary storage and reduce CPU cost due to cheaper compression. Column stores have been adopted for analyzing relational data¹ but to the best of our knowledge have not been extended to nested data models. The columnar storage format that we present is supported by many data processing tools at Google, including MR, Sawzall,²¹ and FlumeJava.⁸

In this paper we make the following contributions:

- We describe a novel columnar storage format for nested data. We present algorithms for dissecting nested records into columns and reassembling them (Section 4).
- We outline Dremel's query language and execution. Both are designed to operate efficiently on column-striped nested data and do not require restructuring of nested records (Section 5).
- We show how execution trees used in Web search systems can be applied to database processing and explain their benefits for answering aggregation queries efficiently (Section 6).
- We present experiments on trillion-record, multi-terabyte datasets, conducted on system instances running on 1000–4000 nodes (Section 7).

This paper is structured as follows. In Section 2, we explain how Dremel is used for data analysis in combination with other data management tools. Its data model is presented in Section 3. The main contributions listed above are covered in Sections 4–8. Related work is discussed in Section 9. Section 10 is the conclusion.

2. BACKGROUND

We start by walking through a scenario that illustrates how interactive query processing fits into a broader data management ecosystem. Suppose that Alice, an engineer at Google, comes up with a novel idea for extracting new kinds of signals from Web pages. She runs an MR job that cranks through the input data and produces a dataset containing the new signals, stored in billions of records in the distributed file system. To analyze the results of her experiment, she launches Dremel and executes several interactive commands:

```
DEFINE TABLE t AS/path/to/data/*
SELECT TOP(signal1, 100), COUNT(*) FROM t
```

Her commands execute in seconds. She inspects the 100 most frequent signals returned by the query. She runs other queries, looking for ways to integrate her signals into Web search. Once she finds enough clues, she sets up a pipeline to process the incoming input data continuously and feeds it to another MR or a serving system. She formulates a few canned SQL queries that aggregate the results of her pipeline across various dimensions and adds them

to an interactive dashboard. Finally, she registers her new dataset in a catalog, so other engineers can locate and query it quickly.

The above scenario requires interoperability between the query processor and other data management tools. The first ingredient for that is a *common storage layer*. GFS¹⁴ is one such distributed storage layer widely used in the company. GFS uses replication to preserve the data despite faulty hardware and achieve fast response times in presence of stragglers. A high-performance storage layer is critical for in situ data management since it allows accessing the data without a time-consuming loading phase. As an added benefit, data in a file system can be conveniently manipulated using standard tools, for example, to transfer to another cluster, change access privileges, or identify a subset of data for analysis based on file names.

The second ingredient for building interoperable data management components is a *shared storage format*. Columnar storage proved successful for flat relational data but making it work for Google required adapting it to a nested data model. Figure 1 illustrates the main idea: all values of a nested field such as A.B.C are stored contiguously. Hence, A.B.C can be retrieved without reading A.E, A.B.D, etc. The challenge that we address is how to preserve all structural information and be able to reconstruct records from an arbitrary subset of fields. Next we discuss our data model and then turn to algorithms and query processing.

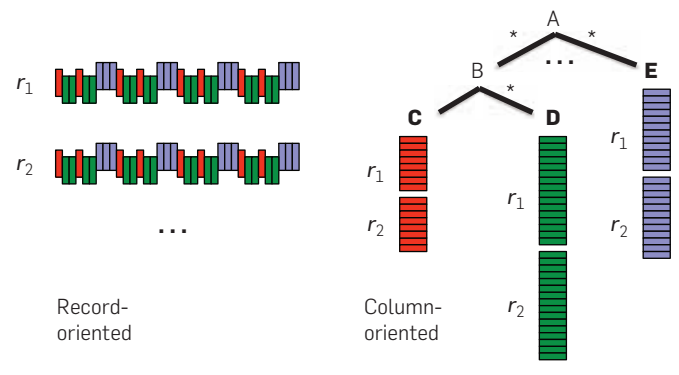
3. DATA MODEL

In this section, we present Dremel's data model and introduce the terminology used later. The data model originated in the context of distributed systems (which explains its name, "Protocol Buffers"²²) is used widely at Google and is available as an open source implementation. The data model is based on strongly typed nested records. Its abstract syntax is given by:

$$\tau = \text{dom} \mid \langle A_1 : \tau[*|?], \dots, A_n : \tau[*|?] \rangle$$

where τ is an atomic type or a record type. Atomic types in **dom** comprise integers, floating-point numbers, strings, etc. Records consist of one or multiple fields. Field i in a

Figure 1. Record-wise vs. columnar representation of nested data.



record has a name A_i and a multiplicity label. *Repeated* fields (label “*”) may occur multiple times in a record; the order of field occurrences is significant. *Optional* fields (label “?”) may be missing from the record. Otherwise, a field is *required*, that is, must appear exactly once.

To illustrate, see Figure 2. It depicts a schema that defines a record type Document, representing a Web document. The schema definition uses the concrete syntax from Protocol Buffers.²² A Document has a required integer DocId and optional Links, containing a list of Forward and Backward entries holding DocIds of other Web pages. A document can have multiple Names, which are different URLs by which the document can be referenced. A Name contains a sequence of Code and (optional) Country pairs. Figure 2 also shows two sample records, r_1 and r_2 , conforming to the schema. The record structure is outlined using indentation. We will use these sample records to explain the algorithms in the next sections. The fields defined in the schema form a tree hierarchy. The full *path* of a nested field is denoted using the usual dotted notation, for example, Name.Language.Code.

The nested data model backs a platform-neutral, extensible mechanism for serializing structured data at Google. Code generation tools produce bindings for programming languages such as C++ or Java. Cross-language interoperability is achieved using a standard binary on-the-wire representation of records, in which field values are laid out sequentially as they occur in the record. This way, an MR program written in Java can consume records from a data source exposed via a C++ library.

4. NESTED COLUMNAR STORAGE

As illustrated in Figure 1, our goal is to store all values of a given field consecutively to improve retrieval efficiency. However, columnar data might eventually be consumed by record-oriented tools such as MR. Therefore, we need a way to assemble records efficiently from any given subset of columns. In this section, we address the following challenges: lossless representation of record structure in a columnar format (Section 4.1), fast encoding (Section 4.2), and efficient record assembly (Section 4.3).

4.1. Repetition and definition levels

In Figure 2, we have seen two documents represented as records. Contrast those to Figure 3. It depicts the same data in a columnar format. The values of every field are stored sequentially as a separate stripe. For each value, we keep extra information, a *repetition level* and a *definition level* (abbreviated as *r* and *d* in the figure). This information encodes the structure of the records.

We explain our encoding using the example of Name.Language.Code in Figure 4. The right-hand side of the figure shows a flattened representation of records r_1 and r_2 obtained as follows. First, we strip away all fields except Name, Language, and Code. Second, we represent the stripped records as a list of root-to-leaf paths. The subscripts denote positions of the respective fields within their enclosing records.

A (repetition level, definition level) pair represents the *delta* between two consecutive paths p_{i-1} and p_i . Repetition

level encodes the length of the common prefix of p_{i-1} and p_i , while definition level encodes the length of p_i (or, alternatively, the length of p_i 's suffix). For example, the common prefix of the first two paths in Figure 4 is r_1 .Name₁ and has length 2.

Path lengths are encoded compactly as follows. The common prefix of two consecutive paths always ends on a repeated field, so we define the repetition level as the

Figure 2. Two sample nested records and their schema.

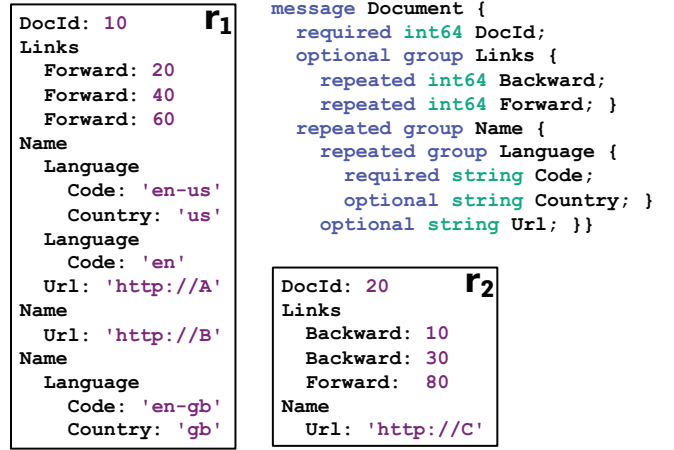


Figure 3. Column-striped representation of the data in Figure 2.

DocId	Name.Url	Links.Forward	Links.Backward
value r d	value r d	value r d	value r d
10 0 0	http://A 0 2	20 0 2	NULL 0 1
20 0 0	http://B 1 2	40 1 2	10 0 2
	NULL 1 1	60 1 2	30 1 2
	http://C 0 2	80 0 2	

Name.Language.Code	Name.Language.Country
value r d	value r d
en-us 0 2	us 0 3
en 2 2	NULL 2 2
NULL 1 1	NULL 1 1
en-gb 1 2	gb 1 3
NULL 0 1	NULL 0 1

Figure 4. Repetition and definition levels: delta between paths.

Name.Language.Code	
value r d	
en-us 0 2	r_1 .Name ₁ .Language ₁ .Code: 'en-us'
en 2 2	r_1 .Name ₁ .Language ₂ .Code: 'en'
NULL 1 1	r_1 .Name ₂
en-gb 1 2	r_1 .Name ₃ .Language ₁ .Code: 'en-gb'
NULL 0 1	r_2 .Name ₁

number of repeated fields in the common prefix (including the first path element identifying the record). The definition level specifies the number of optional and repeated fields in the path (excluding the first path element). We do not count required fields since they are always present. A definition level smaller than the maximal number of repeated and optional fields in a path denotes a NULL. For example, the maximum definition level of Name.Language.Code is 2.

The encoding outlined above preserves the record structure losslessly. We omit the proof for space reasons.

Tablet Layout: A table is stored as a set of tablets. A *tablet* is a self-contained horizontal partition of the table. Figure 5 illustrates the layout of a tablet. In addition to the actual data, the tablet contains the schema and extra meta-data that includes specification of keys, sorting order, value ranges, etc.

Each column is stored as a set of blocks. Each block contains the repetition and definition levels (henceforth, simply called levels) and compressed field values. NULLs are not stored explicitly as they are determined by the definition levels. Definition levels are not stored for values that are always defined. Similarly, repetition levels are stored only if required; for example, definition level 0 implies repetition level 0, so the latter can be omitted. In fact, in Figure 3, no levels are stored for DocId. Levels are packed as bit sequences. We only use as many bits as necessary; for example, if the maximum definition level is 3, we use 2 bits per definition level.

4.2. Splitting records into columns

Above we presented an encoding of the record structure in a columnar format. The next challenge we address is how to produce column stripes with repetition and definition levels efficiently.

The algorithm for computing repetition and definition levels is given in Melnik et al.¹⁸ The algorithm recurs into the record structure and computes the levels for each field value. As illustrated earlier, repetition and definition levels may need to be computed even if field values are missing. Many datasets used at Google are sparse; it is not uncommon to have a schema with thousands of fields, only a hundred of which are used in a given record. Hence, we try to process missing fields as cheaply as possible. To produce column stripes, we create a tree of *field writers*, whose structure matches the field hierarchy in the schema. The basic

Figure 5. Tablet layout.

Schema			Metadata (keys, order, ranges, ...)			
column ₁			...	column _n		
block ₁			...	block _m		
values	r	d				

idea is to update field writers only when they have their own data, and not try to propagate parent state down the tree unless absolutely necessary.

4.3. Record assembly

Assembling records from columnar data efficiently is critical for record-oriented data processing tools (e.g., MR). Given a subset of fields, our goal is to reconstruct the original records as if they contained just the selected fields, with all other fields stripped away. The key idea is we create a finite state machine (FSM) that reads the field values and levels for each field, and appends the values sequentially to the output records. An FSM state corresponds to a field reader for each selected field. State transitions are labeled with repetition levels. Once a reader fetches a value, we look at the next repetition level to decide what next reader to use. The FSM is traversed from the start to end state once for each record.

Figure 6 shows an FSM that reconstructs the complete records in our running example. The start state is DocId. Once a DocId value is read, the FSM transitions to Links.Backward. After all repeated Backward values have been drained, the FSM jumps to Links.Forward, etc.

If only a subset of fields needs to be retrieved, we construct a simpler FSM that is cheaper to execute. Figure 7 depicts an FSM for reading the fields DocId and Name.Language.Country. The figure shows the output records s_1

Figure 6. Complete record assembly automaton. Edges are labeled with repetition levels.

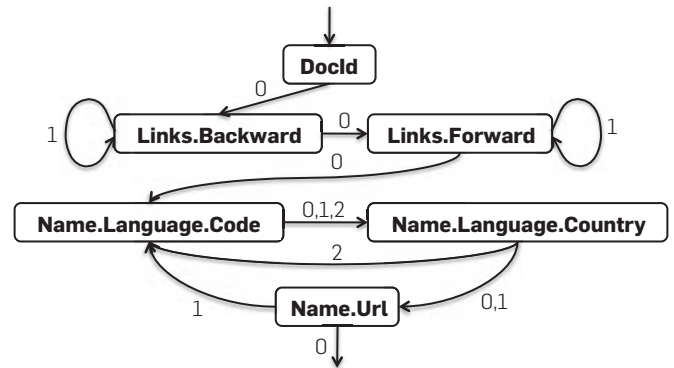
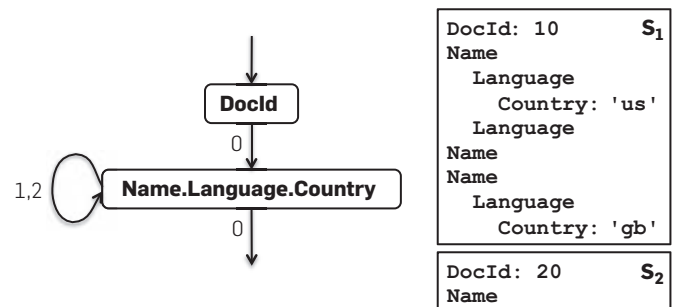


Figure 7. Automaton for assembling records from two fields, and the records it produces.



and s_2 produced by the automaton. Notice that our encoding and the assembly algorithm preserve the enclosing structure of the field Country. This is important for applications that need to access, for example, the Country appearing in the first Language of the second Name. In XPath, this would correspond to the ability to evaluate expressions like `/Name[2]/Language[1]/Country`.

The details of record assembly and FSM construction are in Melnik et al.¹⁸

5. QUERY LANGUAGE

Dremel's query language is based on SQL and is designed to be efficiently implementable on columnar nested storage. Defining the language formally is out of scope of this paper; instead, we illustrate its flavor. Each SQL statement (and algebraic operators it translates to) takes as input one or multiple nested tables and their schemas and produces a nested table and its output schema. Figure 8 depicts a sample query that performs projection, selection, and within-record aggregation. The query is evaluated over the table $t = \{r_1, r_2\}$ from Figure 2. The fields are referenced using path expressions. The query produces a nested result although no record constructors are present in the query.

To explain what the query does, consider the selection operation (the WHERE clause). Think of a nested record as a labeled tree, where each label corresponds to a field name. The selection operator prunes away the branches of the tree that do not satisfy the specified conditions. Thus, only those nested records are retained where Name.Url is defined and starts with http. Next, consider projection. Each scalar expression in the SELECT clause emits a value at the same level of nesting as the most repeated input field used in that expression. So, the string concatenation expression emits Str values at the level of Name.Language.Code in the input schema. The COUNT expression illustrates within-record aggregation. The aggregation is done WITHIN each Name subrecord and emits the number of occurrences of Name.Language.Code for each Name as a nonnegative 64-bit integer (uint64).

The language supports nested subqueries, inter- and intra-record aggregation, top-k, joins, user-defined functions, etc.; some of these features are exemplified in the experimental section.

Figure 8. Sample query, its result, and output schema.

```
SELECT DocId AS Id,
       COUNT(Name.Language.Code) WITHIN Name AS Cnt,
       Name.Url + ',' + Name.Language.Code AS Str
FROM t
WHERE REGEXP(Name.Url, '^http') AND DocId < 20;
```

<pre>Id: 10 Name Cnt: 2 Language Str: 'http://A,en-us' Str: 'http://A,en' Name Cnt: 0</pre>	<pre>t₁</pre>	<pre>message QueryResult { required int64 Id; repeated group Name { optional uint64 Cnt; repeated group Language { optional string Str; } } }</pre>
---	--------------------------	---

6. QUERY EXECUTION

We discuss the core ideas in the context of a read-only system, for simplicity. Many Dremel queries are one-pass aggregations; therefore, we focus on explaining those and use them for experiments in the next section. We defer a detailed discussion of joins, indexing, updates, etc., to future work.

Tree Architecture: Dremel executes queries using a serving tree (see Figure 9). Its purpose is twofold:

1. To parallelize query scheduling and aggregation
2. To provide fault tolerance and deal with stragglers

A root server receives incoming queries, reads metadata from the tables, and routes the queries to the next level in the serving tree. The leaf servers communicate with the storage layer or access the data on local disk.

Consider a simple aggregation query below:

```
SELECT A, COUNT(B) FROM T GROUP BY A
```

When the root server receives the above query, it determines all tablets that comprise T and rewrites the query as follows:

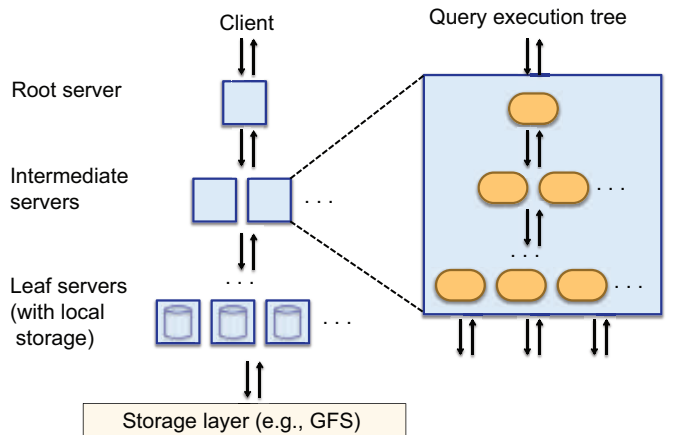
```
SELECT A, SUM(c) FROM (R11 UNION ALL... Rn1) GROUP BY A
```

Tables $R_1^1 \dots R_n^1$ are the results of queries sent to the nodes $1, \dots, n$ at level 1 of the serving tree:

```
Ri1 = SELECT A, COUNT(B) AS c FROM Ti1 GROUP BY A
```

T_i^1 is a disjoint partition of tablets in T processed by server i at level 1. Each serving level performs a similar rewriting. Ultimately, the queries reach the leaves, which scan the tablets in T in parallel. On the way up, intermediate servers perform a parallel aggregation of partial results. The execution model presented above is well suited for aggregation queries

Figure 9. System architecture and execution inside a server node.



returning small- and medium-sized results, which are a very common class of interactive queries. This model also works well for computing approximate results using known one-pass algorithms, such as those for top-k and count-distinct (e.g., see Bar-Yossef et al.⁴).

Beyond One-Pass Aggregation: Dremel supports query processing mechanisms that go beyond one-pass aggregation. These mechanisms are designed to leverage the serving tree architecture, too. For example, one way to execute a query that joins a large partitioned table with small user-defined tables is by sending a copy of the small tables to each leaf server. This strategy is referred to as broadcast join. The serving tree supports such queries efficiently by broadcasting the small tables in parallel down the tree.

As another example, joins that repartition the data (similarly to the “shuffle” phase of MR) maintain a significant amount of distributed execution state. The serving tree helps aggregate their execution state efficiently. Last but not least, SELECT-INTO operations persist query results as new tables in the DFS. The serving tree monitors distributed writes and ensures successful completion. We found that serving trees are a useful building block that complements existing distributed query processing algorithms.

Query Dispatcher: Dremel is a multiuser system, that is, usually several queries are executed simultaneously. A query dispatcher schedules queries based on their priorities and balances the load. Its other important role is to provide *fault tolerance* when one server becomes much slower than others or a tablet replica becomes unreachable.

The amount of data processed in each query is often larger than the number of processing units available for execution, which we call *slots*. A slot corresponds to an execution thread on a leaf server. For example, a system of 3,000 leaf servers each using 8 threads has 24,000 slots. So, a table spanning 100,000 tablets can be processed by assigning about 5 tablets to each slot. During query execution, the query dispatcher computes a histogram of tablet processing times. If a tablet takes a disproportionately long time to process, it reschedules it on another server. Some tablets may need to be redispached multiple times.

The leaf servers read stripes of nested data in columnar representation. The blocks in each stripe are prefetched asynchronously; the read-ahead cache typically achieves hit rates of 95%. Tablets are usually three-way replicated. When a leaf server cannot access one tablet replica, it falls over to another replica.

The query dispatcher honors a parameter that specifies the minimum percentage of tablets that must be scanned before returning a result. As we demonstrate shortly, setting such parameter to a lower value (e.g., 98% instead of 100%) can often speed up execution significantly, especially when using smaller replication factors.

Each server has an internal execution tree, as depicted on the right-hand side of Figure 9. The internal tree corresponds to a physical query execution plan, including evaluation of scalar expressions. Optimized, type-specific code is generated for most scalar functions. An execution plan

for project-select-aggregate queries consists of a set of iterators that scan input columns in lockstep and emit results of aggregates and scalar functions annotated with the correct repetition and definition levels, bypassing the record assembly entirely during query execution. For details, see Melnik et al.¹⁸

7. EXPERIMENTS

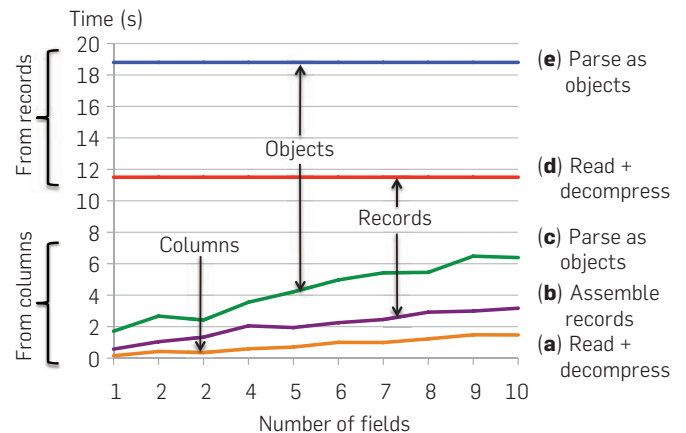
In this section, we evaluate Dremel’s performance on several datasets used at Google and examine the effectiveness of columnar storage for nested data. The properties of the datasets are summarized in Figure 10. In uncompressed, non-replicated form, they occupy about a petabyte of space. All tables are three-way replicated, except one two-way replicated table, and contain from 100K to 800K tablets. We start by examining data access characteristics on a single machine, then show how columnar storage benefits MR execution, and finally focus on Dremel’s performance. The experiments were conducted on system instances running in two data centers next to many other applications, during regular business operation. Unless specified otherwise, execution times were averaged across five runs. Table and field names used below are anonymized.

Local Disk: In the first experiment, we examine performance tradeoffs of columnar vs. record-oriented storage, scanning a 1GB fragment of table T_1 containing about 300K rows (see Figure 11). The data is stored on a local disk and

Figure 10. Datasets used in the experimental study.

Table name	Number of records	Size (unrepl., compressed)	Number of fields	Data center	Repl. factor
T_1	85 billion	87TB	270	A	3×
T_2	24 billion	13TB	530	A	3×
T_3	4 billion	70TB	1200	A	3×
T_4	1+ trillion	105TB	50	B	3×
T_5	1+ trillion	20TB	30	B	2×

Figure 11. Performance breakdown when reading from a local disk (300K-record fragment of Table T_1).



takes about 375MB in compressed columnar representation. The record-oriented format uses heavier compression yet yields about the same size on disk. The experiment was done on a dual-core Intel machine with a disk providing 70MB/s read bandwidth. All reported times are cold; OS cache was flushed prior to each scan.

The figure shows five graphs, illustrating the time it takes to read and uncompress the data, and assemble and parse the records, for a subset of the fields. Graphs (a)–(c) outline the results for columnar storage. Each data point in these graphs was obtained by averaging the measurements over 30 runs, in each of which a set of columns of a given cardinality was chosen at random. Graph (a) shows reading and decompression time. Graph (b) adds the time needed to assemble nested records from columns. Graph (c) shows how long it takes to parse the records into strongly typed C++ data structures. Graphs (d) and (e) depict the time for accessing the data from record-oriented storage, with or without parsing.

The main takeaways of this experiment are the following: when few columns are read, the gains of columnar representation are of about an order of magnitude. Retrieval time for columnar nested data grows linearly with the number of fields. Record assembly and parsing are expensive, each potentially doubling the execution time. We observed similar trends on other datasets. A natural question to ask is when record-wise storage starts outperforming columnar storage. In our experience, the crossover point often lies at dozens of fields but it varies across datasets and depends on whether or not record assembly is required.

MR and Dremel: Next we illustrate an MR and Dremel execution on columnar vs. record-oriented data. We consider a case where a single field is accessed, that is, the performance gains are most pronounced. Execution times for multiple columns can be extrapolated using the results of Figure 11. In this experiment, we count the average number of terms in a field `txtField` of table T_1 . MR execution is done using the following Sawzall²¹ program:

```
numRecs: table sum of int;
numWords: table sum of int;
emit numRecs <- 1;
emit numWords <- CountWords (input.txtField);
```

The number of records is stored in the variable `numRecs`. For each record, `numWords` is incremented by the number of terms in `input.txtField` returned by the `CountWords` function. After the program runs, the average term frequency can be computed as `numWords/numRecs`. In SQL, this computation is expressed as:

```
Q1: SELECT SUM(CountWords(txtField))/COUNT(*)
FROM T1
```

Figure 12 shows the execution times of two MR jobs and Dremel on a logarithmic scale. Both MR jobs are run on 3000 workers. Similarly, a 3000-node Dremel instance is

used to execute Query Q_1 . Dremel and MR-on-columns read about 0.5TB of compressed columnar data vs. 87TB read by MR-on-records. As the figure illustrates, MR gains an order of magnitude in efficiency by switching from record-oriented to columnar storage (from hours to minutes). Another order of magnitude (from minutes to seconds) is achieved by using Dremel, which eliminates the overheads of launching MR jobs, scheduling half a million tasks, and assembling records.

Serving Tree Topology: In the next experiment, we show the impact of the serving tree depth on query execution times. We consider two GROUP BY queries on Table T_2 , which has 24 billion nested records. Each record has a repeated field item containing a numeric amount. The field `item.amount` occurs about 40 billion times. The first query sums up the item amount by country:

```
Q2: SELECT country, SUM(item.amount) FROM T2
GROUP BY country
```

It returns a few hundred records and reads roughly 60GB of compressed data. The second query performs a GROUP BY on a text field domain with a selection condition. It reads about 180GB and produces around 1.1 million distinct domains:

```
Q3: SELECT domain, SUM(item.amount) FROM T2
WHERE domain CONTAINS '.net'
GROUP BY domain
```

Figure 12. MR and Dremel execution on columnar vs. record-oriented storage (3000 nodes, 85 billion records).

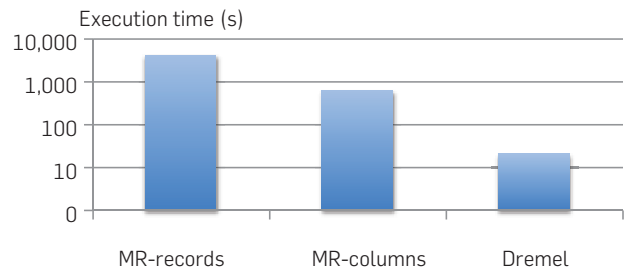


Figure 13. Execution time as a function of serving tree levels for two aggregation queries on T_2 .

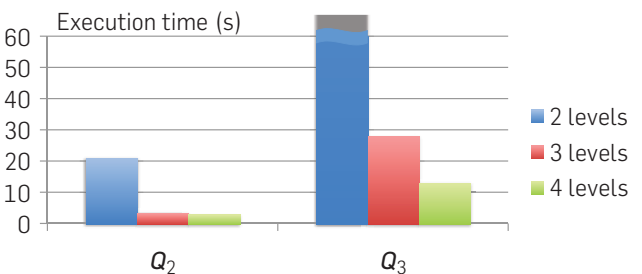


Figure 13 shows the execution times for each query as a function of the server topology. In each topology, the number of leaf servers is kept at 2900 to achieve the same cumulative scan speed. In the two-level topology (1:2900), a single root server communicates directly with the leaf servers. For three levels, we use a 1:100:2900 setup, that is, an extra level of 100 intermediate servers. The four-level topology is 1:10:100:2900.

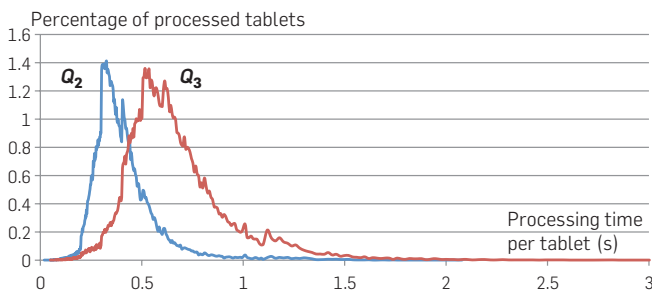
The experiment illustrates that aggregations returning many groups benefit from deeper serving trees. Using two levels is not very effective since the root server needs to aggregate near-sequentially the results received from thousands of nodes. Adding a fourth level halves the execution time of Q_3 due to increased parallelism but does not benefit Q_2 , which returns a small result.

Per-Tablet Histograms: To drill deeper into what happens during query execution consider Figure 14. The figure shows how fast tablets get processed by the leaf servers for a specific run of Q_2 and Q_3 . The time is measured starting at the point when a tablet got scheduled for execution in an available slot, that is, excludes the time spent waiting in the job queue. This measurement methodology factors out the effects of other queries that are executing simultaneously. The area under each histogram corresponds to 100%. As the figure indicates, 99% of Q_2 (or Q_3) tablets are processed under 1 s (or 2 s).

Within-Record Aggregation: As another experiment, we examine the performance of Query Q_4 run on Table T_3 . The query illustrates within-record aggregation: it counts all records where the sum of a.b.c.d values occurring in the record are larger than the sum of a.b.p.q.r values. The fields repeat at different levels of nesting. Due to column striping, only 13GB (out of 70TB) are read from disk and the query completes in 15 s. Without support for nesting, running this query on T_3 would be grossly expensive.

```
Q4: SELECT COUNT(c1 > c2) FROM
      (SELECT SUM(a.b.c.d) WITHIN RECORD AS c1,
              SUM(a.b.p.q.r) WITHIN RECORD AS c2
       FROM T3)
```

Figure 14. Histograms of processing times.



Scalability: The following experiment illustrates the scalability of the system on a trillion-record table. Query Q_5 shown below selects top-20 aid's and their number of occurrences in Table T_4 . The query scans 4.2TB of compressed data.

```
Q5: SELECT TOP(aid, 20), COUNT(*) FROM T4
      WHERE bid = {value1} AND cid = {value2}
```

The query was executed using four configurations of the system, ranging from 1000 to 4000 nodes. The execution times are in Figure 15. In each run, the total expended CPU time is nearly identical, at about 300K seconds, whereas the user-perceived time decreases near-linearly with the growing size of the system. This result suggests that a larger system can be just as effective in terms of resource usage as a smaller one, yet allows faster execution.

Stragglers: Our last experiment shows the impact of stragglers. Query Q_6 below is run on a trillion-row table T_5 . In contrast to the other datasets, T_5 is two-way replicated. Hence, the likelihood of stragglers slowing the execution is higher since there are fewer opportunities to reschedule the work.

```
Q6: SELECT COUNT(DISTINCT a) FROM T5
```

Query Q_6 reads over 1TB of compressed data. The compression ratio for the retrieved field is about 10. As indicated in Figure 16, the processing time for 99% of the

Figure 15. Scaling the system from 1000 to 4000 nodes using a top-k query Q_5 on a trillion-row table T_4 .

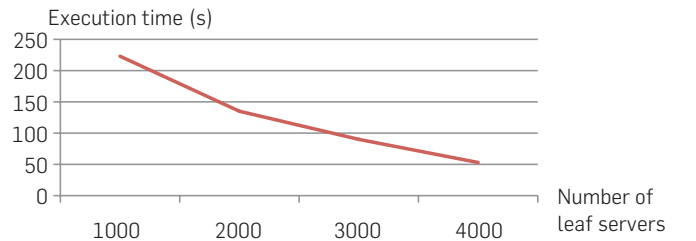
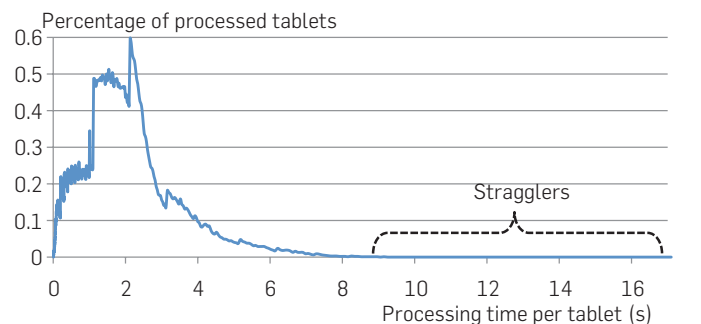


Figure 16. Query Q_6 on T_5 illustrating stragglers at 2× replication.



tablets is below 5 s per tablet per slot. However, a small fraction of the tablets take a lot longer, slowing down the query response time from less than a minute to several minutes, when executed on a 2500 node system. The next section summarizes our experimental findings and the lessons we learned.

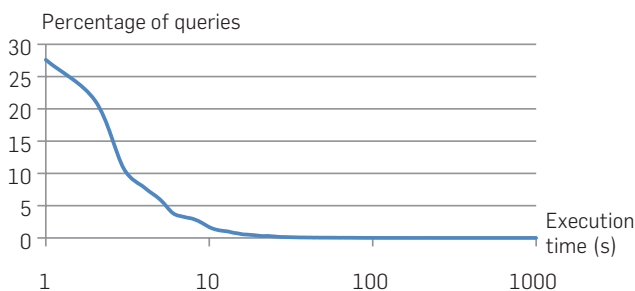
8. OBSERVATIONS

Dremel scans quadrillions of records per month. Figure 17 shows the query response time distribution in a typical monthly workload of one Dremel system, on a logarithmic scale. As the figure indicates, most queries are processed under 10 s, well within the interactive range. Some queries achieve a scan throughput close to 100 billion records per second on a shared cluster, and even higher on dedicated machines. The experimental data presented above suggests the following observations:

- Scan-based queries can be executed at interactive speeds on disk-resident datasets of up to a trillion records.
- Near-linear scalability in the number of columns and servers is achievable for systems containing thousands of nodes.
- MR can benefit from columnar storage just like a DBMS.
- Extreme-scale parallel DBMSs may benefit from the serving tree architecture just like search engines.
- Record assembly and parsing are expensive. Software layers (beyond the query processing layer) need to be optimized to directly consume column-oriented data.
- MR and query processing can be used in a complementary fashion; one layer's output can feed another's input.
- In a multiuser environment, a larger system can benefit from economies of scale while offering a qualitatively better user experience.
- The bulk of a Web-scale dataset can be scanned fast. Getting to the last few percent within tight time bounds is hard.

Dremel's codebase is dense; it comprises fewer than 100K lines of C++, Java, and Python code. The first version was built by Andrey Gubarev as a 20% project.

Figure 17. Query response time distribution in a monthly workload.



9. RELATED WORK

The MR¹² framework was designed to address the challenges of large-scale computing in the context of long-running batch jobs. Like MR, Dremel provides fault-tolerant execution, a flexible data model, and in situ data processing capabilities. The success of MR led to a wide range of third-party implementations (notably open-source Hadoop¹⁵), and a number of hybrid systems that combine parallel DBMSs with MR, offered by vendors like Aster, Cloudera, Greenplum, and Vertica. HadoopDB,³ is a research system in this hybrid category. Recent articles^{13, 23} contrast MR and parallel DBMSs. Our work emphasizes the complementary nature of both paradigms.

Dremel is designed to operate at scale. Although it is conceivable that parallel DBMSs can be made to scale to thousands of nodes, we are not aware of any published work or industry reports that attempted that. Neither are we familiar with prior literature studying MR on columnar storage.

Our columnar representation of nested data builds on ideas that date back several decades: separation of structure from content and transposed representation. A recent review of work on column stores, including compression and query processing, can be found in Abadi et al.¹ Many commercial DBMSs support storage of nested data using XML (e.g., O'Neil et al.²⁰). XML storage schemes attempt to separate the structure from the content but face more challenges due to the flexibility of the XML data model. One system that uses columnar XML representation is XMill.¹⁷ XMill is a compression tool. It stores the structure for all fields combined and is not geared for selective retrieval of columns.

The data model used in Dremel is a variation of the complex value models and nested relational models discussed in Abiteboul et al.² Dremel's query language builds on the ideas from Colby,¹⁰ which introduced a language that avoids restructuring when accessing nested data. In contrast, restructuring is usually required in XQuery and object-oriented query languages, for example, using nested for-loops and constructors. We are not aware of practical implementations of Colby.¹⁰ A recent SQL-like language for nested data is Pig Latin.¹⁹ Other systems for parallel data processing include Scope⁷ and DryadLINQ,²⁴ and are discussed in more detail in Chambers et al.⁸

10. CONCLUSION

We presented Dremel, a distributed system for interactive analysis of large datasets. Dremel is a custom, scalable data management solution built from simpler components. It complements the MR paradigm. We discussed its performance on trillion-record, multiterabyte datasets of real data. The system is widely used at Google and serves as the foundation of BigQuery,⁶ a product launched in preview mode. We outlined the key aspects of Dremel, including its storage format, query language, and execution. In the future, we plan to cover in more depth such areas as formal algebraic specification, joins, extensibility mechanisms, etc.

Acknowledgment

Dremel has benefited greatly from the input of many engineers and interns at Google, in particular Craig Chambers, Ori Gershoni, Rajeev Byrissetti, Leon Wong, Erik Hendriks, Erika Rice Scherpelz, Charlie Garrett, Idan Avraham, Rajesh Rao, Andy Kreling, Li Yin, Madhusudan Hosaagrahara, Dan Belov, Brian Bershad, Lawrence You, Rongrong Zhong, Meelap Shah, Nathan Bales, Ju-yi Kuo, Ovidiu Platon, Nick Kline, Matthew Weaver, Dan Delorey, and Jinyuan Li. We thank Gerhard Weikum for valuable improvement suggestions on the *Communications* article. 

References

1. Abadi, D.J., Boncz, P.A., Harizopoulos, S. Column-oriented database systems. *VLDB* 2, 2 (2009).
2. Abiteboul, S., Hull, R., and Vianu, V. *Foundations of Databases*. Addison Wesley, Reading, PA, 1995.
3. Abouzeid, A., Bajda-Pawlikowski, K., Abadi, D.J., Rasin, A., Silberschatz, A. HadoopDB: An architectural hybrid of MapReduce and DBMS technologies for analytical workloads. *VLDB* 2, 1 (2009).
4. Bar-Yossef, Z., Jayram, T.S., Kumar, R., Sivakumar, D., Trevisan, L. Counting distinct elements in a data stream. In *RANDOM*, 2002, 1–10.
5. Barroso, L.A., Hölzle, U. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan & Claypool Publishers, 2009.
6. BigQuery. <http://code.google.com/apis/bigquery>.
7. Chaiken, R., Jenkins, B., Larson, P.-A., Ramsey, B., Shakib, D., Weaver, S., Zhou, J. SCOPE: Easy and efficient parallel processing of massive data sets. *VLDB* 1, 2 (2008).
8. Chambers, C., Raniwala, A., Perry, F., Adams, S., Henry, R., Bradshaw, R., Weizenbaum, N. FlumeJava: Easy, efficient data-parallel pipelines. In *PLDI*, 2010.
9. Chang, F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T., Fikes, A., Gruber, R. Bigtable: A distributed storage system for structured data. In *OSDI*, 2006.
10. Colby, L.S. A recursive algebra and query optimization for nested relations. In *SIGMOD*, 1989.
11. Dean, J., Challenges in building large-scale information retrieval systems: Invited talk. In *WSDM*, 2009.
12. Dean, J., Ghemawat, S. MapReduce: Simplified data processing on large clusters. In *OSDI*, 2004.
13. Dean, J., Ghemawat, S. MapReduce: A Flexible data processing tool. *Commun. ACM* 53, 1 (2010).
14. Ghemawat, S., Gobioff, H., Leung, S.-T. The Google File System. In *SOSP*, 2003.
15. Hadoop Apache Project. <http://hadoop.apache.org>.
16. Hive. <http://wiki.apache.org/hadoop/Hive>, 2009.
17. Liefke, H., Suciu, D. XMill: An efficient compressor for XML data. In *SIGMOD*, 2000.
18. Melnik, S., Gubarev, A., Long, J.J., Romer, G., Shivakumar, S., Tolton, M., Vassilakis, T. Dremel: Interactive analysis of web-scale datasets. *PVLDB* 3, 1 (2010).
19. Olston, C., Reed, B., Srivastava, U., Kumar, R., Tomkins, A. Pig Latin: A not-so-foreign language for data processing. In *SIGMOD*, 2008.
20. O'Neil, P.E., O'Neil, E.J., Pal, S., Cseri, I., Schaller, G., Westbury, N. ORDPATHS: Insert-friendly XML node labels. In *SIGMOD*, 2004.
21. Pike, R., Dorward, S., Griesemer, R., Quinlan, S. Interpreting the data: Parallel analysis with Sawzall. *Sci. Program.* 13, 4 (2005).
22. Protocol Buffers: Developer Guide. Available at <http://code.google.com/apis/protocolbuffers/docs/overview.html>.
23. Stonebraker, M., Abadi, D., DeWitt, D.J., Madden, S., Paulson, E., Pavlo, A., Rasin, A., MapReduce and parallel DBMSs: Friends or foes? *Commun. ACM* 53, 1 (2010).
24. Yu, Y., Isard, M., Fetterly, D., Budiu, M., Erlingsson, U., Gunda, P.K., Currey, J. DryadLINQ: A system for general-purpose distributed data-parallel computing using a high-level language. In *OSDI*, 2008.

Sergey Melnik (melnik@google.com), Google, Inc.

Andrey Gubarev (andrey@google.com), Google, Inc.

Jing Jing Long (jlong@google.com), Google, Inc.

Geoffrey Romer (gromer@google.com), Google, Inc.

Shiva Shivakumar (shiva@google.com, shiva@cs.stanford.edu), Google, Inc.

Matt Tolton (mtolton@google.com), Google, Inc.

Theo Vassilakis (theov@google.com), Google, Inc.

© 2011 ACM 0001-0782/11/06 \$10.00

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 2011. (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

CAREERS

Jacksonville University **Visiting Assistant Professor** **of Computing Science**

Begins August 2011. Expertise in programming, web app. development & related internet technologies. *Send CV, transcripts, 3 current letters of support. Application screening begins immediately.* JU actively seeks application from individuals with diverse backgrounds & experiences. Master's degree required, Ph.D. preferred. Contact: CS Department, cssearch@ju.edu

Old Dominion University **Assistant Professor**

The College of Sciences at Old Dominion University is undergoing a major expansion to increase research activities and to support the recruiting of outstanding graduate and undergraduate students. The Department of Computer Science has recently hired a new chair and five new faculty members and is looking to strengthen the Medical Image Computing research group with an

Assistant Professor in this area. The position is available as early as Fall, 2011 but a start date of Spring or Fall, 2012 is negotiable.

The candidate must demonstrate the ability to build strong a research program and perform quality teaching. A PhD in computer science or related area is required. The Department is particularly interested in attracting an investigator who can interact and collaborate with researchers in existing departmental research groups in Bioinformatics and Computational Biology, Parallel and High Performance Computing, Wireless Networks, Data Mining, Digital Libraries, or Software Environments and those who can enable further interdisciplinary research and inter-institutional collaborations.

The faculty in the Department of Computer Science is well supported by state funds, peer-reviewed grants from NSF, NIH, Library of Congress, NASA, DOD, DOE and other funding agencies and endowment income. The department offers competitive salaries and start-up packages. The Department has a very strong graduate program that awards both MS and PhD degrees and currently has more than 100 graduate students.

Old Dominion University (www.odu.edu) is a state supported, research intensive institution enrolling more than 24,000 students of which 6,000 are graduate students. Norfolk is a culturally rich, historic city and a major international maritime center in a metropolitan area of over 1.5 million people. Eastern Virginia Medical School, NASA Langley Research Center, Thomas Jefferson National Accelerator Facility, and many other government and research facilities are in near vicinity.

Interested candidates should submit a curriculum vitae, a statement of research activities and future research plans, contact information for four references and a statement of teaching philosophy. Electronic applications are preferred and should be sent to searchcommittee@cs.odu.edu. Paper applications can be sent to:

Search Committee
Department of Computer Science
Old Dominion University
Norfolk, VA 23529-0162

Review of applicants will begin immediately and continue until the position is filled.

Old Dominion University is an affirmative action, equal opportunity institution and requires compliance with the Immigration Reform and Control Act of 1986.

Princeton University **Part-Time and Full-Time Lecturer Positions in** **Computer Science**

Part- and full-time Lecturer positions. The Department of Computer Science seeks applications from outstanding teachers to assist the faculty in teaching our introductory course sequence. There is no application deadline. Candidates should have an exceptional record of classroom instruction and curricular innovation. The primary requirements of the job are to teach recitation sections and to participate in overall management of the introductory sequence. Other responsibilities include supervising graduate student teaching assistants and developing and maintaining online curricular material, classroom demonstrations, and laboratory exercises. An advanced degree in computer science, or related field, is required. PhD preferred. Lecturers are appointed to one-year terms, renewable for up to six years; these may be full- or part-time appointments depending upon departmental need.

Princeton University is an equal opportunity employer and complies with applicable EEO and affirmative action regulations. You may apply online, by submitting a letter of application, resume and names of three references and self-identify at https://jobs.princeton.edu/applicants/jsp/shared/search/Search_css.jsp

Requisition Number: 0110199.



ADVERTISING IN CAREER OPPORTUNITIES

How to Submit a Classified Line Ad: Send an e-mail to acmm mediasales@acm.org. Please include text, and indicate the issue/or issues where the ad will appear, and a contact name and number.

Estimates: An insertion order will then be e-mailed back to you. The ad will be typeset according to CACM guidelines. NO PROOFS can be sent. Classified line ads are NOT commissionable.

Rates: \$325.00 for six lines of text, 40 characters per line. \$32.50 for each additional line after the first six. The MINIMUM is six lines.

Deadlines: 20th of the month/2 months prior to issue date. For latest deadline info, please contact:

acmm mediasales@acm.org

Career Opportunities Online: Classified and recruitment display ads receive a free duplicate listing on our website at:

<http://jobs.acm.org>

Ads are listed for a period of 30 days.

For More Information Contact:

**ACM Media Sales
at 212-626-0686 or
acmm mediasales@acm.org**

ACM Transactions on 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

University of Florida Professor and Endowed Chair

University of Florida, College of Engineering invites applications for the endowed position of Intel/Charles E. Young Leadership Chair in Information Technology. The Intel Foundation established this endowment to honor Dr. Charles E. Young, who served as president of the University of Florida from 1999-2004. The ideal candidate will be a worldwide recognized scientific leader in fundamental and applied IT research with a proven record of excellence in research, education, funded large-scale activities and team building.

This position will be at the full professor level and will be tenured in the Dept. of Electrical and Computer Engineering. Responsibilities will include research, teaching, and service. Successful applicants are expected to be dynamic and visionary and to lead teams of faculty in significant funded research activity. Areas include classical core subjects in IT research as well as emerging important domains such as cybersecurity, cyberphysical systems, virtual reality/simulation, autonomous and adaptable systems, distributed sensing and processing systems, health and medical systems, green/sustainable computing, homeland security. The College of Engineering has excellent ties with industry as evidenced by two NSF Industry-University Cooperative Research Centers in the areas of Autonomic Computing, and High-Performance Reconfigurable Computing.

Candidates must possess a Ph.D. in Electrical and Computer Engineering or related discipline and must have a demonstrated record of exemplary achievement and leadership in their field of research.

Please send a pdf of a curriculum vitae, statement of research interests, and contact information for five references via e-mail to renato@acis.ufl.edu.

The University of Florida is an equal employment opportunity employer. The "government in the sunshine" laws of Florida apply to the search.

Verold Multiuser Engine Developer

Verold Inc. a newly funded, Toronto-based start-up, is looking for software engineers to help us develop a multi-user synchronization framework for an online 3D platform.

Contact us for more details.

Topics:

Distributed system architecture, MMO, P2P, Networking, JIT compilers

Job Requirements:

- Agile software development
- C/C++, Javascript

Apply:

Contact Person: Jad Yaghi
Email Address: jobs@verold.com



LSU

LOUISIANA STATE UNIVERSITY

POSTDOCTORAL RESEARCHER (TWO POSITIONS) DEPARTMENT OF COMPUTER SCIENCE

Required Qualifications: Ph.D. or equivalent degree in Computer Science, Electrical Engineering, Mathematics, or a related discipline. ABD candidates will be considered, but must have Ph.D. by June 30, 2011. Salary will be commensurate with qualifications and experience. The appointment is initially for one year and can be extended based on the future funding situation. An offer of employment is contingent on a satisfactory pre-employment background check. Application deadline is June 16, 2011 or until candidates are selected.

Apply online and view a more detailed ad, including desired qualifications, at:
www.lsusystemcareers.lsu.edu.
Position #'s 031753 and 034471.

LSU SYSTEM IS AN EQUAL
OPPORTUNITY/EQUAL ACCESS EMPLOYER

acmqueue

Content Written
by Experts

Q

Blogs
Articles
Roundtables
Case Studies
Multimedia
RSS



queue.acm.org



DOI:10.1145/1953122.1953151

Peter Winkler

Puzzled Solutions and Sources

Last month (May 2011, p. 120) we posted a trio of brainteasers, including one as yet unsolved, concerning games and their roles and turns, with randomness either removed or inserted. Here, we offer solutions to two of them. How did you do?

1. Deterministic poker.

Solution. Recall that in deterministic poker, all 52 cards are spread out face up. Alice chooses five, followed by Bob choosing five. Alice can then discard any number of cards (which then go out of play) and draw a like number, finishing with five cards. It's then Bob's turn to draw, with the same options. All actions are deterministic and seen by both players who compare hands. Bob wins if the hands are equally good. The question is, who wins the game with best play?

Note that Alice is a goner if she doesn't immediately prevent Bob from constructing a royal flush—A, K, Q, J, 10 of the same suit—since this is the best hand in poker. So, to have a chance, she must pick at least one of A, K, Q, J, 10 from each suit.

Since straight flushes are the critical threat, Alice's best choice for these cards is the four 10s, permanently preventing Bob from getting a straight flush 10-high or better. Bob is doomed, as Alice will be able to get either A-K-Q-J-10 or 10-9-8-7-6 in a suit after drawing. To prevent it, Bob, on his first turn, would have to pick both a card higher than the 10 and a card lower than the 10 in each suit, adding up to eight cards, though he can take only five.

Any initial hand with the four 10s wins for Alice. She can also win with only three 10s, provided she takes A-9, K-9, Q-9, J-9, K-8, Q-8, J-8, Q-7, J-7, or J-6 of the fourth suit as well. This adds up to $(52-4) + 4 \times 10 = 88$ winning initial hands for Alice.

This problem was posed by Martin Gardner in his famous Mathematical Games column in *Scientific American* and included in *The Scientific American Book of Mathematical Puzzles & Diversions*, Simon & Schuster, New York, 1959, 23.

2. Random chess.

Solution. Recall that Clarissa, a computer-science major and president of her university's chess club, has programmed her laptop to play random chess. At each position, all possible legal moves are computed, with one chosen uniformly at random. The program is designed to quit if and only if a checkmate or stalemate is achieved or if just the two kings remain. The question was, how can her program get caught in a loop?

Turns out, lots of ways...In some cases the players are reduced to a sin-

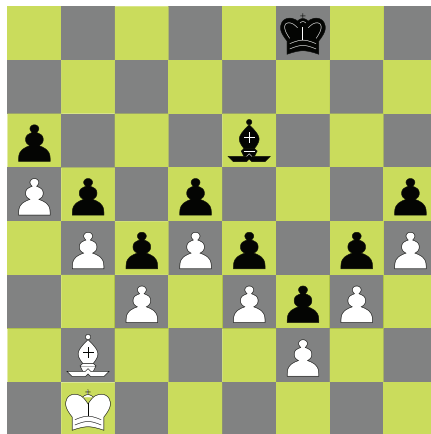
gle forced move at each turn; in others (as in the figure here, a position suggested by Dartmouth student Cole Ott), they have scope but no way to interact.

The chances of a correct program being caught in such a loop are virtually zero. I don't know about Clarissa, but if I wrote the program and it fell into a loop, it would be due to a bug.

3. Winning Hex.

Solution. The unsolved problem was to prove that making one's first move in the center cell leads to a win, with best play, for the first player in Hex. Recall that we already knew (through a "strategy-stealing argument") that the first player has a winning strategy, and it's difficult to believe that playing in the center would undermine it. However, no move by the first player has been identified as part of a winning strategy, even though, amazingly, some moves (near the corners) are known to lose.

Curiously, the game "Random-Turn Hex," in which the players flip coins to determine who moves next, is better understood (and fairer) than the standard alternating game. See the delightful article by Y. Peres et al. "Random-Turn Hex and Other Selection Games" in *American Mathematical Monthly* 114 (May 2007), 373–387.



A standoff.

Peter Winkler (puzzled@cacm.acm.org) is Professor of Mathematics and of Computer Science and Albert Bradley Third Century Professor in the Sciences at Dartmouth College, Hanover, NH.

All readers are encouraged to submit prospective puzzles for future columns to puzzled@cacm.acm.org.

[CONTINUED FROM P. 128] The state of complexity theory at the time made this possible for someone asking the right questions. The main point of PAC learning is that while it has a statistical component, because uncertainty needs to be allowed for, it is the computational aspect that gives it substance, since without that essentially everything would be classified as learnable.

Can you tell me about some of the computational models you developed in conjunction with PAC?

I'll describe the one model that led to consequences far greater than I could have originally imagined. In the course of work that sought to explain why certain easy-looking learning problems, such as learning regular languages, had proved so intractable, Michael Kearns and I formulated the notion of weak learning. This was essentially a gambler's version of PAC learning, where it was sufficient to make predictions with accuracy just better than random guessing, rather than with near certainty. The question of whether this was as powerful as PAC learning became a natural open problem. Successive contributions by Kearns, Schapire, and Freund not only resolved this question in the very unexpected positive direction, but yielded the boosting technique, perhaps the most powerful general learning method now known. This technique, somewhat magically, has been used to improve the performance of almost any basic learning algorithm on real datasets. This is an excellent example of how a theoretical question about abstract models can get at the heart of an important phenomenon and yield rich practical dividends.

You made a number of important contributions to parallel computing in the 1980s. Can you tell me about your more recent work in that arena?

The root problem with parallel machines has not been that any one is inherently bad, but more that it is difficult to make sense of them as a totality. The most striking fact about sequential machines is that they are all the same—they emulate the von Neumann abstraction. This sameness has been vital to the growth of computing

since it has enabled us to write portable software for this abstraction and make machines to emulate it. Thus the von Neumann model has served as the vital “bridging model” between the software and hardware.

I have been long interested in the question of whether a similarly effective bridging model exists for parallel machines and what that might be. Parallelism has now arrived with multiple cores in our laptops. My view is that the irresistible driving forces that have brought this about have been the imperatives of physics. It follows, I think, that we have to look to the driving physical limitations to see what is most fundamental about parallel computing. Computation, communication, synchronization, and memory all need various resources, which in turn are limited by the physical arrangements of the devices. The sought-after bridging model has to account for these costs. We have been spoiled over the last 60-odd years by having an abstraction as simple as the von Neumann model suffice. Life will become a little more complicated now. The absence of an accepted bridging model is evident even at the theoretical level of algorithms.

My most recent work addresses these issues via a particular proposed bridging model, called the Multi-BSP, which tries to capture the physical limitations of machines as simply as possible. It uses barrier synchronization in a hierarchical manner and idealizes a machine as a set of numerical parameters that specify a point in the space of possible machines. Mention of the many architectural details of current machines that are not inevitably suggested by physics is deliberately omitted.

Let's talk about your most recent work in computational neuroscience. Can you explain the “neuroidal model” you developed in your book *Circuits of the Mind*?

The neuroidal model is designed to describe how basic computational tasks are carried out in the brain at the level of neurons. We do not know what changes the brain undergoes when it memorizes a new word or makes a new association. We need some language for describing the alternative algorithms that a network of neurons may

be implementing. Memorizing a new word is easy enough for a computer, but it is still mysterious how the brain does it. Devising algorithms that do not obviously contradict biology is not easy, because biology appears very constrained. Each neuron is connected to only a small fraction of the others, and its influence on each of those is believed to be weak on the average.

You also developed a formal system called robust logics that seeks to reconcile what you've described as “the apparent logical nature of reasoning and the statistical nature of learning.”

Robust logic is aimed at tasks which we do not understand how to do on any computational device, let alone the brain. The tasks in question are those of reasoning using information that is learned from experience and therefore uncertain. Humans manage to make effective decisions even in an uncertain world and even with only partial knowledge. This skill is close to the heart of intelligence, I believe, and understanding it better raises some fundamental scientific questions.

How does computation in the brain differ from computation in a computer?

At the level of neurons the organization is clearly very different—there appears to be no global addressing mechanism for a start. However, there is a good chance, I believe, that at the next level up from neurons, some basic primitives are realized which we will be able to describe in more familiar computational terms. This is the enterprise that the neuroidal model is designed for.

The brain has some general computational character, which we ought to be able to decipher in the foreseeable future. It also contains more specific information acquired through evolution and learning that may be more difficult to uncover. Evolution and learning are subtle adaptive processes, and to emulate their results, we have to understand both the algorithms they use, as well as the environments in which the evolution and learning took place. 

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

© 2011 ACM 0001-0782/11/06 \$10.00

Q&A

A Lifelong Learner

Leslie Valiant discusses machine learning, parallel computing, and computational neuroscience.

LESLIE VALIANT, THE T. Jefferson Coolidge Professor of Computer Science and Applied Mathematics in Harvard University's School of Engineering and Applied Sciences, is the recipient of the 2010 ACM A.M. Turing Award. Valiant is best known for the "probably approximately correct" model of machine learning, which provided an essential framework for studying the learning process and led to major advances in areas from artificial intelligence to computer vision. (A profile of Valiant, "Beauty and Elegance," appears on p. 14.)

Let's talk about your background. What drew you to computer science?

Growing up I was intrigued by both physics and mathematics, and perhaps most of all by the connection between the two. At Cambridge I studied mathematics. I had various tutors, both graduate students and faculty, and when I could I tried to find out what they were working on. Computer science was a way of trying a different direction. My Ph.D. thesis at Warwick was on questions of computability suggested by my advisor, Mike Paterson. In retrospect, this was a good area to cut one's teeth on as it was so quintessentially about computation. Just how rich the field is became clear to me only much later. I consider it my very good fortune to have found my way into it.

How did you first get interested in machine learning?

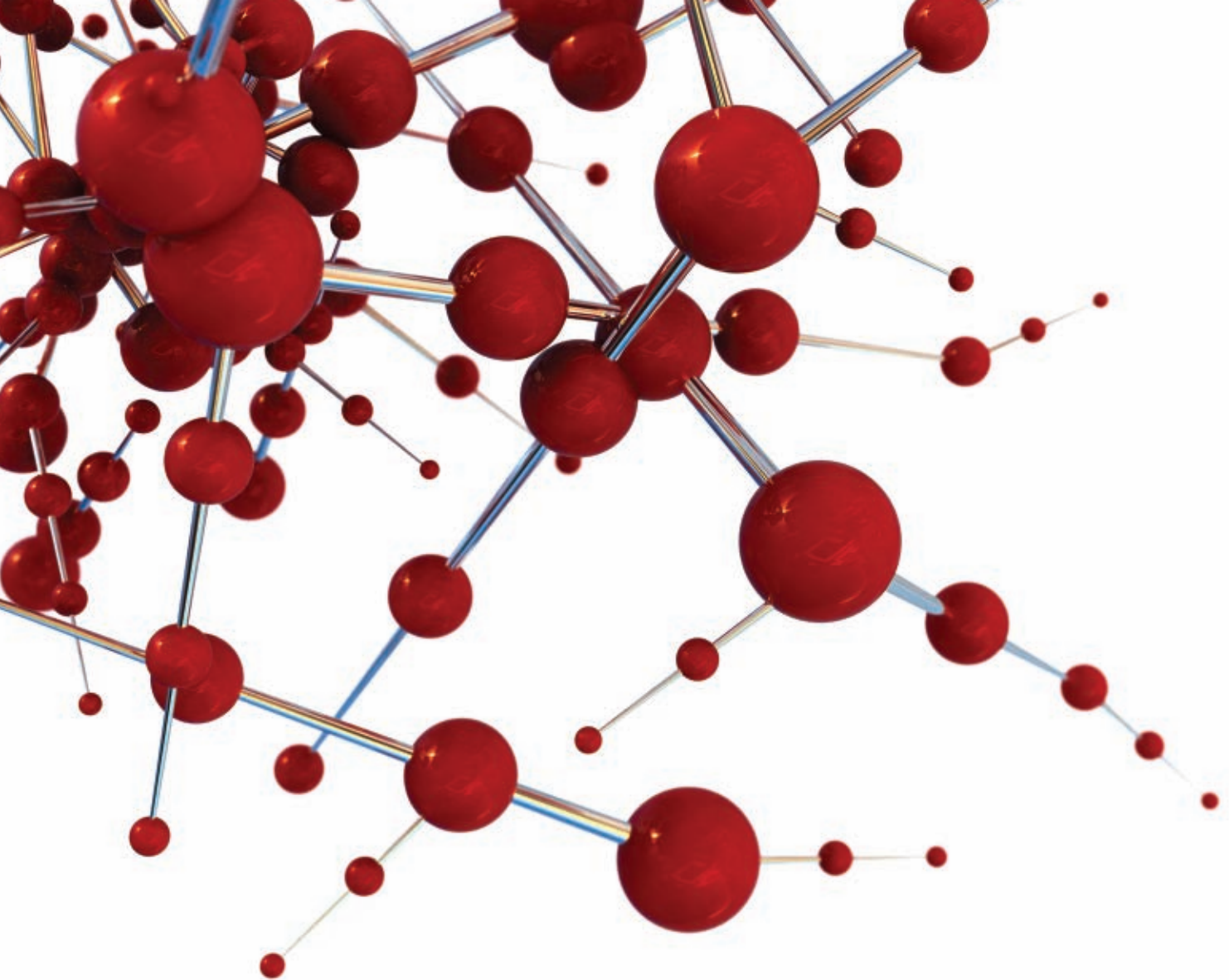
After my Ph.D., I worked in algorithms and complexity theory. I observed work in artificial intelligence



only from a distance. I had accepted the argument that humans provided an existence proof that intelligent behavior is possible by computational mechanisms. However, I recall being puzzled that so many people could be working with such confidence in an area where the concepts seemed so ill-defined. Several topics were studied at that time within AI, such as reasoning, expert systems, planning, natural language, robotics, vision, as well as learning. But which of these was the most basic? Which one had some fundamental definition? I convinced myself somehow that the answer must be learning, but I am not sure how I got there.

Describe how you came to develop the "probably approximately correct," or PAC, model of machine learning.

The main problem, as I saw it, was to specify what needs to be achieved if a learning algorithm is to be declared effective. It was clear that this definition would need to be quantitative if it was to mean anything, since with infinite data and computation there is no real problem. Existing statistical notions, such as Bayesian inference, did not seem enough because they did not distinguish quantitatively between what is learnable in practice and what is not. The final formulation emerged quite rapidly. [CONTINUED ON P. 127]



**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.



From the mind to the marketplace.

Intel works with educators worldwide to revitalize how students think about technology. Because encouraging new ideas fuels innovation.

Collaborating. Investing. Changing the world.

Learn more at Intel.com/software/academicprograms

Sponsors of Tomorrow.™ 