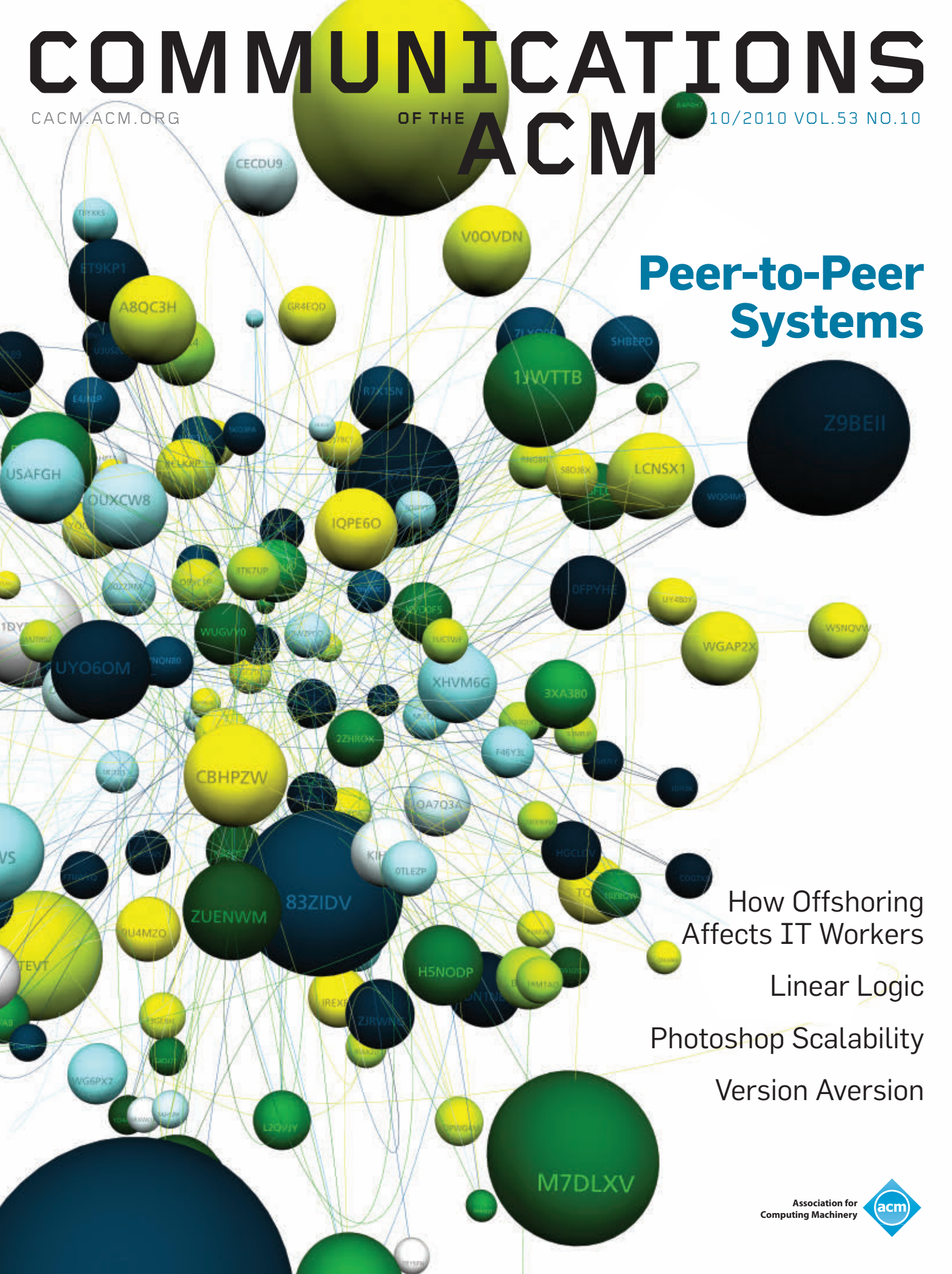


COMMUNICATIONS

CACM.ACM.ORG OF THE

ACM

10/2010 VOL.53 NO.10



Peer-to-Peer Systems

How Offshoring Affects IT Workers

Linear Logic

Photoshop Scalability

Version Aversion

3RD ACM SIGCHI
SYMPOSIUM ON

ENGINEERING INTERACTIVE COMPUTING SYSTEMS

NOVEMBER 22, 2010
Submission deadline
for Long Papers and Workshops

FEBRUARY 10, 2011
Submission deadline
for Late Breaking Results,
Demos, Doctoral Consortium,
Tutorials

LOCATION
Area della Ricerca CNR
Pisa, ITALY

CONFERENCE CHAIR
Fabio Paternò, CNR-ISTI, HIIS
Laboratory

LONG PAPER CHAIRS
Kris Luyten, Hasselt University
Frank Maurer, Univ. of Calgary

Late Breaking Results CHAIRS
Prasun Dewan, UNC Chapel Hill
Carmen Santoro, CNR-ISTI, HIIS
Laboratory

For information, please contact
info@eics2011.org

EICS is sponsored by
ACM SIGCHI



EICS 2011

PISA, Italy

JUNE 13-16, 2011

Topics include but are not limited to:

- Integrating interaction design into the software development process,
- Engineering processes for interactive systems (e.g. design, implementation, prototyping and testing),
- Dynamic generation/composition of interactive systems,
- Requirements engineering for interactive systems,
- Software architectures for interactive systems,
- Modeling interaction and interactive systems,
- Innovative Interactive Applications,
- Interactive systems specification,
- Specifying users' activities,
- Designing Usable Software.

More updated information at
WWW.EICS2011.ORG

Springer References & Key Library Titles



Handbook of Ambient Intelligence and Smart Environments

H. Nakashima, Future University, Hakodate, Hokkaido, Japan;

H. Aghajan, Stanford University, Stanford, CA, USA; **J. C. Augusto**, University of Ulster at Jordanstown, Newtownabbey, UK (Eds.)

Provides readers with comprehensive, up-to-date coverage in this emerging field. Organizes all major concepts, theories, methodologies, trends and challenges into a coherent, unified repository. Covers a wide range of applications relevant to both ambient intelligence and smart environments. Examines case studies of recent major projects to present the reader with a global perspective of actual developments.

2010. XVIII, 1294 p. 100 illus. Hardcover
ISBN 978-0-387-93807-3 ► **\$229.00**



Handbook of Multimedia for Digital Entertainment and Arts

B. Furht, Florida Atlantic University, Boca Raton, FL, USA (Ed.)

The first comprehensive handbook to cover recent research and technical trends in the field of digital entertainment and art. Includes an outline for future research directions within this explosive field. The main focus targets interactive and online games, edutainment, e-performance, personal broadcasting, innovative technologies for digital arts, digital visual and other advanced topics.

2009. XVI, 769 p. 300 illus., 150 in color. Hardcover
ISBN 978-0-387-89023-4 ► **\$199.00**



Handbook of Natural Computing

G. Rozenberg, **T. Bäck**, **J. N. Kok**, Leiden University, The Netherlands (Eds.)

We are now witnessing

an exciting interaction between computer science and the natural sciences. Natural Computing is an important catalyst for this interaction, and this handbook is a major record of this important development.

2011. Approx. 1700 p. (In 3 volumes, not available separately) Hardcover

ISBN 978-3-540-92909-3 ► **\$749.00**

eReference

ISBN 978-3-540-92910-9 ► **\$749.00**

Print + eReference

2011. Approx. 1700 p.

ISBN 978-3-540-92911-6 ► **\$939.00**



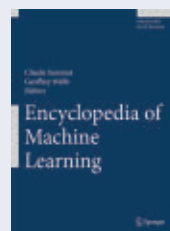
Handbook of Biomedical Imaging

N. Paragios, École Centrale de Paris, France; **J. Duncan**, Yale University, USA; **N. Ayache**, INRIA, France (Eds.)

This book offers a unique guide to the entire chain of biomedical imaging, explaining how image formation is done, and how the most appropriate algorithms are used to address demands and diagnoses. It is an exceptional tool for radiologists, research scientists, senior undergraduate and graduate students in health sciences and engineering, and university professors.

2010. Approx. 590 p. Hardcover

ISBN 978-0-387-09748-0 ► **approx. \$179.00**



Encyclopedia of Machine Learning

C. Sammut, **G. I. Webb** (Eds.)

The first reference work on Machine Learning Comprehensive A-Z

coverage of this complex subject area makes this work easily accessible to professionals and researchers in all fields who are interested in a particular aspect of Machine Learning Targeted literature references provide additional value for researchers looking to study a topic in more detail.

2010. 800 p. Hardcover

ISBN 978-0-387-30768-8 ► **approx. \$549.00**

eReference

2010. 800 p.

ISBN 978-0-387-30164-8 ► **approx. \$549.00**

Print + eReference

2010. 800 p.

ISBN 978-0-387-34558-1 ► **approx. \$689.00**



Handbook of Peer-to-Peer Networking

X. Shen, University of Waterloo, ON, Canada; **H. Yu**, Huawei Technologies, Bridgewater, NJ, USA; **J. Buford**, Avaya

Labs Research, Basking Ridge, NJ, USA; **M. Akon**, University of Waterloo, ON, Canada (Eds.)

Offers elaborate discussions on fundamentals of peer-to-peer computing model, networks and applications. Provides a comprehensive study on recent advancements, crucial design choices, open problems, and possible solution strategies. Written by a team of leading international researchers and professionals.

2010. XLVIII, 1500 p. Hardcover

ISBN 978-0-387-09750-3 ► **\$249.00**

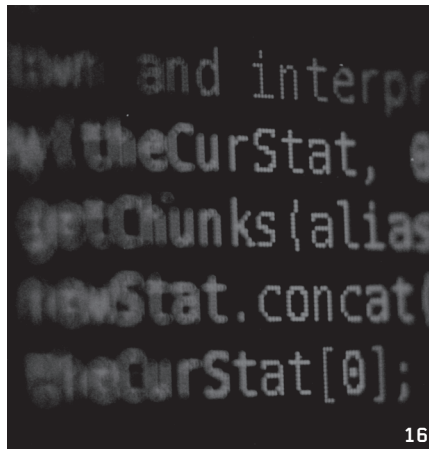
Departments

- 5 **President's Letter**
**ACM is Built on
Volunteers' Shoulders**
By Alain Chesnais
-
- 7 **Letters To The Editor**
**How to Celebrate
Codd's RDBMS Vision**
-
- 8 **BLOG@CACM**
In Search of Database Consistency
Michael Stonebraker discusses the implications of the CAP theorem on database management system applications that span multiple processing sites.
-
- 10 **CACM Online**
The Mobile Road Ahead
By David Roman
-
- 21 **Calendar**
-
- 105 **Careers**

Last Byte

- 112 **Q&A**
Gray's Paradigm
Tony Hey talks about Jim Gray and his vision of a new era of collaborative, data-intensive science.
By Leah Hoffmann

News



- 11 **Linear Logic**
A novel approach to computational logic is reaching maturity, opening up new vistas in programming languages, proof nets, and security applications.
By Alex Wright
-
- 14 **Personal Fabrication**
Open source 3D printers could herald the start of a new industrial revolution.
By Graeme Stemp-Morlock
-
- 16 **Should Code be Released?**
Software code can provide important insights into the results of research, but it's up to individual scientists whether their code is released—and many opt not to.
By Dennis McCafferty

Viewpoints

- 19 **Historical Reflections**
Victorian Data Processing
Reflections on the first payment systems.
By Martin Campbell-Kelly
-
- 22 **Technology Strategy and Management**
**Platforms and Services:
Understanding
the Resurgence of Apple**
Combining new consumer devices and Internet platforms with online services and content is proving to be a successful strategy.
By Michael A. Cusumano
-
- 25 **Inside Risks**
Risks of Undisciplined Development
An illustration of the problems caused by a lack of discipline in software development and our failure to apply what is known in the field.
By David L. Parnas
-
- 28 **Kode Vicious**
Version Aversion
The way you number your releases communicates more than you might think.
By George V. Neville-Neil
-
- 30 **Viewpoint**
**SCORE: Agile Research
Group Management**
Adapting agile software development methodology toward more efficient management of academic research groups.
By Michael Hicks and Jeffrey S. Foster

Practice



32

32 **Photoshop Scalability: Keeping It Simple**

Clem Cole and Russell Williams discuss Photoshop's long history with parallelism, and what is now seen as the chief challenge.
ACM Case Study

39 **Thinking Clearly About Performance, Part 2**

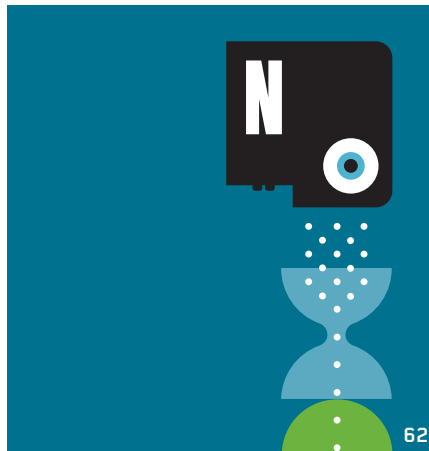
More important principles to keep in mind when designing high-performance software.
By Cary Millsap

46 **Tackling Architectural Complexity with Modeling**

Component models can help diagnose architectural problems in both new and existing systems.
By Kevin Montagne

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

Contributed Articles



62

54 **A Neuromorphic Approach to Computer Vision**

Neuroscience is beginning to inspire a new generation of seeing machines.
By Thomas Serre and Tomaso Poggio

62 **How Offshoring Affects IT Workers**

IT jobs requiring interpersonal interaction or physical presence in fixed locations are less likely to be sent out of the country.
By Prasanna B. Tambe and Lorin M. Hitt

Review Articles

72 **Peer-to-Peer Systems**

Within a decade, P2P has become a technology that enables innovative new services, accounts for a fraction of the Internet traffic, and is used by millions of people every day.
By Rodrigo Rodrigues and Peter Druschel

Research Highlights

84 **Technical Perspective**
A VM 'Engine' That Makes a Difference
By Carl Waldspurger

85 **Difference Engine: Harnessing Memory Redundancy in Virtual Machines**
By Diwaker Gupta, Sangmin Lee, Michael Vrable, Stefan Savage, Alex C. Snoeren, George Varghese, Geoffrey M. Voelker, and Amin Vahdat

94 **Technical Perspective**
Belief Propagation
By Yair Weiss and Judea Pearl

95 **Nonparametric Belief Propagation**
By Erik B. Sudderth, Alexander T. Ihler, Michael Isard, William T. Freeman, and Alan S. Willsky



About the Cover: Peer-to-peer systems have quickly evolved beyond their music sharing, anonymous data sharing, and scientific computing origins to become an efficient means for content distribution and deploying innovative services. As authors Rodrigo Rodrigues and Peter Druschel describe in their cover story beginning on page 72, peer-to-peer systems

are now being used for myriad purposes, including video and telephony, live streaming applications, and to distribute bulk data to many nodes, as depicted in this month's cover imagery by Marius Watz.



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

Executive Director and CEO

John White
Deputy Executive Director and COO
Patricia Ryan

Director, Office of Information Systems
Wayne Graves

Director, Office of Financial Services
Russell Harris

Director, Office of Membership
Lillian Israel

Director, Office of SIG Services
Donna Cappo

Director, Office of Publications
Bernard Rous

Director, Office of Group Publishing
Scott Delman

ACM COUNCIL

President

Wendy Hall

Vice-President

Alain Chesnais

Secretary/Treasurer

Barbara Ryder

Past President

Stuart I. Feldman

Chair, SGB Board

Alexander Wolf

Co-Chairs, Publications Board

Ronald Boisvert and Jack Davidson

Members-at-Large

Carlo Ghezzi;

Anthony Joseph;

Mathai Joseph;

Kelly Lyons;

Bruce Maggs;

Mary Lou Soffa;

Fei-Yue Wang

SGB Council Representatives

Joseph A. Konstan;

Robert A. Walker;

Jack Davidson

PUBLICATIONS BOARD

Co-Chairs

Ronald F. Boisvert; 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; Ricardo Baeza-Yates

ACM U.S. Public Policy Office

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

Computer Science Teachers Association

Chris Stephenson
Executive Director
2 Penn Plaza, Suite 701
New York, NY 10121-0701 USA
T (800) 401-1799; F (541) 687-1840

Association for Computing Machinery (ACM)

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

COMMUNICATIONS OF THE ACM

Trusted insights for computing's leading professionals.

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

STAFF

DIRECTOR OF GROUP PUBLISHING

Scott E. Delman
publisher@cacm.acm.org

Executive Editor

Diane Crawford

Managing Editor

Thomas E. Lambert

Senior Editor

Andrew Rosenbloom

Senior Editor/News

Jack Rosenberger

Web Editor

David Roman

Editorial Assistant

Zarina Strakhan

Rights and Permissions

Deborah Cotton

Art Director

Andrij Borys

Associate Art Director

Alicia Kubista

Assistant Art Director

Mia Angelica Balaquiot

Production Manager

Lynn D'Addesio

Director of Media Sales

Jennifer Ruzicka

Marketing & Communications Manager

Brian Hebert

Public Relations Coordinator

Virginia Gold

Publications Assistant

Emily Eng

Columnists

Alok Aggarwal; Phillip G. Armour;

Martin Campbell-Kelly;

Michael Cusumano; Peter J. Denning;

Shane Greenstein; Mark Guzdial;

Peter Harsha; Leah Hoffmann;

Mari Sako; Pamela Samuelson;

Gene Spafford; Cameron Wilson

CONTACT POINTS

Copyright permission

permissions@cacm.acm.org

Calendar items

calendar@cacm.acm.org

Change of address

acmcoa@cacm.acm.org

Letters to the Editor

letters@cacm.acm.org

WEB SITE

http://cacm.acm.org

AUTHOR GUIDELINES

http://cacm.acm.org/guidelines

ADVERTISING

ACM ADVERTISING DEPARTMENT

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

Director of Media Sales

Jennifer Ruzicka
jen.ruzicka@hq.acm.org

Media Kit acmm mediasales@acm.org

EDITORIAL BOARD

EDITOR-IN-CHIEF

Moshe Y. Vardi
eic@cacm.acm.org

NEWS

Co-chairs

Marc Najork and Prabhakar Raghavan

Board Members

Brian Bershad; Hsiao-Wuen Hon;

Mei Kobayashi; Rajeev Rastogi;

Jeannette Wing

VIEWPOINTS

Co-chairs

Susanne E. Hambrusch; John Leslie King;

J Strother Moore

Board Members

P. Anandan; William Aspray;

Stefan Bechtold; Judith Bishop;

Stuart I. Feldman; Peter Freeman;

Seymour Goodman; Shane Greenstein;

Mark Guzdial; Richard Heeks;

Rachelle Hollander; Richard Ladner;

Susan Landau; Carlos Jose Pereira de Lucena;

Beng Chin Ooi; Loren Terveen



PRACTICE

Chair

Stephen Bourne

Board Members

Eric Allman; Charles Beeler; David J. Brown;

Bryan Cantrill; Terry Coatta; Mark Compton;

Stuart Feldman; Benjamin Fried;

Pat Hanrahan; Marshall Kirk McKusick;

George Neville-Neil; Theo Schlossnagle;

Jim Waldo

The Practice section of the CACM

Editorial Board also serves as

the Editorial Board of [COMMUNIQUE](#).

CONTRIBUTED ARTICLES

Co-chairs

Al Aho and Georg Gottlob

Board Members

Yannis Bakos; Elisa Bertino; Gilles

Brassard; Alan Bundy; Peter Buneman;

Andrew Chien; Anja Feldmann;

Blake Ives; James Larus; Igor Markov;

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

Sriram Rajamani; Jennifer Rexford;

Marie-Christine Rousset; Avi Rubin;

Fred B. Schneider; Abigail Sellen;

Ron Shamir; Marc Snir; Larry Snyder;

Manuela Veloso; Michael Vitale;

Wolfgang Wahlster; Andy Chi-Chih Yao;

Willy Zwaenepoel

RESEARCH HIGHLIGHTS

Co-chairs

David A. Patterson and Stuart J. Russell

Board Members

Martin Abadi; Stuart K. Card; Deborah Estrin;

Shafi Goldwasser; Monika Henzinger;

Maurice Herlihy; Dan Huttenlocher;

Norm Jouppi; Andrew B. Kahng;

Gregory Morrisett; Michael Reiter;

Mendel Rosenblum; Ronitt 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; Jason I. Hong;

Jeff Johnson; Wendy E. MacKay



ACM Copyright Notice

Copyright © 2010 by Association for Computing Machinery, Inc. (ACM). Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and full citation on the first page. Copyright for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or fee. Request permission to publish from permissions@acm.org or fax (212) 869-0481.

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

Subscriptions

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

ACM Media Advertising Policy

Communications of the ACM and other ACM Media publications accept advertising in both print and electronic formats. All advertising in ACM Media publications is at the discretion of ACM and is intended to provide financial support for the various activities and services for ACM members. Current Advertising Rates can be found by visiting <http://www.acm-media.org> or by contacting ACM Media Sales at (212) 626-0654.

Single Copies

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

COMMUNICATIONS OF THE ACM

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

POSTMASTER

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



Association for Computing Machinery



Printed in the U.S.A.



Alain Chesnais

DOI:10.1145/1831407.1831408

ACM is Built on Volunteers' Shoulders

It is a great honor to have been elected ACM President. I must say it's been an interesting road to this juncture. My first ACM role was volunteering to maintain the mailing list of

my local SIGGRAPH chapter in Paris in the mid-1980s. Over the last 25 years, I have continued to volunteer for many different roles within the organization. I am proud to be the first French citizen and the second European to hold this position, as it clearly illustrates that ACM has become a truly international organization. I'm looking forward to the day when we can look back at this time as the beginning of a long chain of volunteer leaders coming from countries throughout every region of the world.

This organization is largely built on the energy and devotion of many dedicated volunteers. I'd like to take this opportunity to share some thoughts on the value of volunteering at ACM. When you look at all of the activities that make up the offerings of our organization, it is amazing to note that members who volunteer their time to deliver the content we provide do the vast majority of the work. There are many opportunities for members to step forward and donate their time to the success of ACM's various endeavors.

I recently attended the annual SIGGRAPH conference in Los Angeles where volunteer efforts are highly visible. With a multimillion-dollar budget, it is by far the largest conference that ACM sponsors, attracting tens of thousands of attendees every year. Though a conference of that size calls upon many highly skilled professional contractors to implement the vision of the

conference organizers, the content is selected and organized by volunteers. I encourage you to visit *Communications'* Web site (<http://cacm.acm.org>) to view a dynamic visual representation of how much work went into the preparation of the conference over a three-year period. Created by Maria Isabel Meirelles, of Northeastern University, Boston, the series of graphs illustrate how volunteer involvement increased dramatically over the 40-month preparatory period as we got closer to the dates of the event. By the time the conference took place a total of over 580 volunteers had put in over 70,000 hours of work to make the conference successful. That's over eight years of cumulated effort!

Not all ACM endeavors require as much volunteer effort as the annual SIGGRAPH conference. There are a multitude of tasks that you, as a member of this organization, can volunteer your services for. You can start by checking out the ACM professional chapter in your area. We have ACM general interest chapters as well as more specialized chapters associated with any of ACM's 34 special interest groups, (SIGs) that can use volunteer support. Tasks cover everything from greeting people at an event hosted by your local chapter to maintaining a Web presence for a particular activity. If a chapter does not yet exist in your area, you can volunteer to establish one. From there you can consider volunteering to help organize a confer-

ence or being a referee to evaluate the quality of submitted content to ACM's 40+ journals and periodicals or the more than 170 conferences and workshops that we sponsor.

By starting off with a task that is small and easily manageable, you can get a sense of the time requirements involved. As you proceed you might find you want to take on more. Working as a volunteer at ACM is extremely rewarding. You can see the value of your effort in the results of the activity you have supported. Little did I imagine, when I first volunteered so long ago to manage a mailing list for my local SIGGRAPH chapter, that I would one day wind up elected to the position of president of the organization.

Over the decades I've held multiple positions including contributing to the publication of the SIGGRAPH quarterly, serving as director for Chapters on the Local Activities Board, chairing the SIG Governing Board, and serving on the organizing committee of the annual SIGGRAPH conference. Each experience has carried with it the satisfaction of giving back to the community that ACM represents. I'd like to encourage you to donate your time to help make ACM a success. Volunteerism is the core of what makes ACM what it is. □

Alain Chesnais (chesnais@acm.org) heads Visual Transitions, a Toronto-based consulting company.

© 2010 ACM 0001-0782/10/1000 \$10.00



Association for
Computing Machinery

Advancing Computing as a Science & Profession

membership application & digital library order form

Priority Code: AD10

You can join ACM in several easy ways:

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

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

Fax
+1-212-944-1318

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

Special rates for residents of developing countries:

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

Special rates for members of sister societies:

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

Please print clearly

Name _____

Address _____

City _____ State/Province _____ Postal code/Zip _____

Country _____ E-mail address _____

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

Purposes of ACM

ACM is dedicated to:

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

I agree with the Purposes of ACM:

Signature _____

ACM Code of Ethics:

<http://www.acm.org/serving/ethics.html>

choose one membership option:

PROFESSIONAL MEMBERSHIP:

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

STUDENT MEMBERSHIP:

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

All new ACM members will receive an
ACM membership card.

For more information, please visit us at www.acm.org

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

RETURN COMPLETED APPLICATION TO:

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

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

Satisfaction Guaranteed!

payment:

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

Visa/MasterCard American Express Check/money order

Professional Member Dues (\$99 or \$198) \$ _____

ACM Digital Library (\$99) \$ _____

Student Member Dues (\$19, \$42, or \$62) \$ _____

Total Amount Due \$ _____

Card # _____ Expiration date _____

Signature _____

DOI:10.1145/1831407.1831409

How to Celebrate Codd's RDBMS Vision

WHILE WE WERE pleased *Communications* celebrated E.F. Codd's seminal article "A Relational Model of Data for Large Shared Data Banks" (June 1970) in "Happy Birthday, RDBMS!" by Gary Anthes (May 2010), we were also dismayed by its inaccuracies and misrepresentations, including about more than just pre-RDBMS history.

For example, saying "Codd's relational model stored data in rows and columns..." (emphasis added) is completely at odds with Codd's goal that "Future users of large data banks must be protected from having to know how data is organized in the machine." Rows and columns are the canonical representation of Codd's relations, not a constraint on physical data structures. Getting this wrong completely undermines Codd's contribution. Moreover, no viable commercial RDBMS has stored data purely in rows and columns, nor has any vendor completely implemented the logical and physical data independence his theory made possible.

Other inaccuracies and misleading statements abound:

DB2 did not "edge out IMS and IDMS." It took a long time for the transaction rates of any commercial RDBMS to compete with those of IMS, which remains an important commercial DBMS;

Ingres and its derivatives did not have the "DEC VAX market to themselves." Interbase, Oracle, and Rdb/VMS were early players (1980s), and Ingres was initially available on VAX/VMS but—like many RDBMS products that preceded the IBM products—introduced on Unix;

The "database wars" raged for almost two decades. Relational repeatedly had to prove itself against network, hierarchical, and object-oriented DBMSs, continuing with XML and Hadoop contenders;

Map/Reduce is a non-declarative programmer's distributed query template, and the Hadoop Distributed File System

is a storage model. Neither rises to the level of data model or programming language;

Whether it was "easier to add the key features of OODBs to the relational model than start from scratch with a new paradigm" never happened. At best, features were added to SQL and SQL-based products, but these misguided additions did violence to the relational model's way of achieving desired capabilities, namely extensible domain support;

"Querying geographically distributed relational databases" is not unsolved. Implementing the relational model's physical data independence solved it;

Since 1980, numerous RDBMS products have provided partial implementation of physical data independence and been widely used in industry. Perhaps David DeWitt [cited by Anthes and director of Microsoft's Jim Gray Systems Laboratory at the University of Wisconsin-Madison] was referring to the problems of querying heterogeneous, distributed data with inadequate metadata, since he was quoted saying databases "created by different organizations" and "almost but not quite alike"; and

Database scalability has always been about numbers of concurrent users and locations, user variety, and manageability, not just data volumes. One of us (McGoveran) published (late 1980s, 1990s) studies evaluating scalability of commercial products along these lines.

David McGoveran, Boulder Creek, CA
C.J. Date, Healdsburg, CA

Author's Response:

E.F. Codd's model let users "see" their data as if it were stored in ordinary tables, rows, and columns. This was easier for them to understand than the pointers and hierarchical trees used in other models. Such simplification was one reason the RDBMS model edged out IMS and IDMS, though IMS is still used in a few narrow (but important) niches. Alas, vendors did not

implement Codd's rules in their purest form, as McGoveran and Date point out.

Gary Anthes, Arlington, VA

Past Future Visions of Two-Way Cable Television

The name "PLATO" in "Celebrating the Legacy of PLATO" by Kirk L. Kroeker (Aug. 2010) triggered my own memories from the early 1970s when I was researching a technology called two-way cable television, whereby interactive broadband services would be possible by blending computers and communications systems. I eventually published *Talk-Back TV: Two-Way Cable Television* (Tab Books, 1976), including an overview of the PLATO system.

I was reminded I had good things to say about the system, including about its plasma-panel display. But in considering PLATO as something that would work in a two-way-television environment, I suggested there would be a problem putting it onto a cable-television network because ordinary televisions could not do many things plasma panels could do. Leaving wiggle room, I added that PLATO researchers had produced considerable material that would work with ordinary CRTs.

Kroeker quoted Brian Dear saying PLATO was a computer system focused on connecting people and an excellent predictor of how the Internet would evolve. Maybe so, but the same could be said about technology being developed or envisioned as "two-way cable television" at the time.

In the same way an exploration of PLATO's history could "enrich everyone's overall perspective" of today's interactive, networked technologies, so, too, could a look back at visions of interactive broadband originally conjured 40 years ago.

Richard H. Veith, Port Murray, NJ

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

© 2010 ACM 0001-0782/10/1000 \$10.00

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:10.1145/1831407.1831411

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

In Search of Database Consistency

Michael Stonebraker discusses the implications of the CAP theorem on database management system applications that span multiple processing sites.



Michael Stonebraker
“Errors in Database Systems, Eventual Consistency, and the CAP Theorem”

<http://cacm.acm.org/blogs/blog-cacm/83396>

Recently, there has been considerable renewed interest in the CAP theorem¹ for database management system (DBMS) applications that span multiple processing sites. In brief, this theorem states that there are three interesting properties that could be desired by DBMS applications:

C: Consistency. The goal is to allow multisite transactions to have the familiar all-or-nothing semantics, commonly supported by commercial DBMSs. In addition, when replicas are supported, one would want the replicas to always have consistent states.

A: Availability. The goal is to support a DBMS that is always up. In other words, when a failure occurs, the system should keep going, switching over to a replica, if required. This feature was popularized by Tandem Computers more than 20 years ago.

P: Partition-tolerance. If there is a

network failure that splits the processing nodes into two groups that cannot talk to each other, then the goal would be to allow processing to continue in both subgroups.

The CAP theorem is a negative result that says you cannot simultaneously achieve all three goals in the presence of errors. Hence, you must pick one objective to give up.

In the NoSQL community, the CAP theorem has been used as the justification for giving up consistency. Since most NoSQL systems typically disallow transactions that cross a node boundary, then consistency applies only to replicas. Therefore, the CAP theorem is used to justify giving up consistent replicas, replacing this goal with “eventual consistency.” With this relaxed notion, one only guarantees that all replicas will converge to the same state eventually, i.e., when network connectivity has been reestablished and enough subsequent time has elapsed for replica cleanup. The justification for giving up C is so that the A and P can be preserved.

The purpose of this blog post is to assert that the above analysis is suspect,

and that recovery from errors has more dimensions to consider. We assume a typical hardware model of a collection of local processing and storage nodes assembled into a cluster using LAN networking. The clusters, in turn, are wired together using WAN networking.

Let's start with a discussion of what causes errors in databases. The following is at least a partial list:

1. Application errors. The application performed one or more incorrect updates. Generally, this is not discovered for minutes to hours thereafter. The database must be backed up to a point before the offending transaction(s), and subsequent activity redone.

2. Repeatable DBMS errors. The DBMS crashed at a processing node. Executing the same transaction on a processing node with a replica will cause the backup to crash. These errors have been termed “Bohr bugs.”²

3. Unrepeatable DBMS errors. The database crashed, but a replica is likely to be ok. These are often caused by weird corner cases dealing with asynchronous operations, and have been termed “Heisenbugs.”²

4. Operating system errors. The OS crashed at a node, generating the “blue screen of death.”

5. A hardware failure in a local cluster. These include memory failures, disk failures, etc. Generally, these cause a “panic stop” by the OS or the DBMS. However, sometimes these failures appear as Heisenbugs.

6. A network partition in a local cluster. The LAN failed and the nodes

can no longer all communicate with each other.

7. A disaster. The local cluster is wiped out by a flood, earthquake, etc. The cluster no longer exists.

8. A network failure in the WAN connecting the clusters together. The WAN failed and clusters can no longer all communicate with each other.

First, note that errors 1 and 2 will cause problems with any high availability scheme. In these two scenarios, there is no way to keep going; i.e., availability is impossible to achieve. Also, replica consistency is meaningless; the current DBMS state is simply wrong. Error 7 will only be recoverable if a local transaction is only committed after the assurance that the transaction has been received by another WAN-connected cluster. Few application builders are willing to accept this kind of latency. Hence, eventual consistency cannot be guaranteed, because a transaction may be completely lost if a disaster occurs at a local cluster before the transaction has been successfully forwarded elsewhere. Put differently, the application designer chooses to suffer data loss when a rare event occurs, because the performance penalty for avoiding it is too high.

As such, errors 1, 2, and 7 are examples of cases for which the CAP theorem simply does not apply. Any real system must be prepared to deal with recovery in these cases. The CAP theorem cannot be appealed to for guidance.

Let us now turn to cases where the CAP theorem might apply. Consider error 6 where a LAN partitions. In my experience, this is exceedingly rare, especially if one replicates the LAN (as Tandem did). Considering local failures (3, 4, 5, and 6), the overwhelming majority cause a single node to fail, which is a degenerate case of a network partition that is easily survived by lots of algorithms. Hence, in my opinion, one is much better off giving up P rather than sacrificing C. (In a LAN environment, I think one should choose CA rather than AP.) Newer SQL OLTP systems appear to do exactly this.

Next, consider error 8, a partition in a WAN network. There is enough redundancy engineered into today's WANs that a partition is quite rare. My experience is that local failures and application errors are way more likely.

“In the NoSQL community, the CAP theorem has been used as the justification for giving up consistency.”

Moreover, the most likely WAN failure is to separate a small portion of the network from the majority. In this case, the majority can continue with straightforward algorithms, and only the small portion must block. Hence, it seems unwise to give up consistency all the time in exchange for availability of a small subset of the nodes in a fairly rare scenario.

Lastly, consider a slowdown either in the OS, the DBMS, or the network manager. This may be caused by a skew in load, buffer pool issues, or innumerable other reasons. The only decision one can make in these scenarios is to “fail” the offending component; i.e., turn the slow response time into a failure of one of the cases mentioned earlier. In my opinion, this is almost always a bad thing to do. One simply pushes the problem somewhere else and adds a noticeable processing load to deal with the subsequent recovery. Also, such problems invariably occur under a heavy load—dealing with this by subtracting hardware is going in the wrong direction.

Obviously, one should write software that can deal with load spikes without failing; for example, by shedding load or operating in a degraded mode. Also, good monitoring software will help identify such problems early, since the real solution is to add more capacity. Lastly, self-reconfiguring software that can absorb additional resources quickly is obviously a good idea.

In summary, one should not throw out the C so quickly, since there are real error scenarios where CAP does not apply and it seems like a bad trade-off in many of the other situations.

References

1. Eric Brewer, “Towards Robust Distributed Systems,” <http://www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf>
2. Jim Gray, “Why Do Computers Stop and What Can Be Done About It,” Tandem Computers Technical Report 85.7, Cupertino, CA, 1985. <http://www.hpl.hp.com/techreports/tandem/TR-85.7.pdf>

Disclosure: Michael Stonebraker is associated with four startups that are producers or consumers of database technology.

Readers' comments

“Degenerate network partitions” is a very good point—in practice I have found that most network partitions in the real world are of this class.

I like to term certain classes of network partitions “trivial.” If there are no clients in the partitioned region, or if there are servers in the partitioned region, it is then trivial. So it could involve more than one machine, but it is then readily handled.

—Dwight Merriman

I think a lot of the discussion about distributed database semantics, much like a lot of the discussion about SQL vs. NoSQL, has been somewhat clouded by a shortage of pragmatism. So an analysis of the CAP theorem in terms of actual practical situations is a welcome change :-)

My company, GenieDB, has developed a replicated database engine that provides “AP” semantics, then developed a “consistency buffer” that provides a consistent view of the database as long as there are no server or network failures; then providing a degraded service, with some fraction of the records in the database becoming “eventually consistent” while the rest remain “immediately consistent.” Providing a degraded service rather than no service is a good thing, as it reduces the cost of developing applications that use a distributed database compared to existing solutions, but that is not something that somebody too blinded by the CAP theorem might consider!

In a similar vein, we’ve provided both NoSQL and SQL interfaces to our database, with different trade-offs available in both, and both can be used at once on the same data. People need to stop fighting over X vs. Y and think about how to combine the best of both in practical ways!

—Alaric Snell-Pym

Michael Stonebraker is an adjunct professor at the Massachusetts Institute of Technology.

© 2010 ACM 0001-0782/10/1000 \$10.00



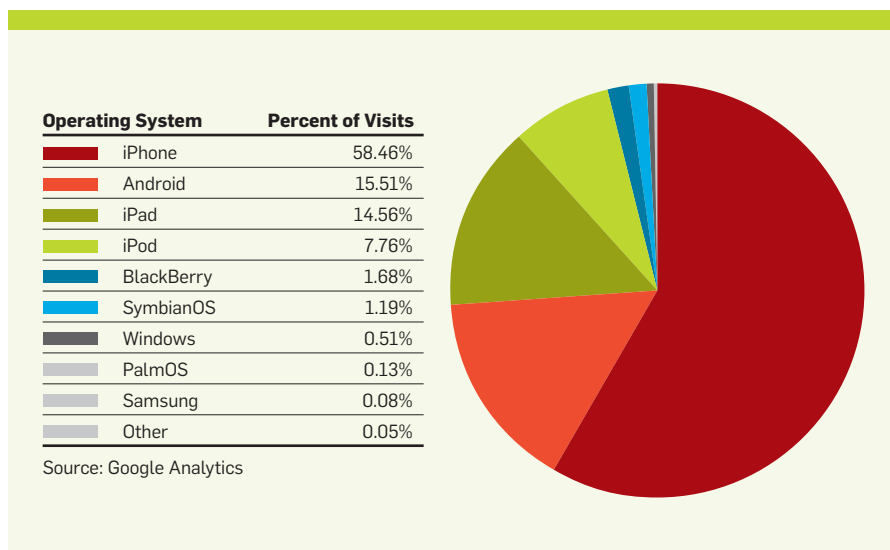
DOI:10.1145/1831407.1831412

David Roman

The Mobile Road Ahead

ACM is planning to launch a mobile version of the *Communications* Web site and a series of mobile applications in the coming months. To determine how best to proceed, we have been directly and indirectly collecting data from members for months to help guide our decision making. We have conducted surveys, met with focus groups, and examined usage statistics.

The following table and pie chart show how users with mobile devices have been getting to the *Communications* Web site. The percentage of users accessing the site with mobile devices is small, but the number is trending upward. This report is based on sampled data.



Here are some of the key takeaways from our research that will influence our decisions going forward:

► **The Time Is Right.** There is a growing need to access *Communications* content from mobile platforms. “[This] is the way things are moving,” said one software engineer.

► **Keep It Simple.** Users want to be able to find the content quickly and easily on their devices, from anywhere anytime.

► **Search, Save, Share.** Most “small screen” mobile devices are being used primarily to find, collect, and share articles.

► **Consumption Devices Are Coming.** Media-ready platforms like the iPad will create new ways to present and consume content.

► **This Is The Beginning.** We’ve collected valuable data on which mobile devices are being used to access *Communications* content, and that is guiding our path in the short term. But for the long term we will remain platform agnostic, and will support the systems that are most heavily used and called for by members.

ACM Member News

SUPERCOMPUTING IN NEW ORLEANS



The 23rd meeting of the world’s largest conference on supercomputing, SC10, takes place this

November 13–19 in New Orleans. “SC10 will have more conference space than any previous SC conference,” says Barry V. Hess, SC10 general chair and deputy chief information officer at Sandia National Laboratories, “and we have expanded our technical program sessions, education program, and exhibit space to showcase the technical advances in high-performance computing [HPC], networking, storage, and analysis, all in one location.”

SC10’s three main thrust areas are climate simulation, which will explore the latest R&D efforts that are taking advantage of HPC systems to enable climate simulation applications and techniques; heterogeneous computing, which will focus on software infrastructure for making effective use of accelerator or heterogeneous supercomputers; and data-intensive computing, whose focus will be on how data is shared and communicated among scientists.

Like the last four SC conferences, SC10 will feature a disruptive technologies program. This year’s focus will be new computing architectures and interfaces that will significantly impact the HPC field.

“New Orleans is about the music, the food, the culture, the people, and the wonderful fusion of all these things,” says Hess. “The SC10 committee has been meeting in New Orleans for three years of planning and has enjoyed all aspects of the city, from the unique food to the historical tours. Our recommendation is to get out and visit the shops, the restaurants, Jackson Square, Café Du Monde, take a cemetery or swamp tour, and enjoy your time in a unique city.”

For more information, visit <http://sc10.supercomputing.org/>.

—Jack Rosenberger

Linear Logic

A novel approach to computational logic is reaching maturity, opening up new vistas in programming languages, proof nets, and security applications.

WHEN THE FRENCH logician Jean-Yves Girard first visited Xerox PARC during a trip to Silicon Valley in 1984, he knew he was in the right place. Seeing computer scientists collaborating with linguists, ethnographers, and other non-programmers, he started to consider the possibilities of bridging computer science with his own branch of philosophy. “What impressed me most was the change of spirit,” he recalls. “It was a very special time.”

Following his trip to California, Girard began work on his breakthrough paper “Linear Logic,” which postulated an entirely new approach to logic, one deeply informed by computational principles. In the ensuing years, the principles of linear logic have found their way into a broad range of other arenas including programming languages, proof nets, security applications, game semantics, and even quantum physics.

In the early 1980s, logicians like Girard were just starting to take an interest in computer science, while a handful of computer scientists were starting to recognize the potential of logical proof systems as a framework for functional programming. Linear logic represented an important step forward



French logician Jean-Yves Girard, author of the seminal paper “Linear Logic.”

for computer science because it challenged the conceptual limitations of traditional classical logic. For thousands of years, the study of logic had hinged on the assumption of permanent Aristotelian truths, or unchanging essences. *A* was *A*, *B* was *B*, and would ever be thus. Through the lens of com-

puter science, Girard began to see a way out of this “foundational aphoria.” His chief insight was that logic could function without this unspoken assumption of perennality. “This was a big shock,” Girard recalls. “The discovery of linear logic went completely against all the things I had been taught in logic.”

Whereas classical logic might support an assertion like type $A \rightarrow B$, computer programs require a set of concrete instructions for transforming A into B , such as applications, variables, or exception handlers. In the eyes of a computer program, then, A is not a permanent entity but a consumable resource. To address this problem, Girard proposed a resource-conscious approach to logic, laying out an entirely new framework capable of describing resources that could be used and depleted during the course of an operation.

In the nearly quarter of a century since Girard published his seminal paper, most of the foundational theoretical work in linear logic has been completed. However, computer scientists continue to find new applications of the theory across a wide range of disciplines like proof nets, categorical semantics, and computer security applications.

At Carnegie Mellon University (CMU), computer science professor Frank Pfenning has been exploring the application of linear logic to distributed security problems. After one of his students introduced him to linear logic, he became convinced it provided the ideal conceptual framework for specifying difficult-to-encode rules like complex privacy policies or resource conservation strategies. "I was most interested in characterizing, logically, complex properties of distributed systems," Pfenning explains.

Working with a team of students, he used the principles of linear logic

"The discovery of linear logic went completely against all the things I had been taught in logic," says Jean-Yves Girard.

to implement a proof-carrying file system (PCFS), featuring an access control policy that is stated as a logical theory, wherein file access is granted on the condition of a logical proof of policy compliance. "Linear logic is tremendously useful here," he explains, "because we can easily represent the change of state that takes place, for example, when you read or write a file."

Working with Symantec, Pfenning and CMU postdoctoral researcher Deepak Garg have applied PCFS to formalize the access control policies of the national intelligence community in the United States. In collaboration with Jamie Morgenstern, an undergraduate student from the University of Chicago, Pfenning is now working on extending the implementation to handle even more complex policies. Pfenning feels the biggest challenges lie in translating

complex real-world rule sets into unambiguous logic. The ideal outcome is what he calls "an abstract logical form that is theoretically tractable and at the same time practically useful."

Proof Nets

Linear logic has also opened new doors in the field of proof nets. Prior to the introduction of linear logic, most computer scientists working in the field relied on intuitionistic logic, following the well-established Curry-Howard Correspondence, which suggested that formal proof calculi shared a common structure with computational models. Before the advent of linear logic, this model had served as the de facto standard for types. "Linear logic enriched this world greatly," says Dale Miller, director of research at INRIA Saclay, who has spent the last several years applying the principles of linear logic to proof systems.

"Originally, proof systems were used to build 'big-step' inference rules from the 'small-step' inference rules of linear logic," Miller explains. Now, he is exploring the possibilities of so-called focused proof systems by using those "small-step" inference rules to build a range of proof systems for classical and intuitionistic logic. "If one has an interpreter for focused linear logic, that interpreter can be used as an interpreter for many proof systems," says Miller, citing the examples of emulating sequent calculus and tableaux. "Different choices yield different and

Obituary

Nicolas Georganas, Multimedia Guru, Dies at 67

Nicolas D. Georganas, a leader in multimedia networking, died on July 27 at age 67. Georganas was founding editor-in-chief, in 2004, of *ACM Transactions on Multimedia Computing, Communications, and Applications (ACM TOMCCAP)*. He promoted the linking of video, audio, and other sensory input—lately focusing on haptics—for a wide variety of distributed interactive systems, from telemedicine to high-level gaming to security.

"He is one of the godfathers of multimedia," says Ralf Steinmetz,

editor-in-chief of *ACM TOMCCAP* and adjunct professor at Technical University Darmstadt. "Whatever we did in this area, particularly with ACM, he was from the beginning involved in it."

"He was incredibly kind and very friendly," says Klara Nahrstedt, a computer science professor at the University of Illinois at Urbana-Champaign, who described Georganas as an elder statesman in a young field. "He truly served many people as a mentor."

Born and educated in Athens, Greece, Georganas

earned a Ph.D. in electrical engineering at the University of Ottawa, where he served on the faculty from 1970 until his death. Georganas' research contributions included ambient multimedia intelligence systems, multimedia communications, and collaborative virtual environments. He published more than 425 technical papers and is co-author of *Queueing Networks—Exact Computational Algorithms: A Unified Theory by Decomposition and Aggregation*.

Georganas was dedicated to building a multimedia

community, and was known for encouraging his students, many of whom are now professors in Canada and elsewhere. Steinmetz said Georganas, who was fluent in English, French, and Greek and spoke some Spanish and German, wanted the community to have an international flavor and championed tolerance among its members. At the same time, he demanded people do their share of the work, and he'd push to get what he wanted. "He tried always to be fair," Steinmetz says. "He was also good at twisting arms."

—Neil Savage

often, known proof systems.”

In recent years, linear logic has also given rise to a new genre of programming languages like Forum, Lolli, and Lygon that incorporate richer forms of expression to allow more powerful approaches to proofs.

Looking ahead, Pfenning believes there is still work to do in improving the state of automation in linear logic. “We need theorem provers, model checkers, and other tools for working with linear logic to make its application to real-world problems easier.” Miller agrees that linear logic has the potential to support the automation of theorem proving. “Focused proof systems give a central role to inference rules that are invertible,” he explains. “When a formula is introduced by an invertible rule, that formula can be discarded. Such information is useful in building theorem provers.”

Miller also sees an opportunity to use linear logic and proof search to provide specifications of algorithms, using proof theory research to help in reasoning about such algorithmic specifications. He also hopes to see the day when a large “logic of unity” might take shape that would encompass classical, intuitionistic, and linear logic in one grand system.

Where could linear logic go from here? Other active research areas include concurrency theory, quantum computing, game semantics, implicit computational complexity, and the verification of imperative programs with heaps using separation logic, a close cousin of linear logic.

With the field maturing, the fundamental principles of linear logic are receding into the background as an area of active inquiry as computer scientists learn to apply the established principles to emerging computational problems. “Linear logic is no longer alive as a specific subject in which you work,” says Girard. “It’s become something classical. It is part of the toolbox.”

Pfenning agrees with Girard’s assessment, but thinks linear logic lacks the widespread exposure it deserves at every level of the computer science curriculum. “It should be part of the standard toolkit,” he says, “but I don’t think it is taught in enough places right now, especially in the United States.”

Girard, meanwhile, has moved on

Active research areas for linear logic include concurrency theory, quantum computing, game semantics, and implicit computational complexity.

from the problems of computing to set his sights on more esoteric quandaries. “I would like to understand why certain things are difficult, why the world is not transparent,” he says. Alas, perhaps some questions are better left to logicians. □

Further Reading

Abramsky, S., Jagadeesan, R., and Malacaria, P. Full abstraction for PCF (extended abstract). *Lecture Notes in Computer Science 789, Proceedings of Conference on Theoretical Aspects of Computer Software, 1994.*

Bowers, K.D., Bauer, L., Garg, D., Pfenning, F., and Reiter, M.K.

Consumable credentials in logic-based access-control systems. *Proceedings of the 14th Annual Network and Distributed System Security Symposium, San Diego, CA, Feb. 28–March 2, 2007.*

Girard, J.-Y.

Linear logic. *Theoretical Computer Science* 50, 1, 1987.

Lincoln, P., Mitchell, J., Scedrov, A., and Shankar, N.

Decision problems for propositional linear logic. *Proceedings of the 31st Annual Symposium on Foundations of Computer Science, St. Louis, MO, Oct. 22–24, 1990.*

Miller, D.

An overview of linear logic programming. *Linear Logic in Computer Science*, Ehrhard, T., Girard, J.-Y., Ruet, P., and Scott, P. (Eds.), Cambridge University Press, London, U.K. 2004.

Alex Wright is a writer and information architect who lives and works in Brooklyn, NY. Patrick Lincoln, SRI International, contributed to the development of this article.

© 2010 ACM 0001-0782/10/1000 \$10.00

Milestones

CS Awards

NEVANLINNA PRIZE

Daniel Spielman, a professor of computer science and applied mathematics at Yale University, won the Rolf Nevanlinna Prize, one of the highest honors in the field of mathematics, from the International Mathematical Union. The Nevanlinna Prize recognizes researchers under the age of 40 for “outstanding contributions in mathematical aspects of information science.” Spielman’s research has included smoothed analysis of linear programming, algorithms for graph-based codes, and applications of graph theory to numerical computing.

“The same way that physicists grow up dreaming about winning the Nobel Prize, I’ve dreamed of winning the Nevanlinna Prize ever since I was a graduate student,” Spielman said in a statement. “I was in shock when László Lovász, the president of the International Mathematical Union, called me up to tell me that I had won. I had to hear him say it a few times before I believed him. It is an incredible honor. Many of my heroes have won this prize.”

MICROSOFT AWARD

Cheryl Arnett from Sunset Elementary School in Craig, CO, and Rawya Shatila from Maskassed Khalil Shehab School in Beirut, Lebanon, won first place in the 2010 U.S. Innovative Education Forum, a Microsoft-sponsored competition for teachers who use technology in their curriculum to improve student learning. Arnett and Shatila’s joint project, called “Digital Stories: A Celebration of Learning and Culture,” connected Arnett’s class of first- and second-graders in Craig, CO, to Shatila’s second-graders in Beirut. The two teachers, who had never met prior to their collaboration, used wikis, blogs, and online mapping tools to share stories and activities to help students increase their global awareness of the similarities and differences between children from different nations. Arnett and Shatila will represent the United States at the Worldwide Innovative Education Forum in South Africa this fall.

—Jack Rosenberger

Personal Fabrication

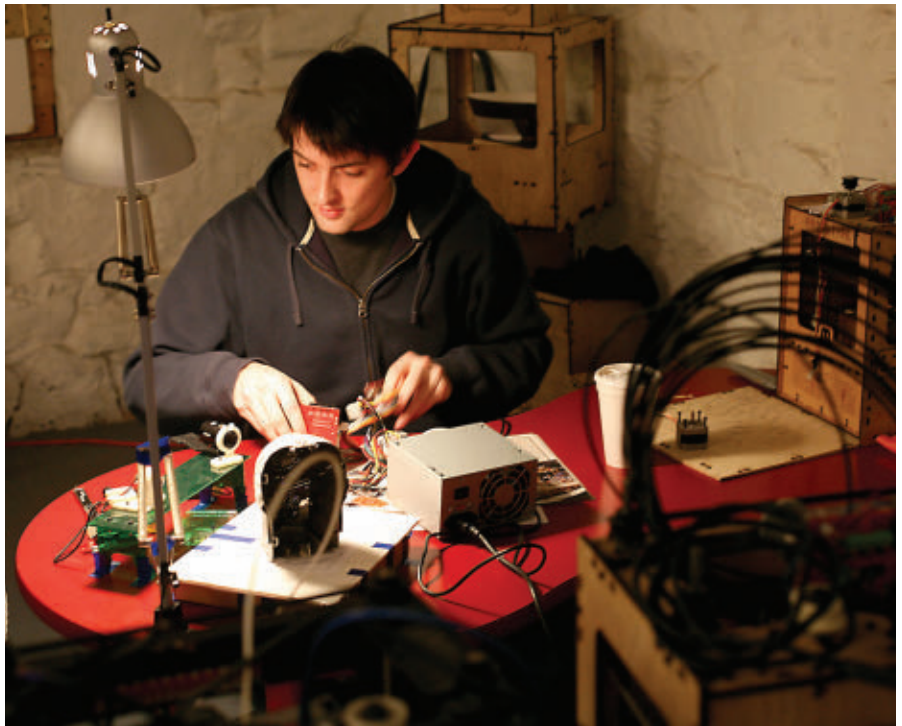
Open source 3D printers could herald the start of a new industrial revolution.

WHILE ATTENDING A health and beauty trade show in the fall of 2009, Nick Starno watched as countless exhibitors struggled with cosmetic tubes, vainly attempting to squeeze the last few drops out of them. Starno, however, is a mechanical design engineer, and familiar with 3D printers. When he got home, he designed a tube squeezer and posted his prototype on a community Web site for 3D printer designs. Within hours, several 3D printer enthusiasts in Europe had downloaded his design and manufactured the tube squeezer. Since then, Starno's design has been downloaded more than 500 times and people around the world have produced his tube squeezer at a cost about 30 cents each.

"I knew that as long as I could model it on the computer, it could be made," says Starno, who now works with Makerbot, a 3D printer company. "No worrying about tooling costs, post processing, surface finishes, packaging, shipping quantities, or advertising. Anyone with a 3D printer could search for my design, download it, and make one on demand without ever leaving their house."

Printing simple devices such as tube squeezers might not seem very exciting or sexy, but it heralds the beginning of a technological revolution involving thousands of hobbyists around the world who are using 3D printers to fabricate wine glasses, toy cars, cooling fans, mechanical arms, and countless types of nuts, bolts, and gears.

To many observers, this revolution mirrors the personal computer revolution, with its kits for hobbyists, of the 1970s. "There are many parallels between personal computing and personal fabrication," says Hod Lipson, an associate professor of mechanical and aerospace engineering and computing and information science at Cornell University. "I think you can look at the history of computers and how they changed our world, and you can anticipate many



Nick Starno, a mechanical design engineer, in the process of building a 3D printer.

aspects of 3D printing and how they will interface with every aspect of our lives."

Open Source Printers

While large-scale, commercial 3D printers have existed for years, personal 3D printers are a recent, fast-spreading phenomenon. Dozens of startup companies are developing and marketing 3D printers, but two of the most widely used 3D printers are open source projects.

Based at the University of Bath, RepRap is the brainchild of Adrian Bowyer, a senior lecturer in the department of mechanical engineering. The other project is Fab@Home, which is led by Lipson.

To design a printable object, a user needs a computer equipped with a computer-assisted design (CAD) program. The different RepRap and Fab@Home 3D printers are the size of a standard office photocopier, and feature off-the-shelf components including a chassis, tool heads, and electronics. The 3D printers work almost the same as a

standard printer, but instead of using multi-colored inks, a printer's mobile arm includes a syringe that ejects melted plastic, slowly building up the "image," layer after layer, into a real object. Simple objects like a gear, for instance, can be completed in less than an hour.

The parts for the latest RepRap printer, Mendel, cost about \$525, but an online network of artists and inventors are constantly modifying and improving Mendel's design. Moreover, Mendel prints about 50% of its own parts, excluding nuts and bolts, so it is almost a self-replicating machine.

"It's designed to copy itself because that's the most efficient way of getting a large number of them out there," says Bowyer, who estimates more than 4,000 RepRap printers have been made since the plans for the original RepRap Darwin printer were first released in 2008. "If you've got something that copies itself, then, in principle, the numbers can grow exponentially fast, and that's fast-

er than any other means of production that humanity currently has.”

New Design Frontiers

In addition to enabling people to manufacture objects they never could before, 3D printers could lead to radically new designs that are not possible with traditional fabrication techniques. “Your first instinct when you have one of these machines is that instead of making something in the machine shop, you are just going to print it,” says Lipson. “But at some point you realize you can make new things with complicated geometry that you cannot make any other way. You don’t have to stick to straight edges and flat surfaces that can be easily machined or thin walls that can be injection molded. You can make absolutely any shape that you want.”

For instance, Lipson’s team has experimented with printing objects with both hard and soft materials. When the materials are printed at a random 50%-50% ratio, the results are ordinary. However, when the dots of hard and soft material are printed in special patterns, the material, when stretched like an elastic, actually gets thicker.

Indeed, one of Lipson’s favorite 3D printer materials is Play-Doh. He recently used it to create miniature copies of the U.S. space shuttle during a school visit as part of the Fab@School project, led by himself and Glen Bull, a professor of instructional technology at the University of Virginia. The Fab@School’s goal is to use 3D printers to show K–12

students the combined power of math, science, and engineering. The MacArthur Foundation and Motorola have awarded \$435,000 to the Fab@School group to develop curriculum, build more 3D printers, and expand the project.

A Larger Ink Palette

Although Play-Doh and other squishy substances can be used in 3D printers, melted plastic remains the primary material. Other desirable materials, including various metals and ceramics, are more challenging to use. Progress has been made in printing with metal, but more experimentation is needed to make the process easier and overcome fundamental properties in the materials like melting point and viscosity.

For Lipson’s Fab@Home project, the ultimate goal is to design a robot that can walk out of the printer. Before that can happen, “inks” for batteries, actuators, wires, transistors, and numerous other pieces must be developed. However, Lipson’s lab has already developed an actuator that operates with low voltage and a printable battery.

Adrian Bowyer at the University of Bath has had success making a printable conductor that melts at a lower temperature than the plastic does. Due to the temperature difference, the 3D printer can manufacture plastic channels that do not melt when filled with the hot conductor for wires or other electrical circuitry.

“At the moment the way we manufacture goods is from economies of

scale,” says Bowyer. “It is more efficient to make lots of one thing in one place and that’s how conventional industry works all over the world. But there are many things we used to do that way that we don’t do anymore. For instance, I’m old enough to remember my parents getting personalized letterhead printed at a local printer, whereas now we have computer printers. Imagine the idea of a whole industry disappearing, and everybody making what they want in their own home. That would be a pretty profound economic change.” ■

Further Reading

Bradshaw, S., Bowyer, A., and Haufe, P. The intellectual property implications of low-cost 3D printing. *SCRIPTed* 7, 1, April 2010.

Hiller, J. and Lipson, H. Design and analysis of digital materials for physical 3D voxel printing. *Rapid Prototyping Journal* 15, 2, 2009.

Malone, E. and Lipson, H. Fab@Home: the personal desktop fabricator kit. *Rapid Prototyping Journal* 13, 4, 2007.

Sells, E., Smith, Z., Bailard, S., Bowyer, A., and Olliver, V.

RepRap: the replicating rapid prototype: maximizing customizability by breeding the means of production. *Handbook of Research in Mass Customization and Personalization*, Piller, F.T. and Tseng, M.M. (Eds.), World Scientific Publishing Company, Singapore, 2009.

Graeme Stemp-Morlock is a science writer based in Elora, Ontario, Canada.

© 2010 ACM 0001-0782/10/1000 \$10.00

Crowdsourcing

Foldit Research Paper’s 57,000+ Co-authors

Since May 2008, tens of thousands of *Foldit* video game players have competed online against each other, and a computer program, in figuring out how 10 different proteins fold into their three-dimensional configurations. In the end, the players managed to outperform the computer program—and are cited as co-authors on the resulting paper, which was published in *Nature*.

While scientists understand the general process of how the primary structure of a protein is transformed into a three-

dimensional structure, the method of using statistical and related software algorithms to predict protein structures is computationally demanding.

“If you were blindfolded and all you’re doing is picking pieces at random, that’s more or less what the computer is doing,” says Zoran Popović, an associate professor of computer science at the University of Washington. “The computational methods are eating up huge amounts of resources.”

Foldit’s top scores are posted online, allowing

the players, who compete individually or in groups, to compare their scores. In the 10 separate protein-folding puzzles, the players matched the results of the computer-generated solutions in three of the puzzles, outscored them in five puzzles, and created significantly better solutions in two puzzles, according to the scientists.

When the results were published in the August 5 issue of *Nature*, Popović and his fellow researchers cited “*Foldit* players” at the end of the paper’s

author list in appreciation of the more than 57,000 players’ contributions “through their feedback and gameplay.”

Such extensive author lists will soon become commonplace given the increasing online collaboration between citizen volunteers and scientists, says Popović, who plans to establish a Center for Game Science at the University of Washington this fall, and will work on problems that can be solved with the symbiosis of human volunteers and computers.

—Phil Scott

Should Code be Released?

Software code can provide important insights into the results of research, but it's up to individual scientists whether their code is released—and many opt not to.

ON ANY GIVEN day, medical researchers at Carnegie Mellon University (CMU) may be investigating new ways to thwart the development of epilepsy or designing an implantable biosensor to improve the early detection of diseases such as cancer and diabetes. As with any disciplined pursuit of science, such work is subject to rigorous rounds of peer review, in which documents revealing methodology, results, and other key details are examined.

But, assuming software was created for the research, should a complete disclosure of the computer code be included in the review process? This is a debate that doesn't arrive with any ready answers—not on the campus grounds of CMU or many other institutions. Scott A. Hissam, a senior member of the technical staff at CMU's Software Engineering Institute, sees validity in both sides of the argument.

"From one perspective, revealing the code is the way it should be in a perfect world, especially if the project is taking public money," says Hissam, who, as a coauthor of *Perspectives on Free and Open Source Software*, has explored the topic. "But, in practice, there are questions. The academic community earns needed credentialing by producing original publications. Do you give up the software code immediately? Or do you wait until you've had a sufficient number of publications? If so, who determines what a sufficient number is?"

Another dynamic that adds complexity to the discussion is that scientific researchers are not software developers. They often write their own code, but generally don't follow the same practices, procedures, and standards as professional software programmers.

"Researchers who are trying to cure



cancer or study tectonic plates will write software code to do a specific task in a lab," Hissam says. "They aren't concerned about the same things that computer programmers are, such as scalability and design patterns and software architecture. So imagine how daunting of a task it would be to review and try to understand how such a code was written."

U.K.'s Climategate

This issue has gained considerable attention ever since Climategate, which involved the illegal hacking of researchers' email accounts last year at the Climate Research Unit at the University of East Anglia, one of world's leading institutions on global climate change. More than 1,000 email messages and 2,000 documents were hacked, and source code was released. Global warming contrarians have contended the email reveals that scientists

manipulated data, among other charges. Climate Research Unit scientists have denied these allegations and independent reviews conducted by both the university and the House of Commons' Science and Technology Select Committee have cleared the scientists of any wrongdoing.

Still, Darrel Ince, professor of computing at the U.K.'s Open University, cited the Climate Research Unit's work as part of his argument that code should be revealed. He wrote in the *Manchester Guardian* that the university's climate-research team depended on code that has been described as undocumented, baroque, and lacking in data needed to pass information from one program and research team to another.

Ince noted that Les Hatton, a professor at the Universities of Kent and Kingston, has conducted an analysis of several million lines of scientific code and found that the software possessed a high level of detectable inconsistencies. For instance, Hatton found that interface inconsistencies between software modules that pass data from one part of a program to another happen, on average, at the rate of one in every seven interfaces in Fortran and one in every 37 interfaces in C.

"This is hugely worrying when you realize that one error—just one—will usually invalidate a computer program," Ince wrote. Those posting comments on the *Guardian* Web site have been largely supportive of his arguments. "The quality of academic software code should absolutely be scrutinized and called out whenever needed," wrote one commenter. "It should be the de facto criteria for accepting papers," wrote another.

Still, not all were in agreement. "I work in scientific software," wrote one commenter. "The sort of good pro-

gramming practices you talk about are things ... [that are] absolutely useless for one person wanting to do a calculation more quickly. That's all the computer models are, fancy calculators. I've seen plenty of Fortran and VB code to do modeling written by academics and it's mostly awful but it also nearly always does the job."

To Share or Not

Efforts to encourage scientists to reveal software code stem from philosophies that began with the birth of computers. Because the big, clunky invention was so expensive, software was freely shared. "There wasn't much that people had written anyway," says John Locke, manager of Freelock Computing, an open-source business services firm. "Sharing code was like sharing scientific ideas, and was treated in the same way."

The U.S. Constitution provides patent and copyright protection to scientists and their sponsors so they can place their work in the public domain while still being able to profit, Locke argues. And this, he says, provides enough protection to open up the code.

"Not sharing your code basically adds an additional burden to others who may try to review and validate your work," Locke says. "If the code is instrumental in testing a hypothesis, keeping it closed can prevent adequate peer review from taking place. After all, source code is nothing more than a very specific set of steps to achieve a desired result. If those steps cannot be reviewed in detail, the whole test is suspect."

There is often hesitancy, however, for these very reasons. Opening up the code essentially throws "the books" open. It further peels away the curtain to reveal how the work was done. These days, scientists are wary of providing additional fodder that could impede their work or damage their reputations.

"There are downsides [to revealing code]," says Alan T. DeKok, a former physicist who now serves as CTO of Mancala Networks, a computer security company. "You may look like a fool for publishing something that's blatantly wrong. You may be unable to exploit new 'secret' knowledge and technology if you publish. You may have better-known people market your idea better than you can, and be cred-

"Not sharing your code basically adds an additional burden to others who may try to review and validate your work," says John Locke.

ited with the work. But in order to be trusted, much of the work should be released. If they can't release key portions, then the rest is suspect."

While ethical considerations and those conveyed in the greater interest of science are often made to encourage more information sharing, those same considerations can be used to state the case that some information needs to remain undisclosed. Had the Manhattan Project happened today, for instance, surely few people would call for an open dissection of its software DNA, says Mike Rozlog, developer tools product manager at Embarcadero Technologies.

Also, science is a highly competitive endeavor, and funding is often based on a track record of success. "If you're forced to release proprietary [code]," Rozlog says, "this could give a significant advantage to rogue organizations that don't follow the same rules."

Opening Up Science

For the past seven years, researchers at Purdue University have attempted to resolve this issue, especially with the study of nanotechnology. Funded by the National Science Foundation, nanoHUB.org has been established as a site where scientists and educators share simulation and modeling tools and run their code on high-performance computer resources, says software architect Michael McLennan, a senior research scientist at Purdue. A toolkit called Rappture standardizes the input and output for the tools and tracks details about execution, such as which user ran which version of the code, the computer used, and

the date of the usage. Simulations run in a cloud of computing resources, and the most demanding computations are sent to national grid computing resources such as the TeraGrid. nanoHUB.org now has a core group of 110,000 users from more than 170 nations, who launch more than 340,000 online simulations each year.

The project encourages users to release their work as open source or under a creative commons license, McLennan says. "But even if the codes are not open source, the unique middleware allows scientists to run the tools and test the behavior of the models," McLennan says. Since launching it, Purdue has developed other hubs using the same software platform to study cancer research and care, biofuels, environmental modeling, pharmaceutical engineering, among other pursuits. It's now constructing a dozen more hubs as well, and some are for outside agencies, such as the Environmental Protection Agency. And researchers at Notre Dame are using the software to build their own hub for biological adaption to climate change.

"Having our software as open source allows these other sites to pick this up and create their own hubs in their own machines," McLennan says. "It shows that this kind of effort can go far beyond nanoHUB.org, and take hold across a wide variety of science and engineering disciplines." ■

Further Reading

Feller, J., Fitzgerald, B., Hissam, S.A., and Lakhani, K.R.

Perspectives on Free and Open Source Software. MIT Press, Cambridge, MA, 2005.

Ince, D.

If you're going to do good science, release the computer code too. *Manchester Guardian*, Feb. 5, 2010.

McLennan, M. and Kennell, R.

HUBzero: a platform for dissemination and collaboration in computational science and engineering. *Computing in Science and Engineering* 12, 2, March/April 2010.

PurdueRCAC

HUBzero Cyberinfrastructure for Scientific Collaboration.

http://www.youtube.com/watch?v=Mr0GA_TluGY

Dennis McCafferty is a Washington, D.C.-based technology writer.

© 2010 ACM 0001-0782/10/1000 \$10.00

Call for Nominations

The ACM Doctoral Dissertation Competition

Rules of the Competition

ACM established the Doctoral Dissertation Award program to recognize and encourage superior research and writing by doctoral candidates in computer science and engineering. These awards are presented annually at the ACM Awards Banquet.

Submissions

Nominations are limited to one per university or college, from any country, unless more than 10 Ph.D.'s are granted in one year, in which case two may be nominated.

Deadline

Submissions must be received at ACM headquarters by **October 31, 2010** to qualify for consideration.

Eligibility

Each nominated dissertation must have been accepted by the department between October 2009 and September 2010. Only English language versions will be accepted. Please send a copy of the thesis in PDF format to emily.eng@acm.org.

Sponsorship

Each nomination shall be forwarded by the thesis advisor and must include the endorsement of the department head. A one-page summary of the significance of the dissertation written by the advisor must accompany the transmittal.

Publication Rights

Each nomination must be accompanied by an assignment to ACM by the author of exclusive publication rights. (Copyright reverts to author if not selected for publication.)

Publication

Winning dissertations will be published by Springer.

Selection Procedure

Dissertations will be reviewed for technical depth and significance of the research contribution, potential impact on theory and practice, and quality of presentation. A committee of five individuals serving staggered five-year terms performs an initial screening to generate a short list, followed by an in-depth evaluation to determine the winning dissertation.

The selection committee will select the winning dissertation in early 2011.

Award

The Doctoral Dissertation Award is accompanied by a prize of \$20,000 and the Honorable Mention Award is accompanied by a prize of \$10,000. Financial sponsorship of the award is provided by Google.

For Submission Procedure

See <http://awards.acm.org/html/dda.cfm>



V



DOI:10.1145/1831407.1831417

Martin Campbell-Kelly

Historical Reflections Victorian Data Processing

Reflections on the first payment systems.

I AM ONE OF those individuals known as a “historian of computing.” Perhaps we are stuck with that appellation, but it can lead one to suppose that all the most significant and important things in information processing happened after the invention of the digital computer. Of course, we usually give a nod to Charles Babbage’s calculating engines and Herman Hollerith’s punched card machines. But this, too, is misleading because it suggests that machinery was always central to data processing. The fact is that the Victorian world was awash with data and with organizations that processed it; and they usually used nothing more technologically advanced than pen and paper. The Bankers’ Clearing House—the first payment system—is just one of many examples.

The Bankers’ Clearing House was established in London in the early 1800s. Interestingly, we owe the first description of the Bankers’ Clearing House to Charles Babbage. Today we think of Babbage primarily as the inventor of calculating machines, but in his lifetime he was better known as a scien-



London bankers’ clerks meet at the Clearing House in Post Office Court, Lombard Street, to exchange cheques and settle accounts, circa 1830.

tist and an economist of international standing. In 1832 he published the first economic treatise on mass production, *The Economy of Machinery and Manufactures*.¹ It is there that he published his account of the Bankers’ Clearing House. When Babbage wrote his book, the Bankers’ Clearing House was a se-

cretive organization that was practically unknown to the general public (not least because the organization handled very large sums of cash). It happened, however, that Babbage was on good terms with Sir John Lubbock, a partner of Lubbock’s Bank and a founder of the Clearing House. Lubbock was an



The New York Clearing House circa 1853.

amateur scientist in his spare time and both he and Babbage were members of the Royal Society. Using this connection, Babbage talked his way in.

Walk Clerks

The origins of the Bankers' Clearing House are obscure, but they date back to at least the late 1700s.³ At that time, when a firm or an individual received a check (still spelled "cheque" in the U.K.), it would be deposited in the recipient's bank. It was then necessary for a clerk to physically present the check to the originating bank, exchange it for cash, and return with the money to his home bank. As the volume of checks grew, each bank employed a "walk clerk" whose job it was to take all the checks due for payment, visit each bank in turn, obtain payment, and return to his bank with a large amount of cash. Walking through the City of London with a large bag of money was, to say the least, unwise, although it went on for many years.

Around 1770, the walk clerks made an informal arrangement to abandon their walks and instead meet at an agreed time in the Five Bells public house in Lombard Street. There they could perform all their financial transactions within the safe confines of four walls. In the early 1800s, the proprietors of the banks at last recognized the merit of this arrangement and formally created the Bankers' Clearing House. When Babbage wrote his account in 1832, it had already

been running for a quarter of a century. Babbage described the operation of the Bankers' Clearing House almost in terms of an algorithm—though one executed by people, not machinery. He wrote: "In a large room in Lombard Street, about 30 clerks from the several London bankers take their stations, in alphabetical order, at desks placed round the room; each having a small open box by his side, and the name of the firm to which he belongs in large characters on the wall above his head. From time to time other clerks from every house enter the room, and, passing along, drop into the box the checks due by that firm to the house from which this distributor is sent."

Thus during the day each bank

Babbage described the operation of the Bankers' Clearing House almost in terms of an algorithm—though one executed by people, not machinery.

dropped off the checks on which it was owed payment and received checks on which it was due to make payment. By adding up all the checks on which it owed money, and all those on which it had to pay out, a bank could calculate exactly the total amount it would have to pay out or would receive that day. At 5 P.M. precisely, the Inspector of the Clearing House took his place on a rostrum, and the debtor banks went up one-by-one to pay what they owed on the day. When this was complete, the banks that were owed money stepped up to the rostrum for payment. When the last bank had been paid, the Inspector was left with a balance of exactly zero. That, of course, assumed that no one had made an arithmetic error. A paper trail of preprinted forms completed by each bank enabled any errors to be traced—but this was a rare occurrence.

Transaction Processing

The amount of money flowing through the Bankers' Clearing House was staggering. In the year 1839, £954 million was cleared—equivalent to \$250 billion in today's currency. However, one of the benefits of the system was that the banks now needed to bring only a relatively small amount of money to the Clearing House. On any day, the totals of checks received and checks paid out would tend to cancel each other out, so that a bank needed only the difference between these two amounts. For example, on the busiest single day of 1839, when £6 million was cleared, only approximately £1/2 million in bank notes was used for the settlement. In his account of the Clearing House, Babbage noted that if the banks were to each open an account with the Bank of England, no money in the form of cash would be needed at all. All that the Clearing House would have to do would be to adjust the account that each bank held with the Bank of England at the close of the business day. This innovation was instituted in 1850, and the physical movement of money was entirely replaced by pen-strokes in an accounting ledger. It was a key moment in both fiscal and information processing history, and Babbage recognized it as such.

The U.S. quickly adopted—and improved on—the British clearing system. The first clearing house was

opened in New York in 1853, located on the fourth floor of the Bank of New York on the corner of Wall Street and William Street. One of the difficulties of the New York clearing operation was that there were over 50 banks in the city and it was realized that the exchanging of checks—as described by Babbage—would create too much confusion and foot traffic. Some nameless genius came up with the brilliant solution depicted in the image on the preceding page of this column. The New York Clearing House constructed a very large oval table, approximately 70 feet in length, with enough working space for each bank. According to a contemporary account,² at 10 o'clock precisely, two clerks from each bank took their places at the table—one seated inside the table and the other standing outside, facing his colleague. At the manager's signal, the clerks outside the table would take one pace forward and perform the day's transactions with the bank they now faced. The process was then repeated, the circle of clerks advancing one pace at a time to the next station "resembling in its movement a military company in lockstep."

After about six minutes the clerks were back in their original positions, the distribution process completed. After that, it was just a matter of balancing the books. If there was a failure to get a zero balance, then there was a system of checks and double-entry accounting so that the error could be detected. Another Yankee innovation, which reputedly cut down on the number of errors, was a system of fines. If an error was found quickly there was no fine, but if it was not detected within an hour a fine of two or three dollars was imposed on the offender, which doubled and quadrupled, the longer it took to find.

The New York Clearing House flourished, and other American financial centers established their own clearing houses—Boston in 1856, Philadelphia in 1858, followed by Chicago and St. Louis some years later.

Persistence of System

You might wonder what happens when you write a check today. In terms of the system, the process is not very different from that of the 19th century. Of course, the technology employed has changed

The longevity of information systems is one of the great lessons of computer history.

beyond recognition. In the 1960s the great innovation was check-reading machines—for which MICR and OCR fonts were designed, and these still appear on the face of a check. Once data had been extracted from the check, it was transferred to magnetic tape for computer processing. It was said at the time that without banking automation it would not have been possible for millions of ordinary Americans to have checking accounts, or to write checks for very small sums of money. By the 1980s, electronic data transfer eliminated much of the physical handling of data. But again, the underlying information system was little altered.

The longevity of information systems is one of the great lessons of computer history. Although new layers of technology are constantly applied to information systems, making transactions faster and cheaper, the underlying systems are remarkably stable and persistent, although of course they do gently evolve over time. We may glory in today's information technology, but one day it will be swept aside—and when it is, and we have logged off for the last time, these venerable systems will survive for another generation of technology. Those Victorian office makers perhaps built better than they knew, and we should salute them. **□**

References

1. Babbage, C. *The Economy of Machinery and Manufactures*. Charles Knight, London, 1832.
2. Gibbons, J.S. *The Banks of New York, their Dealers, the Clearing House, and the Panic of 1857*. Appleton, New York, 1864.
3. Matthews, P.W. *The Banker's Clearing House: What it Is and What it Does*. Pitman, London, 1921.

Martin Campbell-Kelly (M.Campbell-Kelly@warwick.ac.uk) is a professor in the Department of Computer Science at the University of Warwick, where he specializes in the history of computing.

Copyright held by author.

Calendar of Events

October 15–16

Consortium for Computing Sciences
in Colleges (CCSC) Eastern,
Huntingdon, PA,
Contact: Kruse Jerry,
Email: kruse@juniata.edu

October 15–16

Consortium for Computing Sciences
in Colleges Rocky Mountain,
Fort Collins, CO,
Contact: Reeves Tim,
Email: reeves@sanjuancollege.edu

October 16–20

6th Nordic Conference on
Human-Computer Interaction,
Reykjavik, Iceland,
Contact: Ebba Hvannberg,
Email: ebba@hi.is

October 17–21

The 13th ACM International
Conference on Modeling,
Analysis
and Simulation of Wireless and
Mobile Systems,
Bodrum, Turkey,
Contact: Azzedine Boukerche,
Email: boukerch@site.uottowa.ca

October 17–21

Systems Programming
Languages and Applications:
Software for Humanity (formerly
known as OOPSLA),
Reno, NV,
Contact: William R Cook,
Email: wcook@cs.utexas.edu

October 18–20

Symposium on Algorithmic
Game Theory,
Athens, Greece,
Contact: Spirakis Pavlos,
Email: spirakis@cti.gr

October 20–22

International Conference
on Cyberworlds,
Singapore,
Contact: Alexei Sourin,
Email: assourin@ntu.edu.sg

October 21

Workshop on Facial Analysis
and Animation,
Edinburgh, UK,
Contact: Cosker Darren,
Email: D.P.Cosker@cs.bath.ac.uk



Michael A. Cusumano

DOI:10.1145/1831407.1831418

Technology Strategy and Management Platforms and Services: Understanding the Resurgence of Apple

Combining new consumer devices and Internet platforms with online services and content is proving to be a successful strategy.

ON MAY 27, 2010 the technology world experienced a remarkable passing of the baton: Apple went beyond Microsoft to become the world's most valuable technology company in terms of stock market value. It was also on that day the second most valuable U.S. company overall, behind only Exxon Mobil.^a Given Apple's struggles with operating losses and a steep decline in market value in the early 2000s, this resurgence is extraordinary. It reflects not only a series of product innovations but also a shift in strategy that takes advantage of two important trends in the world of high technology: the rising importance and value of an industrywide *platform* company with a large and growing global ecosystem for complementary innovation (versus a standalone *product* company that has to do the lion's share of innovation on its own); and the rising importance and value of services, especially automated services that deliver the digital content and software applications that make these hardware products and platforms so valuable to users.

In terms of platform leadership, Apple has become more like archival Microsoft, but Apple remains a far more innovative and pioneering product company as Steve Jobs and his team have successfully blended computers with consumer electronics

and telephony. The latest transformation began during 2001–2003 with the iPod and iTunes music service. Apple then gained speed from 2007 with the iPhone and App Store. In 2010, the innovations continued with the iPad, which can run existing iPhone ap-

Microsoft and Apple financial comparison, 2000–2009. Units: \$million, %

	Microsoft			Apple		
	Revenues	Operating Profits (%)	Year-End Market Value	Revenues	Operating Profits (%)	Year-End Market Value
2009	\$58,437	34.8%	\$267,323	\$36,537	21.0%	\$190,980
2008	60,420	37.2	149,769	32,479	19.3	118,441
2007	51,122	36.2	287,617	24,006	18.4	74,499
2006	44,282	37.2	251,464	19,315	12.7	45,717
2005	39,788	36.6	233,927	13,931	11.8	29,435
2004	36,835	24.5	256,094	8,279	3.9	8,336
2003	32,187	29.7	252,132	6,207	(loss)	4,480
2002	28,365	29.2	215,553*	5,742	0.3	4,926
2001	25,296	46.3	258,033*	5,363	(loss)	7,924
2000	22,956	47.9	302,326*	7,983	6.5	5,384
1995	5,937	35.3	34,330*	11,062	6.2	4,481

Notes: Fiscal year data. Market value is for calendar year, except when marked with asterisk, then fiscal year, and except for 2009, when market value is as of February 12, 2010.

Source: M. Cusumano, *Staying Power: Six Enduring Principles for Managing Strategy and Innovation in an Unpredictable World* (Oxford University Press, 2010), p. 38. Derived from company Form 10-K annual reports.

a M. Helft and A. Vance, "Apple is No. 1 in Tech, Overtaking Microsoft," *New York Times*, May 27, 2010, p. B1.



plications as well as elegantly display digital content, including books, magazines, and video.^b

Access, Control, and the User Experience

We have seen Apple rise even though its products and services remain under tight corporate control compared to more “open” platforms championed by Microsoft and Intel (the Win-Tel OS and PC device), the Linux community (Linux OS), Nokia and the Symbian alliance (mobile OS and cellphones), and Google (Android, Chrome, and the Open Handset Alliance for mobile applications as well as the Google OpenSocial APIs for social networking applications). For example, Apple has barred some applications from running on the iPhone, including Google Voice. It does not permit its devices to run the most common technology for handling video on the Internet—Adobe Flash. Legal use of the iPhone remains limited to official Apple partners such as AT&T in the U.S. Google

also has criticized Apple’s programming rules for the iPhone and iPad that prohibit application developers from using Google’s advertising technology.^c In my terminology, these kinds of restrictions make Apple’s platforms neither fully open (such as Linux) nor fully closed (such as a proprietary system owned and dominated by one company), but rather “closed, but not closed,” or perhaps “open, but not open.” That is, the platforms are based on proprietary technology, and Apple controls the user experience as well as what applications or content or service contracts can operate on its devices. At the same time, though, Apple has been gradually loosening up access for outside application developers and content providers, especially during 2009–2010.

In an earlier column (“The Puzzle of Apple,” September 2008), I admitted to being frustrated by Apple’s historical reluctance to open up the programming interfaces to its new products and provide easier access to its services or to license its superior software operating system. It pursued this “closed” approach most

famously with the Macintosh, introduced in 1984, but continued this strategy with the initial versions of the iPod, iTunes, the iPhone, and the App Store. Nevertheless, the Apple ecosystems are now as vibrant as any in high technology. Not only are there thousands of applications and accessories available for the iPod made by a wide variety of companies. There were also some 225,000 applications for the iPhone as of mid-2010, many of which work on the iPod and iPad as well as the Macintosh. Apple also was receiving some 15,000 submissions for iPhone applications each week in 30 languages and approving about 95% within seven days.^d By contrast, Google’s Android community had only built approximately 50,000 applications as of mid-2010. To be sure, Apple and Google both trail by far the millions of applications built for Microsoft Windows since the early 1990s. But most computing devices are now mobile phones, and that is where the action lies in software applications development.

^b This article is based on Chapter 1 of M. Cusumano, *Staying Power: Six Enduring Principles for Managing Strategy and Innovation in an Uncertain World* (Oxford University Press, 2010), 30–31, 34–44.

^c S. Morrison and I. Sherr, “Google Blasts Apple over iPhone Ad Changes,” *Wall Street Journal*, June 9, 2010; <http://online.wsj.com/>

^d G. Hora, “95% iPhone Apps Approved in 7 Days,” *Cooltechzone.com*, June 7, 2010; <http://www.cooltechzone.com/2010/06/07/95-iphone-apps-approved-in-7-days/>

Synergies and Network Effects

It is possible that Steve Jobs planned all along to open up the iPod and iPhone programming interfaces and allow more open use of the iPhone beyond a few select partners. The reality is that Apple finally seems to have figured out how to create synergies and powerful network effects across its products and complementary services (see my earlier column “The Evolution of Platform Thinking,” January 2010). The iPod, iPhone, and iPad devices, as well as the iTunes service, all work particularly well with the Macintosh computer, and have some interoperability with Windows. And providing its own essential complements—like Microsoft has always done for DOS and Windows—has become critical to Apple’s success. Apple’s products, despite their elegant designs and unique user interfaces, are not very valuable without external digital content such as music and video files and a variety of applications and accessories. Apple cleverly found a way to provide the key complementary platforms itself—the iTunes Store and the Apple App Store, and now an iBooks store. Moreover, these are *automated* services, with low costs and high potential profit margins. Apple is being smart and encouraging the ecosystem development by sharing most (about 70%) of these revenues with the content owners and application developers.

Apple’s financial break with its past is truly astounding (see the table on the preceding page of this column). In 1995, Apple was nearly *twice* the size of Microsoft in annual revenues (approximately \$11 billion to \$6 billion) but its market valuation was only about *40%* of revenues. By contrast Microsoft’s value was nearly *six times* revenues—reflecting Microsoft’s greater growth prospects as well as operating profit margins that were also about six times Apple’s (35% versus 6%). Indeed, Apple shrunk in subsequent years whereas Microsoft’s sales exploded as Windows 95 became the basis for a new generation of desktop PCs as well as Internet-enabled consumer and enterprise products.

When iPod sales began to surge in 2005, Apple’s revenues, profits, and valuation also began to surge. In fact, by moving beyond the computer business and into consumer electronics

In the long run, the most valuable part of the Apple franchise might end up being its online services and content platforms (iTunes and the App store).

and then mobile phones, Apple’s revenues have risen several times faster than the overall PC industry. Its sales jumped from \$6.2 billion in 2003, with an operating loss, to over \$36 billion in 2009, with a 21% operating profit margin. In addition, Macintosh computers in 2009 made up only 38% of Apple’s revenues, down from 72% in 2003. The iPod accounted for 22% of 2009 revenues, music products 11%, and the iPhone approximately 18%. Software and services as well as hardware peripherals generated the other 12% of sales. It is striking how Apple’s market value remained less than its annual revenues for so many years while Microsoft’s market value was 8 to 13 times revenues. But here too, by 2005, the tide had turned. Apple’s value has continued to rise, reaching five times revenues by the end of 2009 and then finally surpassing Microsoft, whose value has been flat or dropping for a decade due to commoditization of PC hardware and software and its inability to move much beyond the PC. In particular, Microsoft’s attempts to emphasize tablet computers as well as copy the iPod with the Zune digital media player and compete in smartphones with Windows devices have failed miserably.

Current Situation

Not everything is completely smooth for Apple, however. The company has been clashing with Google and its rival mobile OS (Android). Google is the champion of open systems and always tries

to force semi-open or semi-closed platforms to “open up” so that it can get unrestricted access to information on user behavior through searches and thereby sell more and better targeted ads. Apple is also clashing with Adobe, refusing to support the Flash technology on the iPhone or the iPad, even though Flash is used for the vast majority of videos and advertisements on the Web. The U.S. Department of Justice and the Federal Trade Commission are reportedly reviewing Apple’s restrictive policies to see if they violate antitrust laws.^e Apple has near-monopoly shares (approximately 70% or so of the market) for both digital devices (iPod) and digital content services (iTunes). But, for the moment, users continue flocking to Apple products because of their elegance and the superior user experience.

Apple is still less profitable than Microsoft because hardware devices are more expensive to replicate than software products. Apple also has dropped its prices to counter copycat smartphone products from Nokia, Samsung, HTC, and other firms. In the long run, the most valuable part of the Apple franchise might end up being its online services and content platforms (iTunes and App Store). The hardware products may simply become platforms to drive revenue from selling or aggregating high-margin automated digital products. Apple’s acquisition in December 2009 of Lala, the streaming Web music service, also provides “cloud-like” technology that could enable Apple customers to store their music, photos, or videos and listen to or view their content from different devices, anywhere and anytime. In short, rather than in a Microsoft world, we are clearly now living much more in a world defined by Apple as well as Google, Facebook, and other firms that have successfully married new consumer devices and Internet platforms with a variety of online services and content. ■

^e J. Kosman, “An antitrust app: Apple may be in the eye of a regulatory storm,” *New York Post*, May 3, 2010; <http://www.nypost.com/>

Michael A. Cusumano (cusumano@mit.edu) is a professor at the MIT Sloan School of Management and School of Engineering and author of *Staying Power: Six Enduring Principles for Managing Strategy and Innovation in an Uncertain World* (Oxford University Press, 2010).

Copyright held by author.v

Inside Risks

Risks of Undisciplined Development

An illustration of the problems caused by a lack of discipline in software development and our failure to apply what is known in the field.

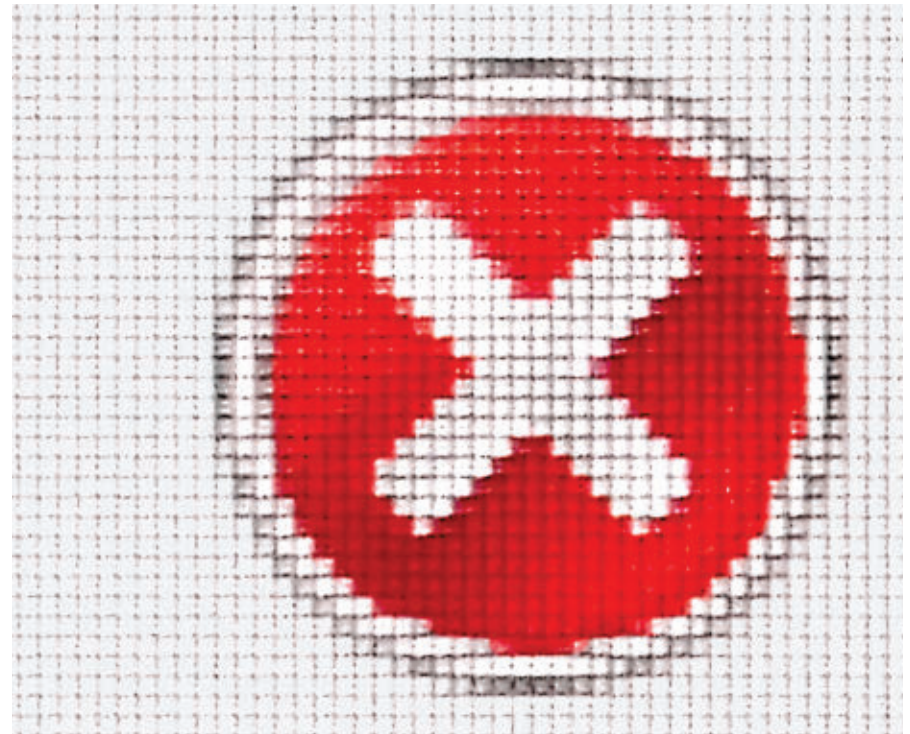
THE BRANCHES OF engineering (such as civil, electrical, and mechanical), are often referred to as disciplines for good reason. Associated with each specialty is a set of rules that specify:

- ▶ checks that must be made;
- ▶ properties that must be measured, calculated, or specified;
- ▶ documentation that must be provided;
- ▶ design review procedures;
- ▶ tests that must be carried out on the product; and
- ▶ product inspection and maintenance procedures.

Like all professional education, engineering education is designed to prepare students to meet the requirements of the authorities that regulate their chosen profession. Consequently, most graduates are taught they must carry out these procedures diligently and are warned they can be deemed guilty of negligence and lose the right to practice their profession if they do not.

Because they are preparing students for a career that can last many decades, good engineering programs teach fundamental principles that will be valid and useful at the end of the graduate's career. Engineering procedures are based on science and mathematics; and graduates are expected to understand the reasons for the rules, not just blindly apply them.

These procedures are intended to



assure that the engineer's product:

- ▶ will be fit for the use for which it is intended;
- ▶ will conform to precise stable standards;
- ▶ is robust enough to survive all foreseeable circumstances (including incorrect input); and
- ▶ is conservatively designed with appropriate allowance for a margin of error.

In some areas, for example building and road construction, the procedures

are enforced by law. In other areas, and when engineers work in industry rather than selling their services directly to the public, employers rely on the professionalism of their employees. Professional engineers are expected to know what must be done and to follow the rules even when their employer wants them to take inappropriate shortcuts.

Anyone who observes engineers at work knows that exercising due diligence requires a lot of "dog work." The dull, but essential, work begins in the



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>



design phase and continues through construction, testing, inspection, commissioning, and maintenance. Licensed engineers are given a unique seal and instructed to use it to signify the acceptability of design documents only after they are sure the required analysis has been completed by qualified persons.

Real-World Experience

Recent experiences reminded me that the activity we (euphemistically) call software engineering does not come close to deserving a place among the traditional engineering disciplines. Replacing an old computer with a newer model of the same brand revealed many careless design errors—errors that in all likelihood could have been avoided if the developers had followed a disciplined design process. None of the problems was safety critical, but the trouble caused was expensive and annoying for all parties.

My “adventure” began when the sales clerk scanned a bar code to initiate the process of creating a receipt and registering my extended warranty. There were three codes on the box; not surprisingly, the sales clerk scanned the wrong one. This is a common occurrence. The number scanned bore no resemblance to a computer serial number but was accepted by the software without any warning to the clerk. The nonsense number was duly printed as the serial number on my receipt. My extended warranty was registered to a nonexistent product. I was billed, and no problem was noted until I phoned the customer care line with a question. When I read the serial num-

Computer science students are not taught to work in disciplined ways. In fact, the importance of disciplined analysis is hardly mentioned.

ber from the receipt, I was told that I had purchased nothing and was not entitled to ask questions. After I found the correct number on the box, I was told that my computer was not yet in their system although a week had passed since the sale.

Correcting the problem required a trip back to the store and tricking the company computer by returning the nonexistent machine and buying it again. In the process, my name was entered incorrectly and I was unable to access the warranty information online. After repeatedly trying to correct their records, the help staff told me it could not be done.

A different problem arose when I used the migration assistant supplied with the new computer to transfer my data and programs to the new machine. Although the description of the migration assistant clearly states that incompatible applications will be moved to a special directory rather than installed, a common software package on the old machine, one that was not usable or needed on the new one, was installed anyway. A process began to consume CPU time at a high rate. Stopping that process required searching the Internet to find an installer for the obsolete product.

The next problem was an error message informing me that a device was connected to a USB 1.1 port and advising me to move it to a USB 2.0 port. My new computer did not have any 1.1 ports so I called the “care” line for advice. They had no list of error messages and could not guess, or find out, which application or component of their software would issue such a message or under what conditions it should be issued. They referred the problem to developers; I am still waiting for a return call.

These incidents are so petty and so commonplace that readers must wonder why I write about them. It is precisely because such events are commonplace, and so indicative of lack of discipline, that such stories should concern anyone who uses or creates software.

As early as the late 1950s, some compilers came with a complete list of error messages and descriptions of the conditions that caused them. Today, such lists cannot be found. Often,

when reviewing a system, I will pick a random message or output symbol and ask, “When does that happen?” I never get a satisfactory answer.

There are methods of design and documentation that facilitate checking that a programmer has considered all possible cases (including such undesired events as incorrect input or the need to correct an earlier transaction) and provided appropriate mechanisms for responding to them. When such methods are used, people find serious errors in software that has been tested and used for years. When I talk or write about such methods, I am often told by colleagues, experienced students, and reviewers that, “Nobody does that.” They are right—that’s the problem!

Much of the fault lies with our teaching. Computer science students are not taught to work in disciplined ways. In fact, the importance of disciplined analysis is hardly mentioned. Of course, just telling students to be diligent is not enough. We need to:

- ▶ teach them what to do and *how* to do it—even in the first course;
- ▶ use those methods ourselves in *every* example we present;
- ▶ insist they use a disciplined approach in *every* assignment in *every* course where they write programs;
- ▶ check they have inspected and tested their programs diligently, and
- ▶ test their ability to check code systematically on examinations.

Many of us preach about the importance of determining the requirements a software product must satisfy, but we do not show students how to organize their work so they can systematically produce a requirements specification that removes all user-visible choices from the province of the programmer.

Some of us advise students to avoid dull work by automating it, but do not explain that this does not relieve an engineer of the responsibility to be sure the work was done correctly.

Innovation and Disciplined Design

It has become modish to talk about teaching creativity and innovation. We need to tell students that inventiveness is not a substitute for disciplined attention to the little details that make the difference between a product we like and a product we curse. Students need to be told how to create and use check-

Even sophisticated and experienced purchasers do not demand the documentation that would be evidence of disciplined design and testing.

lists more than they need to hear about the importance of creativity.

It is obviously important to give courses on picking the most efficient algorithms and to make sure that students graduate prepared to understand current technology and use new technology as it comes along, but neither substitutes for teaching them to be disciplined developers.

Disciplined design is both teachable and doable. It requires the use of the most basic logic, nothing as fancy as temporal logic or any of the best-known formal methods. Simple procedures can be remarkably effective at finding flaws and improving trustworthiness. Unfortunately, they are time-consuming and most decidedly not done by senior colleagues and competitors.

Disciplined software design requires three steps:

1. Determine and describe the set of possible inputs to the software.
2. Partition the input set in such a way that the inputs within each partition are all handled according to a simple rule.
3. State that rule.

Each of these steps requires careful review:

1. Those who know the application must confirm that no other inputs can ever occur.
2. Use basic logic to confirm that every input is in one—and only one—of the partitions.
3. Those who know the application, for example, those who will use the program, must confirm the stated rule is correct for every element of the partition.

These rules seem simple, but reality complicates them:

1. If the software has internal memory, the input space will comprise event sequences, not just current values. Characterizing the set of possible input sequences, including those that should not, but could, happen is difficult. It is very easy to overlook sequences that should not happen.

2. Function names may appear in the characterization of the input set. Verifying the correctness of the proposed partitioning requires knowing the properties of the functions named.

3. The rule describing the output value for some of the partitions may turn out to be complex. This is generally a sign that the partitioning must be revised, usually by refining a partition into two or more smaller partitions. The description of the required behavior for a partition should always be simple but this may imply having more partitions.

Similar “divide and conquer” approaches are available for inspection and testing.

While our failure to teach students to work in disciplined ways is the primary problem, the low standards of purchasers are also a contributing factor. We accept the many bugs we find when a product is first delivered, and the need for frequent error-correcting updates, as inevitable. Even sophisticated and experienced purchasers do not demand the documentation that would be evidence of disciplined design and testing.

We are caught in a catch-22 situation:

- ▶ Until customers demand evidence that the designers were qualified and disciplined, they will continue to get sloppy software.

- ▶ As long as there is no better software, we will buy sloppy software.

- ▶ As long as we buy sloppy software, developers will continue to use undisciplined development methods.

- ▶ As long as we fail to demand that developers use disciplined methods, we run the risk—nay, certainty—that we will continue to encounter software full of bugs. ■

David L. Parnas (parnas@mcmaster.ca) is Professor Emeritus at McMaster University and the University of Limerick as well as President of Middle Road Software. He has been looking for, and teaching, better software development methods for more than 40 years. He is still looking!

Copyright held by author.



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

Kode Vicious Version Aversion

The way you number your releases communicates more than you might think.

Dear KV,

I'm working on a small, open-source project in my free time and trying to figure out a reasonable way to number releases. There are about 10 of us working on the project right now, and there seem to be 10 different opinions about when we should roll out the major and minor numbers for our project. Since the software is only in the alpha stage, I thought it was premature to come up with a numbering scheme; but once someone else posted it on our mailing list we decided we should tackle the problem now so we don't have to think about it again later. When you're working on software, when do you roll out a new version number?

Averse to Aversion

Dear Aversion,

You have 10 developers and you have

A good version-numbering system can be used to track the change in functionality of a piece of software.



only 10 opinions? I was expecting you to say you had 20 opinions, so right from the start it looks like you're not in as bad a shape as you might think. Choosing a versioning scheme is more important than most programmers really understand, in part because a versioning scheme is a form of human communication, and human communication...well, let's just say that many programmers don't get that at all.

A versioning scheme serves a few

important purposes. The most obvious is allowing users to know where they are in the evolution of your software, but version numbers also communicate a good deal more information.

A good version-numbering system can be used to track the change in functionality of a piece of software. A new feature, or a major change to a feature, should always result in a new version number being assigned. I've always been happiest, or perhaps least miser-

able, with three-number versioning—Major.Minor.BugFix—where the major version changes for large features, the minor version changes for small features, and the last number is a bug-fix release. The bug-fix number is perhaps the easiest to understand. After a number of bugs are fixed, you want to release new software into the field so that users can benefit from the work your team has done. Increase the last number and release a new version. Bug fixes should never change an API in a library or introduce or significantly change a feature.

The difference between a major and a minor version change can be tricky depending on the software you are writing. Minor versions can be rolled out so long as there are no backward-incompatible changes to the system. What this means is that if you have a change to your software that will break another piece of software that depends on your code, or that breaks a user assumption about how your system works, that requires a major version change. Additive changes, such as new APIs, or new features that do not break backward compatibility can be rolled into minor version updates.

The slippery slope is figuring out how many minor changes add up to a major change. If your software has 30 features and you add 10 new ones, even if none of them touches the original 30, shouldn't that necessitate a major version change? I think it should, but not everyone agrees. Of course those who don't agree—well, let's just leave that alone, shall we?

One thing that software versions communicate is the rate of change in your software. If your software goes from 1.0 to 2.0 in a month, then either your team is performing miracles, which I find highly suspect, or they're claiming major changes when none has really occurred. A very high rate of minor or bug releases can also indicate problems in a project—in particular, that it is buggy. Although there is no perfect rate for releases, they should definitely slow down a bit as a product matures. Too-frequent releases often mean that a piece of software is immature and perhaps lacks staying power.

Another pattern, in some projects, is never to release a 1.0, but to release a lot of 0.x's. A particularly egregious version of this, pun intended, was the

The difference between a major and a minor version change can be tricky depending on the software you are writing.

Ethereal project, which, after more than 10 years of development, got to the point of releasing a 0.99.5. This was just a way, as far as I could tell, of moving the version number to the right. The software itself is quite good and widely used, but its versioning system was quite odd. Now that the project has been renamed Wireshark, it seems to have moved to a more traditional Major.Minor style of versioning.

Version numbers should also be able to correlate related pieces of software. One of the banes of my existence is the Linux versioning system, although I believe this has more to do with the way Linux itself is developed. The fact that there are now many different operating-system kernels that I might have to choose from to use a piece of related software is simply maddening. A recent example involves having to find the correct bits so I could use a driver for a new piece of hardware. The standard release was 2.6.18-194.3.1.el5, but the version I needed was, 2.6.18-164.el. And just what do those numbers mean? Of course I could work them out with some Web searches, but still, the fact that kernel APIs have changed enough within those minor releases that a driver couldn't work is madness. Even looking at kernel.org, the source of all things Linux kernel, isn't much help. At the time this column was written these are the kernels that are listed as stable for the 2.6 version of the kernel:

2.6.34	2010-05-16
2.6.33.5	2010-05-26
2.6.32.15	2010-06-01
2.6.31.13	2010-04-01
2.6.27.47	2010-05-26

Now, I ask you, how does 2.6.34 come out 10 days before 2.6.33.5, and how can all of these be stable? How do they even relate to each other?

Of course, it's not just open-source projects that have problems with versioning. The biggest software company of them all seems to have one of the most ridiculous versioning schemes of all. Based on the names alone, how does one figure the difference between Windows 95, Windows 98, Windows ME, Windows NT, Windows XP, Windows XP Service Pack 2, and Vista? I can list them in order only because I have watched them all come and go, and, happily, never installed any of them on my own machines. Perhaps if you hash the names just right, then they turn into monotonically increasing version numbers.

One last thing to note is that you should not tie yourself down with your versioning scheme; remember that you may have to be flexible. I once worked on a product with a 1.0.1b release. The b release was necessitated by a not-so-amusing mistake, wherein a developer decided that if a user saved a file without an extension, the developer would provide one. The extension was a four-letter word that is included in George Carlin's list of seven things you can never say on TV, and which one should never really have as a file extension either. I think you get the idea. The developer had meant to remove that particular feature before the code was released but forgot, and so, we had 1.0.1 and 1.0.1b releases. We could have made a 1.0.2 release, but, really, there was just one change—though I do believe we should have made the release 1.0.1f.

KV

Related articles on queue.acm.org

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

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

Making Sense of Revision-Control Systems

Bryan O'Sullivan

<http://queue.acm.org/detail.cfm?id=1595636>

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

SCORE: Agile Research Group Management

Adapting agile software development methodology toward more efficient management of academic research groups.

WORKING WITH AND MENTORING Ph.D. students is the central activity in running an academic research group.

At the start of our careers as assistant professors, we took a fairly typical approach to managing student interactions: once or twice per week, we met with each of our students in prescheduled sessions of approximately half-hour or hour-long duration. However, this approach started breaking down as we gained more students and our other responsibilities increased: our time became fragmented and inefficiently used; hard-earned lessons were not shared effectively among students; and our group lacked any real cohesion or identity. In September 2006, we learned about Scrum,¹ an “agile” software development methodology, and realized we might be able to solve some of the problems we were having by adapting it to our research group.

In this Viewpoint, we briefly describe the resulting process, which we call SCORE (Scrum fOr REsearch). We have been using SCORE for several years, and have discovered it has many benefits, some we intended and some that surprised us. While every situation is different, we hope others may learn from our approach, in idea if not in form, and that we might inspire further discussion of research group management strategies. A longer version of this Viewpoint, with more information and space for feedback, is available at the SCORE Web page.³



SCORE

The major feature of SCORE is its meeting structure, which consists of two parts:

Regular all-hands status meetings. Several times a week (late mornings on Tuesdays, Wednesdays, and Fridays in our group), the group meets for a 15-minute, all-hands status meeting, modeled after the daily “scrum” meeting for which Scrum is named. During the meeting each person gives a brief update on what they did since the last meeting, what problems they encountered, and what they plan to do for the next meeting. If someone cannot physically attend the meeting, they may conference-call in, but physical presence is much preferred. Everyone stands during the meeting, to encourage brevity.

Though brief, status reports are information-rich. Students report on a wide range of activities, such as progress in implementing code, carrying out an experiment, reading a paper, working on a proof, writing up a result, or preparing a talk. We encourage

students to present their status to the group, rather than just to the faculty. Students may also say there has been no change in their status, typically because of classwork or for personal reasons.

On-demand research meetings. Whenever a student or one of us thinks a more in-depth, one-on-one meeting is needed, we schedule it on demand. Since only the status meetings are prescheduled, we are able to reserve large blocks of time (most afternoons) for holding on-demand meetings, and the meetings can be of varying durations—as long, or as short, as required. We often schedule on-demand meetings immediately following a status meeting, typically for the same afternoon.

We kicked off SCORE with a “research review day” of conference-style talks describing ongoing research projects, to provide context for the ensuing status reports. As needed, we inject some of these talks to inform the group of a new project or to “checkpoint” a recent major result. To help increase group spirit further, we have a weekly lunch, and we also hold a reading group one day per week.

Why It Works for Us

Though simple, SCORE works remarkably well for us. After nine months of using SCORE, we surveyed our students for feedback, and their responses were very positive. Since then, colleagues at various institutions have adopted aspects of SCORE, and they have offered us feedback. From these and our

own assessments, we have identified the following list of SCORE's benefits. While none of these benefits is unique to SCORE, it is convenient that SCORE has them all.

More efficient time use for faculty. A major problem we had with prearranged meetings was schedule fragmentation. When a student had only a quick status update, the meeting slot was too long, and the remaining chunks of time were difficult to use. On the other hand, when a student had a deep technical issue to explore, the slots were too short. Because we had so many prescheduled meetings, subsequent discussion might have to wait a day or two, slowing progress. Moreover, context-switching frequently over the course of many meetings was very draining, reducing the meetings' effectiveness.

SCORE solves these issues very well by separating quick status reports from in-depth technical discussions. On-demand meetings have a clear purpose and are therefore much more productive than our weekly meetings used to be.

Improved productivity for students. By our own observations and those of our students, frequent student-adviser contact at SCORE status meetings has improved student morale and productivity. In response to our survey, one student said: "I like the frequency of the status meetings. Frequent meetings make incremental progress necessary: to have something to say at each meeting, you can't goof off for an extended period of time. Also, if you don't know where to go next, there isn't much time before another meeting, when you can get back on track. On the other hand, the frequency of the meetings means that, if something came up and you don't have anything to report for today, it's not a big deal; you'll have something for tomorrow or the next day."

Most graduate students struggle at some point—one study found "At [UC] Berkeley, 67% of graduate students said they had felt hopeless at least once in the last year."² With thrice-weekly status meetings, we can identify struggling students quickly and therefore help them much sooner than we would have when meeting once per week.

Improved group identity and shared knowledge. By giving each person a

On-demand meetings have a clear purpose and are therefore much more productive than our weekly meetings used to be.

window onto the activities of others, participants learn from others' successes and failures, which helps create a group sense of momentum and accomplishment. One student specifically commented he liked hearing about other students' progress: "I can follow other people's research and 'daily research routine.' That helps because it's interesting and I learn things, but also because I can compare my productivity and have a better idea of how I fare."

More than half of the students surveyed specifically cited a "research community" or "sense of belonging" as a benefit of SCORE. The students said they feel the joy of their fellows' successes, which then creates further motivation and enthusiasm for their own work. At the same time, one student mentioned it was consoling to learn that other students hit slow patches, too: "It helped me with the realization that everyone has rough patches and that it is not a big deal." Several students said regular social gatherings and proximate offices were also important in fostering this sense of community. One student said, "Status meetings and the office atmosphere make it worth my while to come to school." Finally, group meetings remove faculty as the bottleneck to developing new ideas or solving technical problems, as students offer advice and begin collaborations with their fellow students based on what they hear in the meetings.

Can SCORE Work for You?

Every research group is different and must find its own process that works best. We hope knowing about SCORE will prove useful to other groups, either as a new process to experiment with or as inspiration for other ideas. For ex-

ample, instead of SCORE's status meetings there may be other good strategies to engender frequent contact and create opportunities for focused, in-depth meetings. Among others, some possible approaches are regular faculty "office hours" in a lab-space that co-locates many students; less formal "coffee hours"; or perhaps co-locating faculty with students. Lessons learned might be communicated more permanently by incorporating group mailing lists, wikis, or blogs. Prescheduled research meetings may also play a role, for example, for external collaborators who do not attend status meetings.

SCORE can also be further improved, as it is silent about some important elements of running a research group. For example, SCORE has no specific process for generating and discussing research ideas. Currently we explore ideas during the on-demand meetings, but doing so does not take advantage of the perspective of the larger group, nor does it give students a broader view of this process.

We encourage interested readers to read the longer version of this Viewpoint at the SCORE Web page,³ and to provide comments on the ideas and issues raised. The long version goes into more detail about running SCORE in practice, describes elements of SCORE that we tried but did not work for us, and reports more in-depth responses to our student survey. We look forward to further exploring strategies for mentoring and working with graduate students to produce high-quality academic research. ■

References

1. Agile Development Methods Inc. About Scrum—Overview (2008); <http://www.controlchaos.com/about/>
2. Fogg, P. Grad-school blues: Students fighting depression and anxiety are not alone. *The Chronicle of Higher Education*, (Feb. 20, 2009).
3. SCORE Web page; <http://www.cs.umd.edu/projects/PL/score>

Michael Hicks (fmwh@cs.umd.edu) is an associate professor in the Computer Science Department and UMIACS, and an affiliate associate professor in the Electrical and Computer Engineering Department, at the University of Maryland, College Park.

Jeffrey S. Foster (jfoster@cs.umd.edu) is an associate professor in the Computer Science Department and UMIACS at the University of Maryland, College Park.

We would like to thank Jens Palsberg, Dan Boneh, David Notkin, Alex Aiken, and Kathryn McKinley for helpful comments on drafts of this Viewpoint. We would also like to thank all those who have given us feedback on SCORE, especially the students in the programming languages group at the University of Maryland.

Copyright held by author.

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

Clem Cole and Russell Williams discuss Photoshop's long history with parallelism, and what is now seen as the chief challenge.

ACM CASE STUDY

Photoshop Scalability: Keeping It Simple

OVER THE PAST two decades, Adobe Photoshop has become the de facto image-editing software for digital photography enthusiasts, artists, and graphic designers worldwide. Part of its widespread appeal has to do with a user interface that makes it fairly straightforward to apply some extremely sophisticated image editing and filtering techniques. Behind that façade, however, stands a lot of complex, computationally demanding code. To improve the performance of these computations, Photoshop's designers became early adopters of parallelism—in the mid-1990s—through efforts to access the extra power offered by the cutting-edge desktop systems of the day that were powered by either two or four processors. At the time, Photoshop was one of the

only consumer desktop applications to offer such a capability.

Photoshop's parallelism, born in the era of specialized expansion cards, has managed to scale well for the two- and four-core machines that have emerged over the past decade. As Photoshop's engineers prepare for the eight- and 16-core machines that are coming, however, they have started to encounter more and more scaling problems, primarily a result of the effects of Amdahl's Law and memory-bandwidth limitations.

In this ACM Case Study, Adobe Photoshop principal scientist **Russell Williams** speaks with **Clem Cole**, architect of Intel's Cluster Ready program, about how the Photoshop team is addressing these challenges. Although in their current positions they represent different aspects of the parallel-computing landscape, both have long histories of tackling parallelism at the operating-system level.

Prior to joining the Photoshop development team, Williams had a long career as an operating-system designer at companies such as Apple, where he worked on the Copland microkernel, and Elxsi, where he helped develop mini-supercomputers. The diversity of that background now allows him to maintain a well-rounded perspective on parallelism at different levels of the stack.

Cole is a veteran operating-system developer with years of experience in Unix kernel and tool development. His current efforts to advance methods that take advantage of multiple processors—using Intel's next generation of multi-core chips—makes him a fitting interviewer for Williams, whose work in large part builds on top of the platforms Cole helps to create at Intel.

While Photoshop comes with a unique set of problems and constraints, many of the engineering challenges it presents will undoubtedly seem familiar to any software engineer who has ever attempted to achieve parallelism in an application. Still, to get a handle on the issues Photoshop's engineers are facing today, we must first consider the appli-



Clem Cole (left) and Russell Williams

ation's history with parallelism over the past 15 years.

COLE: You've been writing software for a long time, and for the past 11 years you've been working with Photoshop and have become increasingly engaged with its parallel aspects. Which parts of that have proved to be easy and what has turned out to be surprisingly hard?

WILLIAMS: The easy part is that Photoshop has actually had quite a bit of parallelism for a long time. At a very simplistic level, it had some worker threads to handle stuff like asynchronous cursor tracking while also managing asynchronous I/O on another thread. Making that sort of thing work has been pretty straightforward because the problem is so simple. There's little data shared across that boundary, and the goal is not to get compute scaling; it's just to get an asynchronous task going.

I should note, however, that even with that incredibly simple task of queuing disk I/O requests so they could be handled asynchronously by another thread, the single longest-lived bug I know of in Photoshop ended up being nestled in that code. It hid out in there for about 10 years. We would turn on the asynchronous I/O and end up hitting that bug. We would search for it for weeks, but then just have to give up and ship the app without the asynchronous I/O being turned on. Every couple of versions we would turn it back on so we could set off looking for the bug again.

COLE: I think it was Butler Lampson who said the wonderful thing about serial machines is you can halt them and look at everything. When we're working in parallel, there's always something else going on, so the concept of stopping everything to examine it is really hard. I'm actually not shocked your bug was able to hide in the I/O system for that long.

WILLIAMS: It turned out to be a very simple problem. Like so many other aspects of Photoshop, it had to do with the fact that the app was written first for the Macintosh and then moved over to Windows. On the Macintosh, the set file position call is atomic—a single call—whereas on Windows, it's a pair of calls. The person who put that in there didn't think about the fact that the pair of calls has to be made atomic whenever you're sharing the file position across threads.

COLE: Now, of course, you can look back and say that's obvious.

WILLIAMS: In fact, the person who originally put that bug in the code was walking down the hallway one of the many times we set off looking for that thing, smacked his forehead, and realized what the problem was—10 years after the fact.

Anyway, the other big area in Photoshop where we've had success with parallelism involves the basic image-processing routines. Whenever you run a filter or an adjustment inside Photoshop, it's broken down into a number of basic image-processing operations, and those are implemented in a library that's accessed through a jump table. Early on, that allowed us to ship accelerated versions of these "bottleneck routines," as they're called. In the 1990s, when companies were selling dedicated DSP (digital signal processor) cards for accelerating Photoshop, we could patch those bottlenecks, execute our routine on the accelerator card, and then return control to the 68KB processor.

That gave us an excellent opportunity to put parallelism into the app in a way that didn't complicate the implementations for our bottleneck-routine algorithms. When one of those routines was called, it would be passed a pointer—or two or three pointers—to bytes in memory. It couldn't access Photoshop's software-based virtual memory and it couldn't call the operating system; it was just a math routine down at the bottom. That gave us a very simple way—prior to getting down to the math routine—of inserting something that would slice up the piece of memory we wanted to process across multiple CPUs and then hand separate chunks of that off to threads on each CPU.

COLE: The key there is you had an object that could be broken apart into smaller objects without the upper-level piece needing to worry about it. It also helps that you had a nice, clean place to make that split.

WILLIAMS: The other nice aspect is that the thing on the bottom didn't need to know about synchronization. It was still nothing more than a math routine that was being passed a source pointer—or maybe a couple of source pointers and counts—along with a destination pointer. All the synchronization lived in that multiprocessor plug-in that inserted

itself into the jump table for the bottleneck routines. That architecture was put into Photoshop in about 1994. It allowed us to take advantage of Windows NT's symmetric multiprocessing architecture for either two or four CPUs, which was what you could get at the time on a very high-end machine. It also allowed us to take advantage of the DayStar multiprocessing API on the Macintosh. You could buy multiprocessor machines from DayStar Digital in the mid- to late-1990s that the rest of the Mac operating system had no way of taking advantage of—but Photoshop could.

Photoshop has well over a decade of using multiple processors to perform the fundamental image-processing work it does on pixels. That has scaled pretty well over the number of CPUs people have typically been able to obtain in a system over the past decade—which is to say either two or four processors. Essentially, no synchronization bugs ended up surfacing in those systems over all that time.

COLE: That's an amazing statement! Is there an insight associated with that that you can share? What do you think the rest of us can learn from that?

WILLIAMS: I think the big win came from not having to write synchronization for the processing routines that were to be parallelized. In other words, people could write convolution kernels or whatever else it was they needed to do in terms of pixel processing without having to worry about getting all those synchronization issues right. If acquiring one asynch I/O thread was all it took for us to introduce a bug that managed to elude us for 10 years, then it's clear that minimizing synchronization issues is very important.

That said, the way synchronization was approached 10 years ago involved the use of far more error-prone synchronization primitives than what we've got available to us today. Things like "enter critical section" and "leave critical section" on Windows could be really fast, but they were also very error prone. Trying to keep track of whether you've put critical sections every place you might need them, and whether or not you've remembered to leave as many times as you entered, that can all tend to be very difficult and error prone.

The nettlesome bug that managed to re-

main obscured within Photoshop's synchronization code for 10 years serves to illustrate just how tricky parallel programming can be. But it also highlights how much progress has been made in terms of improved resources for managing some of this complexity. Had Photoshop's synchronization been written today using C++'s stack-based locking, for example, it's unlikely a bug of that sort would have been introduced. As processors get more cores and grow in complexity, the need will only intensify for new tools and better programming primitives for hiding the complexity from developers and allowing them to code at higher levels of abstraction.

At the same time, software architects also need to keep an eye on some other fundamental issues. For example, despite using less-sophisticated synchronization primitives in the original design, the Photoshop team was able to essentially forget about complex thread-synchronization problems, in part because the image-processing problem itself was so amenable to parallelization. Also, however, Photoshop's architecture made it possible to establish some very clean object boundaries, which in turn made it easy for programmers to slice up objects and spread the resulting pieces across multiple processors. Indeed, the architects of Photoshop were keenly aware of where their best opportunities for parallelization existed, and they designed the application accordingly.

Generalizing on this, it's clear that—with or without advances in tools and programming abstractions—in order for developers to fully leverage the multicore architectures that are coming, they'll need to be adept at identifying those parts of a program that can benefit most from parallelization. It's in these portions of the code that new tools, techniques, and parallel programming abstractions are likely to have the greatest impact.

COLE: As operating-system designers, we both grew up in a world where we had to deal with parallelism. It's not always clear that the solutions we came up with for our operating systems proved to be the right ones. In an earlier conversation, you mentioned your experience with creating and removing mutexes. We've gotten smarter over the years.



RUSSELL WILLIAMS

I think we've made some incremental gains in our ability to deal with parallelism over the past 20 years, just as we've made stepwise progress on all other programming fronts.



We've learned how to do things that are more efficient, but that doesn't mean it has gotten any easier. What do we have up our sleeves to make it easier?

WILLIAMS: Parallelism remains difficult in a couple of ways. It's one thing to ask for a new Photoshop filter for processing a grid of pixels to do that in parallel. It's quite another thing to say, "I'm going to parallelize and thus speed up the way that Photoshop runs JavaScript actions." For example, I've got a JavaScript routine that opens 50 files one after the other and then performs a set of 50 steps on each one. I don't have control over that script. My job is just to make that—or any other script the user has to run—faster.

I could say, "Rewrite all your scripts so we can design a completely new interface that will let you specify that all these images are to be processed in parallel." That's one answer, but it would require a lot of work on the user's part, as well as on our part. And it would still leave us with the problems associated with parallelizing the opening of an image, parsing the image contents, interpreting the JavaScript, running the key command object through the application framework, and updating the user interface—all of which typically is tied into an app framework and thus involves calling some horrendously sequential script interpreter. Once you start looking at the Amdahl's Law numbers on something like that, it soon becomes apparent that trying to get that to parallelize eight ways would just be completely hopeless.

At the other end of the spectrum you might find, for example, a mathematical algorithm that conceptually lends itself to parallelism simply because it has a bunch of data structures it needs to share. So how hard would it be to correctly implement that mathematically parallelizable algorithm?

I think we've made some incremental gains in our ability to deal with parallelism over the past 20 years, just as we've made stepwise progress on all other programming fronts. Remember that back in the 1970s, there was a lot of talk about the "software crisis," regarding how software was getting more and more complicated, to the point where we couldn't manage the bugs anymore. Well, in response to that, there was no huge breakthrough in software produc-

tivity, but we did realize a bunch of incremental gains from object-oriented programming, improved integrated development environments, and the emergence of better symbolic debugging and checker tools that looked for memory leaks. All of that has helped us incrementally improve our productivity and increase our ability to manage complexity.

I think we're seeing much the same thing happen with parallelism. That is, whereas the earliest Photoshop synchronization code was written in terms of "enter critical section, leave critical section," we now have tools such as Boost threads and OpenGL, which essentially are programming languages, to help us deal with those problems. If you look at Pixel Bender [the Adobe library for expressing the parallel computations that can be run on GPUs], you'll find it's at a much higher level and so requires much less synchronization work of the person who's coding the algorithm.

COLE: The key is that you try to go to a higher level each time so you have less and less of the detail to deal with. If we can automate more of what happens below that, we'll manage to become more efficient. You also mentioned that we have better tools now than we did before. Does that suggest we'll need even better tools to take our next step? If so, what are we missing?

WILLIAMS: Debugging multithreaded programs *at all* remains really hard. Debugging GPU-based programming, whether in OpenGL or OpenCL, is still in the Stone Age. In some cases you run it and your system blue-screens, and then you try to figure out what just happened.

COLE: That much we're aware of. We've tried to build stronger libraries so that programmers don't have to worry about a lot of the low-level things anymore. We're also creating better libraries of primitives, such as open source TBB (Threading Building Blocks). Do you see those as the kinds of things developers are looking to suppliers and the research community for?

WILLIAMS: Those things are absolutely a huge help. We're taking a long hard look at TBB right now. Cross-platform tools are also essential. When somebody comes out with something that's Windows only, that's a nonstarter for



CLEM COLE

Locking your data structures is truly only the beginning. The new tuning problem is going to be a real nightmare.



us—unless there is an exact-equivalent technology on the Mac side as well. The creation of cross-platform tools such as Boost or TBB is hugely important to us.

The more we can hide under more library layers, the better off we are. The one thing that ends up limiting the benefit of those libraries, though, is Amdahl's Law. For example, say that as part of some operation we need to transform the image into the frequency domain. There's a parallel implementation of FFT (Fast Fourier Transform) we can just call, and maybe we even have a library on top of that to decide whether or not it makes any sense to ship that all down to the GPU to do a GPU implementation of FFT before sending it back. But that's just one step in our algorithm. Maybe there's a parallel library for the next step, but getting from the FFT step to the step where we call the parallel library is going to require some messing around. It's with all that inter-step setup that Amdahl's Law just kills you. Even if you're spending only 10% of your time doing that stuff, that can be enough to keep you from scaling beyond 10 processors.

Still, the library approach is fabulous, and every parallel library implementation of common algorithms we can lay our hands on is greatly appreciated. Like many of the techniques we have available to us today, however, it starts to run out of steam at about eight to 16 processors. That doesn't mean it isn't worth doing. We're definitely headed down the library path ourselves because it's the only thing we can even imagine working if we're to scale to eight to 16 processors.

For the engineers on the Photoshop development team, the scaling limitations imposed by Amdahl's Law have become all too familiar over the past few years. Although the application's current parallelism scheme has scaled well over two- and four-processor systems, experiments with systems featuring eight or more processors indicate performance improvements that are far less encouraging. That's partly because as the number of cores increases, the image chunks being processed, called *tiles*, end up getting sliced into a greater number of smaller pieces, resulting in increased synchronization overhead. Another issue is that in between each

of the steps that process the image data in parallelizable chunks, there are sequential bookkeeping steps. Because of all this, Amdahl's Law quickly transforms into Amdahl's wall.

Photoshop's engineers tried to mitigate these effects by increasing the tile size, which in turn made each of the sub-pieces larger. This helped to reduce the synchronization overhead, but it presented the developers with yet another parallel-computing bugaboo: memory-bandwidth limitations. Compounding the problem was the fact that Photoshop cannot interrupt pixel-processing operations until an entire tile is complete. So to go too far down the path of increasing tile sizes would surely result in latency issues, as the application would become unable to respond to user input until it had finished processing an entire tile.

Although Williams remains confident his team can continue to improve Photoshop's scaling in the near future through the use of better tools, libraries, and incremental changes to the current approach to parallel processing, eventually those techniques will run out of steam. This means the time has come to start thinking about migrating the application to a different approach altogether that involves new parallel methods and the increased use of GPUs.

COLE: I think you already have some interesting ways of splitting things apart for image processing, but for your base application, have you considered other programming paradigms, such as MPI (message passing interface)?

WILLIAMS: No, we haven't because we've been occupied with moving from the four-core world to the eight- to 16-core world, and what we see is that Photoshop is just going to be stuck in that world for the next few years. Another reason we haven't looked all that seriously at changing to a message-passing-style interface is that it would require such a huge re-architecture effort and it's not at all clear what the win would be for us there.

COLE: The reason I ask is that Intel is obviously looking to enable as many cores in a box as possible, and you mentioned you had previously had problems with memory bandwidth. That's part of the reason why another division

of Intel has become interested in the NUMA (non-uniform memory architecture) way of putting things together. I certainly feel we're going to have applications that have both threadish parts and heavily parallel parts, and we're going to see the processors inside of workstations become more cluster-like in many ways. We may not necessarily go off-chip or out of the box, but we're going to break memory up somehow. And we're going to have to do lots of other things to give back some memory bandwidth just because that's going to have a huge impact for somebody like you.

WILLIAMS: This comes up in a number of different ways. We get asked a lot about how we're going to handle something like Larrabee [the engineering chip for Intel's MIC—Many Integrated Cores—architecture]. The answer is: basically nothing for now. The reason is that any of these future architectures that promise to solve some particular parallelism problem or some particular part of the performance problem are all kind of speculative at this point. Photoshop, on the other hand, is a mass-market application. So, unless we are fairly certain there are going to be millions of one of these platforms out there, we can't afford to bet our software's architectural direction on that. Right now, we don't see desktop architectures moving beyond the mild and cache-coherent form of NUMA we see today.

As a rule, we avoid writing large amounts of processor-specific or manufacturer-specific code, although we do some targeted performance tuning. For us, life will start to get interesting once we can use libraries such as OpenGL, OpenCL, and Adobe's Pixel Bender—or any higher-level libraries that take advantage of these libraries—to get access to all that GPU power in a more general way.

We've also been looking at the change Intel's Nehalem architecture presents in this area. On all previous multicore CPUs, a single thread could soak up essentially all of the memory bandwidth. Given that many of our low-level operations are memory-bandwidth limited, threading them would have only added overhead. Our experience with other multicore CPUs is that they become bandwidth limited with only a couple of threads running, so parallel speedups are limited by mem-

ory bandwidth rather than by Amdahl's Law or the nature of the algorithm. With Nehalem, each processor core is limited to a fraction of the chip's total memory bandwidth, so even a large memcopy can be sped up tremendously by being multithreaded.

COLE: I actually was just trying to make more of an architectural statement than anything. Rest assured, you're going to see just as many cores as we can possibly get in there, but at a certain point, what I refer to as "conservation of memory bandwidth" starts to become the big trade-off; that's when other architectural tricks are going to have to be used that will have some software impact. The question is, if you can't fire a gun and get everybody to change software overnight, at what point does it become economically interesting for a company such as Adobe to say, "OK, if I know I'm going to have to deal with a cluster in a box, I've got to slowly start moving my base so I'll be able to do that"?

WILLIAMS: I suspect we'll end up seeing the same progression we saw with SMP. That is, the hardware and operating-system support will be introduced such that these platforms will be able to run multiple programs, or multiple instances of programs, not yet modified to take advantage of the new architecture. This has already proved to be true with SMP and GPU use. There will be some small handful of applications that will absolutely depend on being able to leverage the brand-new capability right away—as was the case with video games and 3D rendering applications and their need to take advantage of GPUs as soon as possible. The bulk of applications, however, will not start to take significant advantage of new architectures until: (a) there's an installed base; (b) there's software support; and (c) there's a clear direction for where things are heading.

I assume the NUMA and MPI stuff is at the research-lab level at this juncture. And even though the MIC chip is on its way, it's still unclear to us what the programming API will be other than OpenGL and DirectX.

Now, just to throw a question back at you: what do you see the progression being in terms of how the programming API and operating-system support is going to be rolled out, since people like

me are going to need that if we're to take advantage of these kinds of architectural innovations?

COLE: As we develop specialty hardware, be it GPUs or network engines, the machine is becoming essentially a federation of processing elements designed to handle specific tasks. The graphics processor was highly tuned for doing its job of displaying pixels and performing certain mathematical functions that are important to the graphics guys and the gamers. Then other people came along and said, "Hey, I want to be able to do those same functions. Can I get at them?" That's when engineers like me in the operating-systems world start scratching our heads and saying, "Yeah, well, I suppose we could expose that."

But I wonder if that's what you really want. My experience has been that every time we've had a specialty engine like that and we've tried to feed it to the world, you may have been able to write an application library as you did with Photoshop that was able to call some graphics card, but that typically lived for only a couple of generations. That is, it proved to be cost-effective for only that one particular application. So I think the GPU will continue to get smarter and smarter, but it will retain its focus as a graphics engine just the same. That's really where it's going to be most cost-effective.

The rest of the box needs to be more general, maybe with a bunch of specialty execution engines around it that you're able to call up and easily exploit. Then the question becomes: how can the operating system make all those engines available?

Having been one of the early microkernel guys, I smiled when I learned about the early microkernel work you were doing. Many of us in the operating-system community have thought that would be the right way to go.

WILLIAMS: Elxsi was a message-based operating system. It was similar to the GNU Hurd of independent processes in that it talked via messages. One of the things that really hit us hard and is showing up today with GPUs in a different context—and, in fact, was the very first thing I thought of when I started looking at NUMA—is that message-based operations are horrendously expensive relative to everything else you

do. This is something the video apps have run into as well. They went down this path of doing a rendering graph to represent the stack of effects you had on your video frames, and then they would go down and start rendering and compositing those things. As soon as they got to anything they could do on the GPU, they would send it down there to get processed, and then they would work on it until they hit a node in the compositing stack graph that couldn't be handled on the GPU, at which point they would suck it back into the CPU.

What they found was that even with the fastest-bandwidth cards on the fastest-bandwidth interfaces, one trip was all you got. Anything beyond that meant you would have been better off just staying on the CPU in the first place. When you start moving data around, the bandwidth consumption associated with that can quickly overwhelm the cost of doing the computation. But the GPU vendors are continually improving this.

COLE: That's part of why I asked about MPI. I come back to that now only because it seems to be today's popular answer. I'm not saying it is the answer; it's just a way of trying to control what has to get shifted and when and how to partition the data so you can write code that will be able to exploit these execution units without having to move lots of data around.

This is why companies such as Intel are exploring interfaces such as RDMA (remote direct memory access), which is something you find inside of IB (InfiniBand). About a year ago, Intel purchased one of the original iWARP (Internet Wide Area RDMA Protocol) vendors, and the company is also heavily committed to the OpenFabrics Alliance's OFED (OpenFabrics Enterprise Distribution) implementations, so we're now exposing that same RDMA interface you find with InfiniBand in both Ethernet form and IB. I certainly think that kind of hardware is going to start showing up inside the base CPU and will become available to you as you try to move those objects around. So you're going to have computational resources and data movement resources, and the processing power will become the federation of all that stuff underneath.

That means the operating system has got to change. And I think you're right: what will happen then is that the

apps will become richer and will be able to exploit some of those pieces in the hardware base, provided that the operating system exposes it. That's also when you guys at Adobe will want to start exploiting that, since you'll have customers who already have machines with those capabilities built in.

WILLIAMS: When we started to look at NUMA, we ran into some issues with predictability. Ideally, on a big NUMA system, you would want your image to be spread evenly across all the nodes so that when you did an operation, you would be able to fire up each CPU to process its part of the image without having to move things around.

What actually happens, however, is that you've got a stack of images from which somebody makes a selection, and maybe selects some circle or an area from a central portion of the image containing pixels that live on three out of your 10 nodes. In order to distribute the computation the user then invokes on that selection, you now have to copy those things off to all 10 nodes. You quickly get to a point where your data is fragmented all over the place, and any particular selection or operation is unlikely to get nicely distributed across the NUMA nodes. Then you pay a huge cost to redistribute it all as part of starting up your operation. This, along with the more general issue of bandwidth management, is going to prove to be an even harder parallelism problem than the already well-documented problem people have with correctly locking their data structures.

COLE: Yes, we're in violent agreement on that score. Locking your data structures is truly only the beginning. The new tuning problem is going to be exactly the nightmare you just described. ■

Related articles on queue.acm.org

Real-World Concurrency
Bryan Cantrill, Jeff Bonwick
<http://queue.acm.org/detail.cfm?id=1454462>

A Conversation with John Hennessy and David Patterson
<http://queue.acm.org/detail.cfm?id=1189286>

Software and the Concurrency Revolution
Herb Sutter, James Larus
<http://queue.acm.org/detail.cfm?id=1095421>

More important principles to keep in mind when designing high-performance software.

BY CARY MILLSAP

Thinking Clearly About Performance, Part 2

IN PART 1 of this article (*Communications*, Sept. 2010, p. 55), I covered some of the fundamentals of performance. *Performance* is a relation between a *task* and the *time* it consumes. That relation is measurable either as *throughput* or *response time*. Because users feel variance in performance more than they feel

the mean, it's good to express performance goals in a *percentile* format, such as "Task *T* must have response time of *R* seconds or less in *P* proportion or more of executions." To diagnose a performance problem, you need to state your goals objectively, in terms of either throughput or response time, or both.

A *sequence diagram* is a helpful graphical tool for understanding how a task's execution consumes your time. A *profile* is a table that shows details about response time for a single task execution. With a profile, you can learn exactly how much improvement to expect for a proposed investment, but only if you understand the pitfalls of making incorrect assumptions about *skew*.

Minimizing Risk. As mentioned in Part 1, the risk that repairing the per-

formance of one task can damage the performance of another reminds me of something that happened to me once in Denmark. It's a quick story:

Scene: The kitchen table in Måløv, Denmark; *the* oak table, in fact, of Oak Table Network fame, a network of Oracle practitioners who believe in using scientific methods to improve the development and administration of Oracle-based systems.⁸ Roughly 10 people sit around the table, working on their laptops and conducting various conversations.

Cary: Guys, I'm burning up. Would you mind if I opened the window for a little bit to let some cold air in?

Carel-Jan: Why don't you just take off your heavy sweater?

The End.

There is a general principle at work here that humans who optimize know: when everyone is happy except for you, make sure your local stuff is in order before you go messing around with the global stuff that affects everyone else, too.

This principle is why I flinch whenever someone proposes to change a system's Oracle SQL*Net packet size when the problem is really a couple of poorly written Java programs that make unnecessarily many database calls (and, hence, unnecessarily many network I/O calls as well). If everybody is getting along fine except for the user of one or two programs, then the safest solution to the problem is a change whose scope is localized to just those one or two programs.

Efficiency. Even if everyone on the entire system is suffering, you should still focus first on the program that the business needs fixed. The way to begin is to ensure the program is working as efficiently as it can. *Efficiency* is the inverse of how much of a task execution's total service time can be eliminated without adding capacity and without sacrificing required business function.

In other words, efficiency is an inverse measure of waste. Here are some examples of waste that commonly occur in the database application world:

- ▶ A middle-tier program creates a distinct SQL statement for every row it inserts into the database. It executes 10,000 database `prepare` calls (and thus 10,000 network I/O calls) when it could have accomplished the job with one `prepare` call (and thus 9,999 fewer network I/O calls).

- ▶ A middle-tier program makes 100 database `fetch` calls (and thus 100 network I/O calls) to fetch 994 rows. It could have fetched 994 rows in 10 `fetch` calls (and thus 90 fewer network I/O calls).

- ▶ A SQL statement (my choice of article adjective here is a dead giveaway that I was introduced to SQL within the Oracle community) touches the database buffer cache 7,428,322 times to return a 698-row result set. An extra filter predicate could have returned the seven rows that the end user really wanted to see, with only 52 touches upon the database buffer cache.

Certainly, if a system has some global problem that creates inefficiency for broad groups of tasks across the sys-

tem (for example, ill-conceived index, badly set parameter, poorly configured hardware), then you should fix it. Do not tune a system to accommodate programs that are inefficient, however. (Admittedly, sometimes you need a tourniquet to keep from bleeding to death, but do not use a stopgap measure as a permanent solution. Address the inefficiency.) There is a great deal more leverage in curing the program inefficiencies themselves. Even if the programs are commercial off-the-shelf applications, it will benefit you more in the long run to work with your software vendor to make your programs efficient than it will to try to optimize your system to run with an inherently inefficient workload.

Improvements that make your program more efficient can produce tremendous benefits for everyone on the system. It is easy to see how top-line reduction of waste helps the response time of the task being repaired. What many people do not understand as well is that making one program more efficient creates a side effect of performance improvement for other programs on the system that have no apparent relation to the program being repaired. It happens because of the influence of *load* upon the system.

Load is competition for a resource induced by concurrent task executions. It is the reason the performance testing done by software developers does not catch all the performance problems that show up later in production.

One measure of load is *utilization*, which is resource usage divided by resource capacity for a specified time interval. As utilization for a resource goes up, so does the response time a

user will experience when requesting service from that resource. Anyone who has ridden in an automobile in a big city during rush hour has experienced this phenomenon. When the traffic is heavily congested, you have to wait longer at the tollbooth.

The software you use does not actually “go slower” as your car does when you are going 30mph in heavy traffic instead of 60mph on the open road. Computer software always goes the same speed no matter what (a constant number of instructions per clock cycle), but certainly response time degrades as resources on your system get busier.

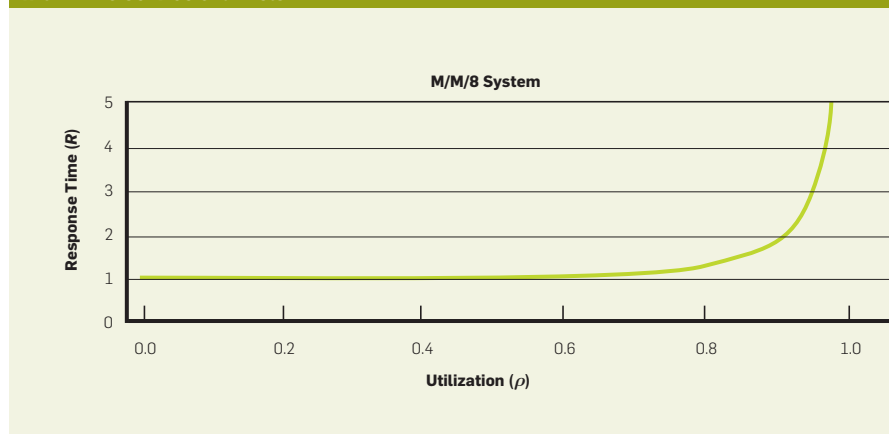
There are two reasons that systems get slower as load increases: *queuing delay* and *coherency delay*.

Queuing delay. The mathematical relationship between load and response time is well known. One mathematical model, called M/M/m, relates response time to load in systems that meet one particularly useful set of specific requirements.⁷ One of the assumptions of M/M/m is the system you are modeling has “theoretically perfect scalability.” This is akin to having a physical system with “no friction,” an assumption that so many problems in introductory physics courses invoke.

Regardless of some overreaching assumptions such as the one about perfect scalability, M/M/m has a lot to teach us about performance. Figure 1 shows the relationship between response time and load using M/M/m.

In the figure, you can see mathematically what you feel when you use a system under different load conditions. At low load, your response time is essentially the same as your response time at

Figure 1. This curve relates response time as a function of utilization for an M/M/m system with $m = 8$ service channels.



no load. As load ramps up, you sense a slight, gradual degradation in response time. That gradual degradation does not really do much harm, but as load continues to ramp up, response time begins to degrade in a manner that's neither slight nor gradual. Rather, the degradation becomes quite unpleasant and, in fact, hyperbolic.

Response time (R), in the perfect scalability $M/M/m$ model, consists of two components: *service time* (S) and *queuing delay* (Q), or $R = S + Q$. Service time is the duration that a task spends consuming a given resource, measured in time per task execution, as in *seconds per click*. Queuing delay is the time that a task spends enqueued at a given resource, awaiting its opportunity to consume that resource. Queuing delay is also measured in time per task execution (for example, seconds per click).

In other words, when you order lunch at Taco Tico, your response time (R) for getting your order is the queuing delay time (Q) that you spend in front of the counter waiting for someone to take your order, plus the service time (S) you spend waiting for your order to hit your hands once you begin talking to the order clerk. Queuing delay is the difference between your response time for a given task and the response time for that same task on an otherwise unloaded system (don't forget our *perfect scalability* assumption).

The Knee

When it comes to performance, you want two things from a system:

- ▶ The best response time you can get: you do not want to have to wait too long for tasks to get done.
- ▶ The best throughput you can get: you want to be able to cram as much load as you possibly can onto the system so that as many people as possible can run their tasks at the same time.

Unfortunately, these two goals are contradictory. Optimizing to the first goal requires you to minimize the load on your system; optimizing to the second goal requires you to maximize it. You can not do both simultaneously. Somewhere in between—at some load level (that is, at some utilization value)—is the optimal load for the system.

The utilization value at which this optimal balance occurs is called the *knee*. This is the point at which

Table 1. $M/M/m$ knee values for common values of m .

Service channel count	Knee utilization
1	50%
2	57%
4	66%
8	74%
16	81%
32	86%
64	89%
128	92%

throughput is maximized with minimal negative impact to response times. (I am engaged in an ongoing debate about whether it is appropriate to use the term *knee* in this context. For the time being, I shall continue to use it; see the sidebar for details.) Mathematically, the knee is the utilization value at which response time divided by utilization is at its minimum. One nice property of the knee is it occurs at the utilization value where a line through the origin is tangent to the response-time curve. On a carefully produced $M/M/m$ graph, you can locate the knee quite nicely with just a straight-edge, as shown in Figure 2.

Another nice property of the $M/M/m$ knee is that you need to know the value of only one parameter to compute it. That parameter is the number of parallel, homogeneous, independent *service channels*. A service channel is a resource that shares a single queue with other identical resources, such as a booth in a toll plaza or a CPU in

an SMP (symmetric multiprocessing) computer.

The italicized lowercase m in the term $M/M/m$ is the number of service channels in the system being modeled. The $M/M/m$ knee value for an arbitrary system is difficult to calculate, but I have done it in Table 1, which shows the knee values for some common service channel counts. (By this point, you may be wondering what the other two Ms stand for in the $M/M/m$ queuing model name. They relate to assumptions about the randomness of the timing of your incoming requests and the randomness of your service times. See http://en.wikipedia.org/wiki/Kendall%27s_notation for more information, or *Optimizing Oracle Performance*⁷ for even more.)

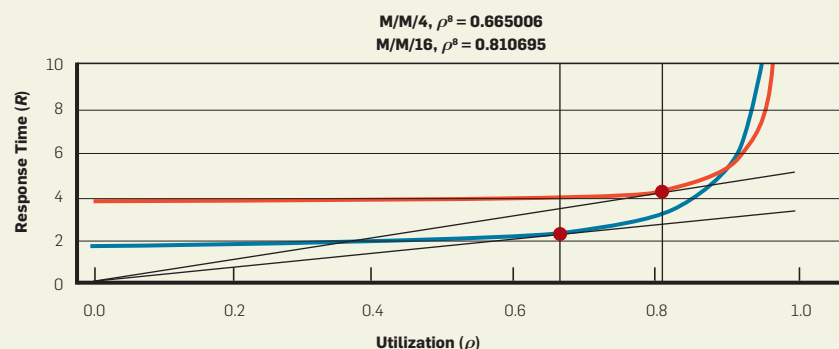
Why is the knee value so important? For systems with randomly timed service requests, allowing sustained resource loads in excess of the knee value results in response times and throughputs that will fluctuate severely with microscopic changes in load. Hence, on systems with random request arrivals, it is vital to manage load so that it will not exceed the knee value.

Relevance of the Knee

How important can this knee concept be, really? After all, as I've told you, the $M/M/m$ model assumes this ridiculously utopian idea that the system you are thinking about scales perfectly. I know what you are thinking: it doesn't.

What $M/M/m$ does give us is the knowledge that even if your system did scale perfectly, you would still be stricken with massive performance problems once your average load exceeded the knee values in Table 1. Your system

Figure 2. The knee occurs at the utilization at which a line through the origin is tangent to the response time curve.



Open Debate About Knees

In this article, I write about knees in performance curves, their relevance, and their application. Whether it is even worthwhile to try to define the concept of *knee*, however, has been the subject of debate going back at least 20 years.

There is significant historical basis to the idea that the thing I have described as a knee in fact is not really meaningful. In 1988, Stephen Samson argued that, at least for M/M/1 queuing systems, no “knee” appears in the performance curve. “The choice of a guideline number is not easy, but the rule-of-thumb makers go right on. In most cases there is not a knee, no matter how much we wish to find one,” wrote Samson.³

The whole problem reminds me, as I wrote in 1999,² of the parable of the frog and the boiling water. The story says that if you drop a frog into a pan of boiling water, he will escape. But if you put a frog into a pan of cool water and slowly heat it, then the frog will sit patiently in place until he is boiled to death.

With utilization, just as with boiling water, there is clearly a “death zone,” a range of values in which you can’t afford to run a system with random arrivals. But where is the border of the death zone? If you are trying to implement a procedural approach to managing utilization, you need to know.

My friend Neil Gunther (see http://en.wikipedia.org/wiki/Neil_J._Gunther for more information about Neil) has debated with me privately that, first, the term *knee* is completely the wrong word to use here, in the absence of a functional discontinuity. Second, he asserts that the boundary value of .5 for an M/M/1 system is wastefully low, that you ought to be able to run such a system successfully at a much higher utilization value than that. Finally, he argues that any such special utilization value should be defined expressly as the utilization value beyond which your average response time exceeds your tolerance for average response time (Figure A). Thus, Gunther argues that any useful not-to-exceed utilization value is derivable only from inquiries about human preferences, not from mathematics. (See http://www.cmg.org/measureit/issues/mit62/m_62_15.html for more information about his argument.)

The problem I see with this argument is illustrated in Figure B. Imagine that your tolerance for average response time is T , which creates a maximum tolerated utilization value of ρT . Notice that even a tiny fluctuation in average utilization near ρT will result in a huge fluctuation in average response time. I believe that your customers feel the variance, not the mean. Perhaps they say they will accept average response times up to T , but humans will not be tolerant

of performance on a system when a 1% change in average utilization over a one-minute period results in, say, a tenfold increase in average response time over that period.

I do understand the perspective that the knee values I’ve listed in this article are below the utilization values that many people feel safe in exceeding, especially for lower-order systems such as M/M/1. It is important, however, to avoid running resources at average utilization values where small fluctuations in utilization yield too-large fluctuations in response time.

Having said that, I do not yet have a good definition for a *too-large fluctuation*. Perhaps, like response-time tolerances, different people have different tolerances for fluctuation. But perhaps there is a fluctuation tolerance factor that applies with reasonable universality across all users. The Apdex

Application Performance Index standard, for example, assumes the response time F at which users become “frustrated” is universally four times the response time T at which their attitude shifts from being “satisfied” to merely “tolerating.”¹

The knee, regardless of how you define it or what we end up calling it, is an important parameter to the capacity-planning procedure that I described earlier in the main text of this article, and I believe it is an important parameter to the daily process of computer system workload management.

I will keep studying.

References

1. Apdex; <http://www.apdex.org>.
2. Millsap, C. Performance management: myths and facts (1999); <http://method-r.com>.
3. Samson, S. MVS performance legends. In *Proceedings of Computer Measurement Group Conference* (1988), 148–159.

Figure A. Gunther’s maximum allowable utilization value ρT is defined as the utilization producing the average response time T .

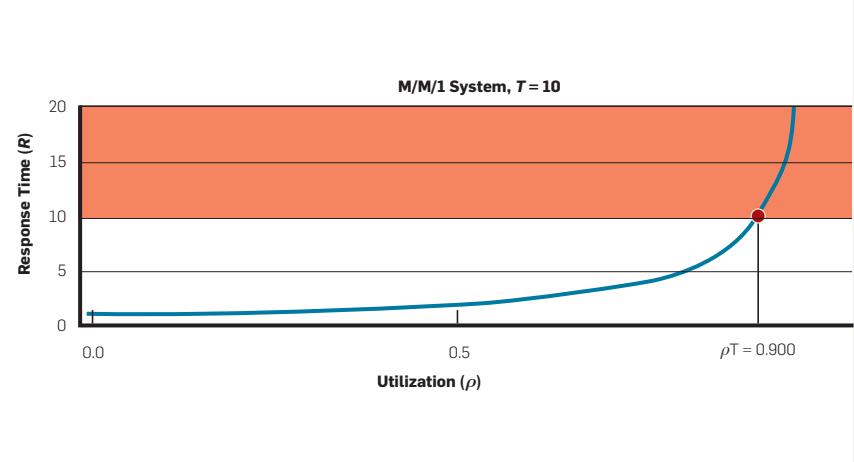
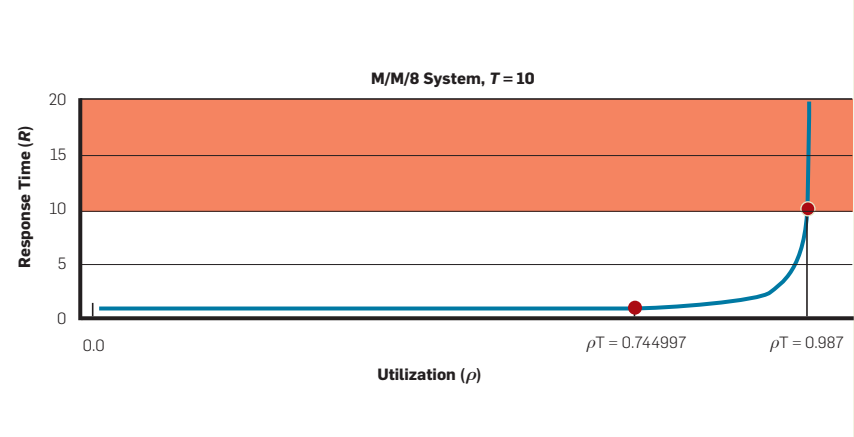


Figure B. Near ρT value, small fluctuations in average utilization result in large response-time fluctuations.



isn't as good as the theoretical systems that $M/M/m$ models. Therefore, the utilization values at which *your* system's knees occur will be more constraining than the values in Table 1. (I use the plural of *values* and *knees*, because you can model your CPUs with one model, your disks with another, your I/O controllers with another, and so on.)

To recap:

- ▶ Each of the resources in your system has a knee.
- ▶ That knee for each of your resources is less than or equal to the knee value you can look up in Table 1. The more imperfectly your system scales, the smaller (worse) your knee value will be.
- ▶ On a system with random request arrivals, if you allow your sustained utilization for any resource on your system to exceed your knee value for that resource, then you will have performance problems.
- ▶ Therefore, it is vital that you manage your load so that your resource utilizations will not exceed your knee values.

Capacity Planning


Understanding the knee can collapse a lot of complexity out of your capacity planning. It works like this:

- ▶ Your goal capacity for a given resource is the amount at which you can comfortably run your tasks at peak times without driving utilizations beyond your knees.
- ▶ If you keep your utilizations less than your knees, your system behaves roughly linearly—no big hyperbolic surprises.
- ▶ If you are letting your system run any of its resources beyond their knee utilizations, however, then you have performance problems (whether you are aware of them or not).
- ▶ If you have performance problems, then you don't need to be spending your time with mathematical models; you need to be spending your time fixing those problems by rescheduling load, eliminating load, or increasing capacity.


That's how capacity planning fits into the performance management process.

Random Arrivals

You might have noticed that I used the term *random arrivals* several times. Why is that important?



The reason the knee value is so important on a system with random arrivals is that these tend to cluster and cause temporary spikes in utilization.



Some systems have something that you probably do not have right now: completely deterministic job scheduling. Some systems—though rare these days—are configured to allow service requests to enter the system in absolute robotic fashion, say, at a pace of one task per second. And by this, I don't mean at an average rate of one task per second (for example, two tasks in one second and zero tasks in the next); I mean one task per second, as a robot might feed car parts into a bin on an assembly line.

If arrivals into your system behave completely deterministically—meaning that you know *exactly* when the next service request is coming—then you can run resource utilizations beyond their knee utilizations without necessarily creating a performance problem. On a system with deterministic arrivals, your goal is to run resource utilizations up to 100% without cramming so much workload into the system that requests begin to queue.

The reason the knee value is so important on a system with *random* arrivals is that these tend to cluster and cause temporary spikes in utilization. These spikes need enough spare capacity to consume so that users don't have to endure noticeable queuing delays (which cause noticeable fluctuations in response times) every time a spike occurs.

Temporary spikes in utilization beyond your knee value for a given resource are OK as long as they don't exceed a few seconds in duration. But how many seconds are too many? I believe (but have not yet tried to prove) that you should at least ensure that your spike durations do not exceed eight seconds. (You will recognize this number if you've heard of the "eight-second rule."²) The answer is certainly that if you're unable to meet your percentile-based response time promises or your throughput promises to your users, then your spikes are too long.

Coherency Delay

Your system does not have theoretically perfect scalability. Even if I have never studied your system specifically, it is a pretty good bet that no matter what computer application system you are thinking of right now, it does not meet the $M/M/m$ "theoretically perfect scalability" assumption. *Coherency delay* is

the factor that you can use to model the imperfection.⁴ It is the duration that a task spends communicating and coordinating access to a shared resource. Like response time, service time, and queuing delay, coherency delay is measured in time per task execution, as in seconds per click.

I will not describe a mathematical model for predicting coherency delay, but the good news is that if you profile your software task executions, you'll see it when it occurs. In Oracle, timed events such as the following are examples of coherency delay:


- ▶ enqueue
- ▶ buffer busy waits
- ▶ latch free

You can not model such coherency delays with $M/M/m$. That is because $M/M/m$ assumes all m of your service channels are parallel, homogeneous, and independent. That means the model assumes that after you wait politely in a FIFO queue for long enough that all the requests that enqueued ahead of you have exited the queue for service, it will be your turn to be serviced. Coherency delays don't work like that, however.


Imagine an HTML data-entry form in which one button labeled "Update" executes a SQL update statement, and another button labeled "Save" executes a SQL commit statement. An application built like this would almost guarantee abysmal performance. That is because the design makes it possible—quite likely, actually—for a user to click Update, look at his calendar, realize "uh-oh, I'm late for lunch," and then go to lunch for two hours before clicking Save later that afternoon.

The impact to other tasks on this system that wanted to update the same row would be devastating. Each task would necessarily wait for a lock on the row (or, on some systems, worse: a lock on the row's page) until the locking user decided to go ahead and click Save—or until a database administrator killed the user's session, which of course would have unsavory side effects to the person who thought he had updated a row.

In this case, the amount of time a task would wait on the lock to be released has nothing to do with how busy the system is. It would be dependent upon random factors that exist outside



All this talk of queuing delays and coherency delays leads to a very difficult question: How can you possibly test a new application enough to be confident that you are not going to wreck your production implementation with performance problems?



of the system's various resource utilizations. That is why you can not model this kind of thing in $M/M/m$, and it is why you can never assume that a performance test executed in a unit-testing type of environment is sufficient for a making a go/no-go decision about insertion of new code into a production system.

Performance Testing

All this talk of queuing delays and coherency delays leads to a very difficult question: How can you possibly test a new application enough to be confident that you are not going to wreck your production implementation with performance problems?

You can model. And you can test.¹ Nothing you do will be perfect, however. It is extremely difficult to create models and tests in which you'll foresee all your production problems in advance of actually encountering those problems in production.

Some people allow the apparent futility of this observation to justify not testing at all. Do not get trapped in that mentality. The following points are certain:

- ▶ You will catch a lot more problems if you try to catch them prior to production than if you do not even try.
- ▶ You will never catch all your problems in preproduction testing. That is why you need a reliable and efficient method for solving the problems that leak through your preproduction testing processes.

Somewhere in the middle between "no testing" and "complete production emulation" is the right amount of testing. The right amount of testing for aircraft manufacturers is probably more than the right amount of testing for companies that sell baseball caps. But don't skip performance testing altogether. At the very least, your performance-test plan will make you a more competent diagnostician (and clearer thinker) when the time comes to fix the performance problems that will inevitably occur during production operation.

Measuring. People feel throughput and response time. Throughput is usually easy to measure, response time is much more difficult. (Remember, throughput and response time are *not* reciprocals.) It may not be difficult to time an end-user action with a stopwatch, but it might

be very difficult to get what you really need, which is the ability to drill down into the details of why a given response time is as large as it is.

Unfortunately, people tend to measure what is easy to measure, which is not necessarily what they *should be* measuring. It's a bug. Measures that aren't what you need, but that are easy enough to obtain and seem related to what you need are called *surrogate measures*. Examples include subroutine call counts and samples of subroutine call execution durations.

I'm ashamed that I do not have greater command over my native language than to say it this way, but here is a catchy, modern way to express what I think about surrogate measures: *surrogate measures suck*.

Here, unfortunately, *suck* doesn't mean *never work*. It would actually be better if surrogate measures never worked. Then nobody would use them. The problem is that surrogate measures work *sometimes*. This inspires people's confidence that the measures they are using should work all the time, and then they don't. Surrogate measures have two big problems.

- ▶ They can tell you your system's OK when it is not. That's what statisticians call *type I error*, the false positive.

- ▶ They can tell you that something is a problem when it is not. That's a *type II error*, the false negative. I have seen each type of error waste years of people's time.

When the time comes to assess the specifics of a real system, your success is at the mercy of how good the measurements are that your system allows you to obtain. I have been fortunate to work in the Oracle market segment, where the software vendor at the center of our universe participates actively in making it possible to measure systems the right way. Getting application software developers to use the tools that Oracle offers is another story, but at least the capabilities are there in the product.

Performance is a Feature

Performance is a software application feature, just like recognizing that it's convenient for a string of the form "Case 1234" to automatically hyperlink over to case 1234 in your bug-tracking system. (FogBugz, which is software that I enjoy using, does this.) Performance, like any

other feature, does not just happen; it has to be designed and built. To do performance well, you have to think about it, study it, write extra code for it, test it, and support it.

Like many other features, however, you can not know exactly how performance is going to work out while you're still writing, studying, designing, and creating the application. For many applications (arguably, for the vast majority), performance is completely unknown until the production phase of the software development life cycle. What this leaves you with is this: since you can't know how your application is going to perform in production, you need to write your application so that it's easy to *fix* performance in production.

As David Garvin has taught us, it's much easier to manage something that's easy to measure.³ Writing an application that is easy to fix in production begins with an application that's easy to measure in production.

Usually, when I mention the concept of production performance measurement, people drift into a state of worry about the measurement-intrusion effect of performance instrumentation. They immediately enter a mode of data-collection compromise, leaving only surrogate measures on the table. Won't software with an extra code path to measure timings be slower than the same software without that extra code path?

I like an answer that I once heard Tom Kyte give in response to this question.⁶ He estimated that the measurement-intrusion effect of Oracle's extensive performance instrumentation is -10% or less (where *or less* means *or better*, as in -20%, -30%, and so on). He went on to explain to a now-vexed questioner that the product is at least 10% faster now because of the knowledge that Oracle Corporation has gained from its performance instrumentation code, more than making up for any "overhead" the instrumentation might have caused.

I think that vendors tend to spend too much time worrying about how to make their measurement code path efficient without figuring out first how to make it effective. It lands squarely upon the idea that Knuth wrote about in 1974 when he said that "premature optimization is the root of all evil."⁵ The software designer who integrates performance

measurement into a product is much more likely to create a fast application and—more importantly—one that will become faster over time.

Acknowledgments

Thank you, Baron Schwartz for the email conversation in which you thought I was helping you, but in actual fact, you were helping me come to grips with the need for introducing coherency delay more prominently into my thinking. Thank you, Jeff Holt, Ron Crisco, Ken Ferlita, and Harold Palacio for the daily work that keeps the company going and for the lunchtime conversations that keep my imagination going. Thank you, Tom Kyte for your continued inspiration and support. Thank you, Mark Farnham for your helpful suggestions. And thank you, Neil Gunther for your patience and generosity in our ongoing discussions about knees. □

Related articles on queue.acm.org

You're Doing It Wrong

Poul-Henning Kamp

<http://queue.acm.org/detail.cfm?id=1814327>

Performance Anti-Patterns

Bart Smaalders

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

Hidden in Plain Sight

Bryan Cantrill

<http://queue.acm.org/detail.cfm?id=1117401>

References

1. CMG (Computer Measurement Group, a network of professionals who study these problems very, very seriously); <http://www.cmg.org>.
2. Eight-second rule; http://en.wikipedia.org/wiki/Network_performance#8-second_rule.
3. Garvin, D. Building a learning organization. *Harvard Business Review* (July 1993).
4. Gunther, N. Universal Law of Computational Scalability (1993); http://en.wikipedia.org/wiki/Neil_J._Gunther#Universal_Law_of_Computational_Scalability.
5. Knuth, D. Structured programming with Go To statements. *ACM Computing Surveys* 6, 4 (1974), 268.
6. Kyte, T. A couple of links and an advert...; <http://tkyte.blogspot.com/2009/02/couple-of-links-and-advert.html>.
7. Millsap, C. and Holt, J. *Optimizing Oracle Performance*. O'Reilly, Sebastopol, CA, 2003.
8. Oak Table Network; <http://www.oaktable.net>.

Cary Millsap is the founder and president of Method R Corporation (<http://method-r.com>), a company devoted to software performance. He is the author (with Jeff Holt) of *Optimizing Oracle Performance* (O'Reilly) and a co-author of *Oracle Insights: Tales of the Oak Table* (Apress). He is the former vice president of Oracle Corporation's System Performance Group and a co-founder of his former company Hotsos. He is also an Oracle ACE Director and a founding partner of the Oak Table Network, an informal association of well-known "Oracle scientists." Millsap blogs at <http://carymillsap.blogspot.com>, and tweets at <http://twitter.com/CaryMillsap>.

© 2010 ACM 0001-0782/10/1000 \$10.00

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

Component models can help diagnose architectural problems in both new and existing systems.

BY KEVIN MONTAGNE

Tackling Architectural Complexity with Modeling

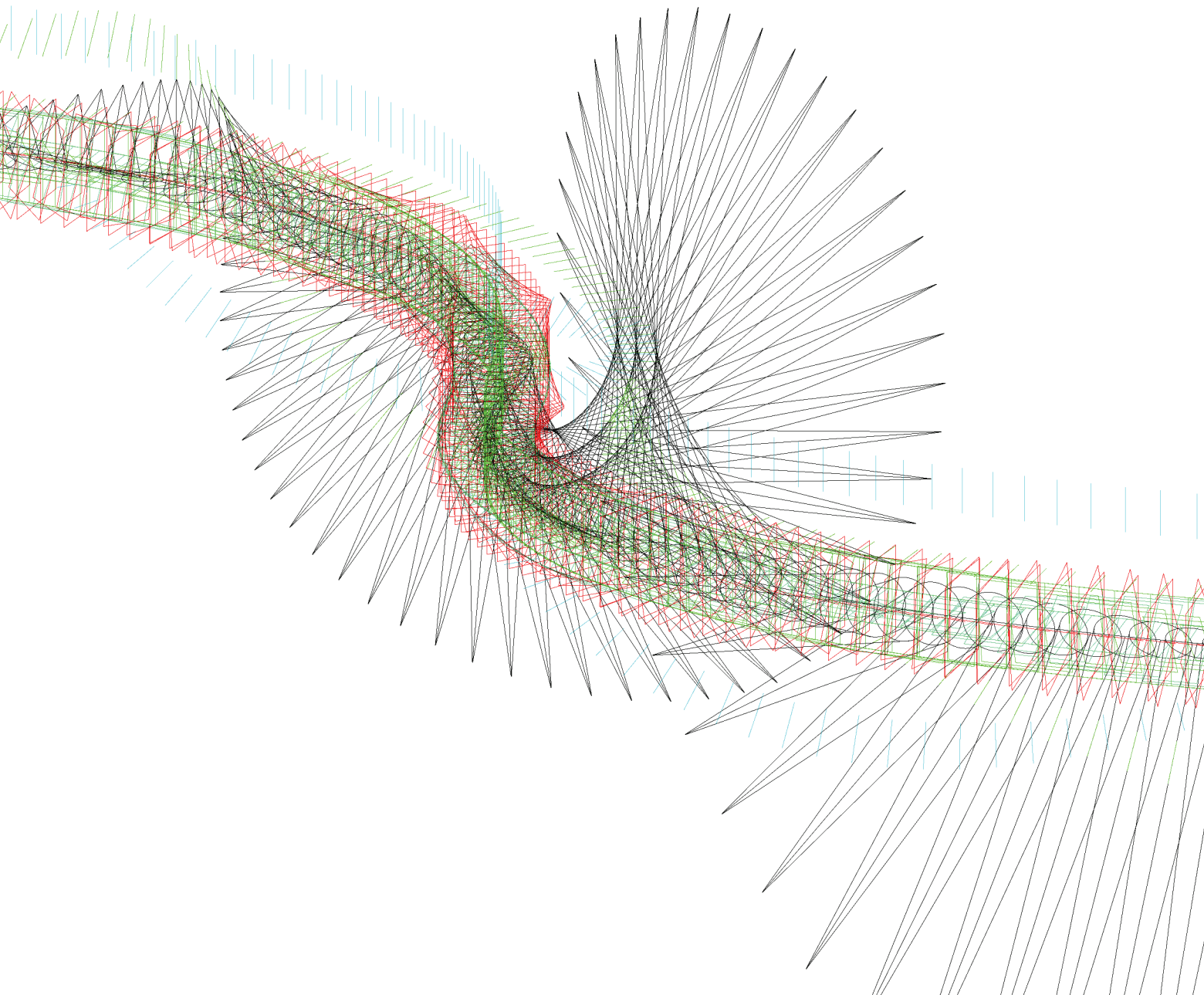
THE EVER-INCREASING MIGHT of modern computers has made it possible to solve problems once considered too difficult to tackle. Far too often, however, the systems for these functionally complex problem spaces have overly complicated architectures. Here, I use the term *architecture* to refer to the overall macro design of a system rather than the details of how the individual parts are implemented. The system architecture is what is behind the scenes of usable functionality, including internal and external communication mechanisms, component boundaries and coupling, and how the system will make use of any underlying infrastructure (databases, networks, among others). The architecture is the “right” answer to the question: how does this system work?

The question is: What can be done about the challenge to understand—or better yet, prevent—the complexity in systems? Many development methodologies (for example, Booch¹) consider nonfunctional aspects, but too often they stop at the diagram stage. The mantra of “we can address [performance, scalability, and so on] later” can be crippling. Individual components (applications) in a system can typically be iterated, but it is often far more difficult to iterate the architecture because of all the interface and infrastructure implications.

In this article, I describe an approach to architectural design when embarking on creating a new system. But what if the system already exists in some form? Much of my architecture work has been with existing systems—many times as an “outsider” who is invited (or sent) in to evaluate and improve the state of the system. These assignments can be quite challenging when dealing with complex systems.

One advantage to modeling an existing system is that the general behavior is already in place so you are not starting from a blank state. You also probably do not have to contend with the creation of the functional parts of the system. This comes at a price, however. There is a fair chance the system’s architecture is complex and not well understood. Additionally, many solutions may not be practical because of the high cost of a system overhaul.

With any type of system the goal is to understand the architecture and system behavior as much as possible. When a large system has been around for years this may seem like a monumental effort. Many techniques are available for discovering how a system works and ways it can be improved. You can ask members of the development and maintenance teams. Diagnostic tools (for example, DTrace) can help make quick work of finding performance or scalability offenders in a system. You can comb through mountains of log files to see what the



developers thought worthy of note. In this article I focus on how modeling the various system components can be used to gain a greater understanding and provide a foundation for evaluating possible changes.

This type of modeling is not just a whiteboard or paper exercise. It is the creation of drivers and components to emulate various aspects of the system. The drivers are used to invoke the various parts of the system to mimic its normal behavior. The idea is to exercise the architecture without the “burden” of ensuring functional correctness. At times these drivers may be scripts written with established tools (for example, WinRunner, JMeter), but I have often found more value in devel-

oping programs specific to the component to be driven. These have allowed me to get the information I needed to make quality decisions. It is important to understand that the model components and the associated drivers are not just simple test programs but are to be used as the basis for exploration and discovery.

The process of modeling the system should start by examining one or two components at a time. The initial targets should be components suspected of negatively impacting the whole system. You can then build independent drivers to interact with the component(s). If a problem component is confirmed, then experimentation with possible changes can begin.

These could span from code changes to infrastructure changes to hardware changes. With the right drivers and component modeling it may become practical to address redesigning some of the components.

Sometimes the functionality contained within a component is so intertwined with the architecture that it's necessary to create a lightweight replica. It is not unusual for some functional aspects of the system to mask the behavior of the underlying technology or infrastructure in responding to the requesting applications. In these cases having a lightweight model can allow the architectural interactions to be explored and better understood. If you discover architectural solutions,

then you can move on to the various functional implementations.

Modeling an Early Windows System

My first experience with modeling involved creating both drivers and mock-up components to explore a new technology. I was working for a large financial institution in the late 1980s when Microsoft Windows 2.1 was released. A group of developers had created a fairly sophisticated suite of Windows applications for telephone-based customer service representatives. The applications provided the ability to retrieve customer information, balances, and so on from several mainframe-based systems (using the now-ancient concept of “screen scraping” the data intended to be displayed on an IBM 3270 dumb terminal) and then present the data in an aggregated view. It also allowed the customer service representatives to place trades on behalf of the customer.

The suite started as a proof of concept, but the prototype demos went so well it was rushed to production. When I joined the team it was already deployed to about 150 representatives. As the programs began to be used all day, problems began to occur frequently. These were manifested in a variety of forms: memory leaks, access violations, spurious error messages, and machine lock-ups (aka freezes).

Our small team was busy adding functionality to meet the rapidly growing wish list and at the same time addressing the stability issues. We navigated through the source, attacking memory leaks and access violations. We struggled to track down the growing list of newly observed error mes-

sage. The most challenging task was “freeze patrol,” where we expended a great deal of time hunting down those machine lock-ups. The problem was that we did not have a really good understanding of how Windows worked behind the scenes.

Those familiar with programming with the early Windows SDKs will remember that documentation (not to mention stability) was not well developed. The API functions were pretty low level and it seemed like there were a bazillion of them. (If it were not for Charles Petzold’s *Programming Windows*,² I am not sure how many Windows applications developed outside of Microsoft would have been completed in the 1980s.) The code base for the applications was already pretty large—at least for applications in those days—and each was implemented slightly differently (they were prototypes, after all). Microsoft offered a few sample programs but nothing close to the complexity of these applications. Therefore, we decided to build components (applications) that imitated the Windows behavior we were trying to achieve.

These components were mostly void of functionality but started off with the basic structure and interface mechanisms similar to the actual applications. The drivers sent fine-grained Windows messages to the model components to simulate key presses and other externally originated actions. They also sent DDE (Dynamic Data Exchange, a primitive way to communicate data between Windows programs) messages throughout the suite of applications. As we matured the model, we began to merge in more of the API

calls (for example, user interface controls) used in the actual programs.

Many of the freezes were tracked down to undocumented idiosyncrasies of Windows Graphics Device Interface (GDI) calls. Examples included sensitivity to the ordering of some API calls, incompatibility between certain calls being made in the same context, and resource exhaustion possibilities. In the early versions of Windows the GDI libraries were tightly interwoven with the kernel libraries. As Windows matured, similar quandaries became error messages, exceptions, or just the offending application locking up.

The result of the modeling was that we gained enough information about this novel Windows technology to morph the programs to where stability was a reasonable expectation. Within 15 months the system was deployed to more than 4,500 workstations and survived well into Windows NT’s life.

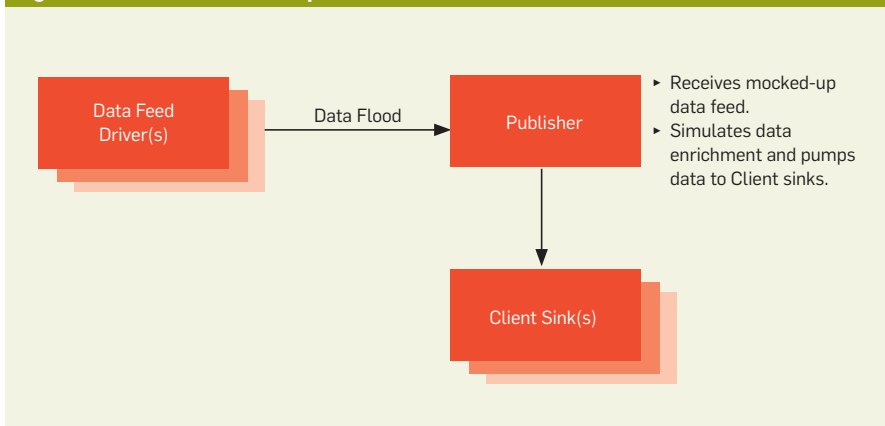
Modeling a “Slave” System

Not all of my modeling experiences resulted in such a positive outcome. Several exposed fundamental flaws in the architectural design, and with a few the only option was to abandon the system and start over. These messages were not typically well received by project management.

One of the more notable examples occurred in a system intended to be a “slave” receiving updates from several existing systems and applying them to a new database. The database would be used by other new systems to form the basis to replace the older systems. The systems would be built using a new technology platform. The technologies were so different and the functional breadth so wide that the development team had grown to more than 60 people for the slave system alone.

I joined the project after the basic architecture and much of the functionality had already been designed and developed, but it was still months away from production. My team’s assignment was to help get the most out of the infrastructure and optimize how the applications interacted with each other. After just a few weeks we suspected that some bad initial assumptions had impacted the architectural design. (I do not mean to disparage any teams in my examples, but

Figure 1. Publisher model component with drivers and sinks.




merely to point out the potential problem with too much focus on functionality at the expense of a solid architectural foundation.) Because it looked like performance and scalability were going to be major concerns, the architecture team began working on some model components and drivers to investigate the design.


We did some research around the incoming rate of messages and the mix in the types of transactions. We also sampled timings from the functional “processors” that had already been built. Then using the same messaging infrastructure as the existing dispatcher, we built a component that would simulate the incoming message dispatcher. Some of the messaging technology was new to the company. At one end of the dispatcher we had drivers to simulate inbound messages. On the other end we simulated the performance of the functional processors (FPs) using pseudo-random numbers clustered around the sampled timings. By design, there was nothing in the modeled components or drivers related to the functional processing in the system.

Once the model was fully functional, we were able to play with various parameters related to the incoming message rates and simulated FP timings. We then began to weight the FP times according to processing cost variations in the mix of incoming message types. Prior to this modeling effort, the design had (wrongly) assumed that the most important performance aspect was the latency of the individual transactions. Several seconds of latency was acceptable to all concerned. After all, it would be quite some time before this slave would become the system of record and drive transactions the other way.

The modeling results were not encouraging. The latency was going to be a challenge, but the overall throughput requirements were going to bury the system. We started exploring ways to address the performance problems. The system was already targeted for the fastest hardware available for the chosen platform, so that option was out. We delayed looking into improving the performance of the individual functional processors; that was deemed to be more costly because of the num-



What can be done about the challenge to understand—or better yet, prevent—the complexity in systems?



ber that had already been written. We thought our chances of quick success could increase with a focus on the common infrastructure pieces.

We worked on new dispatching algorithms but that did not result in enough improvement. We looked at optimizing the messaging infrastructure but still fell short. We then began to benchmark some other message formats and infrastructures, and the results were mildly encouraging. We examined the existing programs to see how easy it was going to be to alter the messaging formats and technology. The programs were too dependent on the message structure for it to be altered within a reasonable timeframe.

Given the still-poor results, we needed to examine the functional algorithms and the database access. We took a few of the midrange and lengthier running processors and inserted some logging to obtain split times of the various steps. Many of the functional algorithms were relatively expensive because of the required complexity for the mapping and restructuring of the data. The database operations seemed to take longer than we logically thought they should. (Over time an architect should develop a sense for a performance budget based on an abstract view of similar functionality where he or she had previously maximized performance.)

We then examined the logical database model. The design was not a pattern that would be performant for the types of programs in the system. The SQL from a few of the algorithms was extracted and placed in stand-alone model components. The idea was to see which types of performance increases were possible. Some increases came from changing some of the SQL statements, which were taking excessive time because the chosen partitioning scheme meant that reading core tables typically involved scanning all partitions. As our simulated database size grew, this became punitive to scalability. The primary problem, however, was not the extended length of time for individual statements but the sheer number of calls. This was a result of taking normalization too far. There were numerous tables with indexes on columns that changed frequently. Additionally, multicolumn keys were

being used instead of artificial keys, sometimes referred to as surrogate keys. The system generates them (typically as integers) to represent the “real” keys. This can improve performance and maintenance when dealing with complex key structures and/or when the actual key values can change.

We determined that material improvements were possible if we restructured the database design and changed the associated SQL statements. The programs were written in such a way that would have made the changes very expensive, however. Our conclusion was that the system would need a major overhaul if it were to be successful. Since the project had already spent well over \$10 million, this recommendation was a hard sell.

After an additional \$5 million, the project was canceled, and my team’s focus was redirected to other efforts. The modeling process had taken only about six weeks. The point to be made here is that it would be possible to use modeling to vet the major architectural decisions before committing large expenditures. It is vastly less expensive to discover that a design will not perform or scale *before* a system is built rather than after it has been placed in production.

Modeling New Systems

It should be standard practice to research the architectural options for new systems—or when making substantial overhauls to existing ones. The experiments should be with lightweight models rather than a full system, but it is vital that these models accurately capture the evolving behavior of the system. Otherwise the value of the modeling process is diminished and may lead to erroneous conclusions.

I typically start by trying to understand the functional problem space in an abstract fashion. Is the primary functionality a user-requested action followed by a system reply (such as, request/reply)? Is it a request followed by a stream of notifications (for example, ticking quotes) or bits (for example, music or video)? Is it to process some input data and send the result to another process or system (such as, flow-through)? Is it to crunch through a massive dataset in search of information (decision support system)? Is it a



It should be standard practice to research the architectural options for new systems—or when making substantial overhauls to existing ones.



combination of these, or something altogether different?

Some may ask: how do I know which portions of the system to model and how much time and effort should be spent in the process? It is a simple case of risk management. The modeling should focus on the areas that would be the most expensive to get wrong. The process should continue until the high-risk decisions can be justified. Make an effort to retest the decisions as often as practical.

One of the most challenging aspects in modeling is in finding the right balance between capturing enough of the system behavior and keeping the model from becoming too complex (and expensive) to implement. This is easier with an existing system. As you progress through the modeling iterations, if the observations begin to mimic aspects of the system, then you are probably pretty close. You can begin to alter the modeling drivers and components to explore more of the behavior. For a new system I typically look to model components that can be used as shells for the real component. The goal is to provide the responsible developer with a starting point that allows the focus to be on the functionality rather than having to explore the critical nuances of the underlying technology and infrastructure.

There are numerous technical modalities to consider when designing or evaluating architecture: performance, availability, scalability, security, testability, maintainability, ease of development, and operability. The priority ordering of these modalities may differ across systems, but each must be considered. How these modalities are addressed and their corresponding technical considerations may vary by system component. For example, with request/reply and streaming updates, latency is a critical performance factor, whereas throughput may be a better performance factor for flow-through message processing or bulk-request functionality. A perhaps subtle but nonetheless important message is to avoid mixing different modality implementations within the same component. Failure to adhere to this lesson puts the architecture on a sure path to complexity.

It is far too common to hear the ex-

cuse: “The system is [going to be] too large to take the time to model its behavior. We just need to start building it.” If the chore of modeling is considered too onerous, then it will probably be very challenging to achieve predictable performance, scalability, and other desirable technical attributes. Some development projects have a strong focus on unit tests, but in my experience it is rare to find a corresponding focus on testing the system architecture as a whole.

Modeling a Sample Component

Describing the modeling of a sample component may provide additional insight into the approach I am advocating. Suppose a new system calls for receiving some stream of data items (for example, stock quotes), enriching the data and publishing it to end users. An architect may suggest that some type of publisher component be built to perform this core requirement. How can this component be modeled before investing money in building a system around it? Data throughput and latency are probably primary concerns. Ideally, we have some target requirements for these. Scalability and availability are also issues that can be addressed with later iterations of the model but before proceeding with the functional development.

Based on this simple example, the model should contain at least two building blocks distinct from the publisher component. The incoming data feed needs to be simulated. A driver should be built to pump data into the publisher. Additionally, some type of client sink is necessary to validate the flow of messages and enable the measuring of throughput and latency. Figure 1 shows a simplified drawing with drivers and sinks for the proposed publisher.

The publisher model component should be built using the proposed target language. It should use any frameworks, libraries, among others, that may affect the model outcome, though it may not be obvious which of these could have an effect. In that case you should take a risk management approach to include those that are core to the operation of the component. Any new technology where the behavior is not already fully understood should be

included as well. Any nonsuspect infrastructure can be added in later iterations. It is important not to get mired in trying to build the functionality too early. As much as possible should be stubbed out.

In some systems a component such as the publisher may present the largest scalability hurdle. In that case we need to know what type of message flow can be handled, what type of latency can be expected, how many clients can be supported, and what type of flow the client applications can handle.

The data-feed driver should accept parameters that allow the message rate to be dialed to arbitrary levels. Any driver should be capable of pushing its target well past any expected high-water mark. The messages do not have to match the intended format, but they should be relatively close in size. Since the driver is tightly coupled with the publisher, it should be written for and run on the same type of platform (language, operating system, among others). This enables the same developer to build both the component and the driver. (I strongly suggest that each developer responsible for a system-level component also create a distinct driver and a possible sink as a standard practice.) The same holds true for the client sink so all three can be packaged together. This provides a cohesiveness that will allow the model to be reused for other purposes in the future.

As the modeling progresses, another model receiver should be built for the target client platform using its expected frameworks and communication mechanism. The reason for the two different platform receiver/sinks is to allow the publisher model component to be tested without involving another platform (for example, scalability testing). The client-platform model receiver can be used to determine if the publisher is interacting with the client platform properly. During future troubleshooting sessions these separate receivers would provide a means to isolate the problem area. All of the drivers and sinks should be maintained as part of the development and maintenance of the publisher.

The next step is to evaluate the publisher model in action with the drivers and sinks. To characterize the performance, some type of instrumentation

needs to be added to the client sink to calculate throughput. Care must be taken with any type of instrumentation so it does not influence the results of the test. For example, logging every single message received with a timestamp is likely to be punitive to performance. Instead, summary statistics can be kept in memory and written out at periodic intervals or when the test ends.

The data-feed driver should output data at a configurable rate while the client sinks count messages and compute the rate of data received. Another instrumentation method could be used to sample the latency. At specified message count intervals, the data-feed driver could log the message number and the originating timestamp. The client sinks could then log the receive timestamp at the same interval. If logged at an appropriate frequency, the samples could give a good representation of the latency without affecting the overall performance. High-resolution timers may be necessary. Testing across multiple machines with a latency requirement lower than the clock synchronization drift would require more sophisticated timing methods.

This model should be exercised at various message rates, including rates that completely overwhelm the publisher and its available resources. In addition to observing throughput and latency, the system resource utilization (CPU, memory, network, and so on) should be profiled. This information could be used later to determine if there are possible benefits in exploring infrastructure tuning.

As mentioned earlier, the publisher is required to do some type of data enrichment as the messages pass through. Throughput, latency, and memory consumption are likely to be impacted by this enrichment. This influence should be estimated and incorporated into the model publisher. If realistic estimates are not available, then purposely estimate high (or following the philosophy of this article, build another model and characterize it). If the cost of enrichment varies by message type, then a pseudorandom delay and memory allocation clustered around the expected averages could be inserted into the model publisher.

Other Uses for Modeling

Modeling is an iterative process. It should not be thought of as just some type of performance test. Here is a list of items that could be added to further the evaluation process.

- ▶ Use the model to evaluate various infrastructure choices. These could include messaging middleware, operating system and database-tuning parameters, network topology, and storage system options.

- ▶ Use the model to create a performance profile for a set of hardware, and use that profile to extrapolate performance on other hardware platforms. Any extrapolation will be more accurate if the model is profiled on more than one hardware platform.

- ▶ Use the performance profiles to determine if multiple instances of the publisher (horizontal scaling) are likely to be required as the system grows. If so, this capability should be built into the design and modeled appropriately. Converting components designed to be singletons could be very expensive.

- ▶ Use the model to explore the set of possible failure scenarios. Availability is one of the primary attributes of a quality system. Waiting to address it after a system is built can cost an order of magnitude more.

The examples used in this article can be seen in abstractions of many systems. Similar modeling approaches should be undertaken for any material component. When interrelated models have been built and tested they can then be combined for more comprehensive system modeling. The approach of building one model at a time allows the system behavioral knowledge to be gained in steps rather than attempting to understand—not to mention build—one all-encompassing model.

One key element present in almost all systems is some type of data store. Evaluating a database design can be complex. There are a number of steps that are similar to the system modeling already discussed, however. Once a draft of the database model (columns, tables, and so on) is available, it can be populated with enough generated data to enable some performance testing. The effort required to write a data generator for this purpose will give an idea of how easy it will be to work with

the database during the development process. If this generator seems too difficult to tackle, it may be a sign the database model is already too complex.

After the tables have been populated, the next step is to create driver(s) that will exercise the queries expected to be most expensive and/or most frequent. These drivers can be used to refine the underlying relational model, storage organization, and tuning parameters. Performing this type of modeling can be priceless. Discovering flaws in the application-level data model after all the queries have been written and the system is running in production is painful. I have worked to improve database performance on dozens of systems. Optimizing queries, storage subsystems, and other database-related items post development can be really challenging. If the system has been in production for some time, then the task is even more difficult. Many times the low-level infrastructure changes could have been determined by early modeling. With the proper design more standard configurations may have sufficed.

Instrumentation and Maintenance

Regardless of the type of driver/component combination, instrumentation is vital to both modeling and the long-lasting health of a system. It is not just a luxury. Flying blind about performance is not advised. Visual flight rules (that is, without instrumentation) can be used only when the skies are clear. How often is that true for modern systems? The functional and technical complexity typically clouds the ability to see clearly what is happening. System performance can be like floating down the river in a raft. If you do not observe the speed of the water periodically, then you might not notice an upcoming waterfall until the raft is hopelessly plunging over the edge. As mentioned previously, when the volume of instrumentation data is too high, consider using “tracers” and/or statistical sampling.

There are numerous advantages to keeping the drivers and model components up to date as a system evolves:

- ▶ They can be used for general regression testing for performance, availability, or scalability, when changes are proposed.

- ▶ They can be used for capacity planning by extrapolating performance from a smaller set of resources. The only practical way to do this is by fully understanding the resource usage characteristics.

- ▶ They can model infrastructure or other large-scale changes that may need to be made to an existing system.

- ▶ At times there are factors outside the control of the maintenance/development team (for example, infrastructure changes). The drivers could be used to test an isolated portion of the system. If any degradation was caused by the outside factors, then the results could provide “defensive” data to have the changes altered or rolled back.

- ▶ When some type of performance, availability, scalability, or other infrastructure problem arises, it would be much quicker to pull out the model and drivers than to take on the possibly overwhelming task of updating them while under pressure to troubleshoot a production problem.

Modeling is an extremely powerful method to understand and improve the overall quality of a system. For systems expected to last for years this improvement translates into real monetary savings. Development organizations can then spend their budgetary money on providing functionality. If the models and associated drivers are sustained, then this functional focus can be widely celebrated. ■

Related articles on queue.acm.org

Hidden in Plain Sight

Bryan Cantrill

<http://queue.acm.org/detail.cfm?id=1117401>

Visualizing System Latency

Brendan Gregg

<http://queue.acm.org/detail.cfm?id=1809426>

Performance Anti-Patterns

Bart Smaalders

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

References

1. Booch, G. *Object-oriented Analysis and Design with Applications* (2nd edition). Benjamin Cummings, Redwood City, CA, 1993.
2. Petzold, C. *Programming Windows*. Microsoft Press, 1988.

Kevin Montagne has over 25 years of experience in the IT field working with large-scale systems where performance and availability were critical. He spent 20 of those years in the financial industry, including more than a decade as an architect of front-office trading systems.

© 2010 ACM 0001-0782/10/1000 \$10.00

ACM's Online Books & Courses Programs!

Helping Members Meet Today's Career Challenges



NEW! 3,200 Online Courses in Multiple Languages Plus 1,000 Virtual Labs from Element K!



ACM's new Online Course Collection includes over **3,200 online courses in multiple languages, 1,000 virtual labs, e-reference tools, and offline capability**. Program highlights:

The ACM E-Learning Catalog - round-the-clock access to 3,200 online courses on a wide range of computing and business topics, in multiple languages.

Exclusive vLab® Virtual Labs - 1,000 unique vLab® exercises place users on systems using real hardware and software allowing them to gain important job-related experience.

Reference Tools - an e-Reference Library extends technical knowledge outside of the classroom, plus online Executive Summaries and quick reference cards to answer on-the-job questions instantly.

Offline Player - members can access assessments and self-study courses offline, anywhere and anytime, without a live Internet connection.

A downloadable Quick Reference Guide and a 15-minute site orientation course for new users are also available to help members get started.

The ACM Online Course Program is open to ACM Professional and Student Members.

600 Online Books from Safari

ACM members are eligible for a **special 40% savings** offer to upgrade to a Premium or Full Library subscription.

For more details visit:
http://pd.acm.org/books/about_sel.cfm

The ACM Online Books Collection includes **full access to 600 online books** from Safari® Books Online, featuring leading publishers including O'Reilly. Safari puts a complete IT and business e-reference library right on your desktop. Available to ACM Professional Members, Safari will help you zero in on exactly the information you need, right when you need it.



Association for
Computing Machinery

Advancing Computing as a Science & Profession

500 Online Books from Books24x7

All Professional and Student Members also have **full access to 500 online books** from Books24x7®, in ACM's rotating collection of complete unabridged books on the hottest computing topics. This virtual library puts information at your fingertips. Search, bookmark, or read cover-to-cover. Your bookshelf allows for quick retrieval and bookmarks let you easily return to specific places in a book.



pd.acm.org
www.acm.org/join

DOI:10.1145/1831407.1831425

**Neuroscience is beginning to inspire
a new generation of seeing machines.**

BY THOMAS SERRE AND TOMASO POGGIO

A Neuromorphic Approach to Computer Vision

IF PHYSICS WAS *the* science of the first half of the 20th century, biology was certainly *the* science of the second half. Neuroscience is now often cited as one of the key scientific focuses of the 21st century and has indeed grown rapidly in recent years, spanning a range of approaches, from molecular neurobiology to neuro-informatics and computational neuroscience. Computer science gave biology powerful new data-analysis tools that yielded bioinformatics and genomics, making possible the sequencing of the human genome. Similarly, computer science techniques are at the heart of brain imaging and other branches of neuroscience.

Computers are critical for the neurosciences, though at a much deeper level, representing the best

metaphor for the central mystery of how the brain produces intelligent behavior and intelligence itself. They also provide experimental tools for information processing, effectively testing theories of the brain, particularly those involving aspects of intelligence (such as sensory perception). The contribution of computer science to neuroscience happens at multiple levels and is well recognized. Perhaps less obvious is that neuroscience is beginning to contribute powerful new ideas and approaches to artificial intelligence and computer science as well. Modern computational neuroscience models are no longer toy models but quantitatively detailed while beginning to compete with state-of-the-art computer-vision systems. Here, we explore how computational neuroscience could become a major source of new ideas and approaches in artificial intelligence.

Understanding the processing of information in our cortex is a significant part of understanding how the brain works and understanding intelligence itself. For example, vision is one of our most developed senses. Primates easily categorize images or parts of images, as in, say, an office scene or a face within a scene, identifying specific objects. Our visual capabilities are exceptional, and, despite decades of engineering, no computer algorithm is yet able to match the performance of the primate visual system.

Our visual cortex may serve as a proxy for the rest of the cortex and thus

» key insights

- **The past century of neuroscience research has begun to answer fundamental questions ranging from the intricate inner workings of individual neurons to understanding the collective behavior of networks of millions of neurons.**
- **A key challenge for the visual cortex is how to deal with the poverty-of-stimulus problem.**
- **A major goal of the visual system is how to adapt to the statistics of its natural environment through visual experience and even evolution.**

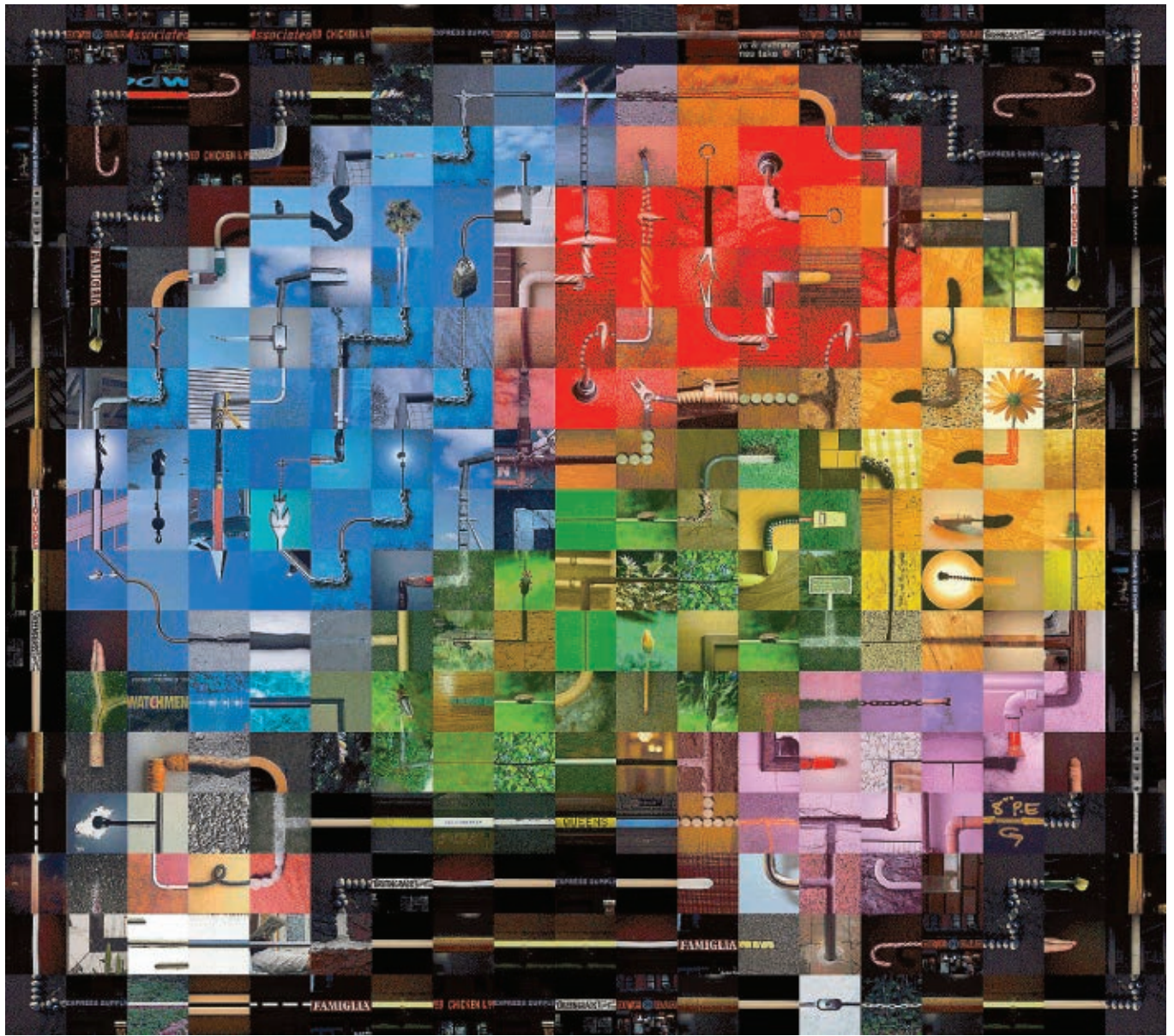


ILLUSTRATION BY KAI SCHREIBER

for intelligence itself. There is little doubt that even a partial solution to the question of which computations are performed by the visual cortex would be a major breakthrough in computational neuroscience and more broadly in neuroscience. It would begin to explain one of the most amazing abilities of the brain and open doors to other aspects of intelligence (such as language and planning). It would also bridge the gap between neurobiology and the various information sciences, making it possible to develop computer algorithms that follow the information-processing principles used by biological organisms and honed by natural evolution.

The past 60 years of experimental work in visual neuroscience has generated a large and rapidly increasing

amount of data. Today's quantitative models bridge several levels of understanding, from biophysics to physiology to behavior. Some of these models compete with state-of-the-art computer-vision systems and are close to human-level performance for specific visual tasks.

Here, we describe recent work toward a theory of cortical visual processing. Unlike other models that address the computations in a given brain area (such as primary visual cortex) or attempt to explain a particular phenomenon (such as contrast adaptation and specific visual illusion), we describe a large-scale model that attempts to mimic the main information-processing steps across multiple brain areas and millions of neuron-like units. A first step toward understanding cortical

functions may take the form of a detailed, neurobiologically plausible model, accounting for the connectivity, biophysics, and physiology of the cortex.

Models provide a much-needed framework for summarizing and integrating existing data and planning, coordinating, and interpreting new experiments. They can be powerful tools in basic research, integrating knowledge across multiple levels of analysis, from molecular to synaptic, cellular, systems, and complex visual behavior. However, models, as we discuss later, are limited in explanatory power but should, ideally, lead to a deeper and more general theory. Here, we discuss the role of the visual cortex and review key computational principles underlying the processing of information dur-

ing visual recognition, then explore a computational neuroscience model (representative of a class of older models) that implements these principles, including some of the evidence in its favor. When tested with natural images, the model performs robust object recognition on par with computer-vision systems and human performance for a specific class of quick visual-recognition tasks. The initial success of this research represents a case in point for arguing that over the next decade progress in computer vision and artificial intelligence promises to benefit directly from progress in neuroscience.

Goal of the Visual System

A key computational issue in object recognition^a is the specificity-invariance trade-off: Recognition must be able to finely discriminate between different objects or object classes (such as the faces in Figure 1) while being tolerant of object transformations (such as scaling, translation, illumination, changes in viewpoint, and clutter), as well as non-rigid transformations (such as variations in shape within a class), as in the change of facial expression in recognizing faces.

A key challenge posed by the visual cortex is how well it deals with the poverty-of-stimulus problem, or simple lack of visual information. Primates are able to learn to recognize an object in quite different images from far fewer labeled examples than are predicted by our present learning theory and algorithms. For instance, discriminative algorithms (such as support vector machines, or SVMs) can learn a complex object-recognition task from a few hundred labeled images. This number is small compared to the apparent dimensionality of the problem (millions of pixels), but a child, even a monkey, is apparently able to learn the same task from a handful of examples. As an example of the prototypical problem in visual recognition, imagine a (naïve) machine is shown an image of a given person and an image of another person. The system’s task is to discrimi-

^a Within recognition, one distinguishes between identification and categorization. From a computational point of view, both involve classification and represent two points on a spectrum of generalization levels.

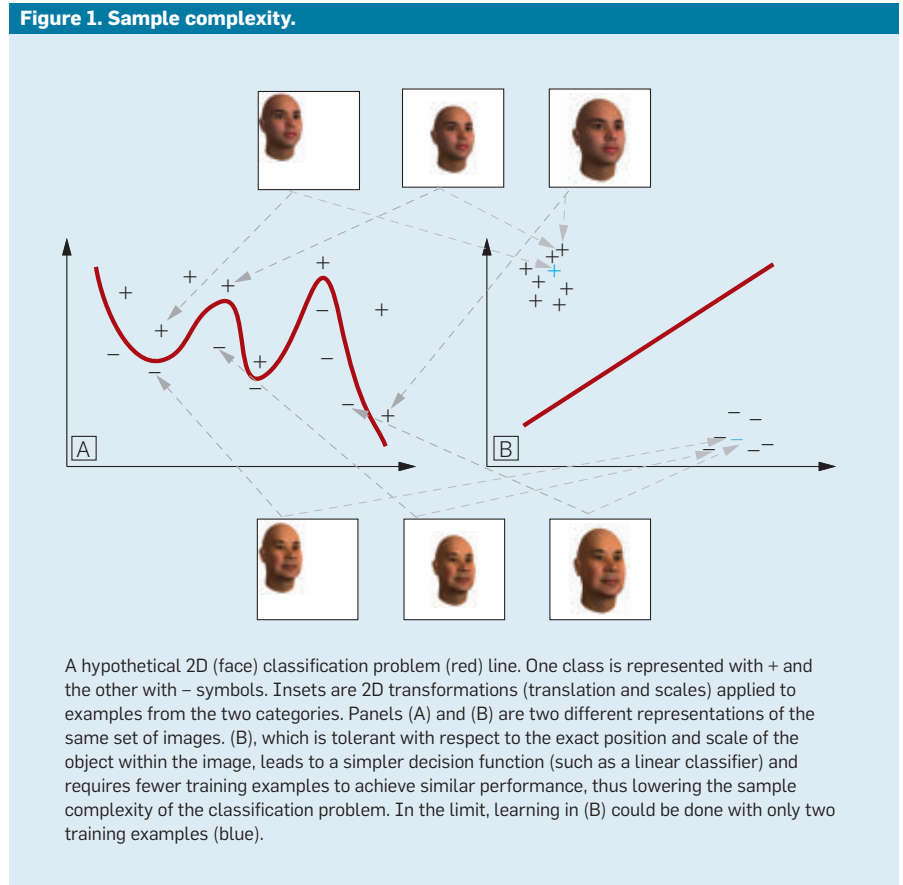
nate future images of these two people without seeing other images of them, though it has seen many images of other people and objects and their transformations and may have learned from them in an unsupervised way. Can the system learn to perform the classification task correctly with just two (or a few) labeled examples?

Imagine trying to build such a classifier from the output of two cortical cells, as in Figure 1. Here, the response of the two cells defines a 2D feature space to represent visual stimuli. In a more realistic setting, objects would be represented by the response patterns of thousands of such neurons. In the figure, we denote visual examples from the two people with + and - signs; panels (A) and (B) illustrate what the recognition problem would look like when these two neurons are sensitive vs. invariant to the precise position of the object within their receptive fields.^b In each case, a separation (the red lines indicate one such possible separation)

^b The receptive field of a neuron is the part of the visual field that (properly stimulated) could elicit a response from the neuron.

can be found between the two classes. It has been shown that certain learning algorithms (such as SVMs with Gaussian kernels) can solve any discrimination task with arbitrary difficulty (in the limit of an infinite number of training examples). That is, with certain classes of learning algorithms we are guaranteed to be able to find a separation for the problem at hand irrespective of the difficulty of the recognition task. However, learning to solve the problem may require a prohibitively large number of training examples.

In separating two classes, the two representations in panels (A) and (B) are not equal; the one in (B) is far superior to the one in (A). With no prior assumption on the class of functions to be learned, the “simplest” classifier that can separate the data in (B) is much simpler than the “simplest” classifier that separates the data in (A). The number of wiggles of the separation line (related to the number of parameters to be learned) gives a hand-wavy estimate of the complexity of a classifier. The sample complexity of the problem derived from the invariant representation in (B) is much lower than that of




the problem in (A). Learning to categorize the data-points in (B) requires far fewer training examples than in (A) and may be done with as few as two examples. The key problem in vision is thus what can be learned effectively with only a small number of examples.^c

The main point is not that a low-level representation provided from the retina would not support robust object recognition. Indeed, relatively good computer-vision systems developed in the 1990s were based on simple retina-like representations and rather complex decision functions (such as radial basis function networks). The main problem of these systems is they required a prohibitively large number of training examples compared to humans.


More recent work in computer vision suggests a hierarchical architecture may provide a better solution to the problem; see also Bengio and Le Cun¹ for a related argument. For instance, Heisele et al.¹⁰ designed a hierarchical system for the detection and recognition of faces, an approach based on a hierarchy of “component experts” performing a local search for one facial component (such as an eye or a nose) over a range of positions and scales. Experimental evidence from Heisele et al.¹⁰ suggests such hierarchical systems based exclusively on linear (SVM) classifiers significantly outperform a shallow architecture that tries to classify a face as a whole, albeit by relying on more complex kernels.

The visual system may be using a similar strategy to recognize objects, with the goal of reducing the sample complexity of the classification problem. In this view, the visual cortex transforms the raw image into a position- and scale-tolerant representation through a hierarchy of processing stages, whereby each layer gradually increases the tolerance to position and scale of the image representation. After several layers of such processing stages, the resulting image representation can be used much more efficiently for task-dependent learning and classi-

^c The idea of sample complexity is related to the point made by DiCarlo and Cox⁴ about the main goal of processing information from the retina to higher visual areas to be “untangling object representations,” so a simple linear classifier can discriminate between any two classes of objects.



The role of the anatomical back-projections present (in abundance) among almost all areas in visual cortex is a matter of debate.



fication by higher brain areas.

These stages can be learned during development from temporal streams of natural images by exploiting the statistics of natural environments in two ways: correlations over images that provide information-rich features at various levels of complexity and sizes; and correlations over time used to learn equivalence classes of these features under transformations (such as shifts in position and changes in scale). The combination of these two learning processes allows efficient sharing of visual features between object categories and makes learning new objects and categories easier, since they inherit the invariance properties of the representation learned from previous experience in the form of basic features common to other objects. In the following sections, we review evidence for this hierarchical architecture and the two correlation mechanisms described earlier.

Hierarchical Architecture and Invariant Recognition

Several lines of evidence (from both human psychophysics and monkey electrophysiology studies) suggest the primate visual system exhibits at least some invariance to position and scale. While the precise amount of invariance is still under debate, there is general agreement as to the fact that there is at least some generalization to position and scale.

The neural mechanisms underlying such invariant visual recognition have been the subject of much computational and experimental work since the early 1990s. One general class of computational models postulates that the hierarchical organization of the visual cortex is key to this process; see also Hegd  and Felleman⁹ for an alternative view. The processing of shape information in the visual cortex follows a series of stages, starting with the retina and proceeding through the lateral geniculate nucleus (LGN) of the thalamus to primary visual cortex (V1) and extrastriate visual areas, V2, V4, and the inferotemporal (IT) cortex. In turn, IT provides a major source of input to the prefrontal cortex (PFC) involved in linking perception to memory and action; see Serre et al.²⁹ for references.

As one progresses along the ventral

stream of the visual cortex, neurons become selective for stimuli that are increasingly complex—from simple oriented bars and edges in early visual area V1 to moderately complex features in intermediate areas (such as a combination of orientations) to complex objects and faces in higher visual areas (such as IT). Along with this increase in complexity of the preferred stimulus, the invariance properties of neurons seem to also increase. Neurons become more and more tolerant with respect to the exact position and scale of the stimulus within their receptive fields. As a result, the receptive field size of neurons increases from about one degree or less in V1 to several degrees in IT.

Compelling evidence suggests that IT, which has been critically linked with a monkey's ability to recognize objects, provides a representation of the image that facilitates recognition tolerant of image transformations. For instance, Logothetis et al.¹⁶ showed that monkeys can be trained to recognize paperclip-like wireframe objects at a specific location and scale. After training, recordings in their IT cortex revealed significant selectivity for the trained objects. Because monkeys were unlikely to have been in contact with the specific paperclip prior to training, this experiment provides indirect evidence of learning. More important, Logothetis et al.¹⁶ found selective neurons also exhibited a range of invariance with respect to the exact position (two to four degrees) and scale (around two octaves) of the stimulus, which was never presented before testing at these new positions and scales. In 2005, Hung et al.¹² showed it was possible to train a (linear) classifier to robustly read out from a population of IT neurons the category information of a briefly flashed stimulus. Hung et al. also showed the classifier was able to generalize to a range of positions and scales (similar to Logothetis et al.'s data) not presented during the training of the classifier. This generalization suggests the observed tolerance to 2D transformation is a property of the population of neurons learned from visual experience but available for a novel object without object-specific learning, depending on task difficulty.

Computational Models of Object Recognition in Cortex

We developed^{26,29} (in close cooperation with experimental labs) an initial quantitative model of feedforward hierarchical processing in the ventral stream of the visual cortex (see Figure 2). The resulting model effectively integrates the large body of neuroscience data (summarized earlier) characterizing the properties of neurons along the object-recognition processing hierarchy. The model also mimics human performance in difficult visual-recognition tasks²⁸ while performing at least as well as most current computer-vision systems.²⁷

Feedforward hierarchical models have a long history, beginning in the 1970s with Marko and Giebel's homogeneous multilayered architecture¹⁷ and later Fukushima's Neocognitron.⁶ One of their key computational mechanisms originates from the pioneering physiological studies and models of Hubel and Wiesel (<http://serre-lab.clps.brown.edu/resources/ACM2010>). The basic idea is to build an increasingly complex and invariant object representation in a hierarchy of stages by progressively integrating, or pooling, convergent inputs from lower levels. Building on existing models (see supplementary notes <http://serre-lab.clps.brown.edu/resources/ACM2010>), we have been developing^{24,29} a similar computational theory that attempts to quantitatively account for a host of recent anatomical and physiological data; see also Mutch and Lowe¹⁹ and Masquelier et al.¹⁸

The feedforward hierarchical model in Figure 2 assumes two classes of functional units: simple and complex. Simple act as local template-matching operators, increasing the complexity of the image representation by pooling over local afferent units with selectivity for different image features (such as edges at different orientations). Complex increase the tolerance of the representation with respect to 2D transformations by pooling over afferent units with similar selectivity but slightly different positions and scales.

Learning and plasticity. How the organization of the visual cortex is influenced by development vs. genetics is a matter of debate. An fMRI study²¹

showed the patterns of neural activity elicited by certain ecologically important classes of objects (such as faces and places in monozygotic twins) are significantly more similar than in dizygotic twins. These results suggest that genes may play a significant role in the way the visual cortex is wired to process certain object classes. Meanwhile, several electrophysiological studies have demonstrated learning and plasticity in the adult monkey; see, for instance, Li and DiCarlo.¹⁵ Learning is likely to be both faster and easier to elicit in higher visually responsive areas (such as PFC and IT¹⁵) than in lower areas.

This learning result makes intuitive sense. For the visual system to remain stable, the time scale for learning should increase ascending the ventral stream.^d In the Figure 2 model, we assumed unsupervised learning from V1 to IT happens during development in a sequence starting with the lower areas. In reality, learning might continue throughout adulthood, certainly at the level of IT and perhaps in intermediate and lower areas as well.

Unsupervised learning in the ventral stream of the visual cortex. With the exception of the task-specific units at the top of the hierarchy ("visual routines"), learning in the model in Figure 2 is unsupervised, thus closely mimicking a developmental learning stage.

As emphasized by several authors, statistical regularities in natural visual scenes may provide critical cues to the visual system for learning with very limited or no supervision. A key goal of the visual system may be to adapt to the statistics of its natural environment through visual experience and perhaps evolution, too. In the Figure 2 model, the selectivity of simple and complex units can be learned from natural video sequences (see supplementary ma-

d In the hierarchical model in Figure 1, learning proceeds layer by layer, starting at the bottom, a process similar to recent work by Hinton¹¹ but that is quite different from the original neural networks that used back-propagation and simultaneously learned all layers at the same time. Our implementation includes the unsupervised learning of features from natural images but assumes the learning of position and scale tolerance, thus hardwired in the model; see Masquelier et al.¹⁸ for an initial attempt at learning position and scale tolerance in the model.

terial <http://serre-lab.clps.brown.edu/resources/ACM2010> for details).

Supervised learning in higher areas. After this initial developmental stage, learning a new object category requires training only of task-specific circuits at the top of the ventral-stream hierarchy, thus providing a position and scale-invariant representation to task-specific circuits beyond IT to learn to generalize over transformations other than image-plane transformations (such as 3D rotation) that must be learned anew for each object or category. For instance, pose-invariant face categorization circuits may be built, possibly in PFC, by combining several units tuned to different face examples, including different people, views, and lighting conditions (possibly in IT).

A default routine may be running in a default state (no specific visual task), perhaps the routine *What is there?* As an example of a simple routine consider a classifier that receives the activity of a few hundred IT-like units, tuned to examples of the target object and distractors. While learning in the model from the layers below is stimulus-driven, the PFC-like classification units are trained in a supervised way following a perceptron-like learning rule.

Immediate Recognition

The role of the anatomical back-projections present (in abundance) among almost all areas in the visual cortex is a matter of debate. A commonly accepted hypothesis is that the basic processing of information is feedforward,³⁰ supported most directly by the short times required for a selective response to appear in cells at all stages of the hierarchy. Neural recordings from IT in a monkey¹² show the activity of small neuronal populations over very short time intervals (as short as 12.5ms and about 100ms after stimulus onset) contains surprisingly accurate and robust information supporting a variety of recognition tasks. While this data does not rule out local feedback loops within an area, it does suggest that a core hierarchical feedforward architecture (like the one described here) may be a reasonable starting point for a theory of the visual cortex, aiming to explain immediate recognition, the initial phase of recognition before eye movement

and high-level processes take place.

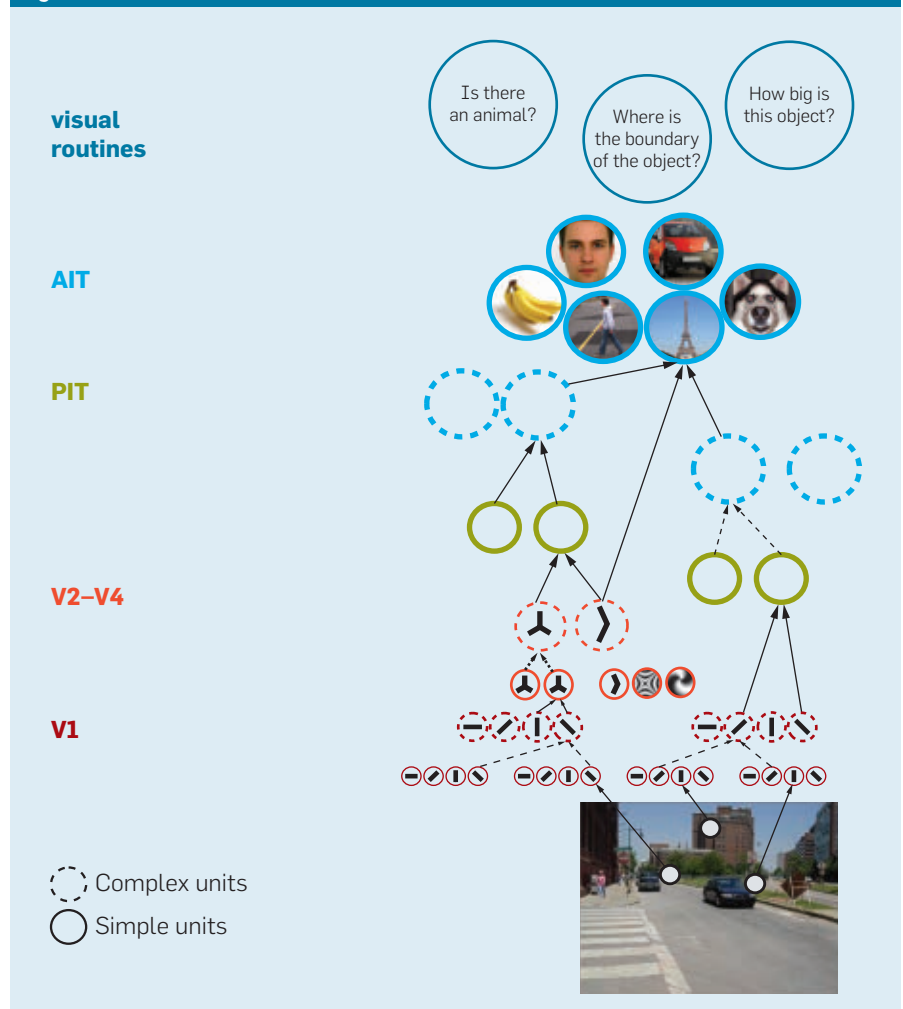
Agreement with experimental data. Since its original development in the late 1990s,^{24,29} the model in Figure 2 has been able to explain a number of new experimental results, including data not used to derive or fit model parameters. The model seems to be qualitatively and quantitatively consistent with (and in some cases predicts²⁹) several properties of subpopulations of cells in V1, V4, IT, and PFC, as well as fMRI and psychophysical data (see the sidebar “Quantitative Data Compatible with the Model” for a complete list of findings).

We compared the performance of the model against the performance of human observers in a rapid animal vs. non-animal recognition task²⁸ for which recognition is quick and cortical back-projections may be less relevant. Results indicate the model predicts human performance quite well during such a task, suggesting the model may

indeed provide a satisfactory description of the feedforward path. In particular, for this experiment, we broke down the performance of the model and human observers into four image categories with varying amounts of clutter. Interestingly, the performance of both the model and the human observers was most accurate (~90% correct for both human participants and the model) on images for which the amount of information is maximal and clutter minimal and decreases monotonically as the clutter in the image increases. This decrease in performance with increasing clutter likely reflects a key limitation of this type of feedforward architecture. This result is in agreement with the reduced selectivity of neurons in V4 and IT when presented with multiple stimuli within their receptive fields for which the model provides a good quantitative fit²⁹ with neurophysiology data (see the sidebar).

Application to computer vision.

Figure 2. Hierarchical feedforward models of the visual cortex.



How does the model²⁹ perform real-world recognition tasks? And how does it compare to state-of-the-art artificial-intelligence systems? Given the specific biological constraints the theory must satisfy (such as using only biophysically plausible operations, receptive field sizes, and a range of invariances), it was not clear how well the model implementation would perform compared to systems heuristically engineered for these complex tasks.

Several years ago, we were surprised to find the model capable of recognizing complex images,²⁷ performing at a level comparable to some of the best existing systems on the CalTech-101 image database of 101 object categories with a recognition rate of about 55% (chance level < 1%); see Serre et al.²⁷ and Mutch and Lowe.¹⁹ A related system with fewer layers, less invariance, and more units had an even better recognition rate on the CalTech data set.²⁰

We also developed an automated system for parsing street-scene images²⁷ based in part on the class of models described earlier. The system recognizes seven different object categories—cars, pedestrians, bikes, skies, roads, buildings, trees—from natural images of street scenes despite very large variations in shape (such as trees in summer and winter and SUVs and compact cars from any point of view).

Content-based recognition and search in videos is an emerging application of computer vision, whereby neuroscience may again suggest an avenue for approaching the problem. In 2007, we developed an initial model for recognizing biological motion and actions from video sequences based on the organization of the dorsal stream of the visual cortex,¹³ which is critically linked to the processing of motion information, from V1 and MT to higher motion-selective areas MST/FST and STS. The system relies on computational principles similar to those in the model of the ventral stream described earlier but that start with spatio-temporal filters modeled after motion-sensitive cells in the primary visual cortex.

We evaluated system performance for recognizing actions (human and animal) in real-world video sequenc-

Quantitative Data Compatible with the Model

Black corresponds to data used to derive the parameters of the model, **red** to data consistent with the model (not used to fit model parameters), and **blue** to actual correct predictions by the model. **Notations:** PFC (prefrontal cortex), V1 (visual area I or primary visual cortex), V4 (visual area IV), and IT (inferotemporal cortex). **Data from these areas corresponds to monkey electrophysiology studies. LOC (Lateral Occipital Complex) involves fMRI with humans. The psychological studies are psychophysics on human subjects.**

Area	Type of data	Ref. biol. data	Ref. model data
Psych.	Rapid animal categorization	(1)	(1)
	Face inversion effect	(2)	(2)
LOC	Face processing (fMRI)	(3)	(3)
PFC	Differential role of IT and PFC in categorization	(4)	(5)
IT	Tuning and invariance properties	(6)	(5)
	Read out for object category	(7)	(8,9)
	Average effect in IT	(10)	(10)
V4	MAX operation	(11)	(5)
	Tuning for two-bar stimuli	(12)	(8,9)
	Two-spot interaction	(13)	(8)
	Tuning for boundary conformation	(14)	(8,15)
	Tuning for Cartesian and non-Cartesian gratings	(16)	(8)
V1	Simple and complex cells tuning properties	(17–19)	(8)
	MAX operation in subset of complex cells	(20)	(5)

- Serre, T., Oliva, A., and Poggio, T. *Proc. Natl. Acad. Sci.* 104, 6424 (Apr. 2007).
- Riesenhuber, M. et al. *Proc. Biol. Sci.* 271, S448 (2004).
- Jiang, X. et al. *Neuron* 50, 159 (2006).
- Freedman, D.J., Riesenhuber, M., Poggio, T., and Miller, E.K. *Journ. Neurosci.* 23, 5235 (2003).
- Riesenhuber, M. and Poggio, T. *Nature Neuroscience* 2, 1019 (1999).
- Logothetis, N.K., Pauls, J., and Poggio, T. *Curr. Biol.* 5, 552 (May 1995).
- Hung, C.P., Kreiman, G., Poggio, T., and DiCarlo, J.J. *Science* 310, 863 (Nov. 2005).
- Serre, T. et al. *MIT AI Memo 2005-036 / CBCL Memo 259* (2005).
- Serre, T. et al. *Prog. Brain Res.* 165, 33 (2007).
- Zoccolan, D., Kouh, M., Poggio, T., and DiCarlo, J.J. *Journ. Neurosci.* 27, 12292 (2007).
- Gawne, T.J. and Martin, J.M. *Journ. Neurophysiol.* 88, 1128 (2002).
- Reynolds, J.H., Chelazzi, L., and Desimone, R. *Journ. Neurosci.* 19, 1736 (Mar. 1999).
- Taylor, K., Mandon, S., Freiwald, W.A., and Kreiter, A.K. *Cereb. Cortex* 15, 1424 (2005).
- Pasupathy, A. and Connor, C. *Journ. Neurophysiol.* 82, 2490 (1999).
- Cadieu, C. et al. *Journ. Neurophysiol.* 98, 1733 (2007).
- Gallant, J.L. et al. *Journ. Neurophysiol.* 76, 2718 (1996).
- Schiller, P.H., Finlay, B.L., and Volman, S.F. *Journ. Neurophysiol.* 39, 1288 (1976).
- Hubel, D.H. and Wiesel, T.N. *Journ. Physiol.* 160, 106 (1962).
- De Valois, R.L., Albrecht, D.G., and Thorell, L.G. *Vision Res.* 22, 545 (1982).
- Lampl, I., Ferster, D., Poggio, T., and Riesenhuber, M. *Journ. Neurophysiol.* 92, 2704 (2004).

es,¹³ finding that the model of the dorsal stream competed with a state-of-the-art action-recognition system (that outperformed many other systems) on all three data sets.¹³ A direct extension of this approach led to a computer system for the automated monitoring and analysis of rodent behavior for behavioral phenotyping applications that perform on par with human manual scoring. We also found the learning in

this model produced a large dictionary of optic-flow patterns that seems consistent with the response properties of cells in the medial temporal (MT) area in response to both isolated gratings and plaids, or two gratings superimposed on one another.

Conclusion

Demonstrating that a model designed to mimic known anatomy and physiol-

ogy of the primate visual system leads to good performance with respect to computer-vision benchmarks may suggest neuroscience is on the verge of providing novel and useful paradigms to computer vision and perhaps to other areas of computer science as well. The feedforward model described here can be modified and improved by taking into account new experimental data (such as more detailed properties of specific visual areas like V1²⁵), implementing some of its implicit assumptions (such as learning invariances from sequences of natural images), taking into account additional sources of visual information (such as binocular disparity and color), and extension to describe the detailed dynamics of neural responses. Meanwhile, the recognition performance of models of this general type can be improved by exploring parameters (such as receptive field sizes and connectivity) by, say, using computer-intensive iterations of a mutation-and-test cycle.

However, it is important to realize the intrinsic limitations of the specific computational framework we have described and why it is at best a first step toward understanding the visual cortex. First, from the anatomical and physiological point of view the class of feedforward models we've described here is incomplete, as it does not account for the massive back-projections found in the cortex. To date, the role of cortical feedback remains poorly understood. It is likely that feedback underlies top-down signals related to attention, task-dependent biases, and memory. Back-projections must also be taken into account in order to describe visual perception beyond the first 100msec–200msec.

Given enough time, humans use eye movement to scan images, and performance in many object-recognition tasks improves significantly over that obtained during quick presentations. Extensions of the model to incorporate feedback are possible and under way.² Feedforward models may well turn out to be approximate descriptions of the first 100msec–200msec of the processing required by more complex theories of vision based on back-projections.^{3,5,7,8,14,22,31} However, the computations involved in

the initial phase are nontrivial but essential for any scheme involving feedback. A related point is that normal visual perception is much more than classification, as it involves interpreting and parsing visual scenes. In this sense, the class of models we describe is limited, since it deals only with classification tasks. More complex architectures are needed; see Serre et al.²⁶ for a discussion.

Finally, we described a class of models, *not* a theory. Computational models are not sufficient on their own. Our model, despite describing (quantitatively) aspects of monkey physiology and human recognition, does not yield a good understanding of the computational principles of the cortex and their power. What is yet needed is a mathematical theory to explain the hierarchical organization of the cortex.

Acknowledgments

We thank Jake Bouvrie for his useful feedback on the manuscript, as well as the referees for their valuable comments. C

References

1. Bengio, J. and Le Cun, Y. Scaling learning algorithms towards AI. In *Large-Scale Kernel Machines*, L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, J., Eds. MIT Press, Cambridge, MA, 2007, 321–360.
2. Chikkerur, S., Serre, T., Tan, C., and Poggio, T. *What and Where: A Bayesian Inference Theory of Attention* (in press). Vision Research, 2010.
3. Dean, T. A computational model of the cerebral cortex. In *Proceedings of the 20th National Conference on Artificial Intelligence* (Pittsburgh, PA, July 9–13, 2005), 938–943.
4. DiCarlo, J.J. and Cox, D.D. Untangling invariant object recognition. *Trends in Cognitive Science* 11, 8 (Aug. 2007), 333–341.
5. Epshtein, B., Lifshitz, I., and Ullman, S. Image interpretation by a single bottom-up top-down cycle. *Proceedings of the National Academy of Sciences* 105, 38 (Sept. 2008), 14298–14303.
6. Fukushima, K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics* 36, 4 (Apr. 1980), 193–202.
7. George, D. and Hawkins, J. A hierarchical Bayesian model of invariant pattern recognition in the visual cortex. In *Proceedings of the International Joint Conference on Neural Networks* 3, (Montréal, July 31–Aug. 4). IEEE Press, 2005, 1812–1817.
8. Grossberg, S. Towards a unified theory of neocortex: Laminar cortical circuits for vision and cognition. *Progress in Brain Research* 165 (2007), 79–104.
9. Hegd , H. and Felleman, D.J. Reappraising the functional implications of the primate visual anatomical hierarchy. *The Neuroscientist* 13, 5 (2007), 416–421.
10. Heisele, B., Serre, T., and Poggio, T. A component-based framework for face detection and identification. *International Journal of Computer Vision* 74, 2 (Jan. 1, 2007), 167–181.
11. Hinton, G.E. Learning multiple layers of representation. *Trends in Cognitive Sciences* 11, 10 (Oct. 2007), 428–434.
12. Hung, C.P., Kreiman, G., Poggio, T., and DiCarlo, J.J. Fast read-out of object identity from macaque inferior

- temporal cortex. *Science* 310, 5749 (Nov. 4, 2005), 863–866.
13. Jhuang, H., Serre, T., Wolf, L., and Poggio, T. A biologically inspired system for action recognition. In *Proceedings of the 11th IEEE International Conference on Computer Vision* (Rio de Janeiro, Brazil, Oct. 14–20). IEEE Press, 2007.
14. Lee, T.S. and Mumford, D. Hierarchical Bayesian inference in the visual cortex. *Journal of the Optical Society of America* 20, 7 (July 2003), 1434–1448.
15. Li, N. and DiCarlo, J.J. Unsupervised natural experience rapidly alters invariant object representation in visual cortex. *Science* 321, 5895 (Sept. 12, 2008), 1502–1507.
16. Logothetis, N.K., Pauls, J., and Poggio, T. Shape representation in the inferior temporal cortex of monkeys. *Current Biology* 5, 5 (May 1, 1995), 552–563.
17. Marko, H. and Giebel, H. Recognition of handwritten characters with a system of homogeneous layers. *Nachrichtentechnische Zeitschrift* 23 (1970), 455–459.
18. Masquelier, T., Serre, T., Thorpe, S., and Poggio, T. *Learning Complex Cell Invariance from Natural Videos: A Plausibility Proof*. MIT Center for Biological & Computational Learning Paper #269/MIT-CSAIL-TR #2007-060, Cambridge, MA, 2007.
19. Mutch, J. and Lowe, D. Multiclass object recognition using sparse, localized features. In *Proceedings of the Computer Vision and Pattern Recognition* (New York, June 17–22, 2006).
20. Pinto, N., Cox, D.D., and DiCarlo, J.J. Why is real-world visual object recognition hard? *PLoS Computational Biology* 4, 1 (Jan. 1, 2008), e27.
21. Polk, T.A., Park, J.E., Smith, M.R., and Park, D.C. Nature versus nurture in ventral visual cortex: A functional magnetic resonance imaging study of twins. *Journal of Neuroscience* 27, 51 (2007), 13921–13925.
22. Rao, R.P. and Ballard, D.H. Predictive coding in the visual cortex: A functional interpretation of some extra-classical receptive-field effects. *Nature Neuroscience* 2, 1 (1999), 79–87.
23. Reynolds, J.H., Chelazzi, L., and Desimone, R. Competitive mechanisms subserve attention in macaque areas V2 and V4. *Journal of Neuroscience* 19, 5 (Mar. 1, 1999), 1736–1753.
24. Riesenhuber, M. and Poggio, T. Hierarchical models of object recognition in cortex. *Nature Neuroscience* 2, 11 (1999), 1019–1025.
25. Rolls, E.T. and Deco, G. *Computational Neuroscience of Vision*. Oxford University Press, Oxford, U.K., 2002.
26. Serre, T., Kreiman, G., Kouh, M., Cadieu, C., Knoblich, U., and Poggio, T. A quantitative theory of immediate visual recognition. *Progress in Brain Research* 165 (2007), 33–56.
27. Serre, T., Wolf, L., Bileschi, S., Riesenhuber, M., and Poggio, T. Object recognition with cortex-like mechanisms. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 29, 3 (2007), 411–426.
28. Serre, T., Oliva, A., and Poggio, T. A feedforward architecture accounts for rapid categorization. *Proceedings of the National Academy of Sciences* 104, 15 (Apr. 10, 2007), 6424–6429.
29. Serre, T., Kouh, M., Cadieu, C., Knoblich, U., Kreiman, G., and Poggio, T. *A Theory of Object Recognition: Computations and Circuits in the Feedforward Path of the Ventral Stream in Primate Visual Cortex*. MIT AI Memo 2005-036 / CBCL Memo 259, AI Memo 2005-036 / CBCL Memo 259 2005. Cambridge, MA, 2005.
30. Thorpe, S., Fize, D., and Marlot, C. Speed of processing in the human visual system. *Nature* 381, 6582 (1996), 520–522.
31. Yuille, A. and Kersten, D. Vision as Bayesian inference: Analysis by synthesis? *Trends in Cognitive Science* 10, 7 (July 2006), 301–308.

Thomas Serre (thomas_serre@brown.edu) is an assistant professor in the Department of Cognitive, Linguistic & Psychological Sciences at Brown University, Providence, RI.

Tomaso Poggio (tp@ai.mit.edu) is the Eugene McDermott Professor in the Department of Brain and Cognitive Sciences in the McGovern Institute for Brain Research at the Massachusetts Institute of Technology, Cambridge, MA.

DOI:10.1145/1831407.1831426

IT jobs requiring interpersonal interaction or physical presence in fixed locations are less likely to be sent out of the country.

BY PRASANNA B. TAMBE AND LORIN M. HITT

How Offshoring Affects IT Workers

THOUGH THE OUTSOURCING of IT services has long been a topic of academic interest,²² the potential for the global sourcing of IT services to have a long-term effect on the domestic IT work force continues to attract significant interest from the media, public, and academic community.^{1,3,6,15,19,24} Here, we use survey data collected in 2007 to characterize the effect offshoring has had on the U.S. IT work force (see the sidebar “Key Survey Questions”) and estimate how it will affect the demand for skills among U.S. IT workers in the future.

Understanding the effect of offshoring on domestic employment is potentially important for anticipating the training needs of existing and future IT workers and for enabling policymakers to frame initiatives that ease the transition to a global IT work force. However, our current understanding is limited by

a paucity of data on firms’ offshoring activities. Most discussion of offshoring relies on anecdotes, press reports, and theoretical arguments. Indeed, the U.S. government acknowledges development of better offshoring data is a pressing policy concern.⁹

The primary contribution of this study is the collection and analysis of data describing how offshoring affects the U.S. work force. That data comes from two complementary, unusually large surveys carried out in late 2007, one involving 3,014 human resources managers and the other more than 6,000 U.S. workers employed in a variety of occupations. The data allows us to provide general statistics about the overall rate of U.S. IT offshoring and address two main questions: Do the rates of IT worker offshoring differ significantly from the offshoring rates for workers in other occupations? And is the pattern of IT offshoring consistent with the theory that jobs are less readily offshored if they require face-to-face contact with U.S.-based consumers or co-workers or require employees to perform hands-on work with U.S.-based assets?

Our interest in the second question was motivated by work suggesting that job characteristics (such as the need for customer contact or physical presence or information intensity) are closely related to the potential rate of offshoring.^{2,4,11,18} In the study, we combined data on offshoring-related displacement by occupation with Blinder’s classification⁴ of “offshorability” of various occupations to understand

» key insights

- **Offshoring is most common in high-tech firms and IT functions.**
- **IT workers reported offshoring-related displacement at a rate of 8%, more than double that of workers in other occupations.**
- **Technical occupations reliant on skills that can be delivered with relatively little face-to-face contact are more easily offshored, suggesting a coming shift in the skill base of the domestic IT work force.**

N



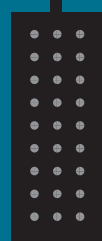
OFFSHORING



EAST



SOUTH



Key Survey Questions

QUESTIONS FOR EMPLOYERS

Did your company outsource work to third-party vendors outside the country in 2007?

- Yes No I don't know

Did your company offshore job functions to its foreign affiliates in 2007?

- Yes No I don't know

What positions are your company most likely to offshore?

- Sales Managers
- HR Managers
- Sales Agents
- Systems Administrators
- Financial Specialists
- Marketing Managers
- General Managers
- Hardware Engineers
- Software Engineers
- Computer-Support Specialists
- Computer-Systems Analysts
- Data-Entry Keyers
- Computer Programmers
- Network Analysts
- Database Administrators
- Software Developers
- Graphic Designers
- Financial Service Providers
- Customer Service Providers

Where do you offshore?

Check all that apply.

[drop-down list of countries]⬇

QUESTIONS FOR WORKERS

Have you ever been displaced from your job because your position was offshored?

- Yes No

In which state is your company headquartered?

[drop-down list of U.S. states]⬇

In what industry do you currently work?

[drop-down list of industries]⬇

What is your current profession?

[drop-down list of professions]⬇

What is your job level?

- Professional/Technical Staff Member
- Entry level/Administrative/Clerical
- Director/Manager/Supervisor/Team Leader
- Vice President
- Senior Management

What is your current salary?

[drop-down list of income ranges] ⬇

You indicated you were displaced because your position was offshored. What happened as a result of your last displacement?

- I was placed somewhere else in the company
- I was let go
- Other

separate questionnaires, both administered in the winter of 2007 by a third-party survey firm on behalf of one of the largest online recruitment and career-advancement companies in the U.S. The first focused on the offshoring practices of 3,016 individual firms, including whether and why they offshore and what types of work and to what countries they offshore. It was conducted within the U.S. among hiring managers and human-resource professionals employed full-time with significant involvement in hiring decisions. Respondents were also asked about firm characteristics (such as size and industry). The second was administered to individual workers and included questions relating to whether or not they had been displaced due to offshoring. It was also conducted within the U.S. (online) among 6,704 employees employed full-time and included both firm characteristics (such as size and industry) and employee characteristics (such as age, salary, and job level).

To test the hypothesis that job characteristics affect the likelihood of a job being offshored, we used probit models in which the dependent variable was 1 if an employee reported being displaced due to offshoring or an employer reported offshoring a particular type of work; we also included a measure of the importance of face-to-face contact or physical presence as an independent variable. Rather than restrict our sample to IT workers, we included all occupations in the analysis to increase the variation in the skill content of jobs, employing Huber-White robust (clustered) standard errors to account for possible random firm effects.

We captured the importance of face-to-face contact and physical presence in our regression models by including index values computed in a study of the offshorability of various occupations.⁴ Blinder's index is derived by placing jobs into categories depending on whether they require face-to-face interaction (such as child-care workers) and whether they require workers (such as in the construction trades) to be in a particular location. To maintain consistency with Blinder's classification, we adopted the term "personally delivered"

how job characteristics correlate with offshoring rates.

About 15% of all firms and 40% of technology firms we surveyed engaged in some offshoring activity, with about 30% offshoring IT workers. About 8% of IT workers reported having been displaced due to offshoring, more than twice the percentage of any other type of employee in the survey. However, this rate implies an annual displacement rate of about 1% per year, a relatively small fraction of the annual worker turnover rate in the U.S. economy. In addition, the offshoring of some IT occupations (such as programmers and software developers) was especially likely to be associated with domestic job displacement. Other occupations requiring more interpersonal interaction (such as systems analysts) were less likely to be offshored, and overseas employment

in other occupations (such as sales and management) may have been directed at serving offshore customers and therefore were also less likely to be associated with job displacement in the U.S.

We make three separate contributions toward understanding how offshoring affects domestic IT workers: quantify the extent to which offshoring has affected IT workers; show a relationship between occupational attributes and offshoring-related displacement, providing empirical support for emerging theories of how offshoring drives the global disaggregation of skills^{2,4}; and contribute to the literature demonstrating the growing importance of managerial and interpersonal skills for IT workers.^{13,18,21}

Data and Methods

Our primary data comes from two

or “personal” services to describe tasks requiring customer contact or physical presence and “impersonal” services to describe tasks requiring neither of these characteristics. Higher values on this scale indicate workers in these jobs provide fewer personally delivered services, or “impersonal” jobs, and are therefore more likely to be offshored, all else being equal.

We also included additional variables in our regressions to control for other factors that might affect an employee’s chances of being displaced due to offshoring. Since the relative benefit of offshoring a particular worker depends on the cost of the worker to the firm, we included a measure of employee salary (coded in discrete levels). Employees are less likely to be displaced if they have more firm-specific knowledge or experience with the firm. Though we did not have access to organizational tenure variables, we included the individual’s job level, coded in discrete levels. We included demographic variables for employees, as there is evidence that such factors as race, age, and gender influence displacement; see Kletzer¹² for a review of the job-displacement literature. We also included the number of employees at the firm to control for firm size, as well as a dummy variable indicating the industry in which the firm competes. Finally, in some regressions, we also included the state within the U.S. in which the firm operated in 2007, to control for regional differences.

Results

Here, we present some statistics and results from the regression analyses aiming to identify the factors that most affect offshoring:

Employer statistics. Table 1 reports the overall incidence of offshoring by industry in 2007. The proportion of firms that reported offshoring any type of work across all industries was 15.2%. However, within technology-services and telecommunications industries, over 40% of firms in the sample reported offshoring some type of work. The hypothesis that offshoring is more common in high-tech industries than in other industries is significant at the $p < .01$ level ($X^2(1)=100.5$).

The figure here shows that offshoring rates vary significantly by job type. Over 30% of respondents reported offshoring computer programmers and software developers, but only about half of them, or 15.5%, reported offshoring systems analysts. About 24% of employers offshore customer service, and a smaller percentage (less than 10%) offshore management, sales, and marketing functions. A test of the hypothesis that employers offshore IT workers more than other types of workers is significant at the .01 level ($X^2(1)=86.6$). Among IT workers, the hypothesis that computer programmers and software developers are offshored in greater numbers

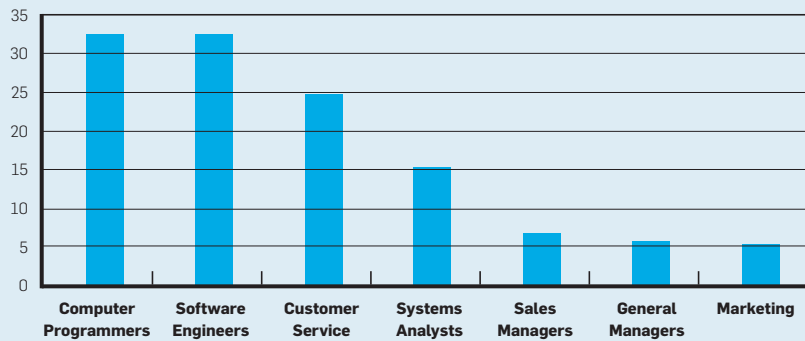
than systems analysts is significant at the $p < .01$ level ($X^2(1)=30.9$).

Employers also reported offshoring different types of work for very different reasons. A test of the hypothesis that occupation and reason for offshoring are independent is rejected at the $p < .01$ level ($X^2(36)=165.2$). Table 2 lists correlations between type of work and reasons for offshoring it. Jobs involving close interaction with the markets they serve (such as management and sales) are offshored for quality reasons in firms that are expanding geographically, while firms appear to offshore computer and technical work and customer service jobs primarily for cost savings and for ac-

Table 1. Percent of surveyed firms by industry reporting offshoring work.

Industry	N	Offshore to Third Party	Offshore to Foreign Affiliate	Total Offshore
Technology services	126	35.7	29.4	42.1
Telecommunications	43	37.2	16.3	41.9
Insurance	93	32.3	11.8	32.3
Manufacturing	248	24.6	19	31.1
Engineering services	82	22	17.1	28.1
Banking and finance	135	22.2	11.1	24.4
Oil	18	16.7	11.1	22.2
Travel	33	21.2	9.1	21.2
Utilities	29	17.2	6.9	20.7
Communications	34	20.6	8.8	20.6
Advertising/Marketing	41	14.6	9.8	17.1
Research services	35	17.1	0	17.1
Transportation and warehousing	74	13.5	5.4	16.2
Administrative-support services	56	14.3	8.9	16.1
Automotive services	45	8.9	8.9	15.6
Wholesale trade	66	13.6	6.1	15.2
Arts, entertainment, recreation	61	11.5	1.6	13.1
Agricultural/Forestry/Fishing/Hunting	25	12	0	12.0
Printing trade	26	11.5	3.9	11.5
Other	383	9.7	4.4	11.2
Retail trade	269	10	3	11.2
Construction	86	7	4.7	9.3
Other services	110	7.3	6.4	9.1
Waste Management and Remediation Services	12	8.3	0	8.3
Legal services	52	5.8	0	5.8
Accommodation and Food services	125	4	2.4	5.6
Health Care and Social Assistance	381	5.2	0.8	5.5
Real estate	49	4.1	0	4.1
Religious/nonprofit	139	2.9	2.2	3.6
Education	123	3.3	0.8	3.3
Gas	13	0	0	0.0
Mining	4	0	0	0.0
Total	3,016	13.1	7.0	15.2

Percent of firms reporting offshoring by worker type.



Types of Worker Being Offshored	Computer Programmer	Software Engineers	Customer Service	Systems Analysts	Sales Managers	General Managers	Marketing
	31.4%	31.4%	24.9%	15.5%	7.6%	6.1%	5.5%

N=458

for IT work, and, of all the countries in our sample, offshoring to India was most associated with cost savings.

Employee statistics. The findings from the employer data are supported by statistics from the employee surveys. Table 4 includes the percentage of workers, by occupation, who reported having been displaced due to offshoring. Across all occupations, slightly over 4% of workers reported having been displaced due to offshoring. Of occupations with at least 100 observations in the sample, engineers, machine operators, and IT workers reported the highest rates of offshoring-related job displacement. Of the five occupations with the highest displacement rates, all but machine operators were technology-related. Furthermore, unlike computer jobs, which in 2007 were increasing as a proportion of employment worldwide, machine-operator employment numbers have declined, suffering from unusually high displacement rates.^{10,16} These results support the common perception that U.S. IT workers have experienced higher rates of offshoring-related displacement than other U.S. workers.

Table 5 compares the displacement frequency of IT workers with that of all other types of worker. At about 8%, IT workers were (in 2007) displaced at twice the rate of other workers. A test of the hypothesis that displacement rates differ between IT workers and non-IT workers is significant at the .01 level ($X^2=27.5$, $p<.01$). However, because these numbers reflect the percentage of workers who have ever been offshored, an 8% displacement rate implies an annual average offshoring-related displacement rate of about 1% for IT workers, assuming that many U.S. firms began offshoring in 2000.^a Surveys conducted 1995–2005 suggest average IT staff turnover rates vary from 10% to 15%.^{7,8}

Among occupations with at least

^a We computed this average estimate by dividing the total displacement rate by the number of years since 2000, because many companies viewed the potential Y2K problem as a trigger. The annual displacement rate is slightly less if firms engaged in substantial offshoring before 2000. Due to data limitations, we can say little about how the actual displacement rate changed from year to year 2000–2007.

Table 2. Correlations between occupation and reasons for offshoring.

Occupation	Reason for Offshoring					Total
	Cost Savings	Skills	Service Quality	Expansion	Other	
Computer programmer	.24**	.14**	-.07	.03	-.09	144
Software developer	.22**	.14**	-.01	.03	-.13**	144
Systems analyst	.15**	.13**	.14**	.03	-.11*	71
Customer service	.14**	.05	.11*	.00	-.16**	114
Graphic designer	.04	.08	.13**	.08	-.11*	35
General manager	.01	.15**	.12**	.26**	-.09	28
Sales manager	-.06	.10*	.15**	.29**	-.08	35
Marketing personnel	-.08	.17**	.22**	.27**	-.11*	25
HR personnel	.07	.03	.10*	.15**	-.12**	31
Total (Percentage)	63.7	26.9	13.9	19.1	18.7	

Reported correlations are partial correlations between whether or not a firm offshores a position and the reasons it gives for firms report for offshoring, controlling for firm size and industry, * significant at the 5% level or greater, ** significant at the 1% level or greater. For all correlations, N=458.

cess to skills. These correlations suggest that offshoring may have more direct implications for U.S.-based IT workers than the offshoring of other types of workers.

Table 3 further supports these ideas, showing firms' offshoring destinations, as well as correlations between destination and type of work being offshored. A test of the hypothesis that occupation and offshoring destination are independent is rejected at the $p<.01$ level ($X^2(171)=298.2$),

indicating that particular types of work are best suited for offshoring only to certain countries. IT work and customer-service work appear to be much more concentrated than sales, management, and marketing, which are spread over a larger number of countries, consistent with the idea that jobs involving personally delivered services are often co-located with overseas customers. In 2007, India was the most popular destination for offshoring any type of work, especially

100 observations, we observed the lowest displacement rates among sales representatives and nurses. Moreover, a number of occupations with fewer observations (such as real estate agents, veterinarians, professors, and religious professionals) reported no offshoring displacement. In addition to providing a basis for comparison with the IT worker population, data from these other occupations provides preliminary support for the hypothesis that employees providing more personally delivered services are generally less vulnerable to offshoring than occupations that need not be in a fixed location (such as computer programmers and machine operators).

Occupational Attributes

We provide a more rigorous test of the hypothesis that “impersonal” jobs are more vulnerable to offshoring. Table 6 reports the results of some regressions using the survey data. The unit of observation in the regression is the employer-occupation offshoring combination, taking a value of 1 if the employer reported offshoring a particular occupation and 0 otherwise. Column (1) lists the results from a regression that includes all firms in our sample. Employers were less likely to offshore jobs in which employees provide personal services ($t=9.0$). The size of the employer ($t=2.0$) and the local cost of doing business ($t=3.5$) for the employer both significantly increase the probability that it will offshore a particular job. Column (2) lists results from a regression that includes only the firms that reported offshoring work in 2007. The extent to which an employee provided personal services is still significantly and negatively associated with whether a job was offshored ($t=14.0$).

Column (3) adds a covariate indicating whether the employer was expanding geographically, along with an interaction term between geographic expansion and personal interaction. Our estimates indicate that firms expanding geographically are more likely to hire offshore workers ($t=6.71$). Furthermore, geographic expansion moderates the type of work being offshored, consistent with the hypothesis that provision of personal

services must be co-located with the markets being served. If a firm’s customers are all located in the U.S., it will offshore jobs that do not require

personal interaction with the U.S. market. However, if a firm does business in overseas markets, employers will also hire offshore workers who

Table 3. Correlations between offshoring destinations and type of work.

Country	Total	Type of Position Offshored					Reason
		Customer					Wages
		IT	Service	Marketing	Management	Sales	
India	236	.43**	.20**	-.14**	-.08	-.16**	.40**
China	128	-.08	-.03	.03	.05	.05	.11**
U.S. Territories	74	-.06	.02	.18**	.16**	.15**	-.30**
Mexico	63	-.08	.04	.10	.05	.05	.09*
Canada	47	.01	.11	.22**	.26**	.04	-.01
Other	43	-.06	-.04	-.07	.03	-.05	-.17**
Germany	41	.04	.05	.20**	.26**	.15**	-.09*
Philippines	37	.08	.20**	.01	.03	-.02	.09*
United Kingdom	37	-.06	.09	.22**	.20**	.24**	-.15**
France	33	-.01	.03	.23**	.17**	.10	-.13**
Brazil	31	-.00	.03	.27**	.17**	.18	-.09*
Argentina	23	.00	.06	.13**	.07	.09	-.01
Italy	23	.02	.08	.29**	.22**	.19**	-.07
Japan	23	.01	.07	.15**	.18**	.20**	-.07
Australia	22	.02	.07	.26**	.29**	.20**	-.06
Other Europe	21	-.03	.02	.00	.14**	.07	-.07
Poland	20	.02	.11	.01	.05	.03	.05
Russia	20	.09	.07	.11**	.05	-.01	-.04
Taiwan	20	.05	.00	.11**	.10*	.08	.06

** p<.01 *p<.05, N=458.

For “Type of Position,” we report correlations between whether a firm reports offshoring to a country and whether it reports offshoring a particular worker type. For “Reason,” we report correlations between whether a firm reports offshoring to a country and whether wages are a principal reason it did so.

Table 4. Worker displacement levels by occupation.

Occupation	N	Displaced
Engineer	180	10.00%
Machine Operator/Assembly	359	8.36%
IT Manager/Network Administrator	123	8.13%
Other Computer or Internet Specialty	248	7.26%
Engineering Technician	222	6.76%
Other Health Care Professional	265	5.28%
Maintenance/Mechanic/Repair Worker	141	4.96%
Sales Representative, Retail	270	4.07%
Food Preparation/Service Worker	150	4.00%
Other Profession	1,431	3.91%
Administrative Assistant/Secretary	725	3.45%
Other Financial Professional	281	2.85%
Transportation/Equipment Operator	213	2.82%
Sales Representative, Other	334	2.40%
Nurse, Nurse Practitioner, or Physician's Assistant	305	0.98%
Total	5,247	5.0%

^aTable includes only occupations with at least 100 samples in the survey.

Table 5. Offshoring-related displacement rates for IT and non-IT workers.

	Non-IT Workers	Displaced
Not Displaced	5,704	712
Displaced	227	61
	3.8%	7.9%

A chi-squared test of the hypothesis of equality between displacement averages is rejected at the $p < .01$ level $X^2(1) = 27.5$, $p < .01$.

can provide personal services directly to overseas customers.

Table 7 reports the results of the primary regressions from the survey data relating the personal services provided in one's occupation to the likelihood of offshoring-related displacement. The results in column (1) support the hypothesis that employment in a job providing personal services significantly decreases the likelihood of being displaced due to offshoring ($t = 3.5$). Somewhat surprisingly, the coefficient estimate on salary level is negative and significant, suggesting workers with higher salaries are less likely to be offshored ($t = 2.09$). However, in the absence of human-capital data, the salary term in the regressions also reflects human-capital variables (such as education and experience). We therefore interpret the negative coefficient as indicating that, conditional on job level, workers with more human capital are less likely to be offshored, an effect that dominates any direct gains from offshoring more expensive workers. The results also suggest that older workers ($t = 4.9$), males ($t = 2.21$), and workers in simpler jobs ($t = 2.15$) requiring less firm-specific capital are likely to be offshored.

After including industry dummies in column (2), the coefficient estimate on gender is no longer significant, indicating our earlier estimate on gender may have reflected high offshoring intensity in such industries as IT with a higher fraction of men and low offshoring intensity in such industries as health care with a higher fraction of women. However, the estimates on the other coefficients remain significant. Dummy variables for state and race in column (3) do not

significantly alter the coefficient estimates on any other variable.

Column (4) shows the marginal effects of the estimates from our baseline regression in column (1) where all variables are standardized so effect sizes are comparable. A one standard deviation increase in our personal-services index measure decreases the probability of offshoring-related displacement by about 1%, a 25% increase over the U.S. national base rate of 4% in 2007. The effect of a one standard deviation decrease in our personal-services index appears to be similar in magnitude to a one standard deviation increase in age, also increasing the likelihood of offshoring-related displacement by slightly over 1%. Older workers who do not provide personal services are thus particularly vulnerable to offshoring-related displacement.

The sample in column (5) is restricted to IT workers. The measure on personal interaction is insignificant because there is little variation in this index within the IT workers in our surveyed population. Of the remain-

ing variables, only age is significant, suggesting that among IT workers, older workers are at the greatest risk of offshoring-related displacement ($t = 3.10$).

Table 8 explores how the level of personal interaction in a prior job affects outcomes for workers after being displaced due to offshoring. Employees in occupations providing fewer personal services are more likely to be separated from their employers, while those providing more personal services are more likely to be retained for other positions ($t = 2.0$). This suggests that firms might retain and move workers with interpersonal or management skills not as easy to source globally. Column (2) shows how much change in the personal-delivery measure affects retention. A one standard deviation increase in the personal-skills index increases the chance of being retained by a firm by about 6%.

Discussion

Although about 15% of firms in the U.S. offshored in 2007, firms in high-

Table 6. Probit analysis, employer offshoring.

	All Employers	Only Employers That Offshore	Only Employers That Offshore
	Probit Estimates	Probit Estimates	Probit Estimates
Employer offshores job type	(1)	(2)	(3)
Impersonal ^a	.009 (.001)**	.014 (.001)**	.019 (.002)**
Number of Employees	.002 (.001)**	.006 (.002)**	.006 (.002)**
Local Cost of Doing Business	.007 (.002)**	-.001 (.002)	-.001 (.002)
Geographic Expansion			1.55 (.231)**
Geographic Expansion * Impersonal			-.015 (.003)**
Controls	Industry	Industry	Industry
Pseudo-R ²	.11	.07	.09
N	26,568	4,041	4,041

Huber-White standard errors are clustered on firm and shown in parentheses. ** $p < .01$

^a Composite skill index taken from Blinder.⁴ Higher values indicate that less face-to-face contact or physical presence are needed for job.

Column (1) is a probit regression of employer and job characteristics against the likelihood the employer offshores a particular type of job.

Column (2) is a probit regression of employer and job characteristics against the likelihood the employer offshores a particular type of job, only for employers are offshoring.

Column (3) is similar to Column (2) but includes variables related to employer expansion plans.

tech industries offshored at rates higher than 40%, and IT work was the most commonly offshored type of work. IT workers in the U.S. have experienced offshoring-related displacement at a rate of 8%, more than double the percentage in other occupations. Firms offshore for a number of reasons, but IT workers appear to be offshored primarily for cost or access to skills. Therefore, compared to sales workers offshored to provide customer contact to overseas markets, the offshoring of IT workers should lead to greater displacement of U.S.-based IT workers. Our results also provide empirical support for the hypothesis proposed in earlier work⁴ that employees in jobs requiring face-to-face contact or physical presence in a fixed location are less likely to be offshored. This suggests that IT workers are especially vulnerable to offshoring because IT jobs generally require less customer contact or interaction with fixed physical assets.

Our estimates imply an average displacement rate of about 1% per year

for U.S.-based IT workers. However, as offshoring grows more popular, our findings, which suggest that workers who do not provide personal services are being displaced at a higher rate, are consistent with emerging work providing evidence for a potentially significant long-term shift in the relative demand for skills within the IT labor market.²³ These results suggest that technical occupations reliant on skills that can be delivered with relatively little face-to-face contact are more easily offshored. Other scholars have noted that interpersonal or managerial skills are increasingly valuable for IT workers,^{13,17} so our findings suggest that offshoring will continue to drive a secular increase in the direction of this trend. IT workers concerned about offshoring-related displacement may find more robust career paths in IT professions that require personal delivery.

These results also have policy implications. First, the relatively low level of offshoring suggests any policy prescription for addressing the

adverse consequences of offshoring should be concerned with the potential growth of offshoring rather than the existing level of offshoring-related displacement. Annual rates of offshoring-related displacement in the survey were on the order of 10% of aggregate IT-worker turnover. While unclear at what level offshoring shifts from a trend affecting mostly individual workers to a concern for all workers in an occupation, the trends should be measured and monitored.

Proposed policy interventions attempting to reduce the adverse effects of worker displacement (such as worker retraining and government compensation to offset wage losses associated with moving to new industries) could focus on specific occupations. Furthermore, training programs could focus on the movement of displaced workers toward work that combines existing skills with those that involve elements of personal delivery. Private or public educational institutions can potentially adjust their curricula to address

Table 7. Probit analysis, worker displacement.

Probit Displacement from Offshoring	All Workers	All Workers	All Workers	All Workers	IT Workers Only
	Probit Estimates	Probit Estimates	Probit Estimates	Marginal Effects	Probit Estimates
	(1)	(2)	(3)	(4)	(5)
Impersonal ^a	.007 (.002)**	.004 (.002)*	.004 (.002)**	.011 (.002)**	.000 (.004)
Job Level	-.103 (.048)*	-.095 (.049)*	-.092 (.050)*	-.006 (.003)**	-.231 (.172)
Salary	-.048 (.023)*	-.059 (.024)**	-.056 (.025)*	-.130 (.059)**	-.021 (.058)
Male?	.137 (.062)*	.066 (.065)	.070 (.067)	.006 (.003)**	-.138 (.183)
Age	.197 (.040)**	.194 (.041)**	.195 (.043)**	.014 (.003)**	.319 (.103)**
Number of Employees	-.001 (.001)	-.001 (.001)	-.000 (.001)	-.002 (.003)	.003 (.003)
Controls		Industry	Industry State Race		
Pseudo-R ²	.03	.05	.06	.06	.04
N	5,790	5,790	5,790	5,471	672

Standard errors are in parentheses, **p<.01 *p<.05

^a Composite skill index from Blinder.⁴ Higher values indicate less face-to-face contact or physical presence needed for job.

Table 8. Probit analysis of outcomes for all displaced workers.

Separated from Firm	Probit Estimates	Marginal Effects
	(1)	(2)
Impersonal ^a	.012 (.006)**	.060 (.029)**
Job Level	-.180 (.180)	-.038 (.038)
Salary	.097 (.087)	.863 (.772)
Male	-.357 (.217)	-.050 (.030)
Age	-.193 (.145)	-.044 (.032)
Controls	Industry	Industry
Pseudo-R ²	.09	.09
N	222	222

Standard errors in parentheses. ** p<.01

^a Composite skill index taken from Blinder.⁴ Higher values indicate less face-to-face contact or physical presence needed for job.


N=222.57 employees retained by their employers after job displacement.

this emerging need. Our findings are consistent with broader calls from education scholars who have advocated (in response to recent waves of technological change) emphasizing “softer” skills (such as complex communication)¹⁴ in the U.S. educational system; for example, educators could interweave existing material in the IT curriculum with projects that promote teamwork, negotiation, and presentation skills.

In the future, this area of research would benefit from improved offshoring data, including more fine-grain measures of the task content of individual jobs. Data at the task level would allow researchers to test more nuanced models of which attributes make a job vulnerable to offshoring (such as those considering the modularity, codifiability, or information intensity of a worker’s task set). These tests would also provide insight into how jobs and educational programs can be designed so U.S.-based workers maximize the value they provide to the global economy. Furthermore, although our survey data was unique because it allowed us to capture fine-grain outcomes, a limitation of the data was its reliance on self-reported responses from employees and hiring managers that might be subject to bias. Evidence from other data sources could therefore be useful in validating these results.

Finally, although our study focused on job displacement, offshoring may also affect workers through reduced wages. A more comprehensive understanding of the full effects of offshoring on IT workers and the demand for particular skills could therefore be provided through analyses of job displacement and wage effects.

Acknowledgments

We thank Peter Cappelli, Eric Clemmons, Lori Rosenkopf, and three anonymous reviewers for their guidance and comments. 

References

1. Amiti, M. and Wei, S. Fear of service outsourcing: Is it justified? *Economic Policy* 20, (Apr. 2005), 308–348.
2. Apte, U. and Mason, R. Global disaggregation of information-intensive services. *Management Science* 41, 7 (July 1995), 1250–1262.
3. Aspray, W., Mayadas, F., and Vardi, M. *Globalization and Offshoring of Software: A Report of the ACM Job Migration Task Force*. ACM Press, New York, 2006.
4. Blinder, A. *How Many U.S. Jobs Might Be*



Over 30% of respondents reported offshoring computer programmers and software developers, but only about half of them, or 15.5%, reported offshoring systems analysts.



Offshorable? CEPS Working Paper No. 142, (Mar. 2007).

5. Blinder, A. Offshoring: The next industrial revolution. *Foreign Affairs* 85, 2 (Mar./Apr. 2006), 113–128.
6. Carmel, E. and Tjia, P. *Offshoring Information Technology: Sourcing and Outsourcing to a Global Workforce*. Cambridge University Press, Cambridge, England, 2005.
7. Ferrat, T., Agarwal, R., Brown, C., and Moore, J.E. IT human resource management configurations and IT turnover: Theoretical synthesis and empirical analysis. *Information Systems Research* 16, 3 (Sept. 2005), 237–255.
8. George, T. Report: IT turnover high despite concerns over job stability. *InformationWeek* (May 6, 2002); <http://www.informationweek.com/news/global-cio/compensation/showArticle.jhtml?articleID=6502404>
9. Government Accountability Office. *Current Government Data Provide Limited Insight into Offshoring of Services*. GAO-04-932. Washington D.C., 2004.
10. Herz, D. Worker displacement still common in the late 1980s. *Monthly Labor Review* 114 (May 1991), 3–9.
11. Jensen, J. and Kletzer, L. *Measuring Tradable Services and the Task Content of Offshorable Services Jobs*. Working Paper, 2007.
12. Kletzer, L. Job displacement. *Journal of Economic Perspectives* 12, 1 (Winter 1998), 115–136.
13. Lee, D.M.S., Trauth, E., and Farwell, D. Critical skills and knowledge requirements of IS professionals: A joint academic/industry investigation. *MIS Quarterly* 19, 3 (Sept. 1995), 313–340.
14. Levy, F. and Murnane, R. *Teaching the New Basic Skills: Principles for Educating Children to Thrive in a Changing Economy*. Free Press, New York, 1996.
15. Mann, C. What global sourcing means for U.S. IT workers and for the U.S. economy. *Commun. ACM* 47, 7 (July 2004), 33–35.
16. Mark, J. Technological change and employment: Some results from BLS research. *Monthly Labor Review* 110 (1987), 26–29.
17. Mithas, S. and Krishnan, M.S. Human capital and institutional effects in the compensation of information technology professionals in the United States. *Management Science* 54, 3 (Mar. 2008), 415–428.
18. Mithas, S. and Whitaker, J. Is the world flat or spiky? Information intensity, skills, and global disaggregation. *Information Systems Research* 18, 3 (Sept. 2007), 237–259.
19. Panko, R. IT employment prospects: Beyond the dotcom bubble. *European Journal of Information Systems* 17 (2008), 182–197.
20. Phillips, D. and Clancy, K. Some effects of ‘social desirability’ in survey studies. *The American Journal of Sociology* 77, 5 (Mar. 1972), 921–940.
21. Ramasubbu, N., Mithas, S., and Krishnan, M.S. High tech, high touch: The effect of employee skills and customer heterogeneity on customer satisfaction with enterprise system support services. *Decision Support Systems* 44, 2 (Jan. 2008), 509–523.
22. Slaughter, S. and Ang, S. Employment outsourcing in information systems. *Commun. ACM* 39, 7 (July 1996), 47–54.
23. Tambe, P. and Hitt, L. *Now I.T.’s ‘Personal’: Offshoring and the Shifting Skill Composition of the U.S. Information Technology Workforce*. Working Paper, 2010.
24. Zwieg, P., Kaiser, K., Beath, C. et al. The information technology workforce: Trends and implications 2005–2008. *MIS Quarterly Executive* (2006); <http://misqe.org/ojs2/index.php/misqe/article/view/104>

Prasanna B. Tambe (ptambe@stern.nyu.edu) is an assistant professor of information, operations, and management sciences in the Stern School of Business, New York University, New York.

Lorin M. Hitt (lhitt@wharton.upenn.edu) is the Class of 1942 Professor of Operations and Information Management in The Wharton School, University of Pennsylvania, Philadelphia, PA.



Group Term Life Insurance

10- or 20-Year Group Term
Life Insurance

Group Disability Income Insurance

Group Accidental Death &
Dismemberment Insurance

Group Catastrophic Major
Medical Insurance

Group Dental Plan

Long-Term Care Plan

Major Medical Insurance

Short-Term Medical Plan

Who has time to think about insurance?

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

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

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

Call Marsh U.S. Consumer, a service of Seabury & Smith, Inc., at 1-800-503-9230 or visit www.personal-plans.com/promo/acm/49771.

49771 (2010) ©Seabury & Smith, Inc. 2010

Administered by:

MARSH



MARSH MERCER KROLL
GUY CARPENTER OLIVER WYMAN

d/b/a in CA Seabury & Smith Insurance Program Management
CA Ins. Lic. #0633005
AR Ins. Lic. #245544

Within a decade, P2P has proven to be a technology that enables innovative new services and is used by millions of people every day.

BY RODRIGO RODRIGUES AND PETER DRUSCHEL

Peer-to-Peer Systems

PEER-TO-PEER (P2P) COMPUTING has attracted significant interest in recent years, originally sparked by the release of three influential systems in 1999: the Napster music-sharing system, the Freenet anonymous data store, and the SETI@home volunteer-based scientific computing projects. Napster, for instance, allowed its users to download music directly from each other's computers via the Internet. Because the bandwidth-intensive music downloads occurred directly between users' computers, Napster avoided significant operating costs and was able to offer its service to millions of users for free. Though unresolved legal issues ultimately sealed Napster's fate, the idea of cooperative resource sharing among peers found its way into many other applications.

More than a decade later, P2P technology has gone far beyond music sharing, anonymous data storage, or scientific computing; it now enjoys significant research attention and increasingly widespread use in open software communities and industry alike. Scientists, companies, and open-software

organizations use BitTorrent to distribute bulk data such as software updates, data sets, and media files to many nodes;⁵ commercial P2P software allows enterprises to distribute news and events to their employees and customers;²⁹ millions of people use Skype to make video and phone calls;¹ and hundreds of TV channels are available using live streaming applications such as PPLive,¹⁷ CoolStreaming,³⁸ and the BBC's iPlayer.⁴

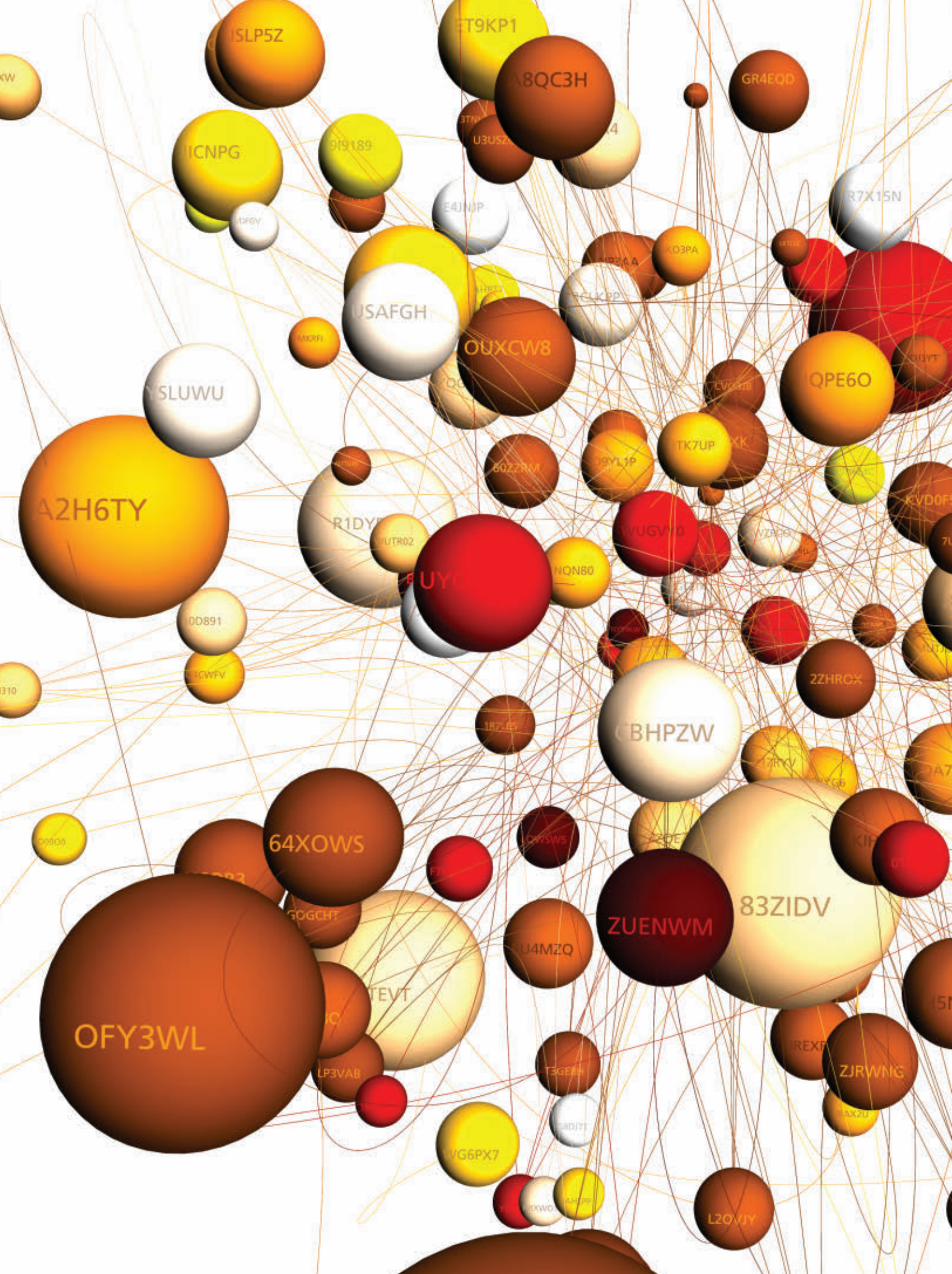
The term P2P has been defined in different ways, so we should clarify what exactly we mean by a P2P system. For the purposes of this article, a P2P system is a distributed system with the following properties:

High degree of decentralization. The peers implement both client and server functionality and most of the system's state and tasks are dynamically allocated among the peers. There are few if any dedicated nodes with centralized state. As a result, the bulk of the computation, bandwidth, and storage needed to operate the system are contributed by participating nodes.

Self-organization. Once a node is introduced into the system (typically by providing it with the IP address of a participating node and any necessary

>> key insights

- P2P leverages the computing resources of cooperating users to achieve scalability and organic growth, thus lowering the deployment barrier for innovative new services.
- Originally invented for music/data sharing and volunteer computing, P2P systems now enjoy widespread commercial and non-commercial use in content distribution, IPTV, and IP telephony.
- The strength of P2P—its independence of dedicated infrastructure and centralized control—is also its weakness, as it presents new technical, commercial, and legal challenges.
- P2P technology may turn out to be most valuable as a low-cost deployment vector for experimental, innovative services; those services that prove to be commercially viable can be subsequently combined with centralized, infrastructure-based components.



key material), little or no manual configuration is needed to maintain the system.

Multiple administrative domains.

The participating nodes are not owned and controlled by a single organization. In general, each node is owned and operated by an independent individual who voluntarily joins the system.

P2P systems have several distinctive characteristics that make them interesting:

Low barrier to deployment. Because P2P systems require little or no dedicated infrastructure, the upfront investment needed to deploy a P2P service tends to be low when compared to client-server systems.

Organic growth. Because the resources are contributed by participating nodes, a P2P system can grow almost arbitrarily without requiring a “fork-lift upgrade” of existing infrastructure, for example, the replacement of a server with more powerful hardware.

Resilience to faults and attacks. P2P systems tend to be resilient to faults because there are few if any nodes that are critical to the system’s operation. To attack or shut down a P2P system, an attacker must target a large proportion of the nodes simultaneously.

Abundance and diversity of resources. Popular P2P systems have an abundance of resources that few organizations would be able to afford individually. The resources tend to be diverse in terms of their hardware and software architecture, network attachment, power supply, geographic location and jurisdiction. This diversity reduces their vulnerability to correlated failure, attack, and even censorship.

As with other technologies (for example, cryptography), the properties of P2P systems lend themselves to desirable and undesirable use. For instance, P2P systems’ resilience may help citizens avoid censorship by a totalitarian regime; at the same time, it can be abused to try and hide criminal activity from law enforcement agencies. The scalability of a P2P system can be used to disseminate a critical software update efficiently at a planetary scale, but can also be used to facilitate the illegal distribution of copyrighted content.

Despite having acquired a negative reputation for some of its initial pur-

poses, P2P technologies are increasingly being used for legal applications with enormous business potential, and there is consensus about their ability to lower the barrier for the introduction of innovative technologies. Nevertheless, P2P technology faces many challenges. The decentralized nature of P2P systems raises concerns about manageability, security, and law enforcement. Moreover, P2P applications are affecting the traffic experienced by Internet service providers (ISPs) and threaten to disrupt the current Internet economics. In this article, we briefly sketch important highlights of the technology, its applications, and the challenges it faces.

Applications

Here, we discuss some of the most successful P2P systems and also mention promising P2P systems that have not yet received as much attention.

Sharing and distributing files. Presently, the most popular P2P applications are file sharing (for example, eDonkey) and bulk data distribution (for example, BitTorrent).

Both types of systems can be viewed as successors of Napster. In Napster, users shared a subset of their disk files with other participants, who were able to search for keywords in the file names. Users would then download any of the files in the query results directly from the peer that shared it.

Much of the content shared by Napster users was music, which led to copyright infringement lawsuits. Napster was found guilty and had to shut down its services. Simultaneously, a series of similar P2P systems appeared, most notably Gnutella and FastTrack (better known by one of its client applications, Kazaa). Gnutella, unlike Napster, has no centralized components and is not operated by any single entity (perhaps in part to make it harder to prosecute).

The desire to reduce the download time for very large files led to the design of BitTorrent,¹⁰ which enables a large set of users to download bulk data quickly and efficiently. The system uses spare upload bandwidth of concurrent downloaders and peers who already have the complete file (either because they are data sources or have finished the download) to assist other downloaders in the system. Unlike file-shar-

ing applications, BitTorrent and other P2P content distribution networks do not include a search component, and users downloading different content are unaware of each other, since they form separate networks. The protocol is widely used for disseminating data, software, or media content.

Streaming media. An increasingly popular P2P application is streaming media distribution and IPTV (delivering digital television service over the Internet). As in file sharing, the idea is to leverage the bandwidth of participating clients to avoid the bandwidth costs of server-based solutions.

Streaming media distribution has stricter timing requirements than downloading bulk data because data must be delivered before the playout deadline to be useful.

Example systems include academic efforts with widespread adoption such as PPLive¹⁷ and CoolStreaming,³⁸ and commercial products such as BBC’s iPlayer⁴ and Skinkers LiveStation.²⁹

Telephony. Another major use of P2P technology on the Internet is for making audio and video calls, popularized by the Skype application. Skype exploits the resources of participating nodes to provide seamless audiovisual connectivity to its users, regardless of their current location or type of Internet connection. Peers assist those without publicly routable IP addresses to establish connections, thus working around connectivity problems due to firewalls and network address translation, without requiring a centralized infrastructure that handles and forwards calls. Skype reported 520 million registered users at the end of 2009.

Volunteer computing. A fourth important P2P application is volunteer computing. In these systems, users donate their spare CPU cycles to scientific computations, usually in fields such as astrophysics, biology, or climatology. The first system of this type was SETI@home. Volunteers install a screen saver that runs the P2P application when the user is not active. This application downloads blocks containing observational data collected at the Arecibo radio telescope from the SETI@home server. Then the application analyzes this data, searching for possible radio transmissions, and sends the results back to the server.


The success of SETI@home and similar projects led to the development of the BOINC platform,³ which has been used to develop many cycle-sharing P2P systems in use today. At the time of this writing, BOINC has more than half a million active peers computing on average 5.42 petaFLOPS (floating-point operations per second). For comparison, a modern PC performs on the order of a few tens of GFLOPS (about five orders of magnitude fewer), and the world's fastest supercomputer as of August 2010 has a performance of about 1.76 petaFLOPS.

Other applications. Other types of P2P applications have seen significant use, at least temporarily, but have not reached the same levels of adoption as the systems we describe here. Among them are applications that leverage peer-contributed disk space to provide distributed storage. Freenet⁹ aims to combine distributed storage with content distribution, censorship resistance, and anonymity. It is still active, but the properties of the system make it difficult to estimate its actual use. MojoNation³⁶ was a subsequent project for building a reliable P2P storage system, but it was shut down after proving unable to ensure the availability of data due to unstable membership and other problems.


P2P Web content distribution networks (CDNs) such as CoralCDN¹⁶ and CoDeeN³⁵ were deployed as research prototypes but gained widespread use. In these systems, a set of cooperating users form a network of Web caches and name servers that replicates Web content as users access it, thereby reducing the load on servers hosting popular content. During its peak usage, CoralCDN received up to 25 million hits per day from one million unique IP addresses.

Many more P2P systems have been designed and prototyped, but either were not deployed publicly or had small deployments. Examples include systems for distributed data monitoring, management and mining,^{26,37} massively distributed query processing,¹⁹ cooperative backup,¹¹ bibliographic databases,³³ serverless email,²⁴ and archival storage.²³

Technology developed for P2P applications has also been incorporated into other types of systems. For in-



While the earliest and most visible P2P systems were mainly file-sharing applications, current uses of P2P technology are much more diverse and include the distribution of data, software, media content, as well as Internet telephony and scientific computing.



stance, Dynamo,¹³ a storage substrate that Amazon uses internally for many of its services and applications, uses distributed hash tables (DHTs), which we will explain later. Akamai's NetSession^a client uses P2P downloads to increase performance and reduce the cost of delivering streaming content. Even though these systems are controlled by a single organization and thus do not strictly satisfy our definition of a P2P system, they are based on P2P technology.

While P2P systems are a recent invention, technical predecessors of P2P systems have existed for a long time. Early examples include the NNTP and SMTP news and mail distribution systems, and the Internet routing system. Like P2P systems, these are mostly decentralized systems that rely on resource contributions from their participants. However, the peers in these systems are organizations and the protocols are not self-organizing.

While the earliest and most visible P2P systems were mainly file-sharing applications, current uses of P2P technology are much more diverse and include the distribution of data, software, media content, as well as Internet telephony and scientific computing. Moreover, an increasing number of commercial services and products rely on P2P technology.

How Do P2P Systems Work?

Here, we sketch some of the most important techniques that make P2P systems work. We discuss fundamental architectural choices like the degree of centralization and the structure of the overlay network. As you will see, one of the key challenges is to build an overlay with a routing capability that works well in the presence of a high membership turnover (usually referred to as churn), which is typical of deployed P2P system.²⁸ We then present solutions to specific problems addressed in the context of P2P systems: application state maintenance, application-level node coordination, and content distribution.

Note that our intention in this presentation is to provide representative

^a See Akamai NetSession Interface Overview at http://www.akamai.com/html/misc/akamai_client/netsession_interface.html/.

examples of the most interesting techniques rather than try to be exhaustive or precise about a particular system or protocol.

Degree of centralization. We can broadly categorize the architecture of P2P systems according to the presence or absence of centralized components in the system design.

Partly centralized P2P systems have a dedicated controller node that maintains the set of participating nodes and controls the system. For instance, Napster had a Web site that maintained the membership and a content index; early versions of BitTorrent have a “tracker,” which is a node that keeps track of the set of nodes uploading and downloading the same content, and periodically provides nodes with a set of peers they can connect to;¹⁰ the BOINC platform

for volunteer computing has a site that maintains the membership and assigns compute tasks;³ and Skype has a central site that provides log-in, account management, and payment.

Resource-intensive operations like transmitting content or computing application functions do not involve the controller. Like general P2P systems, partly centralized P2P systems can provide organic growth and abundant resources. However, they do not necessarily offer the same scalability and resilience because the controller forms a potential bottleneck and a single point of failure and attack. Partly centralized P2P systems are relatively simple and can be managed by a single organization via the controller.

Decentralized P2P system. In a decentralized P2P system, there are no

dedicated nodes that are critical for the operation of the system. Decentralized P2P systems have no inherent bottlenecks and can potentially scale very well. Moreover, the lack of dedicated nodes makes them potentially resilient to failure, attack, and legal challenge.

In some decentralized P2P systems, nodes with plenty of resources, high availability and a publicly routable IP address act as supernodes. These supernodes have additional responsibilities, such as acting as a rendez-vous point for nodes behind firewalls, storing state or keeping an index of available content. Supernodes can increase the efficiency of a P2P system, but may also increase its vulnerability to node failure.

Overlay maintenance. P2P systems maintain an overlay network, which can be thought of as a directed graph $G = (N, E)$, where N is the set of participating computers and E is a set of *overlay links*. A pair of nodes connected by a link in E is aware of each other’s IP address and communicates directly via the Internet. Here, we discuss how different types of P2P systems maintain their overlay.

In partly centralized P2P systems, new nodes join the overlay by connecting to the controller located at a well-known domain name or IP address (which can be, for instance, hardcoded in the application). Thus, the overlay initially has a star-shaped topology with the controller at the center. Additional overlay links may be formed dynamically among participants that have been introduced by the controller.

In decentralized overlays, newly joining nodes are expected to obtain, through an outside channel, the network address (for example, IP address and port number) of some node that already participates in the system. The address of such a bootstrap node can be obtained, for instance, from a Web site. To join, the new node contacts the bootstrap node.

We distinguish between systems that maintain an unstructured or a structured overlay network.

Unstructured overlays. In an unstructured P2P system, there are no constraints on the links between different nodes, and therefore the overlay graph does not have any particular structure. In a typical unstructured P2P system,

Figure 1. An example KBR implementation.

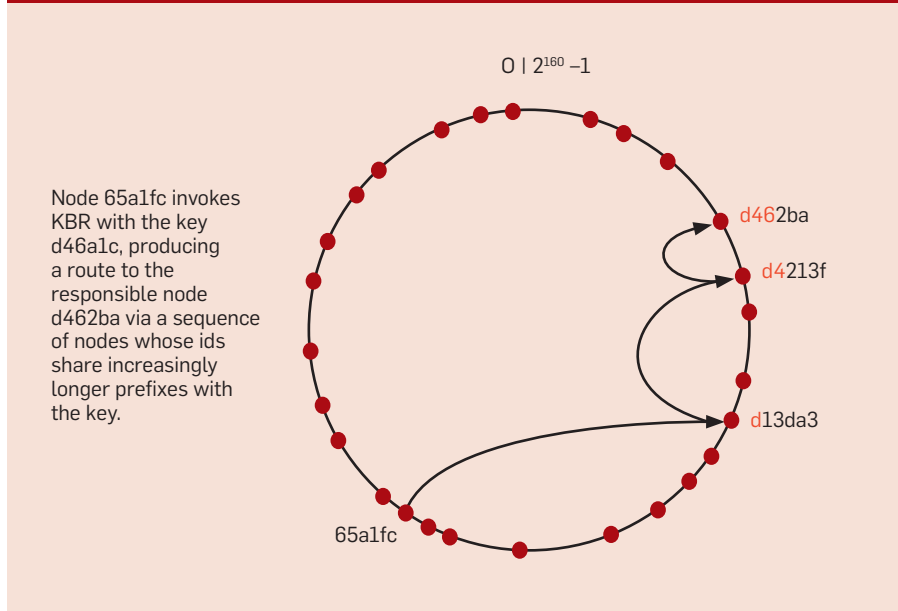
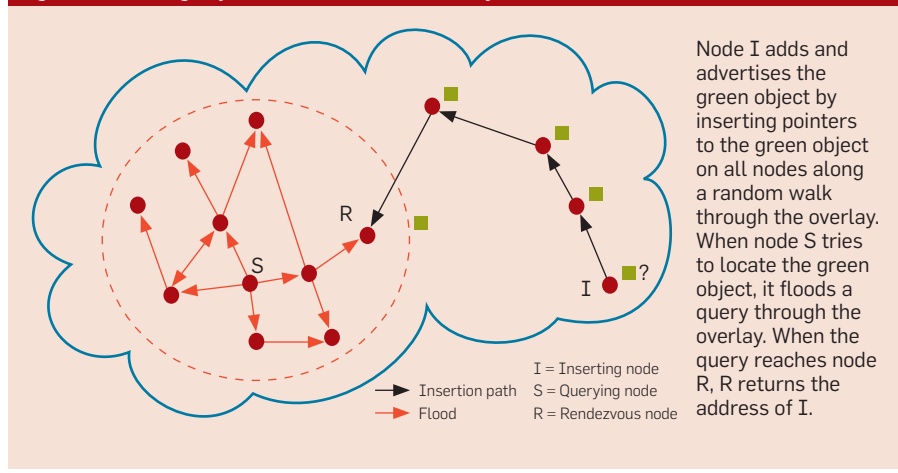


Figure 2. Locating objects in unstructured overlays.



a newly joining node forms its initial links by repeatedly performing a random walk through the overlay starting at the bootstrap node and requesting a link to the node where the walk terminates. Nodes acquire additional links (for example, by performing more random walks) whenever their degree falls below the desired minimum; they refuse link requests when their current degree is at its maximum.

The minimum node degree is typically chosen to maintain connectivity in the overlay despite node failures and membership churn. A maximum degree is maintained to bound the overhead associated with maintaining overlay links.

Structured overlays. In a structured overlay, each node has a unique identifier in a large numeric key space, for example, the set of 160-bit integers. Identifiers are chosen in a way that makes them uniformly distributed in that space. The overlay graph has a specific structure; a node's identifier determines its position within that structure and constrains its set of overlay links.

Keys are also used when assigning responsibilities to nodes. The key space is divided among the participating nodes, such that each key is mapped to exactly one of the current overlay nodes via a simple function. For instance, a key may be mapped to the node whose identifier is the key's closest counterclockwise successor in the key space. In this technique the key space is considered to be circular (that is, the id zero succeeds the highest id value) to account for the fact that there may exist keys greater than all node identifiers.

The overlay graph structure is chosen to enable efficient key-based routing. Key-based routing implements the primitive $KBR(n_0, k)$. Given a starting node n_0 and a key k , KBR produces a path, that is, a sequence of overlay nodes that ends in the node responsible for k . As will become clear in subsequent sections, KBR is a powerful primitive.

Many implementations of key-based routing exist.^{18,27,32} In general, they strike a balance between the amount of routing state required at each node and the number of forwarding hops required to deliver a message. Typical implementations require an

amount of per-node state and a number of forwarding hops that are both logarithmic in the size of the network.

Figure 1 illustrates an example of a key-based routing scheme. Node 65a1fc invokes KBR with the key d46a1c, producing a route via a sequence of nodes whose ids share increasingly longer prefixes with the key. Eventually the message reaches the node with id d462ba, which has sufficient knowledge about its neighboring nodes to determine that it is responsible for the target key. Though not depicted, the reply can be forwarded directly to the invoking node.

Summary. We have seen how the overlay network is formed and maintained in different types of P2P systems. In partly centralized P2P systems, the controller facilitates the overlay formation.

In other P2P systems, overlay maintenance is fully decentralized. Compared to an unstructured overlay network, a structured overlay network invests additional resources to maintain a specific graph structure. In return, structured overlays are able to perform key-based routing efficiently.

The choice between an unstructured and a structured overlay depends on how useful key-based routing is for the application, and also on the frequency of overlay membership events. As we will discuss, key-based routing can reliably and efficiently locate uniquely identified data items and maintain spanning trees among member nodes. However, maintaining a structured overlay in a high-churn environment has an associated cost, which may not be worth paying if the application does not require the functionality provided by key-based routing.

Some P2P systems use both structured and unstructured overlays. A recent ("trackerless") version of BitTorrent, for instance, uses key-based routing to choose tracker nodes, but builds an unstructured overlay to disseminate the content.

Distributed state. Most P2P systems maintain some application-specific distributed state. Without loss of generality, we consider that state as a collection of objects with unique keys. Maintaining this collection of state objects in a distributed manner, that is, providing mechanisms for object

placement and locating objects, are key tasks in such systems.

Partly centralized systems. In partly centralized P2P systems, an object is typically stored at the node that inserted the object, as well as any nodes that have subsequently downloaded the object. The controller node maintains information about which objects exist in the system, their keys, names and other attributes, and which nodes are currently storing those objects. Queries for a given key, or a set of keywords that match an object's name or attributes, are directed to the controller, which responds with a set of nodes from which the corresponding object(s) can be downloaded.

Unstructured systems. As in partly centralized systems, content is typically stored at the node that introduced the content to the system, and replicated at other downloaders. To make it easier to find content, some systems place copies of (or pointers to) an inserted object on additional nodes, for instance, along a random walk path through the overlay.

To locate an object, a querying node typically floods a request message through the overlay. The query can specify the desired object by its key, metadata, or keywords. A node that receives a query and has a matching object (or a pointer to a matching object), responds to the querying node. Figure 2 illustrates this process. In this case, node I inserts an object into the system and holds its only copy, but inserts pointers to the object on all nodes along a random walk that ends in node R . When node S tries to locate the object, it floods a query, first, to all nodes that are at a distance of one hop, then to all nodes two hops away. In the last step the query reaches node R , which returns the address of I .

Often, the scope of the flood (that is, the maximal number of hops from the querying nodes that a flood message is forwarded) is limited to trade recall (the probability that an object that exists in the system is found) for overhead (the number of messages required by the flood). An alternative to flooding is for the querying node to send a request message along a random walk through the overlay.

Gnutella was the first example of a decentralized, unstructured network

that used flooding to locate content in a file sharing system.

Structured overlays. In structured overlays, distributed state is maintained using a *distributed hash table* (DHT) abstraction. The DHT has the same put/get interface as a conventional hash table. Inserted key/value pairs are distributed among the participating nodes in the structured overlay using a simple placement function. For instance, that function can position replicas of the key/value pair on the set of r nodes whose identifiers succeed the key in the circular key space. Note that in our terminology, the values correspond to the state objects main-

tained by the system.

Given this replica placement policy, the DHT's put and get operations can be implemented using the KBR primitive in a straightforward manner. To insert (put) a key/value pair, we use the KBR primitive to determine the responsible node for the key k and store the pair on that node, which then propagates it to the set of replicas for k . To look up (get) a value, we use the KBR primitive to fetch the value associated with a given key. The responsible node can respond to the fetch request or forward it to one of the nodes in the replica set. Figure 3 shows an example put operation, where the value is initially

pushed to the node responsible for key k , which is discovered using KBR, and this node pushes the value to its three immediate successors.

When a DHT experiences churn, pairs have to be moved between nodes as the mapping of keys to nodes changes. To minimize the required network communication, large data values are typically not inserted directly into a DHT; instead, an indirection pointer is inserted under the value's key, which points to the node that actually stores the value.

DHTs are used, for instance, in file sharing networks such as eDonkey, and also in some versions of BitTorrent.

Summary. Unstructured overlays tend to be very efficient at locating widely replicated objects, while KBR-based techniques can reliably and efficiently locate any object that exists in the system, no matter how rare it may be. Put another way, unstructured overlays are good at finding “hay” while structured overlays are good at finding “needles.” On the other hand, unstructured networks support arbitrary keyword-based queries, while KBR-based systems directly support only key-based queries.

Distributed coordination. Frequently, a group of nodes in a P2P application must coordinate their actions without centralized control. For instance, the set of nodes that replicate a particular object must inform each other of updates to the object. In another example, a node that is interested in receiving a particular streaming content channel may wish to find, among the nodes that currently receive that channel, one that is nearby and has available upstream network bandwidth. We will look at two distinct approaches to this problem: epidemic techniques where information spreads virally through the system, and tree-based techniques where distribution trees are formed to spread the information.

We focus only on decentralized overlays, since coordination can be accomplished by the controller node in partly centralized systems.

Unstructured overlays. In unstructured overlays, coordination typically relies on epidemic techniques. In these protocols, information is spread through the overlay in a manner simi-

Figure 3. Inserting a value into a DHT.

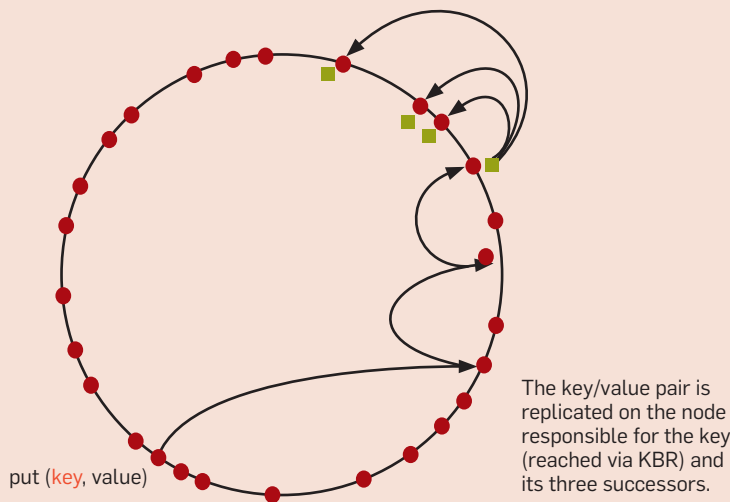
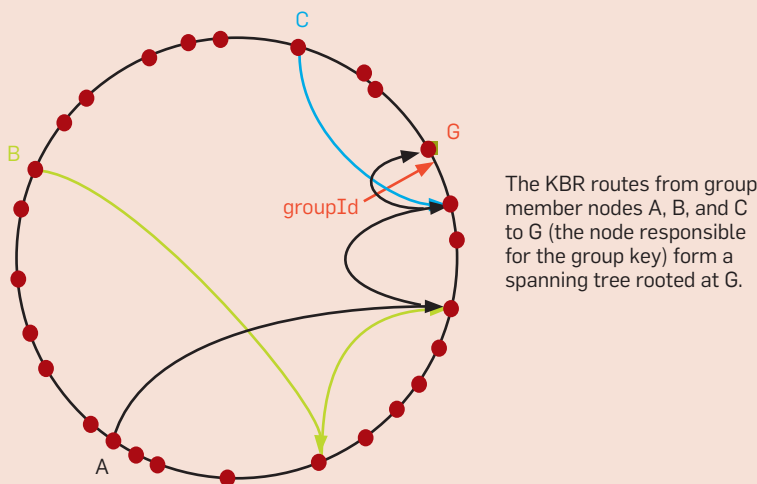


Figure 4. An example KBR tree.



lar to the way an infection spreads in a population: the node that produced the information sends it to (some of) its overlay neighbors, who send it to (some of) their neighbors, and so on. This method of dissemination is very simple and robust. As in all epidemic techniques, there is a trade-off between the speed of information dissemination and overhead. Moreover, if a given piece of information is of interest only to a subset of nodes and these nodes are widely dispersed within the overlay, then the information ends up being needlessly delivered to all nodes.

A more efficient way to coordinate the actions among a group of nodes is to form a spanning tree among the nodes. The spanning tree is embedded in the overlay graph, using a decentralized algorithm for spanning tree formation. This tree can then be used to multicast messages to all members, or to compute summaries (for example, sums, averages, minima, or maxima) of state variables within the group. However, this added coordination efficiency must be balanced against the overhead of maintaining the spanning tree in the unstructured overlay network.

Structured overlays. In structured overlays, spanning trees among any group of overlay nodes can be formed and maintained very efficiently using the KBR primitive, making trees the preferred method of coordination in these overlays. To join a spanning tree, a node uses KBR to route to a unique key associated with the group. The resulting union of the paths from all group members form a spanning tree rooted at the node responsible for the group's key. This *KBR tree* is then used to aggregate and disseminate state associated with the group, and to implement multicast and anycast. Figure 4 illustrates an example KBR tree formed by the union of the KBR routes from nodes *A*, *B*, and *C* to the key corresponding to the group id. This tree is rooted at node *G*, which is the responsible node for that key.

Because a join message terminates as soon as it intercepts the tree, group membership maintenance is decentralized, that is, the arrival or departure of a node is noted only by the node's parent and children in the tree. As a result, the technique scales to large numbers of groups, as well as large and



Unstructured overlays are good at finding “hay,” while structured overlays are good at finding “needles.”



highly dynamic groups.

Summary. The epidemic techniques typically used for coordination in unstructured overlays are simple and robust to overlay churn, but they may not scale to large overlays or large numbers of groups, and information tends to propagate slowly. Spanning trees can increase the efficiency of coordination, but maintaining a spanning tree in an unstructured overlay adds costs.

The additional overhead for maintaining a *structured* overlay is proportional to the churn in the total overlay membership. Once that overhead is paid, KBR trees enable efficient and fast coordination among potentially numerous, large and dynamic subgroups within the overlay.

Content Distribution

Another common task in P2P systems is the distribution of bulk data or streaming content to a set of interested nodes. P2P techniques for content distribution can be categorized as tree-based (where fixed distribution trees are formed either with the aid of a structured overlay or embedded in an unstructured overlay), or swarming protocols (which have no notion of a fixed tree for routing content and usually form an unstructured overlay). Due to space constraints, we focus on the swarming protocols popularized by the BitTorrent protocol.¹⁰

In swarming protocols, the content is divided into a sequence of blocks, and each block is individually multicast to all overlay nodes such that different blocks are disseminated along different paths.

The basic operation of a swarming protocol is simple: once every swarming interval (say, one second), overlay neighbors exchange information indicating which content blocks they have available. (In streaming content distribution, only the most recently published blocks are normally of interest.) Each node intersects the availability information received from its neighbors, and then requests a block it does not already have from one of the neighbors who has it.

It is important that blocks are well distributed among the peers, to ensure neighboring peers tend to have blocks they can swap and that blocks remain available when some peers leave the

system. To achieve such a distribution, the system can randomize both the choice of block to download and the choice of a neighbor from whom to request the block. In one possible strategy, a node chooses to download the rarest block among all blocks held by its overlay neighbors.¹⁰

The best known and original swarming protocol for bulk content distribution is BitTorrent.¹⁰ Examples of swarming protocols used for streaming content include PPLive¹⁷ and the original version of CoolStreaming.³⁸

Challenges

Much of the promise of P2P systems stems from their independence of dedicated infrastructure and centralized control. However, these very properties also expose P2P systems to some unique challenges not faced by other types of distributed systems. Moreover, given the popularity of P2P systems, they become natural targets for misuse or attack. Here, we give an overview of challenges and attacks that P2P systems may face, and corresponding defense techniques. As you will see, some of the issues have been addressed to varying degrees, and others remain open questions.

Controlling membership. Most P2P systems have open or loosely controlled membership. This lack of strong user identities allows an attacker to populate a P2P system with nodes under his control, by creating many distinct identities (such action was termed a *Sybil attack*¹⁵). Once he controls a large number of “virtual” peers, an attacker can defeat many kinds of defenses against node failure or misbehavior, for example, those that rely on replication or voting. For instance, an attacker who wishes to suppress the value associated with some key k from a DHT can add virtual nodes to the system until he controls all of the nodes that store replicas of the value. These nodes can then deny the existence of that key/value pair when a get operation for key k is issued.

Initial proposals to address Sybil attacks required proof of work (for example, solving a cryptographic puzzle or downloading a large file) before a new node could join the overlay.^{15,34} While these approaches limit the rate at which an attacker can obtain iden-



Much of the promise of P2P systems stems from their independence of dedicated infrastructure and centralized control. However, these very properties also expose P2P systems to some unique challenges not faced by other types of distributed systems.



ties, they also make it more difficult for legitimate users to join. Moreover, an attacker with enough resources or access to a botnet can still mount Sybil attacks.

Another solution requires certified identities,⁷ where a trusted authority vouches for the correspondence between a peer identity and the corresponding real-world entity. The disadvantage of certified identities is that a trusted authority and the necessary registration process may be impractical or inappropriate in some applications.

Protecting data. Another aspect of P2P system robustness is the availability, durability, integrity, and authenticity of the data stored in the system or downloaded by a peer. Different types of P2P systems have devised different mechanisms to address these problems.

Integrity and authenticity. In the case of DHTs, data integrity is commonly verified using *self-certifying* named objects. DHTs take advantage of the fact that they have flexibility in the choice of the keys for values stored in the DHT. By setting $key=hash(value)$ during the put operation, the downloader can verify the retrieved data is correct by applying the cryptographic hash function to the result of the get operation and comparing it to the original key. Systems that store mutable data and systems that allow users to choose arbitrary names for inserted content can instead use cryptographic signatures to protect the integrity and authenticity of the data. However, such systems require an infrastructure to manage the cryptographic keys.

Studies show that systems that do not protect the integrity of inserted data (including many file sharing systems) tend to be rife with mislabeled or corrupted content.^{8,22} One possible approach to counter the problem of content pollution is for peers to vote on the authenticity of data. For example, a voting system called Credence was developed by researchers and used by several thousands of peers in the Gnutella file sharing network.³⁴ However, the problem remains challenging given the possibility of Sybil attacks to defeat the voting.

Availability and durability. The next challenge is how to ensure the avail-

ability and durability of data stored in a P2P system. Even in the absence of attacks, ensuring availability can prove difficult due to churn. For a data object to be available, at least one node that stores a replica must be online at all times. To make sure an object remains available under churn, a system must constantly move replicas to live nodes, which can require significant network bandwidth. For this reason, a practical P2P storage system cannot simultaneously achieve all three goals of scalable storage, high availability, and resilience to churn.⁶

Another challenge is that the long-term membership of a P2P storage system (that is, the set of nodes that periodically come online) must be non-decreasing to ensure the durability of stored data. Otherwise, the system may lose data permanently, since the storage space available among the remaining members may fall below that required to store all the data.

Incentives. Participants in a P2P system are expected to contribute resources for the common good of all peers. However, users don't necessarily have an incentive to contribute if they can access the service for free. Such users, called free riders, may wish to save their own disk space, bandwidth, and compute cycles, or they may prefer not to contribute any content in a file-sharing system.

Free riding is reportedly widespread in many P2P systems. For instance, in 2000 and 2001, studies of the Gnutella system found a large fraction of free riders.^{2,28} More recently, a study of a DHT used in the eMule file-sharing system found large clusters of peers (with more than 10,000 nodes) that had modified their client software to produce the same node identifier for all nodes, which means these nodes are not responsible for any keys.³¹

The presence of many free riders reduces the resources available to a P2P system, and can deteriorate the quality of the service the system is able to provide to its users. To address this issue, incentive schemes have been incorporated in the design of P2P systems.

BitTorrent uses a tit-for-tat strategy, where to be able to download a file from a peer, a peer must upload another part of the same file in return, or risk being disconnected from that peer.¹⁰

This provides a strong incentive for users to share their upload bandwidth, since a peer that does not upload data will have poor download performance. A number of other incentive mechanisms have been proposed, which all try to tie the quality of the service a peer receives to how much that peer contributes.^{12,25}

Managing P2P systems. Whether P2P systems are easier to manage than other distributed systems is an open question.

On the one hand, P2P systems adapt to a wide range of conditions with respect to workload and resource availability, they automatically recover from most node failures, and participating users look after their hardware independently. As a result, the burden associated with the day-to-day operation of P2P systems appears to be low compared to server-based solutions, as evidenced by the fact that graduate students have been able to deploy and manage P2P systems that attract millions of users.¹⁶

On the other hand, there is evidence that P2P systems can experience widespread disruptions that are difficult to manage. For instance, on Aug. 16, 2007, the Skype overlay network collapsed and remained unavailable for several days. The problem was reportedly triggered by a Microsoft Windows Update patch that caused many of the peers to reboot around the same time, causing a lack of resources that, combined with a software bug, prevented the overlay from recovering.³⁰ This type of problem may indicate the lack of centralized control over available resources and participating nodes makes it difficult to manage systemwide disruptions when they occur. However, more research and long-term practical experience with deployed systems is needed to settle this question.

Some of the challenges P2P systems face (for example, data integrity and authenticity) are largely solved, while others (for example, membership control and incentives) have partial solutions that are sufficient for important applications. However, some problems remain wide open (for example, data durability and management issues). Progress on these problems may be necessary to further expand the range

of applications of P2P technology.

Peer-to-Peer and ISPs

Internet service providers have witnessed the success of P2P applications with mixed feelings. On one hand, P2P is fueling demand for network bandwidth. Indeed, P2P accounts for the majority of bytes transferred on the Internet.²⁹ On the other hand, P2P traffic patterns are challenging certain assumptions that ISPs have made when engineering their networks and when pricing their services.

To understand this tension, we must consider the Internet's structure and pricing. The Internet is a roughly hierarchical conglomeration of independent network providers. Local ISPs typically connect to regional ISPs, who in turn connect to (inter-)national backbone providers. ISPs at the same level of the hierarchy (so-called peer ISPs) may also exchange traffic directly. In particular, the backbone providers are fully interconnected.

Typically, peer ISPs do not charge each other for traffic they exchange directly, but customers pay for the bits they send to their providers. An exception is residential Internet connections that are usually offered at a flat rate by ISPs.

This pricing model originated at a time when client-server applications dominated the traffic in the Internet. Commercial server operators pay their ISPs for the bandwidth used, who in turn pay their respective providers. Since residential customers rarely operate servers (in fact, their terms of use do not allow them to operate commercial servers), it was reasonable to assume they generate little upstream traffic, keeping costs low for local ISPs and enabling them to offer flat-rate pricing.

With P2P content distribution applications, however, residential P2P nodes upload content to each other. Unless the P2P nodes happen to connect to the same ISP or to two ISPs that peer directly with each other, the uploading node's ISP must forward the data to its own provider. This incurs costs that the ISP cannot pass on to its flat-rate customers.²⁰ As a result of this tension, some ISPs have started to traffic shape and even block BitTorrent traffic.¹⁴ Whether network operators

should be required to disclose such practices, and if they should be allowed at all to discriminate among different traffic types is the subject of an ongoing debate.

Independent of the outcome of this debate, the tension will have to be resolved in a way that allows P2P applications to thrive while ensuring the profitability of ISPs. A promising technical approach is to bias the peer selection in P2P applications toward nodes connected to the same ISP or to ISPs that peer with each other.²⁰ Another solution is for ISPs to change their pricing model.

A more fundamental tension is that some ISPs view many of the currently deployed P2P applications as competing with their own value-added services. For instance, ISPs that offer conventional telephone service may view P2P VoIP applications as competition, and cable ISP may view P2P IPTV applications as competing with their own IPTV services. In either case, such ISP's market share in the more profitable value-added services is potentially diminished in favor of carrying more plain bits.

In the long term, however, ISPs will likely benefit, directly and indirectly, from the innovation and emergence of new services that P2P systems enable. Moreover, ISPs may find new revenue sources by offering infrastructure support for successful services that initially developed as P2P applications.

Conclusion

In this article, we have sketched the promise, technology, and challenges of P2P systems. As a disruptive technology, P2P creates significant opportunities and challenges for the Internet, industry, and society. Arguably the most significant promise of P2P technology lies in its ability to significantly lower the barrier for innovation. But the great strength of P2P, its independence of dedicated infrastructure and centralized control, may also be its weakness, as it creates new challenges that must be dealt with through technical, commercial, and legal means.

One possible outcome is that P2P will turn out to be especially valuable as a proving ground for new ideas and services, in addition to keeping its role as a platform for grassroots services

that enable free speech and the unregulated exchange of information. Services that turn out to be popular, legal, and commercially viable may then be transformed into more infrastructure-based, commercial services. Here, ideas from P2P systems may be combined with traditional, centralized approaches to build highly scalable and dependable systems. ■

References

- About Skype: 100 Billion Skype-to-Skype Minutes Served; http://about.skype.com/2008/02/100_billion_skypetoskype_minut.html.
- Adar, E. and Huberman, B.A. Free riding on Gnutella. *First Monday* 5, 10 (Oct. 2000).
- Anderson, D.P. BOINC: A system for public-resource computing and storage. In *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing* (2004), 4–10.
- BBC News. One million viewers use iPlayer. <http://news.bbc.co.uk/2/hi/technology/7187967.stm>.
- Bittorrent (protocol). Wikipedia; [http://en.wikipedia.org/wiki/BitTorrent_\(protocol\)#Adoption](http://en.wikipedia.org/wiki/BitTorrent_(protocol)#Adoption).
- Blake, C. and Rodrigues, R. High availability, scalable storage, dynamic peer networks: Pick two. In *Proceedings of the 9th Workshop on Hot Topics in Operating Systems* (May 2003).
- Castro, M., Druschel, P., Ganesh, A., Rowstron, A. and Wallach, D.S. Security for structured peer-to-peer overlay networks. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation* (Dec. 2002).
- Christin, N., Weigend, A.S. and Chuang, J. Content availability, pollution and poisoning in file sharing peer-to-peer networks. In *Proceedings of the 6th ACM Conference on Electronic Commerce* (June 2005).
- Clarke, I., Sandberg, O., Wiley, B. and Hong, T.W. Freenet: A distributed anonymous information storage and retrieval system. In *Proceedings of the Designing Privacy Enhancing Technologies—International Workshop on Design Issues in Anonymity and Unobservability* (July 2000).
- Cohen, B. Incentives build robustness in BitTorrent. In *Proceedings of the 1st International Workshop on Economics of P2P Systems* (June 2003).
- Cox, L.P., Murray, C.D. and Noble, B.D. Pastiche: Making backup cheap and easy. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation* (Dec. 2002).
- Cox, L.P. and Noble, B.D. Samsara: honor among thieves in peer-to-peer storage. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles* (Oct. 2003).
- DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P. and Vogels, W. Dynamo: Amazon's highly available key-value store. In *Proceedings of the 21st ACM Symposium on Operating Systems Principles* (Oct. 2007).
- Dischinger, M., Mislove, A., Haebleren, A. and Gummadi, K.P. Detecting BitTorrent blocking. In *Proceedings of the 8th Internet Measurement Conference* (Oct. 2008).
- Douceur, J. The Sybil attack. In *Proceedings of the First International Workshop on Peer-to-Peer Systems* (Mar. 2002).
- Freedman, M.J., Freudenthal, E. and Mazières, D. Democratizing content publication with Coral. In *Proceedings of the 1st USENIX Symposium on Networked Systems Design and Implementation* (Mar. 2004).
- Hei, X., Liang, C., Liang, J., Liu, Y. and Ross, K.W. Insights into PPLive: A measurement study of a large-scale P2P IPTV system. In *Proceedings of the 15th International World Wide Web Conference, IPTV Workshop* (May 2006).
- Hildrum, K., Kubiatowicz, J.D., Rao, S. and Zhao, B.Y. Distributed object location in a dynamic network. In *Proceedings of the 14th Annual ACM Symposium on Parallel Algorithms and Architectures* (2002), 41–52.
- Huebsch, R., Hellerstein, J.M., Lanham, N., Loo, B.T., Shenker, S. and Stoica, I. Querying the Internet with PIER. In *Proceedings of the 29th International Conference on Very Large Data Bases* (Sept. 2003).
- Karagiannis, T., Rodriguez, P., and Papagiannaki, K. Should Internet service providers fear peer-assisted content distribution? In *Proceedings of the Internet Measurement Conference* (Oct. 2005).
- Li, B., Xie, S., Qu, Y., Keung, G., Lin, C., Liu, J. and Zhang, X. Inside the new coolstreaming: Principles, measurements and performance implications. In *Proceedings of INFOCOM* (2008).
- Liang, J., Kumar, R., Xi, Y. and Ross, K.W. Pollution in P2P file sharing systems. In *Proceedings of INFOCOM* (Mar. 2005).
- Maniatis, P., Roussopoulos, M., Giuli, T.J., Rosenthal, D.S.H. and Baker, M. The LOCKSS peer-to-peer digital preservation system. *ACM Transactions on Computer Systems* 23, 1 (2005), 2–50.
- Mislove, A. Post, A., Haebleren, A. and Druschel, P. Experiences in building and operating ePOST, a reliable peer-to-peer application. In *Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems* (Apr. 2006).
- Nandi, A., Ngan, T.-W.J., Singh, A., Druschel, P. and Wallach, D.S. Scrivener: Providing incentives in cooperative content distribution systems. In *Proceedings of the ACM/IFIP/USENIX 6th International Middleware Conference* (Nov. 2005).
- Renesse, R.V., Birman, K.P. and Vogels, W. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Transactions on Computer Systems* 21, 2 (2003), 164–206.
- Rowstron, A. and Druschel, P. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms* (Nov. 2001).
- Saroiu, S., Gummadi, P.K., and Gribble, S.D. A measurement study of peer-to-peer file sharing systems. In *Proceedings of the SPIE/ACM Conference on Multimedia Computing and Networking* (Jan. 2002).
- Skinkers: Enterprise communication management; http://www.skinkers.com/About_us/About_Skinkers.
- Skype: What happened on August 16; http://heartbeat.skype.com/2007/08/what_happened_on_august_16.html.
- Steiner, M., Biersack, E.W. and Ennajary, T. Actively monitoring peers in KAD. In *Proceedings of the 6th International Workshop on Peer-to-Peer Systems* (Feb. 2007).
- Stoica, I., Morris, R., Karger, D., Kaashoek, M.F. and Balakrishnan, H. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proceedings of SIGCOMM '01*, (Aug. 2001).
- Stribling, J. Li, J., Councill, I.G., Kaashoek, M.F. and Morris, R. Overcite: A distributed, cooperative citeseer. In *Proceedings of the 3rd Symposium on Networked Systems Design and Implementation* (May 2006).
- Walsh, K. and Sirer, E.G. Experience with an object reputation system for peer-to-peer file sharing. In *Proceedings of the 3rd Symposium on Networked Systems Design and Implementation* (May 2006).
- Wang, L., Park, K., Pang, R., Pai, V.S., and Peterson, L. Reliability and security in the CoDeeN content distribution network. In *Proceedings of the USENIX 2004 Annual Technical Conference* (June 2004).
- Wilcox-O'Hearn, B. Experiences deploying a large-scale emergent network. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems* (Mar. 2002).
- Yalagandula, P. and Dahlin, M. A scalable distributed information management system. In *Proceedings of SIGCOMM '04* (2004).
- Zhang, X., Liu, J., Li, B. and Yum, T.-S.P. CoolStreaming/DONet: A data-driven overlay network for peer-to-peer live media streaming. In *Proceedings of INFOCOM '05* (2005).

Rodrigo Rodrigues (rodrigo@mpi-sws.org) is a tenure-track faculty member at the Max Planck Institute for Software Systems (MPI-SWS), where he heads the dependable systems group.

Peter Druschel (druschel@mpi-sws.org) is the founding director of the Max Planck Institute for Software Systems (MPI-SWS), where he heads the distributed systems group.

© 2010 ACM 0001-0782/10/1000 \$10.00

research highlights

P. 84

**Technical
Perspective**
**A VM 'Engine' That
Makes a Difference**

By Carl Waldspurger

P. 85

**Difference Engine:
Harnessing Memory Redundancy
in Virtual Machines**

By Diwaker Gupta, Sangmin Lee, Michael Vrable, Stefan Savage,
Alex C. Snoeren, George Varghese, Geoffrey M. Voelker, and Amin Vahdat

P. 94

**Technical
Perspective**
Belief Propagation

By Yair Weiss and Judea Pearl

P. 95

**Nonparametric
Belief Propagation**

By Erik B. Sudderth, Alexander T. Ihler, Michael Isard,
William T. Freeman, and Alan S. Willsky

Technical Perspective

A VM ‘Engine’ That Makes a Difference

By Carl Waldspurger

THE PAST DECADE has witnessed a renaissance in server virtualization, which is transforming enterprise computing by offering new capabilities and efficiencies. The following paper by Diwaker Gupta et al. presents a novel approach for significantly improving the efficiency of virtualized servers. Their “Difference Engine” eliminates memory redundancy by exploiting similarity both within and across virtual machines.

A virtual machine (VM) is a software abstraction that behaves like hardware. The classic definition by Popek and Goldberg is “an efficient, isolated duplicate of a real machine.” For example, a VM that presents the illusion of being a physical x86 server may run an unmodified operating system designed for that platform, such as Windows or Linux. Neither the OS nor its users need be aware they are interacting with a VM instead of dedicated hardware.

Little more than a decade ago, virtual machines were considered a fairly exotic mainframe technology. Today, VMs are pervasive in corporate datacenters, and serve as the foundation for cloud-computing platforms. The commercial success of virtual machines has influenced the design of high-volume processor architectures, which now contain special-purpose hardware to accelerate virtualization.

Why have VMs proliferated so rapidly? One reason is that virtualization is an extremely versatile technology.

There is a well-known adage: “All problems in computer science can be solved by another level of indirection.” The virtualization software layer, known as a hypervisor, provides this level of indirection, decoupling an OS and its applications from physical hardware. Eliminating the traditional “one machine, one OS” constraint opens up numerous possibilities.

Initially, the most compelling use of VMs was basic partitioning and server consolidation. In typical unvirtualized environments, individual servers were

grossly underutilized. Virtualization allowed many servers to be consolidated as VMs onto a single physical machine, resulting in significantly lower capital and management costs. This ability to “do more with less” fueled the rapid adoption of virtualization, even through economic downturns.

As virtualization became more mainstream, innovations arose for managing distributed systems consisting of many virtualized servers. Since VMs are independent of the particular hardware on which they execute, they are inherently portable. Live, running VMs can migrate between different physical servers, enabling zero-downtime infrastructure maintenance, and supporting automated dynamic load balancing in production datacenters and clouds.

Additional virtualization features leverage indirection to offer capabilities beyond those of physical platforms. By interposing on VM operations transparently, no changes are required to the software running within the VM. Examples include improving security by adding checks that cannot be defeated by compromised software within the VM, and replicating VM state across physical machines for fault tolerance.

While core virtualization techniques are now reasonably mature, researchers continue to develop innovative ways to optimize VM efficiency and improve server utilization. Today, limited hardware memory often constrains the degree of server consolidation on modern machines equipped with many processor cores. The Difference Engine cleverly exploits the extra level of indirection in virtualized memory systems to reduce the memory footprint of VMs. Since higher consolidation ratios translate directly into cost savings, such techniques are incredibly valuable.


Due to consolidation, many VMs on the same physical machine typically run similar OS instances and applications, or contain common data. The Difference Engine extends the hyper-

visor with several mechanisms that reclaim memory by eliminating redundancy. First, when identical memory pages are found, they are deduplicated by retaining only a single instance that is shared copy-on-write, similar to the page-sharing feature that we introduced in VMware’s hypervisor.

However, the Difference Engine goes much further, taking advantage of deduplication opportunities that are left on the table when sharing is restricted to completely-identical pages. By observing that many more pages are nearly identical, sharing at sub-page granularity becomes very attractive. Candidates for sub-page sharing are identified by hashing small portions of pages, and patches are generated against reference pages to store near-duplicates compactly. When pages are not sufficiently similar, a conventional compression algorithm is applied to wring out any remaining intra-page redundancy.

By combining these mechanisms to eliminate full-page, sub-page, and intra-page redundancy, the Difference Engine achieves impressive space savings—more than twice as much as full-page sharing alone for VMs running disparate workloads. Of course, these savings aren’t free; compressed pages and sub-pages still incur page faults, and hashing, patching, and compression are compute-intensive operations.

But given current trends, it’s a safe bet that spare processor cycles will be easier to find than spare memory pages. The emergence of dense flash memory, phase-change memory, and other technologies will surely shift bottlenecks and trade-offs, ensuring this research area remains interesting.

Given the long history and extensive literature associated with both virtualization and memory management, it’s refreshing to find a paper that is both stimulating and practical. As virtual machines become increasingly ubiquitous, I’m confident that similar ideas will be leveraged by both commercial and research hypervisors. I strongly urge you to get a glimpse of this future now by reading this paper. 

Carl Waldspurger (carl@vmware.com) is a Principal Engineer at VMware, Palo Alto, where he oversees core resource management and virtualization technologies.

© 2010 ACM 0001-0782/10/1000 \$10.00

Difference Engine: Harnessing Memory Redundancy in Virtual Machines

By Diwaker Gupta, Sangmin Lee, Michael Vrible, Stefan Savage, Alex C. Snoeren, George Varghese, Geoffrey M. Voelker, and Amin Vahdat

Abstract

Virtual machine monitors (VMMs) are a popular platform for Internet hosting centers and cloud-based compute services. By multiplexing hardware resources among virtual machines (VMs) running commodity operating systems, VMMs decrease both the capital outlay and management overhead of hosting centers. Appropriate placement and migration policies can take advantage of statistical multiplexing to effectively utilize available processors. However, main memory is not amenable to such multiplexing and is often the primary bottleneck in achieving higher degrees of consolidation.

Previous efforts have shown that content-based page sharing provides modest decreases in the memory footprint of VMs running similar operating systems and applications. Our studies show that significant additional gains can be had by leveraging both subpage level sharing (through page patching) and incore memory compression. We build *Difference Engine*, an extension to the Xen VMM, to support each of these—in addition to standard copy-on-write full-page sharing—and demonstrate substantial savings across VMs running disparate workloads (up to 65%). In head-to-head memory-savings comparisons, *Difference Engine* outperforms VMware ESX server by a factor 1.6–2.5 for heterogeneous workloads. In all cases, the performance overhead of *Difference Engine* is less than 7%.

1. INTRODUCTION

Virtualization technology has improved dramatically over the past decade to become pervasive within the service-delivery industry. Virtual machines are particularly attractive for server consolidation. Their strong resource and fault isolation guarantees allow multiplexing of hardware among individual services, each with (potentially) distinct software configurations. Anecdotally, individual server machines often run at 5–10% CPU utilization. Operators' reasons are manifold: because of the need to over-provision for peak levels of demand, because fault isolation mandates that individual services run on individual machines, and because many services often run best on a particular operating system configuration. The promise of virtual machine technology for server consolidation is to run multiple services on a single physical machine while still allowing independent configuration and failure isolation.

While physical CPUs are frequently amenable to multiplexing, main memory is not. Many services run comfortably on a machine with 1GB of RAM; multiplexing 10 VMs on that same host, however, would allocate each just 100MB of RAM. Increasing a machine's physical memory is often both difficult and expensive. Incremental upgrades in memory capacity are subject to both the availability of extra slots on the motherboard and the ability to support higher-capacity modules: such upgrades often involve replacing—as opposed to just adding—memory chips. Moreover, not only is high-density memory expensive, it also consumes significant power. Furthermore, as many-core processors become the norm, the bottleneck for VM multiplexing will increasingly be the memory, not the CPU. Finally, both applications and operating systems are becoming more and more resource intensive over time. As a result, commodity operating systems require significant physical memory to avoid frequent paging.

Not surprisingly, researchers and commercial VM software vendors have developed techniques to decrease the memory requirements for virtual machines. Notably, the VMware ESX server implements content-based page sharing, which has been shown to reduce the memory footprint of multiple, homogeneous virtual machines by 10–40%.¹⁹ We find that these values depend greatly on the operating system and configuration of the guest VMs. We are not aware of any previously published sharing figures for mixed-OS ESX deployments. Our evaluation indicates, however, that the benefits of ESX-style page sharing decrease as the heterogeneity of the guest VMs increases, due in large part to the fact that page sharing requires the candidate pages to be *identical*.

The premise of this work is that there are significant additional benefits from sharing at a subpage granularity, i.e., there are many pages that are *nearly* identical. We show that it is possible to efficiently find such similar pages and coalesce them into a much smaller memory footprint. Among the set of similar pages, we are able to store most as *patches* relative to a single baseline page. We also compress

The original version of this paper is entitled “Difference Engine: Harnessing Memory Redundancy in Virtual Machines” and was presented at USENIX OSDI, December 2008. An extended abstract entitled “Difference Engine” appeared in USENIX *login*; volume 34, number 2.

those pages that are unlikely to be accessed in the near future; an efficient implementation of compression nicely complements page sharing and patching.

In this paper, we present Difference Engine, an extension to the Xen VMM⁵ that not only shares identical pages, but also supports subpage sharing and in-memory compression of infrequently accessed pages. Our results show that for a heterogeneous setup (different operating systems hosting different applications), Difference Engine can reduce memory usage by nearly 65%. In head-to-head comparisons against VMware's ESX server running the same workloads, Difference Engine delivers a factor of 1.5 more memory savings for a homogeneous workload and a factor of 1.6–2.5 more memory savings for heterogeneous workloads.

Critically, we demonstrate that these benefits can be obtained without negatively impacting application performance: in our experiments across a variety of workloads, Difference Engine imposes less than 7% overhead. We further show that Difference Engine can leverage improved memory efficiency to increase aggregate system performance by utilizing the free memory to create additional virtual machines in support of a target workload.

2. RELATED WORK

Difference Engine builds upon substantial previous work in page sharing, delta encoding, and memory compression. In each instance, we attempt to leverage existing approaches where appropriate.

2.1. Page sharing

Two common approaches in the literature for finding redundant pages are content-based page sharing, exemplified by VMware's ESX server,¹⁹ and explicitly tracking page changes to build knowledge of identical pages, exemplified by "transparent page sharing" in Disco.⁷ Transparent page sharing can be more efficient, but requires several modifications to the guest OS, in contrast to ESX server and Difference Engine which require no modifications.

To find sharing candidates, both Difference Engine and ESX hash contents of each page and use hash collisions to identify potential duplicates. Once shared, our system can manage page updates in a copy-on-write fashion, as in Disco and ESX server. We build upon earlier work on *flash cloning*¹⁸ of VMs, which allows new VMs to be cloned from an existing VM in milliseconds; as the newly created VM writes to its memory, it is given private copies of the shared pages. An extension by Kloster et al. studied page sharing in Xen¹¹ and we build upon this experience, adding support for fully virtualized (HVM) guests, integrating the global clock, and improving the overall reliability and performance.

2.2. Delta encoding

Our initial investigations into page similarity were inspired by research in leveraging similarity across files in large file systems. In GLIMPSE,¹⁴ Manber proposed computing Rabin fingerprints over fixed-size blocks at multiple offsets in a file. Similar files will then share some fingerprints. Thus the maximum number of common fingerprints is a strong indicator of similarity. However, in a dynamically evolving virtual

memory system, this approach does not scale well since every time a page changes its fingerprints must be recomputed as well. Further, it is inefficient to find the maximal intersecting set from among a large number of candidate pages. Broder adapted Manber's approach to the problem of identifying documents (in this case, Web pages) that are nearly identical using a combination of Rabin fingerprints and sampling based on minimum values under a set of random permutations.⁶

While these techniques can be used to identify similar files, they do not address how to efficiently encode the differences. Douglass and Iyengar explored using Rabin fingerprints and delta encoding to compress similar files in the DERD system,¹⁰ but only considered whole files. Kulkarni et al.¹² extended the DERD scheme to exploit similarity at the block level. Difference Engine also tries to exploit memory redundancy at several different granularities.

2.3. Memory compression

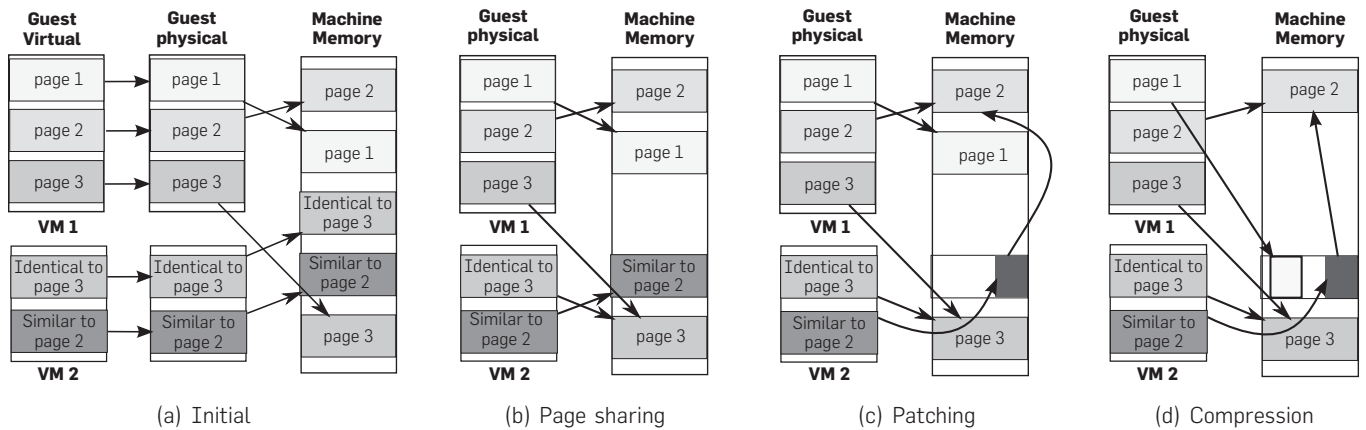
In-memory compression is not a new idea. Douglass et al.⁹ implemented memory compression in the Sprite operating system with mixed results. In their experience, memory compression was sometimes beneficial, but at other times the performance overhead outweighed the memory savings. Subsequently, Wilson et al. argued Douglass' mixed results were primarily due to slow hardware.²⁰ They also developed new compression algorithms (leveraged by Difference Engine) that exploit the inherent structure present in virtual memory, whereas earlier systems used general-purpose compression algorithms. Tudu et al.¹⁷ implemented a compressed cache for Linux that adaptively manages the amount of physical memory devoted to compressed pages using a simple algorithm shown to be effective across a wide variety of workloads.

3. ARCHITECTURE

Difference Engine uses three distinct mechanisms that work together to realize the benefits of memory sharing, as shown in Figure 1. In this example, two VMs have allocated five pages total, each initially backed by distinct pages in machine memory (Figure 1a). For brevity, we only show how the mapping from guest physical memory to machine memory changes; the guest virtual to guest physical mapping remains unaffected. First, for identical pages across the VMs, we store a single copy and create references that point to the original. In Figure 1b, one page in VM-2 is identical to one in VM-1. For pages that are similar, but not identical, we store a patch against a reference page and discard the redundant copy. In Figure 1c, the second page of VM-2 is stored as a patch to the second page of VM-1. Finally, for pages that are unique and infrequently accessed, we compress them in memory to save space. In Figure 1d, the remaining private page in VM-1 is compressed. The actual machine memory footprint is now less than three pages, down from five pages originally.

In all three cases, efficiency concerns require us to select candidate pages that are unlikely to be accessed in the near future. We employ a global clock that scans memory in the background, identifying pages that have not been recently

Figure 1. The three different memory conservation techniques employed by Difference Engine: page sharing, page patching, and compression. In this example, five physical pages are stored in less than three machine memory pages for a savings of roughly 50%.



used. In addition, reference pages for sharing or patching must be found quickly without introducing performance overhead. Difference Engine uses full-page hashes and hash-based fingerprints to identify good candidates. Finally, we implement a demand paging mechanism that supplements main memory by writing VM pages to disk to support overcommitment, allowing the total memory required for all VMs to temporarily exceed the physical memory capacity.

3.1. Page sharing

Difference Engine's implementation of content-based page sharing is similar to those in earlier systems. We walk through memory looking for identical pages. As we scan memory, we hash each page and index it based on its hash value. Identical pages hash to the same value and a collision indicates that a potential matching page has been found. We perform a byte-by-byte comparison to ensure that the pages are indeed identical before sharing them.

Upon identifying target pages for sharing, we reclaim one of the pages and update the virtual memory to point at the shared copy. Both mappings are marked read-only, so that writes to a shared page cause a page fault that will be trapped by the VMM. The VMM returns a private copy of the shared page to the faulting VM and updates the virtual memory mappings appropriately. If no VM refers to a shared page, the VMM reclaims it and returns it to the free memory pool.

3.2. Patching

Traditionally, the goal of page sharing has been to eliminate redundant copies of *identical* pages. Difference Engine considers further reducing the memory required to store *similar* pages by constructing patches that represent a page as the difference relative to a reference page. To motivate this design decision, we provide an initial study into the potential savings due to subpage sharing, both within and across virtual machines. First, we define the following two heterogeneous workloads, each involving three 512MB virtual machines:

- MIXED-1: Windows XP SP1 hosting RUBiS8; Debian 3.1 compiling the Linux kernel; Slackware 10.2 compiling Vim 7.0 followed by a run of the lmbench benchmark.¹⁵
- MIXED-2: Windows XP SP1 running Apache 2.2.8 hosting approximately 32,000 static Web pages crawled from Wikipedia, with httpperf running on a separate machine requesting these pages; Debian 3.1 running the SysBench database benchmark¹ using 10 threads to issue 100,000 requests; Slackware 10.2 running dbench² with 10 clients for 6 min followed by a run of the IOZone benchmark.³

We designed these workloads to stress the memory-saving mechanisms since opportunities for identical page sharing are reduced. In this first experiment, for a variety of configurations, we suspend the VMs after completing a benchmark, and consider a static snapshot of their memory to determine the number of pages required to store the images using various techniques. Table 1 shows the results of our analysis for the MIXED-1 workload.

The first column breaks down these 393,120 pages into three categories: 149,038 zero pages (i.e., the page contains all zeros), 52,436 sharable pages (the page is not all zeros, and there exists at least one other identical page), and 191,646 unique pages (no other page in memory is exactly the same). The second column shows the number of pages required to store these three categories of pages

Table 1. Effectiveness of page sharing across three 512MB VMs running Windows XP, Debian, and Slackware Linux using 4KB pages.

Pages	Initial	After Sharing	After Patching
Unique	191,646	191,646	
Sharable (non-zero)	52,436	3,577	
Zero	149,038	1	
Total	393,120	195,224	88,422
Reference		50,727	50,727
Patchable		144,497	37,695

using traditional page sharing. Each unique page must be preserved; however, we only need to store one copy of a set of identical pages. Hence, the 52,436 nonunique pages contain only 3,577 distinct pages—implying there are roughly 14 copies of every nonunique page. Furthermore, only one copy of the zero page is needed. In total, the 393,120 original pages can be represented by 195,224 distinct pages—a 50% savings.

The third column depicts the additional savings available if we consider subpage sharing. Using a cut-off of 2KB for the patch size (i.e., we do not create a patch if it will take up more than half a page), we identify 144,497 distinct pages eligible for patching. We store the 50,727 remaining pages as is and use them as reference pages for the patched pages. For each of the similar pages, we compute a patch using Xdelta.¹³ The average patch size is 1,070 bytes, allowing them to be stored in 37,695 4KB pages, saving 106,802 pages. In sum, subpage sharing requires only 88,422 pages to store the memory for all VMs instead of 195,224 for fullpage sharing or 393,120 originally—an impressive 77% savings, or almost another 50% over full-page sharing. We note that this was the least savings in our experiments; the savings from patching are even higher in most cases. Further, a significant amount of page sharing actually comes from zero pages and, therefore, depends on their availability. For instance, the same workload when executed on 256MB VMs yields far fewer zero pages. Alternative mechanisms to page sharing become even more important in such cases.

One of the principal complications with subpage sharing is identifying candidate reference pages. Difference Engine uses a parameterized scheme to identify similar pages based upon the hashes of several 64-byte portions of each page. In particular, `HashSimilarityDetector(k, s)` hashes the contents of $(k \cdot s)$ 64-byte blocks at randomly chosen locations on the page, and then groups these hashes together into k groups of s hashes each. We use each group as an index into a hash table. In other words, higher values of s capture *local* similarity while higher k values incorporate *global* similarity. Hence, `HashSimilarityDetector(1,1)` will choose one block on a page and index that block; pages are considered similar if that block of data matches. `HashSimilarityDetector(1,2)` combines the hashes from two different locations in the page into one index of length two. `HashSimilarityDetector(2,1)` instead indexes each page twice: once based on the contents of a first block, and again based on the contents of a second block. Pages that match at least one of the two blocks are chosen as candidates. For each scheme, the number of candidates, c , specifies how many different pages the hash table tracks for each signature. With one candidate, we only store the first page found with each signature; for larger values, we keep multiple pages in the hash table for each index. When trying to build a patch, Difference Engine computes a patch between all matching pages and chooses the best one.

We have evaluated this scheme for a variety of parameter settings on the two workloads described above. For both the workloads, `HashSimilarityDetector(2,1)` with one candidate does surprisingly well. There is a substantial gain due to hashing two distinct blocks in the page separately, but

little additional gain by hashing more blocks. Combining blocks does not help much, at least for these workloads. Furthermore, storing more candidates in each hash bucket also produces little gain. Hence, Difference Engine indexes a page by hashing 64-byte blocks at two fixed locations in the page (chosen at random) and using each hash value as a separate index to store the page in the hash table. To find a candidate similar page, the system computes hashes at the same two locations, looks up those hash table entries, and chooses the better of the (at most) two pages found there.

3.3. Compression

Finally, for pages that are not significantly similar to other pages in memory, we consider compressing them to reduce the memory footprint. Compression is useful only if the compression ratio is reasonably high, and, like patching, if selected pages are accessed infrequently, otherwise the overhead of compression/decompression will outweigh the benefits. We identify candidate pages for compression using a global clock algorithm (Section 4.2), assuming that pages that have not been recently accessed are unlikely to be accessed in the near future.

Difference Engine supports multiple compression algorithms, currently LZO and WKdm as described in Wilson et al.²⁰; we invalidate compressed pages in the VM and save them in a dynamically allocated storage area in machine memory. When a VM accesses a compressed page, Difference Engine decompresses the page and returns it to the VM uncompressed. It remains there until it is again considered for compression.

3.4. Paging machine memory

While Difference Engine will deliver some (typically high) level of memory savings, in the worst case all VMs might actually require all of their allocated memory. Setting aside sufficient physical memory to account for this case prevents using the memory saved by Difference Engine to create additional VMs. Not doing so, however, may result in temporarily overshooting the physical memory capacity of the machine and cause a system crash. We therefore require a demand-paging mechanism to supplement main memory by writing pages out to disk in such cases.

A good candidate page for swapping out would likely not be accessed in the near future—the same requirement as compressed/patched pages. In fact, Difference Engine also considers compressed and patched pages as candidates for swapping out. Once the contents of the page are written to disk, the page can be reclaimed. When a VM accesses a swapped out page, Difference Engine fetches it from disk and copies the contents into a newly allocated page that is mapped appropriately in the VM's memory.

Since disk I/O is involved, swapping in/out is an expensive operation. Further, a swapped page is unavailable for sharing or as a reference page for patching. Therefore, swapping should be an infrequent operation. Difference Engine implements the core mechanisms for paging, and leaves policy decisions—such as when and how much to swap—to user space tools.

4. IMPLEMENTATION

We have implemented Difference Engine on top of Xen 3.0.4 in roughly 14,500 lines of code. An additional 20,000 lines come from ports of existing patching and compression algorithms (Xdelta, LZO, WKdm) to run inside Xen.

4.1. Modifications to Xen

Xen and other platforms that support fully virtualized guests use a mechanism called “shadow page tables” to manage guest OS memory.¹⁹ The guest OS has its own copy of the page table that it manages believing that they are the hardware page tables, though in reality it is just a map from the guest’s virtual memory to its notion of physical memory (V2P map). In addition, Xen maintains a map from the guest’s notion of physical memory to the machine memory (P2M map). The shadow page table is a cache of the results of composing the V2P map with the P2M map, mapping guest virtual memory directly to machine memory.

Difference Engine relies on manipulating P2M maps and the shadow page tables to interpose on page accesses. For simplicity, we do not consider any pages mapped by Domain-0 (the privileged, control domain in Xen), which, among other things, avoids the potential for circular page faults.

4.2. Clock

Difference Engine implements a not-recently-used (NRU) policy¹⁶ to select candidate pages for sharing, patching, compression, and swapping out. On each invocation, the clock scans a portion of the memory, checking and clearing the *referenced* (R) and *modified* (M) bits on pages. Thus, pages with the R/M bits set must have been referenced/modified since the last scan. We ensure that successive scans of memory are separated by at least 4 s in the current implementation to give domains a chance to set the R/M bits on frequently accessed pages. In the presence of multiple VMs, the clock scans a small portion of each VM’s memory in turn for fairness. The external API exported by the clock is simple: return a list of pages (of some maximum size) that have not been accessed in some time.

Using the R/M bits, we can annotate each page with its “freshness.” By default, we employ the following policy. Recently modified pages are ignored; pages accessed recently but not modified are considered for sharing and to be reference pages for patching, but cannot be patched or compressed themselves; pages not recently accessed can be shared or patched; and pages not accessed for an extended period of time are eligible for everything, including compression and swapping.

4.3. Page sharing

Difference Engine uses the SuperFastHash⁴ function to compute digests for each scanned page and inserts them along with the page-frame number into a hash table. Ideally, the hash table should be sized so that it can hold entries for all of physical memory. The hash table is allocated out of Xen’s heap space, which is quite limited in size: the code, data, and heap segments in Xen must all fit in a 12MB region

of memory. Changing the heap size requires pervasive code changes in Xen, and will likely break the application binary interface (ABI) for some OSes. We therefore restrict the size of the page-sharing hash table so that it can hold entries for only 1/5 of physical memory. Hence Difference Engine processes memory in five passes, as described by Kloster et al.¹¹ In our test configuration, this partitioning results in a 1.76MB hash table. We divide the space of hash function values into five intervals, and only insert a page into the table if its hash value falls into the current interval. A complete cycle of five passes covering all the hash value intervals is required to identify all identical pages.

4.4. Page-similarity detection

The goal of the page-similarity component is to find pairs of pages with similar content, and, hence, make candidates for patching. We implement a simple strategy for finding similar pages based on hashing short blocks within a page, as described in Section 3.2. Specifically, we use the HashSimilarityDetector(2,1) described there, which hashes short data blocks from two locations on each page, and indexes the page at each of those two locations in a separate page-similarity hash table, distinct from the page-sharing hash table described above. We use the 1-candidate variation, where at most one page is indexed for each block hash value.

Recall that the clock makes a complete scan through memory in five passes. The page-sharing hash table is cleared after each pass, since only pages *within* a pass are considered for sharing. However, two similar pages may appear in different passes if their hash values fall in different intervals. Since we want to only consider pages that have not been shared in a full cycle for patching, the page-similarity hash table is *not* cleared on every pass. This approach also increases the chances of finding better candidate pages to act as the reference for a patch.

4.5. Compression

Compression operates similarly to patching—in both cases the goal is to replace a page with a shorter representation of the same data. The primary difference is that patching makes use of a reference page, while a compressed representation is self contained. There is one important interaction between compression and patching: once we compress a page, the page can no longer be used as a reference for a later patched page. A naive implementation that compresses all nonidentical pages as it goes along will almost entirely prevent page patches from being built. Compression of a page should be postponed at least until all pages have been checked for similarity against it. A complete cycle of a page sharing scan will identify similar pages, so a sufficient condition for compression is that *no page should be compressed until a complete cycle of the page sharing code finishes.*

4.6. Paging machine memory

Recall that any memory freed by Difference Engine cannot be used reliably without supplementing main memory with secondary storage. That is, when the total allocated memory of all VMs exceeds the system memory capacity, some pages will have to be swapped to disk.

The Xen VMM does not perform any I/O (delegating all I/O to Domain-0) and is not aware of any devices. Thus, it is not possible to build swap support directly in the VMM. Further, since Difference Engine supports unmodified OSes, we cannot expect any support from the guest OS. Hence, we implement a single swap daemon (`swapd`) running as a user process in Domain-0 to manage swap space. For each VM in the system, `swapd` creates a separate thread to handle swap-in requests.

To swap out a page, `swapd` makes a hypercall into Xen, where a victim page is chosen by invoking the global clock. If the victim is a compressed or patched page, we first reconstruct it. We pause the VM that owns the page and copy the contents of the page to a page in Domain-0's address space (supplied by `swapd`). Next, we remove all entries pointing to the victim page in the P2M and M2P maps, and in the shadow page tables. We then mark the page as swapped out in the corresponding page table entry. Meanwhile, `swapd` writes the page contents to the swap file and inserts the corresponding byte offset in a hash table keyed by `<Domain ID, guest page-frame number>`. Finally, we free the page, return it to the domain heap, and reschedule the VM.

When a VM tries to access a swapped page, it incurs a page fault and traps into Xen. We pause the VM and allocate a fresh page to hold the swapped in data. We populate the P2M and M2P maps appropriately to accommodate the new page. Xen dispatches a swap-in request to `swapd` containing the domain ID and the faulting page-frame number. The handler thread for the faulting domain in `swapd` receives the request and fetches the location of the page in the swap file from the hash table. It then copies the page contents into the newly allocated page frame within Xen via another hypercall. At this point, `swapd` notifies Xen, and Xen restarts the VM at the faulting instruction.

5. EVALUATION

We first present micro-benchmarks to evaluate the cost of individual operations, the performance of the global clock and the behavior of each of the three mechanisms in isolation. Next, we evaluate whole system performance: for a range of workloads, we measure memory savings and the impact on application performance. We present head-to-head comparisons with the VMware ESX server. Finally, we demonstrate how our memory savings can be used to boost the aggregate system performance. Unless otherwise mentioned, all experiments are run on dual-processor, dual-core 2.33-GHz Intel Xeon machines and the page size is 4KB.

5.1. Cost of individual operations

Before quantifying the memory savings provided by Difference Engine, we measure the overhead of various functions involved. Table 2 shows the overhead imposed by the major Difference Engine operations. As expected, collapsing identical pages into a copy-on-write shared page (`share_page`) and recreating private copies (`cow_break`) are relatively cheap operations, taking approximately 6 and 25 μ s, respectively. Perhaps more surprising, however, is

Table 2. CPU overhead of different functions.

Function	Mean execution time (μ s)
<code>share_pages</code>	6.2
<code>cow_break</code>	25.1
<code>compress_page</code>	29.7
<code>uncompress</code>	10.4
<code>patch_page</code>	338.1
<code>unpatch</code>	18.6
<code>swap_out_page</code>	48.9
<code>swap_in_page</code>	7151.6

that compressing a page on our hardware is fast, requiring slightly less than 30 μ s on average. Patching, on the other hand, is almost an order of magnitude slower: creating a patch (`patch_page`) takes over 300 μ s. This time is primarily due to the overhead of finding a good candidate base page and constructing the patch. Both decompressing a page and reconstructing a patched page are also fairly fast, taking 10 and 18 μ s respectively.

Swapping out takes approximately 50 μ s. However, this does *not* include the time to actually write the page to disk. This is intentional: once the page contents have been copied to user space, they *are* immediately available for being swapped in; and the actual write to the disk might be delayed because of file system and OS buffering in Domain-0. Swapping in, on the other hand, is the most expensive operation, taking approximately 7 ms. There are a few caveats, however. First, swapping in is an asynchronous operation and might be affected by several factors, including process scheduling within Domain-0; it is *not* a tight bound. Second, swapping in might require reading the page from disk, and the seek time will depend on the size of the swap file, among other things.

5.2. Real-world applications

We now present the performance of Difference Engine on a variety of workloads. We seek to answer two questions. First, how effective are the memory-saving mechanisms at reducing memory usage for real-world applications? Second, what is the impact of those memory-sharing mechanisms on system performance? Since the degree of possible sharing depends on the software configuration, we consider several different cases of application mixes.

To put our numbers in perspective, we conduct head-to-head comparisons with VMware ESX server for three different workload mixes. We run ESX Server 3.0.1 build 32,039 on a Dell PowerEdge 1950 system. Note that even though this system has two 2.3-GHz Intel Xeon processors, our VMware license limits our usage to a single CPU. We therefore restrict Xen (and, hence, Difference Engine) to use a single CPU for fairness. We also ensure that the OS images used with ESX match those used with Xen, especially the file system and disk layout. Note that we are only concerned with the effectiveness of the memory sharing mechanisms—not in comparing the application performance across the two hypervisors. In an effort to compare the performance of Difference Engine to ESX in its most aggressive configuration, we configure both to scan 10,000 pages/s, the highest rate allowed by ESX.^a

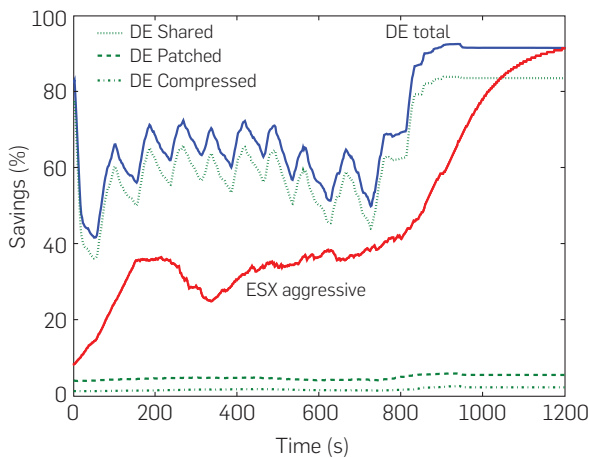
Base Scenario—Homogeneous VMs: In our first set of benchmarks, we test the base scenario where all VMs on a machine run the same OS and applications. This scenario is common in cluster-based systems where several services are replicated to provide fault tolerance or load balancing. Our expectation is that significant memory savings are available and that most of the savings will come from page sharing.

On a machine running standard Xen, we start from 1 to 6 VMs, each with 256MB of memory and running RUBiS⁸—an e-commerce application designed to evaluate application server performance—on Debian 3.1. We use the PHP implementation of RUBiS; each instance consists of a Web server (Apache) and a database server (MySQL). Two distinct client machines generate the workload, each running the standard RUBiS workload generator simulating 100 user sessions. The benchmark runs for roughly 20 min. The workload generator reports several metrics at the end of the benchmark, in particular the average response time and the total number of requests served.

We then run the same set of VMs with Difference Engine enabled. Both the total number of requests and the average response time remain unaffected while Difference Engine delivers 65%–75% memory savings. In this case, the bulk of memory savings comes from page sharing. Recall that Difference Engine tries to share as many pages as it can before considering pages for patching and compression, so sharing is expected to be the largest contributor in most cases, particularly in homogeneous workloads.

We next compare Difference Engine performance with the VMware ESX server. We set up four 512MB virtual machines running Debian 3.1. Each VM executes `dbench`² for 10 min followed by a stabilization period of 20 min. Figure 2 shows the amount of memory saved as a function of time. First, note that *eventually* both ESX and Difference

Figure 2. Four identical VMs execute `dbench`. Both Difference Engine and ESX eventually yield similar savings, but DE extracts more savings while the benchmark is in progress.



⁸ After initial publication of our results, we were informed by VMware that this version of ESX silently limits the effective page-sharing rate to a maximum of 450 pages/sec per vm regardless of the configured scan rate.

Engine reclaim roughly the same amount of memory (the graph for ESX plateaus beyond 1,200 s). However, *while* `dbench` is executing, Difference Engine delivers approximately 1.5 times the memory savings achieved by ESX. As before, the bulk of Difference Engine savings come from page sharing for the homogeneous workload case.

Heterogeneous OS and Applications: Given the increasing trend toward virtualization, both on the desktop and in the data center, we envision that a single physical machine will host significantly different types of operating systems and workloads. While smarter VM placement and scheduling will mitigate some of these differences, there will still be a diverse and heterogeneous mix of applications and environments, underscoring the need for mechanisms other than page sharing. We now examine the utility of Difference Engine in such scenarios, and demonstrate that significant additional memory savings result from employing patching and compression in these settings.

Figures 3 and 4 show the memory savings as a function of time for the two heterogeneous workloads—MIXED-1 and MIXED-2 described in Section 3.2. We make the following observations. First, in steady state, Difference Engine delivers a factor of 1.6–2.5 more memory savings than ESX. For instance, for the MIXED-2 workload, Difference Engine could host the three VMs allocated 512MB of physical memory each in approximately 760MB of machine memory; ESX would require roughly 1,100MB of machine memory. The remaining, significant, savings come from patching and compression. And these savings come at a small cost. Table 3 summarizes the performance of the three benchmarks in the MIXED-1 workload. The baseline configuration is regular Xen without Difference Engine. In all cases, performance overhead of Difference Engine is within 7% of the baseline. For the same workload, we find that performance under ESX with aggressive page sharing is also within 5% of the ESX baseline with no page sharing.

Increasing Aggregate System Performance: Difference Engine goes to great lengths to reclaim memory in a system, but eventually this extra memory needs to actually get used in a productive

Figure 3. Memory savings for MIXED-1. Difference Engine saves up to 45% more memory than ESX.

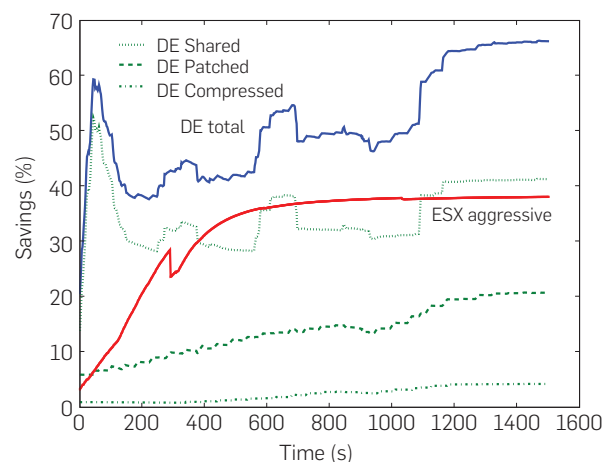


Figure 4. Memory savings for Mixed-2. Difference Engine saves almost twice as much memory as ESX.

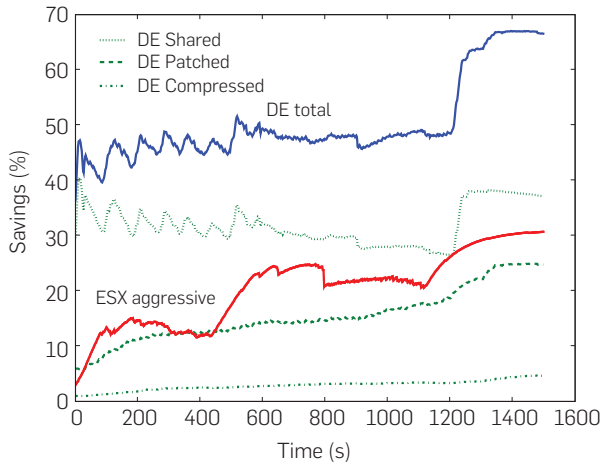


Table 3. Application performance under Difference Engine for the heterogeneous workload Mixed-1 is within 7% of the baseline.

	Kernel Compile (s)	Vim compile, lmbench (s)	RUBiS requests	RUBiS response time(ms)
Baseline	670	620	3149	1280
DE	710	702	3130	1268

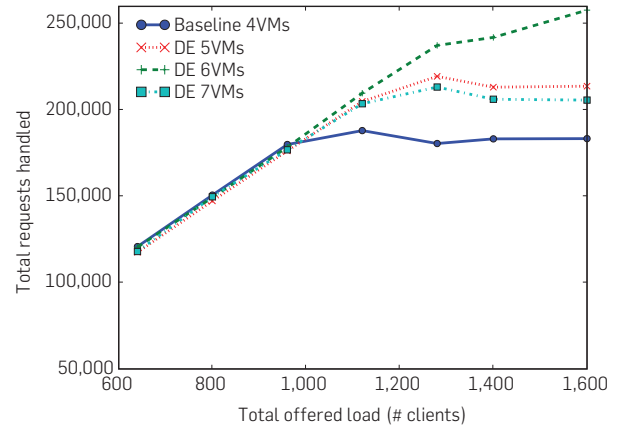
manner. One can certainly use the saved memory to create more VMs, but does that increase the aggregate system performance?

To answer this question, we created four VMs with 650MB of RAM each on a physical machine with 2.8GB of free memory (excluding a memory allocated to Domain-0). For the baseline (without Difference Engine), Xen allocates memory statically. Upon creating all the VMs, there is clearly not enough memory left to create another VM of the same configuration. Each VM hosts a RUBiS instance. For this experiment, we used the Java Servlets implementation of RUBiS. There are two distinct client machines per VM to act as workload generators.

The goal is to increase the load on the system to saturation. The solid line in Figure 5 shows the total requests served for the baseline, with the total offered load marked on the X-axis. Beyond 960 clients, the total number of requests served plateaus at around 180,000 while the average response time (not shown) increases sharply. Upon investigation, we find that for higher loads all of the VMs have more than 95% memory utilization and some VMs actually start swapping to disk (within the guest OS). Using fewer VMs with more memory (e.g., 2 VMs with 1.2GB RAM each) did not improve the baseline performance for this workload.

Next, we repeat the same experiment with Difference Engine, except this time we utilize reclaimed memory to create additional VMs. As a result, for each data point on the X-axis, the per VM load decreases, while the aggregate offered load remains the same. We expect that since each VM individually has lower load compared to the baseline, the system will

Figure 5. Up to a limit, Difference Engine can help increase aggregate system performance by spreading the load across extra VMs.



deliver better aggregate performance. The remaining lines show the performance with up to three extra VMs. Clearly, Difference Engine enables higher aggregate performance compared to the baseline. However, beyond a certain point (two additional VMs in this case), the overhead of managing the extra VMs begins to offset the performance benefits: Difference Engine has to manage 4.5GB of memory on a system with 2.8GB of RAM to support seven VMs. In each case, beyond 1,400 clients, the VM's working set becomes large enough to invoke the paging mechanism: we observe between 5,000 pages (for one extra VM) to around 20,000 pages (for three extra VMs) being swapped out, of which roughly a fourth get swapped back in.

6. CONCLUSION

One of the primary bottlenecks to higher degrees of virtual machine multiplexing is main memory. Earlier work shows that substantial memory savings are available from harvesting identical pages across virtual machines when running homogeneous workloads. The premise of this work is that there are significant additional memory savings available from locating and patching similar pages and in-memory page compression. We present the design and evaluation of Difference Engine to demonstrate the potential memory savings available from leveraging a combination of whole page sharing, page patching, and compression. Our performance evaluation shows that Difference Engine delivers an additional factor of 1.6–2.5 more memory savings than VMware ESX server for a variety of workloads, with minimal performance overhead. Difference Engine mechanisms might also be leveraged to improve single OS memory management; we leave such exploration to future work.

Acknowledgments

In the course of the project, we received invaluable assistance from a number of people at VMware. We would like to thank Carl Waldspurger, Jennifer Anderson, and Hemant Gaidhani, and the Performance Benchmark group for feedback and discussions on the performance of ESX server. Also, special thanks are owed to Kiran Tati for assisting

with ESX setup and monitoring and to Emil Sit for providing insightful feedback on the paper. Finally, we would like to thank Michael Mitzenmacher for his assistance with min-wise hashing, our OSDI shepherd Fred Douglass for his insightful feedback and support, Rick Farrow at ;login; and the anonymous OSDI '08 reviewers for their valuable comments. This work was supported in part by NSF CSR-PDOS Grant No. CNS-0615392, the UCSD Center for Networked Systems (CNS), and UC Discovery Grant 07-10237. Vrable was supported in part by an NSF Graduate Research Fellowship. □

References

1. <http://sysbench.sourceforge.net/>.
2. <http://samba.org/ftp/tridge/dbench/>.
3. <http://www.iozone.org/>.
4. <http://www.azillionmonkeys.com/qed/hash.html>.
5. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A. Xen and the art of virtualization. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles* (2003).
6. Broder, A.Z. Identifying and filtering near-duplicate documents. In *Proceedings of the 11th Annual Symposium on Combinatorial Pattern Matching* (2000).
7. Bugnion, E., Devine, S., Rosenblum, M. Disco: Running commodity operating systems on scalable multiprocessors. In *Proceedings of the 16th ACM Symposium on Operating System Principles* (1997).
8. Cecchet, E., Marguerite, J., Zwaenepoel, W. Performance and scalability of EJB applications. In *Proceedings of the 17th ACM Conference on Object-oriented Programming, Systems, Languages, and Applications* (2002).
9. Douglass, F. The compression cache: Using on-line compression to extend physical memory. In *Proceedings of the USENIX Winter Technical Conference* (1993).
10. Douglass, F., Iyengar, A. Application-specific delta-encoding via resemblance detection. In *Proceedings of the USENIX Annual Technical Conference* (2003).
11. Kloster, J.F., Kristensen, J., Mejlholm, A. On the feasibility of memory sharing. Master's thesis, Aalborg University (2006).
12. Kulkarni, P., Douglass, F., Lavoie, J.,

- Tracey, J.M. Redundancy elimination within large collections of files. In *Proceedings of the USENIX Annual Technical Conference* (2004).
13. MacDonald, J. xdelta. <http://www.xdelta.org/>.
14. Manber, U., Wu, S. GLIMPSE: A tool to search through entire file systems. In *Proceedings of the USENIX Winter Technical Conference* (1994).
15. McVoy, L., Staelin, C. Imbench: Portable tools for performance analysis. In *Proceedings of the USENIX Annual Technical Conference* (1996).
16. Tanenbaum, A.S. *Modern Operating Systems*. Prentice Hall (2007).
17. Tudu, I.C., Gross, T. Adaptive main memory compression. In *Proceedings of the USENIX Annual Technical Conference* (2005).
18. Vrable, M., Ma, J., Chen, J., Moore, D., VandeKieft, E., Snoeren, A.C., Voelker, G.M., Savage, S. Scalability, fidelity and containment in the Potemkin virtual honeyfarm. In *Proceedings of the 20th ACM Symposium on Operating System Principles* (2005).
19. Waldspurger, C.A. Memory resource management in VMware ESX server. In *Proceedings of the 5th ACM/USENIX Symposium on Operating System Design and Implementation* (2002).
20. Wilson, P.R., Kaplan, S.F., Smaragdakis, Y. The case for compressed caching in virtual memory systems. In *Proceedings of the USENIX Annual Technical Conference* (1999).

Diwaker Gupta (diwaker@asterdata.com), University of California, San Diego, Currently at Aster Data.

Sangmin Lee (sangmin@cs.utexas.edu), University of California, San Diego, Currently at UT Austin.

Michael Vrable (mvrable@cs.ucsd.edu), University of California, San Diego.

Stefan Savage (savage@cs.ucsd.edu), University of California, San Diego.

Alex C. Snoeren (snoeren@cs.ucsd.edu), University of California, San Diego.

George Varghese (varghese@cs.ucsd.edu), University of California, San Diego.

Geoffrey M. Voelker (voelker@cs.ucsd.edu), University of California, San Diego.

Amin Vahdat (vahdat@cs.ucsd.edu), University of California, San Diego.

© 2010 ACM 0001-0782/10/1000 \$10.00



Association for
Computing Machinery

Advancing Computing as a Science & Profession



You've come a long way.
Share what you've learned.



ACM has partnered with MentorNet, the award-winning nonprofit e-mentoring network in engineering, science and mathematics. MentorNet's award-winning **One-on-One Mentoring Programs** pair ACM student members with mentors from industry, government, higher education, and other sectors.

- Communicate by email about career goals, course work, and many other topics.
- Spend just **20 minutes a week** - and make a huge difference in a student's life.
- Take part in a lively online community of professionals and students all over the world.



Make a difference to a student in your field.
Sign up today at: www.mentornet.net
Find out more at: www.acm.org/mentornet

MentorNet's sponsors include 3M Foundation, ACM, Alcoa Foundation, Agilent Technologies, Amylin Pharmaceuticals, Bechtel Group Foundation, Cisco Systems, Hewlett-Packard Company, IBM Corporation, Intel Foundation, Lockheed Martin Space Systems, National Science Foundation, Naval Research Laboratory, NVIDIA, Sandia National Laboratories, Schlumberger, S.D. Bechtel, Jr. Foundation, Texas Instruments, and The Henry Luce Foundation.

Technical Perspective

Belief Propagation

By Yair Weiss and Judea Pearl

WHEN A PAIR of nuclear-powered Russian submarines was reported patrolling off the eastern seaboard of the U.S. last summer, Pentagon officials expressed wariness over the Kremlin's motivations. At the same time, these officials emphasized their confidence in the U.S. Navy's tracking capabilities: "We've known where they were," a senior Defense Department official told the *New York Times*, "and we're not concerned about our ability to track the subs."

While the official did not divulge the methods used by the Navy to track submarines, the *Times* added that such tracking "can be done from aircraft, ships, underwater sensors, or other submarines." But the article failed to mention perhaps the most important part of modern tracking technology—the algorithm that fuses different measurements at different times. Nearly every modern tracking system is based on the seminal work of Rudolf Kalman¹ who developed the optimal fusion algorithm for linear dynamics under Gaussian noise. This algorithm, now known simply as the "Kalman filter" is used in a remarkably broad range of real-world applications—from patient monitoring to spaceship navigation. But in the 50 years since Kalman first published his algorithm, it has become apparent that the problem it addresses is a special case of a much more general problem.

This general problem, known as "Bayesian inference in graphical models," is defined on a graph where the nodes denote random variables and edges encode direct probabilistic dependencies. Each node has access to a noisy measurement about its state. In the case of tracking a submarine, the t th node will represent the location of a submarine at time t , and edges will connect node t to node $t+1$ in a temporal chain, representing the fact that a submarine's current location is highly dependent on its location in the previous time. Kalman's algorithm allows one to efficiently compute the optimal estimate of the submarine's location, given


all the measurements. It assumes the probabilistic dependencies are Gaussian and the graph is a temporal chain.

The generalization of Kalman's algorithm to arbitrary graphical models is called "belief propagation"² and it originated in the late 1970s after Judea Pearl read a paper by the cognitive psychologist David Rumelhart on how children comprehend text.³ Rumelhart presented compelling evidence that text comprehension must be, first, a collaborative computation among a vast number of autonomous, neural-like modules, each doing an extremely simple and repetitive task and, second, that some kind of friendly "handshaking" must take place between top-down and bottom-up modes of inference, for example, the meaning of a sentence helps disambiguate a word while, at the same time, recognizing a word helps disambiguate a sentence. This disambiguation is similar to what happens in a Kalman filter (where measurements at one time can disambiguate measurements at another time), but the dependency structure is certainly not a temporal chain.

Not caring much about generality, Pearl pieced together the simplest structure he could think of (that is, a tree) and tried to see if anything useful can be computed by assigning each variable a simple processor, forced to communicate only with its neighbors. This gave rise to the tree-propagation algorithm and, a year later, to belief propagation on poly-trees, which supported bi-directional inferences and interactions known as "explaining-away."

Although several algorithms were later developed to perform Bayesian updating in general, "loopy" structures, the prospects of achieving such updating by local message passing process remained elusive. Out of total frustration, yet still convinced that such algorithms must guide many of our cognitive abilities, Pearl imagined a "shortsighted" algorithm that totally ignores the loopy structure of the graph and propagates messages

as if each module is situated in a poly-tree environment. He then assigned as a homework exercise² the task of evaluating the extent to which this uninformed algorithm could serve as an approximation to the exact Bayesian inference problem. This "homework exercise" was partially solved by different researchers in the last decade and loopy belief propagation is now used successfully in applications ranging from satellite communication to driver assistance.

The success of loopy belief propagation, however, has been limited to discrete state spaces. In the following paper, Sudderth et al. provide an elegant algorithm that handles continuous variables. Unlike the Kalman filter, it does not require the probabilistic dependencies to be Gaussian, relying instead on stochastic algorithms known as "Monte Carlo" algorithms. An extension to Kalman filters called "particle filters" also uses Monte Carlo algorithms, but the authors provide an algorithm that can work with any dependency structure, not just a temporal chain. They show how their algorithm successfully solves some important "loopy" problems in computer-vision and sensor networks. One only wonders if in the future such algorithms will be used to solve the really difficult problems—figuring out the Kremlin's intent from partial, noisy observations, or reading text as children do. 

References

1. Kalman, R.E. A new approach to linear filtering and prediction problems. *Transactions of the ASME—Journal of Basic Engineering* 82, Series D (1960) 35-45.
2. Pearl, J. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.
3. Rumelhart, D. Toward an interactive model of reading. Technical Report CHIP-56, Center for Human Information Processing, University of California, San Diego, 1976.

Yair Weiss (yweiss@cs.huji.ac.il) is a professor in the School of Computer Science and Engineering at The Hebrew University of Jerusalem, Israel.

Judea Pearl (Judea@cs.ucla.edu) is a professor of computer science and director of the Cognitive Systems Laboratory at University of California, Los Angeles.

© 2010 ACM 0001-0782/10/1000 \$10.00

Nonparametric Belief Propagation

By Erik B. Sudderth, Alexander T. Ihler, Michael Isard, William T. Freeman, and Alan S. Willsky

Abstract

Continuous quantities are ubiquitous in models of real-world phenomena, but are surprisingly difficult to reason about automatically. Probabilistic graphical models such as Bayesian networks and Markov random fields, and algorithms for approximate inference such as belief propagation (BP), have proven to be powerful tools in a wide range of applications in statistics and artificial intelligence. However, applying these methods to models with continuous variables remains a challenging task. In this work we describe an extension of BP to continuous variable models, generalizing particle filtering, and Gaussian mixture filtering techniques for time series to more complex models. We illustrate the power of the resulting nonparametric BP algorithm via two applications: kinematic tracking of visual motion and distributed localization in sensor networks.

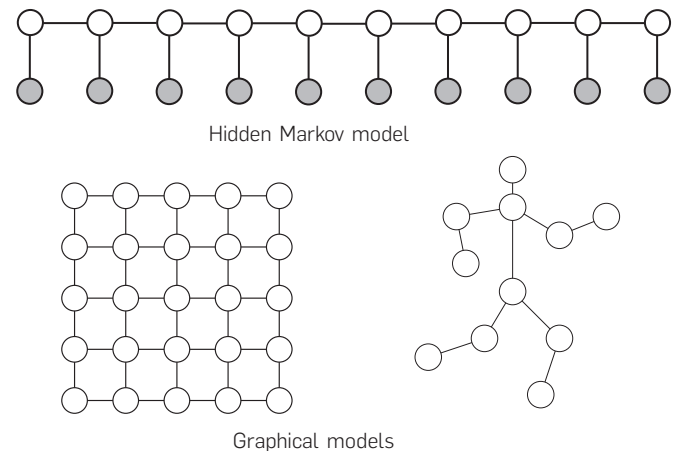
1. INTRODUCTION

Graphical models provide a powerful, general framework for developing statistical models in such diverse areas as bioinformatics, communications, natural language processing, and computer vision.²⁸ However, graphical formulations are only useful when combined with efficient algorithms for inference and learning. Such algorithms exist for many discrete models, such as those underlying modern error correcting codes and machine translation systems.

For most problems involving high-dimensional continuous variables, comparably efficient and accurate algorithms are unavailable. Alas, these are exactly the sorts of problems that arise frequently in areas like computer vision. Difficulties begin with the continuous surfaces and illuminants that digital cameras record in grids of pixels, and that geometric reconstruction algorithms seek to recover. At a higher level, the articulated models used in many tracking applications have dozens of degrees of freedom to be estimated at each time step.^{41,45} Realistic graphical models for these problems must represent outliers, bimodalities, and other non-Gaussian statistical features. The corresponding optimal inference procedures for these models typically involve integral equations for which no closed form solution exists. It is thus necessary to develop families of approximate representations, and algorithms for fitting those approximations.

In this work we describe the *nonparametric belief propagation* (NBP) algorithm. NBP combines ideas from Monte Carlo³ and particle filtering^{6,11} approaches for representing complex uncertainty in time series, with the popular belief propagation (BP) algorithm³⁷ for approximate inference in complex graphical models. Unlike discretized approximations to continuous variables, NBP is not limited to low-dimensional domains. Unlike classical Gaussian approximations, NBP's particle-based messages can represent, and consistently reason about, the multimodal

Figure 1. Particle filters assume variables are related by a hidden Markov model (top). The NBP algorithm extends particle filtering techniques to arbitrarily structured graphical models, such as those for arrays of image pixels (bottom left) or articulated human motion (bottom right).



distributions induced by many real-world datasets. And unlike particle filters, NBP can exploit the rich nonsequential structure of more complex graphical models, like those in Figure 1.

We begin in Section 2 by reviewing graphical models, BP, Monte Carlo methods, and particle filters. Section 3 then develops the two stages of the NBP algorithm: a belief fusion step which combines information from multiple particle sets, and a message propagation step which accounts for dependencies among random variables. We review a pair of previous real-world applications of NBP in Section 4: kinematic tracking of visual motion (Figures 6 and 7) and distributed localization in sensor networks (Figure 8). Finally, we conclude in Section 5 by surveying algorithmic and theoretical developments since the original introduction of NBP.

2. INFERENCE IN GRAPHICAL MODELS

Probabilistic graphical models decompose multivariate distributions into a set of local interactions among small subsets of variables. These local relationships produce conditional independencies which lead to efficient learning and inference algorithms. Moreover, their modular

The original versions of this paper were entitled “Nonparametric Belief Propagation,” by E. Sudderth, A. Ihler, W. Freeman, and A. Willsky, and “PAMPAS: Real-Valued Graphical Models for Computer Vision,” by M. Isard. Both appeared in the *IEEE Conference on Computer Vision and Pattern Recognition*, June 2003.

structure provides an intuitive language for expressing domain-specific knowledge about variable relationships and facilitates the transfer of modeling advances to new applications.

Several different formalisms have been proposed that use graphs to represent probability distributions.^{28, 30, 47, 50} *Directed* graphical models, or *Bayesian networks*, are widely used in artificial intelligence to encode causal, generative processes. Such directed graphs provided the basis for the earliest versions of the BP algorithm.³⁷ Alternatively, *undirected* graphical models, or *Markov random fields* (MRFs), provide popular models for the symmetric dependencies arising in such areas as signal processing, spatial statistics, bioinformatics, and computer vision.

2.1. Pairwise Markov random fields

An undirected graph \mathcal{G} is defined by a set of nodes \mathcal{V} and a corresponding set of undirected edges \mathcal{E} (see Figure 1). Let $\Gamma(i) \triangleq \{j \mid (i, j) \in \mathcal{E}\}$ denote the *neighborhood* of a node $i \in \mathcal{V}$. MRFs associate each node $i \in \mathcal{V}$ with an unobserved, or hidden, random variable $x_i \in \mathcal{X}_i$. Let $x = \{x_i \mid i \in \mathcal{V}\}$ denote the set of all hidden variables. Given evidence or observations y , a *pairwise MRF* represents the posterior distribution $p(x \mid y)$ in factored form:

$$p(x \mid y) \propto p(x, y) \propto \prod_{(i,j) \in \mathcal{E}} \psi_{ij}(x_i, x_j) \prod_{i \in \mathcal{V}} \psi_i(x_i, y). \quad (1)$$

Here, the proportionality sign indicates that $p(x, y)$ may only be known up to an uncertain normalization constant, chosen so that it integrates to one. The positive *potential functions* $\psi_{ij}(x_i, x_j) > 0$ can often be interpreted as soft, local constraints. Note that the local evidence potential $\psi_i(x_i, y)$ does *not* typically equal the marginal distribution $p(x_i \mid y)$, due to interactions with other potentials.

In this paper, we develop algorithms to approximate the conditional marginal distributions $p(x_i \mid y)$ for all $i \in \mathcal{V}$. These densities lead to estimates of x_i , such as the posterior mean $\mathbb{E}[x_i \mid y]$, as well as corresponding measures of uncertainty. We focus on pairwise MRFs due to their simplicity and popularity in practical applications. However, the nonparametric updates we present may also be directly extended to models with higher-order potentials.

2.2. Belief propagation

For graphs that are acyclic or tree-structured, the desired conditional distributions $p(x_i \mid y)$ can be directly calculated by a local message-passing algorithm known as *belief propagation* (BP).^{37, 50} At each iteration of the BP algorithm, nodes $j \in \mathcal{V}$ calculate messages $m_{ji}(x_i)$ to be sent to each neighboring node $i \in \Gamma(j)$:

$$m_{ji}(x_i) \propto \int_{\mathcal{X}_j} \psi_{ij}(x_i, x_j) \psi_j(x_j, y) \prod_{k \in \Gamma(j) \setminus i} m_{kj}(x_j) dx_j. \quad (2)$$

The outgoing message is a positive function defined on \mathcal{X}_i . Intuitively, it is a (possibly approximate) sufficient statistic of the information that node j has collected regarding x_i . At any iteration, each node can produce an approximation $q_i(x_i)$ to the marginal distribution $p(x_i \mid y)$ by combining incoming messages with the local evidence potential:

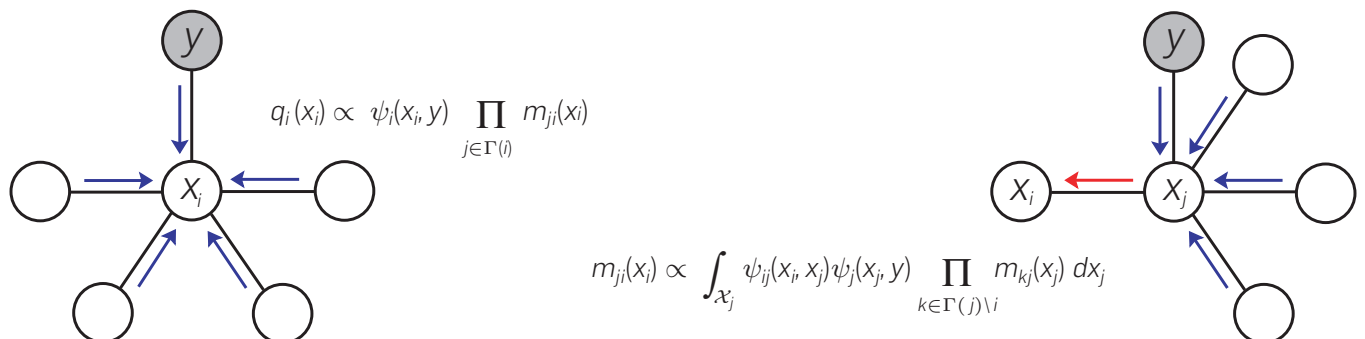
$$q_i(x_i) \propto \psi_i(x_i, y) \prod_{j \in \Gamma(i)} m_{ji}(x_i). \quad (3)$$

These updates are graphically summarized in Figure 2. For tree-structured graphs, the approximate marginals, or *beliefs*, $q_i(x_i)$ will converge to the true marginals $p(x_i \mid y)$ once messages from each node have propagated across the graph. With an efficient update schedule, the messages for each distinct edge need only be computed once, and BP can be seen as a distributed variant of dynamic programming.

Because each iteration of the BP algorithm involves only local message updates, it can be applied even to graphs with cycles. For such graphs, the statistical dependencies between BP messages are not accounted for, and the sequence of beliefs $q_i(x_i)$ will *not* converge to the true marginals. In many applications, however, the resulting *loopy BP* algorithm³⁷ exhibits excellent empirical performance.^{8, 14, 15, 49} Recently, several theoretical studies have provided insight into the approximations made by loopy BP, establishing connections to other *variational* inference algorithms⁴⁷ and partially justifying its application to graphs with cycles.^{20, 23, 34, 50, 51}

The BP algorithm implicitly assumes messages $m_{ji}(x_i)$ have a finite parameterization, which can be tractably updated via the integral of Equation 2. Most implementations

Figure 2. Message-passing recursions underlying the BP algorithm. Left: Approximate marginal (belief) estimates combine the local observation potential with messages from neighboring nodes. Right: A new outgoing message (red) is computed from all other incoming messages (blue).



assume each hidden variable x_i takes one of K discrete values ($|\mathcal{X}_i| = K$), so that messages and marginals can be represented by K -dimensional vectors. The message update integral then becomes a matrix–vector product, which in general requires $\mathcal{O}(K^2)$ operations. This variant of BP is sometimes called the *sum–product algorithm*.³⁰

For graphical models with continuous hidden variables, closed-form evaluation of the BP update integral is only possible when the posterior is jointly Gaussian. The resulting Gaussian BP algorithm, which uses linear algebra to update estimates of posterior mean vectors and covariance matrices, generalizes Kalman smoothing algorithms for linear dynamical systems.² More generally, a fixed K -point discretization sometimes leads to an effective histogram approximation of the true continuous beliefs.^{13, 14} However, as K must in general grow exponentially with the dimension of \mathcal{X}_i , computation of the discrete messages underlying such approximations can be extremely demanding.

2.3. Monte Carlo methods

By using random samples to simulate probabilistic models, *Monte Carlo methods*³ provide flexible alternatives to variational methods like BP. Given a target distribution $p(x | y)$, many inference tasks can be expressed via the expected value $\mathbb{E}_p[f(x)]$ of an appropriately chosen function. Given L independent samples $\{x^{(\ell)}\}_{\ell=1}^L$ from $p(x | y)$, the desired statistic can be approximated as follows:

$$\mathbb{E}_p[f(x)] = \int_{\mathcal{X}} f(x) p(x | y) dx \approx \frac{1}{L} \sum_{\ell=1}^L f(x^{(\ell)}). \quad (4)$$

This estimate is unbiased, and converges to $\mathbb{E}_p[f(x)]$ almost surely as $L \rightarrow \infty$. For the graphical models of interest here, however, exact sampling from $p(x | y)$ is intractable.

Importance sampling provides an alternative based on a *proposal distribution* $q(x)$, chosen so that $q(\bar{x}) > 0$ wherever $p(\bar{x} | y) > 0$. Defining the importance weight function as $w(x) = \bar{p}(x | y)/q(x)$, where $p(x | y) \propto \bar{p}(x | y)$ up to some potentially unknown normalization constant, the expectation of Equation 4 can be rewritten as follows:

$$\mathbb{E}_p[f(x)] = \frac{\int_{\mathcal{X}} f(x) w(x) q(x) dx}{\int_{\mathcal{X}} w(x) q(x) dx} \approx \sum_{\ell=1}^L w^{(\ell)} f(x^{(\ell)}), \quad (5)$$

$$x^{(\ell)} \sim q(x), \quad w^{(\ell)} \triangleq \frac{w(x^{(\ell)})}{\sum_{m=1}^L w(x^{(m)})}.$$

Importance sampling thus estimates the target expectation via a collection of L *weighted* samples $\{x^{(\ell)}, w^{(\ell)}\}_{\ell=1}^L$.

For high-dimensional models like the full joint distribution of Equation 1, designing tractable proposal distributions that closely approximate $p(x | y)$ is extremely challenging. Even minor discrepancies can produce widely varying importance weights $w^{(\ell)}$, which may in turn cause the estimator of Equation 5 to have a huge variance even for large L . Instead, we use importance sampling to *locally* approximate intermediate computations in the BP algorithm.

2.4. Particle filters

Our approach is inspired by particle filters, an approximate inference algorithm specialized for *hidden Markov models* (HMMs). As depicted graphically in Figure 1, an HMM models a sequence of T observations $y = \{y_1, \dots, y_T\}$ via a corresponding set of hidden states x :

$$p(x, y) = p(x_1) p(y_1 | x_1) \prod_{t=2}^T p(x_t | x_{t-1}) p(y_t | x_t). \quad (6)$$

Recognizing this factorization as a special case of the pairwise MRF of Equation 1, the “forward” BP messages passed to subsequent time steps are defined via the recursion

$$m_{t,t+1}(x_{t+1}) \propto \int_{\mathcal{X}_t} p(x_{t+1} | x_t) p(y_t | x_t) m_{t-1,t}(x_t) dx_t \\ \propto p(x_{t+1} | y_1, \dots, y_t). \quad (7)$$

For continuous \mathcal{X}_t where this update lacks a closed form, *particle filters*^{6, 11} approximate the forward BP messages $m_{t-1,t}(x_t)$ via a collection of L weighted samples, or particles, $\{(x_t^{(\ell)}, w_t^{(\ell)})\}_{\ell=1}^L$. Importance sampling is used to recursively update the particle locations and weights via a single, forward pass through the observations. A variety of proposal distributions $q(x_{t+1} | x_t, y_{t+1})$, which aim to approximate $p(x_{t+1} | x_t, y_{t+1})$, have been suggested.⁶ For example, the “bootstrap filter” samples $x_{t+1}^{(\ell)} \sim p(x_{t+1} | x_t^{(\ell)})$, and incorporates evidence via weights $w_{t+1}^{(\ell)} \propto w_t^{(\ell)} p(y_t | x_t^{(\ell)})$.

For the simple algorithm sketched above, each message update introduces additional approximations, so that the expected variance of the importance weights $w_t^{(\ell)}$ increases over time. Particle filters avoid such sample depletion via a *resampling* operation, in which the highest-weight particles at time t determine a larger proportion of the outgoing message particles $x_{t+1}^{(\ell)}$. The bootstrap filter then becomes:

$$x_{t+1}^{(\ell)} \sim p(x_{t+1} | x_t^{(z_\ell)}), \Pr[z_\ell = m] \propto w_t^{(m)} p(y_t | x_t^{(m)}).$$

After such resampling, outgoing message particles are equally weighted as $w_{t+1}^{(\ell)} = 1/L$, $\ell = 1, \dots, L$. By stochastically selecting the highest-weight particles multiple times, resampling dynamically focuses the particle filter’s computational resources on the most probable regions of the state space.

3. NONPARAMETRIC BP

Although particle filters can be adapted to an extremely wide range of dynamical models and observation types, they are specialized to the structure of temporal filtering problems. Conversely, loopy BP can in principle be applied to graphs of any structure, but is only analytically tractable when all hidden variables are discrete or jointly Gaussian. In this section, we describe an NBP algorithm^{26, 44} that generalizes sequential Monte Carlo methods to arbitrary graphs. As in regularized particle filters,¹¹ we approximate the true BP messages and beliefs by nonparametric density estimates. Importance sampling and MCMC approximations then update these sample-based messages, propagating information from local observations throughout the graph.

3.1. Nonparametric representations

Consider again the BP algorithm of Section 2.2, and suppose

that messages $m_{ji}(x_i)$ are approximated by a set of weighted, discrete samples. If \mathcal{X}_i is continuous and these messages are constructed from independent proposal distributions, their particles will be distinct with probability one. For the message product operation underlying the BP algorithm to produce sensible results, some interpolation of these samples to nearby states is thus needed.

We accomplish this interpolation, and ensure that messages are smooth and strictly positive, by convolving raw particle sets with a Gaussian distribution, or kernel:

$$m_{ji}(x_i) = \sum_{\ell=1}^L w_{ji}^{(\ell)} \mathcal{N}(x_i; x_{ji}^{(\ell)}, \Lambda_{ji}), \quad (9)$$

$$q_i(x_i) = \sum_{\ell=1}^L w_i^{(\ell)} \mathcal{N}(x_i; x_i^{(\ell)}, \Lambda_i). \quad (10)$$

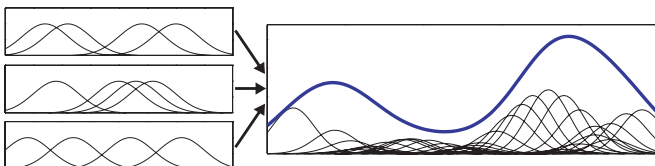
Here, $\mathcal{N}(x; \mu, \Lambda)$ denotes a normalized Gaussian density with mean μ and covariance Λ , evaluated at x . As detailed later, we use methods from the nonparametric density estimation literature to construct these mixture approximations.⁴² The product of two Gaussians is itself a scaled Gaussian distribution, a fact which simplifies our later algorithms.

3.2. Message fusion

We begin by assuming that the observation potential is a Gaussian mixture. Such representations arise naturally from learning-based approaches to model identification.¹⁴ The BP belief update of Equation 3 is then defined by a product of $d = (|\Gamma(i)| + 1)$ mixtures: the observation potential $\psi_i(x_i, y)$, and messages $m_{ji}(x_i)$ as in Equation 9 from each neighbor. As illustrated in Figure 3, the product of d Gaussian mixtures, each containing L components, is itself a mixture of L^d Gaussians. While in principle this belief update could be performed exactly, the exponential growth in the number of mixture components quickly becomes intractable.

The NBP algorithm instead approximates the product mixture via a collection of L independent, possibly importance weighted samples $\{(x_i^{(\ell)}, w_i^{(\ell)})\}_{\ell=1}^L$ from the “ideal” belief of Equation 3. Given these samples, the bandwidth Λ_i of the nonparametric belief estimate (Equation 10) is determined via a method from the extensive kernel density estimation literature.⁴² For example, the simple “rule of thumb” method combines a robust covariance estimate with an asymptotic formula that assumes the target density is Gaussian. While fast to compute, it often oversmooths

Figure 3. A product of three mixtures of $L = 4$ 1D Gaussians. Although the $4^3 = 64$ components in the product density (thin lines) vary widely in position and weight (scaled for visibility), their normalized sum (thick line) has a simple form.



multimodal distributions. In such cases, more sophisticated cross-validation schemes can improve performance.

In many applications, NBP’s computations are dominated by the cost of sampling from such products of Gaussian mixtures. Exact sampling by explicit construction of the product distribution requires $\mathcal{O}(L^d)$ operations. Fortunately, a number of efficient approximate samplers have been developed. One simple but sometimes effective approach uses an evenly weighted mixture of the d input distributions as an importance sampling proposal. For higher-dimensional variables, iterative Gibbs sampling algorithms are often more effective.⁴⁴ Multiscale KD-tree density representations can improve the mixing rate of Gibbs samplers, and also lead to “epsilon-exact” samplers with accuracy guarantees.²⁵ More sophisticated importance samplers⁵ and multiscale simulated or parallel tempering algorithms³⁹ can also be highly effective. Yet more approaches improve efficiency by introducing an additional message approximation step.^{19, 22, 31} By first reducing the complexity of each message, the product can be approximated more quickly, or even computed exactly. When $\psi_i(x_i, y)$ is a non-Gaussian analytic function, we can use any of these samplers to construct an importance sampling proposal from the incoming Gaussian mixture messages.

3.3. Message propagation

The particle filter of Section 2.4 propagates belief estimates to subsequent time steps by sampling $x_{t+1}^{(\ell)} \sim p(x_{t+1} | x_t^{(\ell)})$. The consistency of this procedure depends critically on the HMM’s factorization into properly normalized conditional distributions, so that $\int p(x_{t+1} | x_t) dx_{t+1} = 1$ for all $x_t \in \mathcal{X}_t$. By definition, such conditional distributions place no biases on x_t . In contrast, for pairwise MRFs, the clique potential $\psi_{ij}(x_i, x_j)$ is an arbitrary nonnegative function that may influence the values assumed by either linked variable. To account for this, we quantify the *marginal influence* of $\psi_{ij}(x_i, x_j)$ on x_j via the following function:

$$\varphi_{ij}(x_j) = \int_{\mathcal{X}_i} \psi_{ij}(x_i, x_j) dx_i. \quad (11)$$

If $\psi_{ij}(x_i, x_j)$ is a Gaussian mixture, $\varphi_{ij}(x_j)$ is simply the mixture obtained by marginalizing each component. In the common case where $\psi_{ij}(x_i, x_j) = \tilde{\psi}_{ij}(x_i - x_j)$ depends only on the difference between neighboring variables, the marginal influence is constant and may be ignored.

As summarized in the algorithm of Figure 4, NBP approximates the BP message update of Equation 2 in two stages. Using the efficient algorithms discussed in Section 3.2, we first draw L independent samples $\tilde{x}_j^{(\ell)}$ from a partial belief estimate combining the marginal influence function, observation potential, and incoming messages. For each of these auxiliary particles $\tilde{x}_j^{(\ell)}$, we then interpret the clique potential as a joint distribution and sample particles $x_{ji}^{(\ell)} \in \mathcal{X}_i$ from the conditional density proportional to $\psi_{ij}(x_i, x_j = \tilde{x}_j^{(\ell)})$.

Particle-based approximations are only meaningful when the corresponding BP messages $m_{ji}(x_i)$ are finitely integrable. Some models, however, contain nonnormalizable

potentials that nevertheless encode important constraints. For example, the kinematic tracking and sensor localization applications considered in Section 4 both involve “repulsive” potentials, that encourage pairs of variables to *not* take similar values. In such cases, the NBP algorithm in Figure 4 instead stores the weighted particles needed to evaluate $m_{ji}(\bar{x}_i)$ at any location \bar{x}_i of interest. These messages then influence subsequent iterations via importance weighting.

As illustrated in Figure 2, the BP update of message $m_{ji}(x_i)$ is most often expressed as a transformation of the incoming messages from all *other* neighboring nodes $k \in \Gamma(j) \setminus i$. From Equations 2 and 3, however, it can also be expressed as

$$m_{ji}(x_i) \propto \int_{x_j} \psi_{ij}(x_i, x_j) \frac{q_j(x_j)}{m_{ij}(x_j)} dx_j. \quad (12)$$

This transformation suggests an alternative *belief sampling* form of the NBP algorithm, in which the latest belief estimate provides a proposal distribution for auxiliary particles $\tilde{x}_j^{(l)} \sim q_j(x_j)$. Overcounting of $m_{ij}(x_j)$ may then be avoided via importance weights $\tilde{w}_j^{(l)} \propto 1/m_{ij}(\tilde{x}_j^{(l)})$. Computationally, belief sampling offers clear advantages: computation of new outgoing messages to d neighbors requires $\mathcal{O}(dL)$ operations, versus the $\mathcal{O}(d^2L)$ cost of the approach in Figure 4. Statistically, belief sampling also has potentially desirable properties,^{26, 29} but can be less stable when the number of particles L is small.²²

Figure 4. Nonparametric BP update for the message $m_{ji}(x_i)$ sent from node j to node i , as in Figure 2.

Given input messages $m_{kj}(x_j)$ for each $k \in \Gamma(j) \setminus i$, which may be either kernel densities $m_{kj}(x_j) = \{x_{kj}^{(l)}, w_{kj}^{(l)}, \Lambda_{kj}\}_{l=1}^L$ or analytic functions, construct an output message $m_{ji}(x_i)$ as follows:

1. Determine the marginal influence $\varphi_j(x_j)$ of Equation (11).
2. Draw L independent, weighted samples from the product

$$(\tilde{x}_j^{(l)}, \tilde{w}_j^{(l)}) \sim \varphi_j(x_j) \psi_j(x_j, y) \prod_{k \in \Gamma(j) \setminus i} m_{kj}(x_j).$$

Optionally resample by drawing L particles with replacement according to $\Pr[\tilde{x}_j^{(l)}] \propto \tilde{w}_j^{(l)}$, giving evenly weighted particles.

3. If $\psi_{ij}(x_i, x_j)$ is normalizeable ($\int \psi_{ij}(x_i, x_j) dx_i < \infty$ for all $\bar{x} \in \mathcal{X}_i$), construct a kernel-based output message:

- (a) For each auxiliary particle $\tilde{x}_j^{(l)}$, sample an outgoing particle

$$x_{ji}^{(l)} \sim \psi_{ij}(x_i, x_j = \tilde{x}_j^{(l)})$$

Using importance sampling or MCMC methods as needed.

- (b) Set $w_{ji}^{(l)}$ to account for importance weights in steps 2–4(a).
- (c) Set Δ_j via some bandwidth selection method (see Silverman⁴²).
4. Otherwise, treat $m_{ji}(x_i)$ as an analytic function

$$m_{ji}(x_i) \propto \sum_{l=1}^L \tilde{w}_j^{(l)} \psi_{ij}(x_i, \tilde{x}_j^{(l)})$$

parameterized by the auxiliary particles $\{\tilde{x}_j^{(l)}, \tilde{w}_j^{(l)}\}_{l=1}^L$.

4. ILLUSTRATIVE APPLICATIONS

In this section we show several illustrative examples of applications that use NBP to reason about structured collections of real-valued variables. We first show examples of kinematic tracking problems in computer vision, in which the variables represent the spatial position of parts of an object. We then show how a similar formulation can be used for collaborative self-localization and tracking in wireless sensor networks. Other applications of NBP include deformable contour tracking for medical image segmentation,⁴⁶ image denoising and super-resolution,³⁸ learning flexible models of dynamics and motor response in robotic control,¹⁷ error correcting codes defined for real-valued codewords,^{31, 43} and sparse signal reconstruction using compressed sensing principles.⁴ NBP has also been proposed as a computational mechanism for hierarchical Bayesian information processing in the visual cortex.³²

4.1. Visual tracking of articulated motion

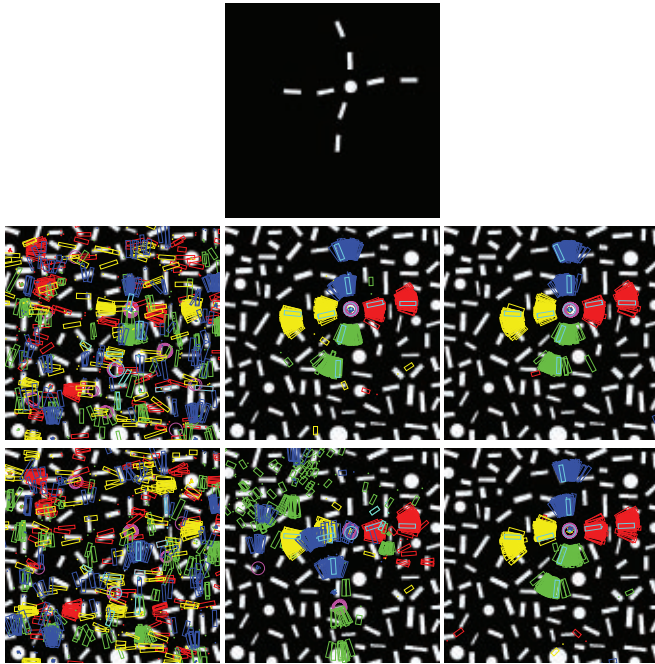
Visual tracking systems use video sequences from one or more cameras to estimate object motion. Some of the most challenging tracking applications involve *articulated* objects, whose jointed motion leads to complex pose variations. For example, human motion capture is widely used in visual effects and scene understanding applications.³³ Estimates of human, and especially hand, motion are also used to build more expressive computer interfaces.⁴⁸

To illustrate the difficulties, we consider a toy 2D object localization problem in Figure 5. The model consists of nine nodes: a central circle, and four jointed arms composed of two rectangular links. The circle node’s state x_0 encodes its position and radius, while each rectangular link node’s state x_i encodes its position, angle, width, and height. Each arm prefers one of the four compass directions, arms pivot around their inner joints, and geometry is loosely enforced via Gaussian pairwise potentials $\psi_{ij}(x_i, x_j)$; for details see Isard.²⁶

Our goal is to find the object in a sea of clutter (white shapes in Figure 5) whose elements look exactly like components of the object. This mimics the difficulties faced in real video sequences: statistical detectors for individual object parts often falsely fire on background regions, and global geometric reasoning is needed to disambiguate them. Applied to this model, NBP’s particles encode hypotheses about the pose x_i of individual object parts, while messages use geometric constraints to propagate information between parts. When all of the true object’s parts are visible, NBP localizes it after a single iteration. By using Gaussian mixture potentials $\psi_i(x_i, y)$ that allow occasional outliers in observed part locations, NBP remains successful even when the central circle is missing. In this case, however, it takes more iterations to propagate information from the visible arms.

Kinematic tracking of real hand motion poses far greater challenges. Even coarse models of the hand’s geometry have 26 continuous degrees of freedom: each finger’s joints have four angles of rotation, and the palm

Figure 5. Detection of a toy, four-armed articulated object (top row) in clutter. We show NBP estimates after 0, 1, and 3 iterations (columns), for cases where the circular central joint is either visible (middle row) or occluded (bottom row).



may take any 3D position and orientation.⁴⁸ The graphical models in Figure 6 instead encode hand pose via the 3D pose of 16 rigid bodies.⁴⁵ Analytic pairwise potentials then capture kinematic constraints (phalanges are connected by revolute joints), structural constraints (two fingers cannot occupy the same 3D volume), and Markov temporal dynamics. The geometry of individual rigid bodies is modeled via quadric surfaces (a standard approach in computer graphics), and related to observed images via statistical color and edge cues.⁴⁵

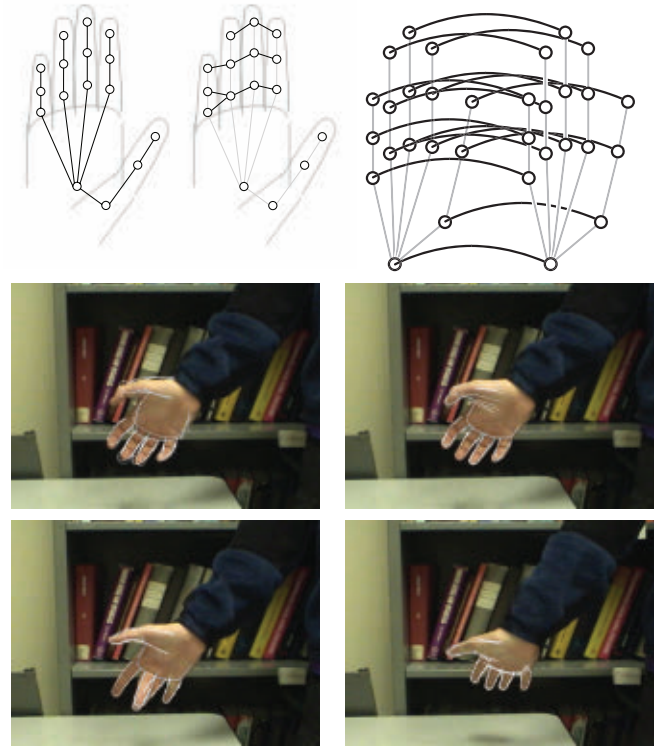
Because different fingers are locally similar in appearance, global inference is needed to accurately associate hand components to image cues. Discretization of the 6D pose variable for each rigid body is intractable, but as illustrated in Figure 6, NBP’s sampling-based message approximations often lead to accurate hand localization and tracking. While we project particle outlines to the image plane for visualization, we emphasize NBP’s estimates are of 3D pose.

Finally, Figure 7 illustrates a complementary approach to multicamera tracking of 3D person motion.⁴¹ While the hand tracker used rigid kinematic potentials, this graphical model of full-body pose is explicitly “loose limbed,” and uses pairwise potentials estimated from calibrated, 3D motion capture data. Even without the benefit of dynamical cues or highly accurate image-based likelihoods, we see that NBP successfully infers the full human body pose.

4.2. Sensor self-localization

Another problem for which NBP has been very successful

Figure 6. Articulated 3D hand tracking with NBP. Top: Graphical models capturing the kinematic, structural, and temporal constraints relating the hand’s 16 rigid bodies. Middle: Given a single input image, projected estimates of hand pose after one (left) and four (right) NBP iterations. Bottom: Two frames showing snapshots of tracking performance from a monocular video sequence.

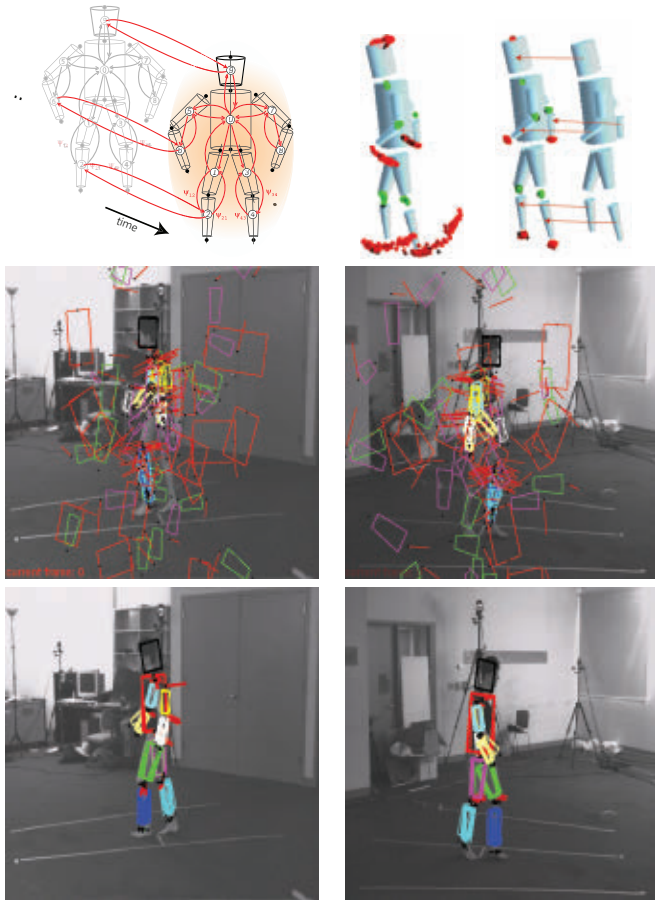


is sensor localization.²² One of the critical first tasks in using ad-hoc networks of wireless sensors is to determine the location of each sensor; the high cost of manual calibration or specialized hardware like GPS units makes self-localization, or estimating position based on local in-network information, very appealing. As with articulated tracking, we will be estimating the position of a number of objects (sensors) using joint information about the objects’ relative positions. Specifically, let us assume that some subset of pairs of sensors $(i, j) \in \mathcal{E}$ are able to measure a noisy estimate of their relative distance (e.g., through signal strength of wireless communication or measuring time delays of acoustic signals). Our measurements y_{ij} tell us something about the relative positions x_i, x_j of two sensors; assuming independent noise, the likelihood of our measurements is

$$p(y | x) = \prod_{(i,j) \in \mathcal{E}} p(y_{ij} | x_i, x_j) = \prod_{(i,j) \in \mathcal{E}} \psi_{ij}(x_i, x_j).$$

We can see immediately that this likelihood has the form of a pairwise graphical model whose edges are the pairs of sensors with distance measurements. Typically we assume a small number of *anchor* sensors with known or partially known position to remove translational, rotational, and mirror image ambiguity from the geometry.

Figure 7. Articulated 3D person tracking with NBP.⁴¹ Top: Graphical model encoding kinematic and dynamic relationships (left), and spatial and temporal potential functions (right) learned from mocap data. Middle: Bottom-up limb detections, as seen from two of four camera views. Bottom: Estimated body pose following 30 iterations of NBP.



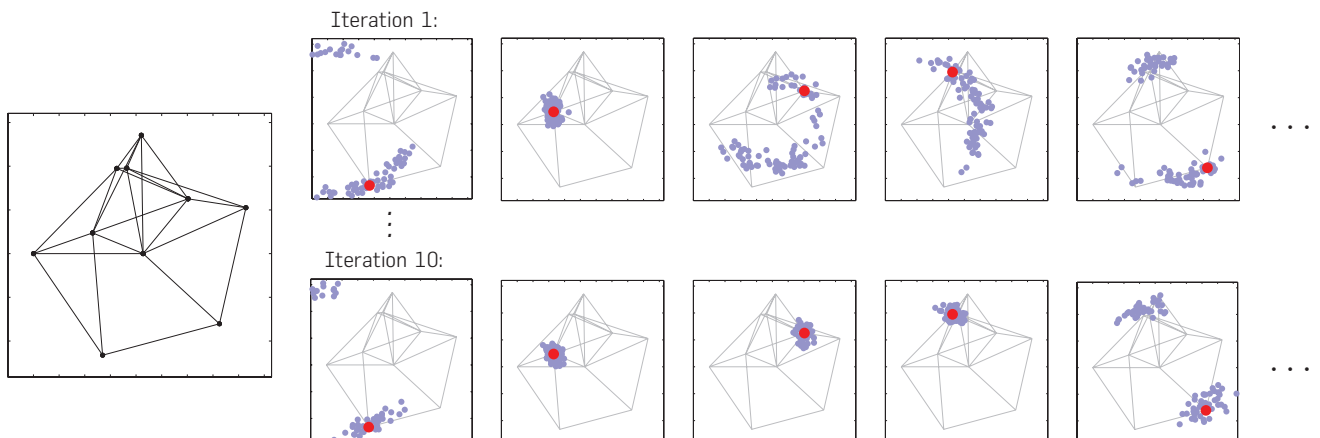
A small 10-sensor network with 24 edges is shown in Figure 8, indicating both the true 2D sensor positions (nodes) and inter-sensor measurements (edges). The beliefs obtained using NBP are displayed on the right, by showing 500 samples from the estimated belief; the true sensor positions are also superimposed (red dots). The initial beliefs are highly non-Gaussian and often fairly diffuse (top row). As information propagates through the graph and captures more of the inter-sensor dependencies, these beliefs tend to shrink to good estimates of the sensor positions. However, in some cases, the measurements themselves are nearly ambiguous, resulting in a bimodal posterior distribution. For example, the sensor located in the bottom right has only three, nearly colinear neighbors, and so can be explained almost as well by “flipping” its position across the line. Such bimodalities indicate that the system is not fully constrained, and are important to identify as they indicate sensors with potentially significant errors in position.

5. DISCUSSION

The abundance of problems that involve continuous variables has given rise to a variety of related algorithms for estimating posterior probabilities and beliefs in these systems. Here we describe several influential historical predecessors of NBP, and then discuss subsequent work that builds on or extends some of the same ideas.

As mentioned in Section 2.2, direct discretization of continuous variables into binned “histogram” potentials can be effective in problems with low-dimensional variables.⁴ In higher-dimensional problems, however, the number of bins grows exponentially and quickly becomes intractable. One possibility is to use domain specific heuristics to exclude those configurations that appear unlikely based on local evidence.^{8, 14} However, if the local evidence used to discard states is inaccurate

Figure 8. NBP for self-localization in a small network of 10 sensors. Left: Sensor positions, with edges connecting sensor pairs with noisy distance measurements. Right: Each panel shows the belief of one sensor (scatterplot), along with its true location (red dot). After the first iteration of message passing, beliefs are diffuse with non-Gaussian uncertainty. After 10 iterations, the beliefs have stabilized near the true values. Some beliefs remain multimodal, indicating a high degree of uncertainty in that sensor’s position due to near-symmetries that remain ambiguous given the measurements.



or misleading, these approximations will heavily distort the resulting estimates.

One advantage of Monte Carlo and particle filtering methods lies in the fact that their discretization of the state space is obtained stochastically, and thus has excellent theoretical properties. Examples include statistical consistency, and convergence rates that do not depend on the dimension.¹⁰ While particle filters are typically restricted to “forward” sequential inference, the connection to discrete inference has been exploited to define smoothing (forward and backward) algorithms,⁶ and to perform resampling to dynamically improve the approximation.³⁵ Monte Carlo approximations were also previously applied to other tree-structured graphs, including junction trees.^{9,29}

Gaussian mixture models also have a long history of use in inference. In Markov chains, an algorithm for forward inference using Gaussian mixture approximations was first proposed by Alspach and Sorenson¹; see also Anderson and Moore.² Regularized particle filters smooth each particle with a small, typically Gaussian kernel to produce a mixture model representation of forward messages.¹¹ For Bayesian networks, Gaussian mixture-based potentials and messages have been applied to junction tree-based inference.¹²

NBP combines many of the best elements of these methods. By sampling, we obtain probabilistic approximation properties similar to particle filtering. Representing messages as Gaussian mixture models provides smooth estimates similar to regularized particle filters, and interfaces well with Gaussian mixture estimates of the potential functions.^{12,14,17,38} NBP extends these ideas to “loopy” message passing and approximate inference.

Since the original development of NBP, a number of algorithms have been developed that use alternative representations for inference on continuous, or hybrid, graphical models. Of these, the most closely related is particle BP, which uses a simplified importance sampling representation of messages, more closely resembling the representation of (unregularized) particle filters. This form enables the derivation of convergence rates similar to those available for particle filtering,²¹ and also allows the algorithm to be extended to more general inference techniques such as reweighted message-passing algorithms.²⁴

Other forms of message representation have also been explored. Early approaches to deterministic discrete message approximation would often mistakenly discard states in the early stages of inference, due to misleading local evidence. More recently, dynamic discretization techniques have been developed to allow the inference process to recover from such mistakes by re-including states that were previously removed.^{7,27,36} Other approaches substitute alternative, smoother message representations, such as Gaussian process-based density estimates.⁴⁰

Finally, several authors have developed additional ways of combining Monte Carlo sampling with the principles of exact inference. AND/OR importance sampling,¹⁶ for example, uses the structure of the graph to improve the statistical

efficiency of Monte Carlo estimates given a set of samples. Another example, Hot Coupling,¹⁸ uses a sequential ordering of the graph’s edges to define a sequence of importance sampling distributions.

The intersection of variational and Monte Carlo methods for approximate inference remains an extremely active research area. We anticipate many further advances in the coming years, driven by increasingly varied and ambitious real-world applications.

Acknowledgments

The authors thank L. Sigal, S. Bhatia, S. Roth, and M. Black for providing the person tracking results in Figure 7. Work of WTF supported by NGA NEGI-1582-04-0004 and by MURI Grant N00014-06-1-0734. Work of ASW supported by AFOSR Grant FA9559-08-1-1080 and a MURI funded through AFOSR Grant FA9550-06-1-0324. □

References

- Alspach, D.L. and Sorenson, H.W. Nonlinear Bayesian estimation using Gaussian sum approximations. *Morgan Kaufmann. IEEE Trans. AC 17*, 4 (Aug. 1972), 439–448.
- Anderson, B.D.O., Moore, J.B. *Optimal Filtering*. Prentice Hall, New Jersey, 1979.
- Andrieu, C., de Freitas, N., Doucet, A., Jordan, M.I. An introduction to MCMC for machine learning. *Mach. Learn. 50* (2003), 5–43.
- Baron, D., Sarvotham, S., Baraniuk, R.G. Bayesian compressive sensing via belief propagation. *IEEE Trans. Sig. Proc. 58*, 1 (2010), 269–280.
- Briers, M., Doucet, A., Singh, S.S. Sequential auxiliary particle belief propagation. In *ICIF* (2005), 705–711.
- Cappé, O., Godsill, S.J., Moulines, E. An overview of existing methods and recent advances in sequential Monte Carlo. *Proc. IEEE 95*, 5 (May 2007), 899–924.
- Coughlan, J., Shen, H. Dynamic quantization for belief propagation in sparse spaces. *Comput. Vis. Image Underst. 106*, 1 (2007), 47–58.
- Coughlan, J.M., Ferreira, S.J. Finding deformable shapes using loopy belief propagation. In *ECCV*, vol. 3, (2002), 453–468.
- Dawid, A.P., Kjærulff, U., Lauritzen, S.L. Hybrid propagation in junction trees. In *Advances in Intelligent Computing* (1995), 87–97.
- Del Moral, P. *Feynman-Kac Formulae: Genealogical and Interacting Particle Systems with Applications*. Springer-Verlag, New York, 2004.
- Doucet, A., de Freitas, N., Gordon, N., eds. *Sequential Monte Carlo Methods in Practice*. Springer-Verlag, New York, 2001.
- Driver, E., Morrell, D. Implementation of continuous Bayesian networks using sums of weighted Gaussians. In *UAI* (1995), 134–140.
- Felzenszwalb, P.F., Huttenlocher, D.P. Pictorial structures for object recognition. *IJCV 61*, 1 (2005), 55–79.
- Freeman, W.T., Pasztor, E.C., Carmichael, O.T. Learning low-level vision. *IJCV 40*, 1 (2000), 25–47.
- Frey, B.J., MacKay, D.J.C. A revolution: Belief propagation in graphs with cycles. In *NIPS 10* (1998), MIT Press, 479–485.
- Gogate, V., Dechter, R. AND/OR importance sampling. In *UAI* (2008), 212–219.
- Grimes, D.B., Rashid, D.R., Rao, R.P. Learning nonparametric models for probabilistic imitation. In *NIPS* (2007), MIT Press, 521–528.
- Hamze, F., de Freitas, N. Hot coupling: A particle approach to inference and normalization on pairwise undirected graphs of arbitrary topology. In *NIPS 18* (2006), MIT Press, 491–498.
- Han, T.X., Ning, H., Huang, T.S. Efficient nonparametric belief propagation with application to articulated body tracking. In *CVPR* (2006), 214–221.
- Heskes, T. On the uniqueness of loopy belief propagation fixed points. *Neural Comp. 16* (2004), 2379–2413.
- Ihler, A., McAllester, D. Particle belief propagation. In *AI Stat. 12* (2009).
- Ihler, A.T., Fisher, J.W., Moses, R.L., Willsky, A.S. Nonparametric belief propagation for self-localization of sensor networks. *IEEE J. Sel. Areas Commun. 23*, 4 (Apr. 2005), 809–819.
- Ihler, A.T., Fisher, J.W., Willsky, A.S. Loopy belief propagation: Convergence and effects of message errors. *JMLR 6* (2005), 905–936.
- Ihler, A.T., Frank, A.J., Smyth, P. Particle-based variational inference for continuous systems. In *NIPS 22* (2009), 826–834.
- Ihler, A.T., Sudderth, E.B., Freeman, W.T., Willsky, A.S. Efficient multiscale sampling from products of Gaussian mixtures. In *NIPS 16* (2004), MIT Press.
- Isard, M. PAMPAS: Real-valued graphical models for computer vision. In *CVPR*, vol. 1 (2003), 613–620.
- Isard, M., MacCormick, J., Achan, K. Continuously-adaptive discretization for message-passing algorithms. In *NIPS* (2009), MIT Press, 737–744.
- Jordan, M.I. Graphical models. *Stat.*

- Sci. 19, 1 (2004), 140–155.
29. Koller, D., Lerner, U., Angelov, D. A general algorithm for approximate inference and its application to hybrid Bayes nets. In *UAI 15* (1999), Morgan Kaufmann, 324–333.
 30. Kschischang, F.R., Frey, B.J., Loeliger, H.-A. Factor graphs and the sum-product algorithm. *IEEE Trans. IT* 47, 2 (Feb. 2001), 498–519.
 31. Kurkoski, B., Dauwels, J. Message-passing decoding of lattices using Gaussian mixtures. In *ISIT* (July 2008).
 32. Lee, T.S., Mumford, D. Hierarchical Bayesian inference in the visual cortex. *J. Opt. Soc. Am. A* 20, 7 (July 2003), 1434–1448.
 33. Moeslund, T.B., Hilton, A., Kruger, V. A survey of advances in vision-based human motion capture and analysis. *Comput. Vision Image Underst.* 104 (2006), 90–126.
 34. Mooij, J.M., Kappen, H.J. Sufficient conditions for convergence of the sum-product algorithm. *IEEE Trans. IT* 53, 12 (Dec. 2007), 4422–4437.
 35. Neal, R.M., Beal, M.J., Roweis, S.T. Inferring state sequences for non-linear systems with embedded hidden Markov models. In *NIPS 16* (2004), MIT Press.
 36. Neil, M., Tailor, M., Marquez, D. Inference in hybrid Bayesian networks using dynamic discretization. *Stat. Comput.* 17, 3 (2007), 219–233.
 37. Pearl, J. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufman, San Mateo, 1988.
 38. Rajaram, S., Gupta, M.D., Petrovic, N., Huang, T.S. Learning-based nonparametric image super-resolution. *EURASIP J. Appl. Signal Process.* (2006), 229–240.
 39. Rudoy, D., Wolf, P.J. Multi-scale MCMC methods for sampling from products of Gaussian mixtures. In *ICASSP*, vol. 3 (2007), IIII-1201–III-1204.
 40. Seeger M. Gaussian process belief propagation. In *Predicting structured data* (2007), 301–318.
 41. Sigal, L., Bhatia, S., Roth, S., Black, M.J., Isard, M. Tracking loose-limbed people. In *CVPR* (2004).
 42. Silverman, B.W. *Density Estimation for Statistics and Data Analysis*. Chapman & Hall, London, 1986.
 43. Sommer, N., Feder, M., Shalvi, O. Low-density lattice codes. *IEEE Trans. Info. Theory* 54, 4 (2008), 1561–1585.
 44. Sudderth, E.B., Ihler, A.T., Freeman, W.T., Willsky, A.S. Nonparametric belief propagation. In *CVPR*, vol. 1 (2003), 605–612.
 45. Sudderth, E.B., Mandel, M.I., Freeman, W.T., Willsky, A.S. Visual hand tracking using nonparametric belief propagation. In *CVPR Workshop on Generative Model Based Vision* (June 2004).
 46. Sun, W., Cetin, M., Chan, R., Willsky, A.S. Learning the dynamics and time-recursive boundary detection of deformable objects. *IEEE Trans. IP* 17, 11 (Nov. 2008), 2186–2200.
 47. Wainwright, M.J., Jordan, M.I. Graphical models, exponential families, and variational inference. *Foundations Trends Mach. Learn.* 1, (2008), 1–305.
 48. Wu, Y., Huang, T.S. Hand modeling, analysis, and recognition. *IEEE Signal Proc. Mag.* (May 2001), 51–60.
 49. Yanover, C., Weiss, Y. Approximate inference and protein-folding. In *NIPS 16* (2003), MIT Press, 1457–1464.
 50. Yedidia, J.S., Freeman, W.T., Weiss, Y. Understanding belief propagation and its generalizations. In G. Lakemeyer and B. Nebel, eds. *Exploring Artificial Intelligence in the New Millennium*. Morgan Kaufmann, 2002.
 51. Yedidia, J.S., Freeman, W.T., Weiss, Y. Constructing free energy approximations and generalized belief propagation algorithms. *IEEE Trans. IT* 51, 7 (July 2005), 2282–2312.

Erik B. Sudderth (sudderth@cs.brown.edu), Brown University, Providence, RI.

Alexander T. Ihler (ihler@ics.uci.edu), University of California, Irvine.

Michael Isard (misard@microsoft.com), Microsoft Research, Mountain View, CA.

William T. Freeman (billf@mit.edu), Massachusetts Institute of Technology, Cambridge, MA.

Alan S. Willsky (willsky@mit.edu), Massachusetts Institute of Technology, Cambridge, MA.

© 2010 ACM 0001-0782/10/1000 \$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 2010. (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**.

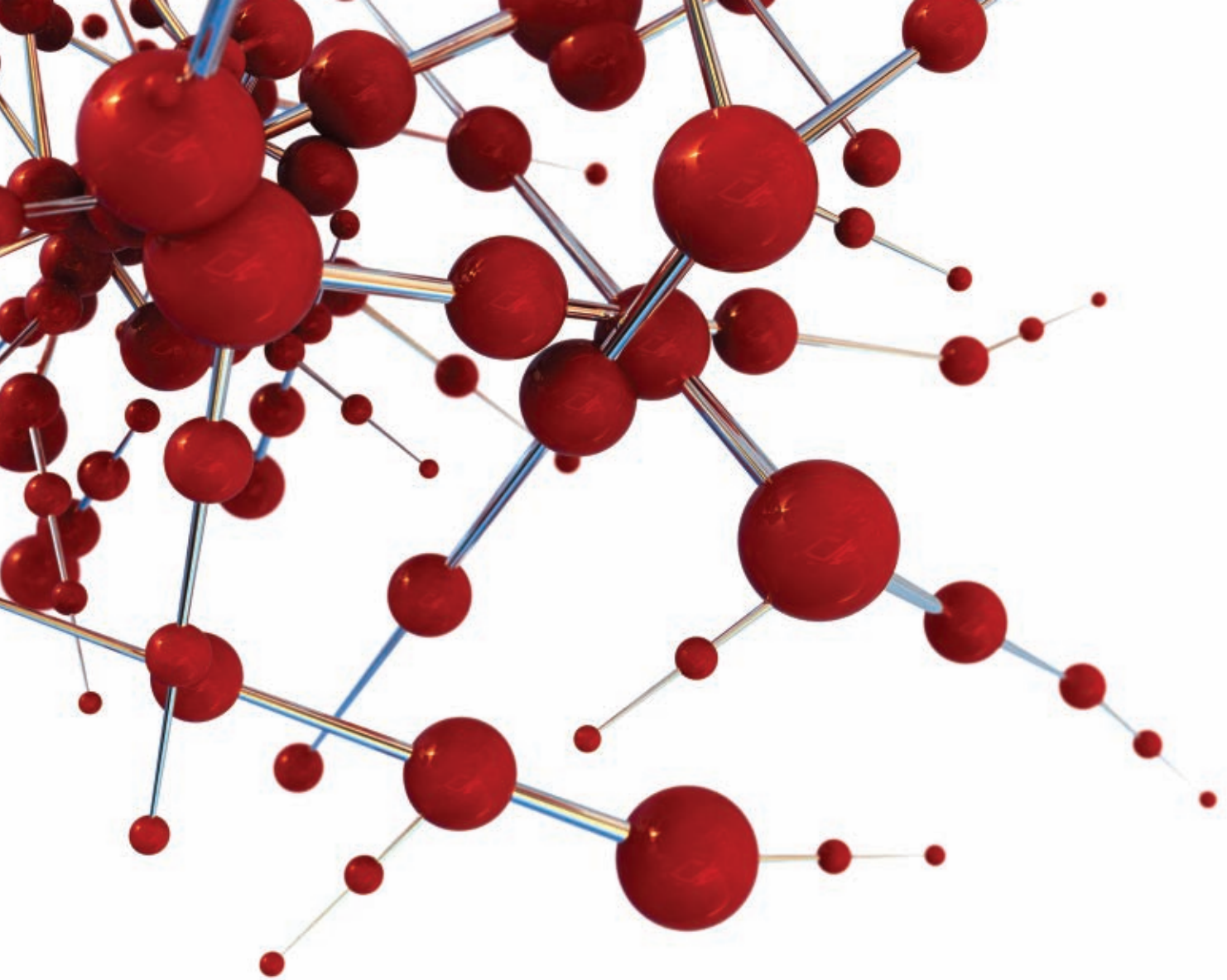
Learn more and apply at:

<http://www.acm.org/life>



Association for
Computing Machinery

Advancing Computing as a Science & Profession



**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.

Auburn University
Department of Computer Science
and Software Engineering
Assistant/Associate Professor

The Department of Computer Science and Software Engineering (CSSE) invites applications for a tenure-track faculty position at the Assistant / Associate Professor level to begin Spring 2011 or Fall 2011. We encourage candidates from all areas of computer science and software engineering to apply. We are especially interested in candidates specializing in software engineering and cyber security. *Candidates selected for these positions must be able to meet eligibility requirements to work in the United States at the time appointment is scheduled to begin and continue working legally for the proposed term of employment; excellent communication skills required.*

Applicants should submit a current curriculum vita, research vision, teaching philosophy, and the names and addresses of three references to Kai H. Chang, Professor and Chair, kchang@eng.auburn.edu (with copy to bjl0002@auburn.edu).

The applicant review process will begin October 15 2010. Detailed announcement of this position can be found at: <http://www.eng.auburn.edu/csse/> Auburn University is an Affirmative Action/Equal Opportunity Employer. **Women and minorities are encouraged to apply.**

Azusa Pacific University
Instructor / Lab Manager

Azusa Pacific University, an evangelical Christian university, announces the opening of a full-time faculty position in the Department of Computer Science beginning fall 2010. The ideal candidate should have a Master's degree in Computer Science or Computer Information Systems and be able to provide evidence of teaching excellence. Women and minorities are encouraged to apply. Position subject to final funding. For a full job description please go to <http://www.apu.edu/provost/employment/positions>.

Duke University
Department of Computer Science
Tenure-track Faculty Positions - Assistant Professor Level

The Department of Computer Science at Duke University invites applications and nominations for tenure-track faculty positions at an assistant professor level, to begin August 2011. We are interested in strong candidates in all active research areas of computer science -- including algorithms, artificial intelligence, computer architecture, computer vision, database systems, distributed systems, machine learning, operating systems, optimization, programming languages, and security -- as well as interdisciplinary areas

such as computational economics and computational biology.

The department is committed to increasing the diversity of its faculty, and we strongly encourage applications from women and minority candidates.

A successful candidate must have a solid disciplinary foundation and demonstrate promise of outstanding scholarship in every respect, including research and teaching. Please refer to www.cs.duke.edu for information about the department and to www.provost.duke.edu/faculty/ for information about the advantages that Duke offers to faculty.

Applications should be submitted online at www.cs.duke.edu/facsearch. A Ph.D. in computer science or related area is required. To guarantee full consideration, applications and letters of reference should be received by November 1, 2010.

Durham, Chapel Hill, and the Research Triangle of North Carolina are vibrant, diverse, and thriving communities, frequently ranked among the best places in the country to live and work. Duke and the many other universities in the area offer a wealth of education and employment opportunities for spouses and families.

Duke University is an affirmative action, equal opportunity employer.

Michigan Technological University
Computer Network Systems Administration
Faculty Opening

The School of Technology at Michigan Technological University in Houghton, Michigan invites applications for a faculty position in the Computer Network Systems Administration (CNSA) program starting January 2010. Primary responsibilities are to instruct students in the CNSA program and establish a record of sustained scholarship. Information about the School of Technology, along with the curriculum, course descriptions, and complete announcement, can be found on-line at: <http://www.tech.mtu.edu/>. Send requested application materials to Dean Frendey at: jimf@mtu.edu

Michigan Technological University is an Equal Opportunity Educational Institution/Equal Opportunity Employer.

Middlebury College
Visiting Assistant Professor

Middlebury College invites applications for a three-year faculty position in computer science, at the rank of Visiting Assistant Professor, beginning September 2011. Specialization is open, with preference for candidates working in systems or interdisciplinary areas. For more information, see <http://www.cs.middlebury.edu/job>.

Review of applications will begin October 15, 2010, and continue until the position is filled.

Middlebury College is an Equal Opportunity Employer, committed to hiring a diverse faculty

to complement the increasing diversity of the student body.

Montana State University
RightNow Technologies Professorships in
Computer Science

The **Montana State University Computer Science Department** is searching for two faculty members at either the Assistant, Associate or Full level, based on experience. Candidates at the Associate or Full level must have established or rising prominence in their field. A three-year start-up package is being provided by RightNow Technologies. Montana State University is a Carnegie Foundation RU/VH research university with an enrollment of approximately 13,000. The website www.cs.montana.edu/faculty-vacancies has information on position requirements and application procedures. ADA/EO/AA/Veterans Preference.

NEC Laboratories America, Inc.
Research Staff Member - Distributed Systems

NEC Laboratories America, Inc. (<http://www.nec-labs.com>) conducts research in support of NEC U.S. and global businesses. The research program covers many areas--reflecting the breadth of NEC business--and maintains a balanced mix of fundamental and applied research.

The Large-Scale Distributed Systems group conducts advanced research in the area of design, analysis, modeling and evaluation of distributed systems. Our current focus is to create innovative technologies to build next generation large-scale computing platforms and to simplify and automate the management of complex IT systems and services. Our researchers have expertise in networking, statistics, modeling, distributed systems, and operating systems. Our group has many ongoing projects, especially in the emerging Cloud Computing area. The group strongly believes in publishing our research and advancing the state-of-the-art. We also build technologies that solve real world problems and ultimately help industry business needs. Many of our research results have been/will be transferred into industry products.

The group is seeking a member to work in the area of distributed systems. The candidate must have deep knowledge and extensive experience in system architecture design and implementation. He/she must have a PhD in CS/CE with strong publication records in the following areas:

- ▶ distributed systems, operating systems
- ▶ virtualization, resource provisioning
- ▶ performance, reliability, dependability and security
- ▶ data centers and cloud computing

For consideration, please forward your resume and a research statement to recruit@nec-labs.com and reference "ASDS-RSM" in the subject line. EOE/AA/MFDV.

Nuance Communications
Senior Software Engineer

Senior Software Engineer wanted to develop speech recognition software for OEM-based mobility handsets. Must have Master's deg. in Comp. Sci, Engineering or a rel. field & 2 yrs. software programming or engineering involving C/C++ programming & debugging & incl. embedded software development. Must have strong proficiency in C language, as demonstrated through employer screening test. Must have strong interpersonal skills for dealing directly with customers both verbally & in writing. Send resume to Melissa Cornell, Employment Specialist, Nuance Communications, Inc., One Wayside Road, Burlington, MA 01803-4613.

Princeton University
Computer Science
Post Doctoral Research Associate
Postdoc Positions In Compilation For Parallel Architecture Performance And Reliability

The Liberty Research Group (<http://liberty.princeton.edu/>) at Princeton University's Department of Computer Science is soliciting applications for a post-doctoral research positions in Compilation for Parallel Architecture Performance and Reliability. The position is a one-year position, with possibility of renewal, starting immediately. The ideal candidate will have recently completed a Ph.D. in Computer Science and will have

expertise and experience with one or more of the following:

- ▶ Compiler Analysis and Optimization
- ▶ Affine Transformations
- ▶ Automatic Parallelization
- ▶ Parallel Applications
- ▶ Parallel Architectures
- ▶ Software Fault Tolerance
- ▶ Programming Language Design
- ▶ The LLVM Compiler

Princeton University is an equal opportunity employer and complies with applicable EEO and affirmative action regulations You may apply online at: <https://jobs.cs.princeton.edu/postdoc-august/>. Questions regarding this position can be sent by e-mail to David August (<http://www.cs.princeton.edu/~august/>), starting the subject line with "POSTDOC"

Rutgers University
Tenure-Track Position

The Department of Computer Science at Rutgers University invites applications for faculty positions at all ranks, with appointments starting in September 2011. The search focuses on theoretical and applied cryptography, although all candidates whose research deals with security will be considered. Applicants for this position must have completed or be in the process of completing a dissertation in Computer Science or a closely related field, and should show evidence of exceptional research

promise, potential for developing an externally funded research program, and commitment to quality advising and teaching at the graduate and undergraduate levels. A hired candidate who has not defended his or her Ph.D. by September will be hired at the rank of Instructor, and must complete the Ph.D. by December 31, 2011 to be eligible for tenure-track title retroactive to start date.

Applicants should go to <http://www.cs.rutgers.edu/employment/> to apply. Required materials are a curriculum vitae, a research statement addressing both past work and future plans, a teaching statement, and three references.

Applications should be received by November 15, 2010 for full consideration.

Rutgers values academic diversity and encourages applications from individuals with a variety of experiences, perspectives, and backgrounds. Females, minorities, dual-career couples, and persons with disabilities are encouraged to apply.

Rutgers is an affirmative action/equal opportunity employer.

Stanford University
Department of Computer Science
Faculty Opening

The Department of Computer Science at Stanford University invites applications for a tenure-track faculty position at the junior level (Assistant or untenured Associate Professor). We give higher priority to the overall originality and promise of the candidate's work than to the candidate's sub-

VCU

Virginia Commonwealth University

ASSISTANT/ASSOCIATE PROFESSOR

School of Engineering Computer Science

The Computer Science Department at Virginia Commonwealth University (VCU) invites applications for a tenure-track position at the rank of Assistant/Associate Professor level. Candidates must have a Ph.D. in computer science and demonstrate the ability to establish, or continue an active, externally funded research program. All specializations will be considered with particular interest in parallel/cloud computing, information security, bioinformatics, and software engineering. VCU, the largest university in Virginia, is an urban Carnegie research I extensive institution (ranked in the top 100 universities in federal R&D expenditures) with a richly diverse community and commitment to multicultural opportunities.

The deadline for applications is November 15, 2010. Candidates are to submit applications electronically to: cmscsearch@vcu.edu as a single pdf file that includes (in order) a cover letter, resume, research and teaching statement, and the names and e-mail addresses of three references. (Reference letters should be provided only upon the request of the search committee).

Virginia Commonwealth University is an Equal Opportunity/Affirmative Action employer. Women, minorities and persons with disabilities are strongly encouraged to apply.



Los Alamos National Laboratory is a premier national security research institution, delivering innovative scientific and engineering solutions for the nation's most crucial and complex problems. Currently, we have an excellent opportunity available for:

ASSOCIATE DIRECTOR, IT

As a key member of the Laboratory's senior management team, the Associate Director for Information Technology will provide leadership, management, oversight, strategic planning and execution of information systems and technology programs and services.

The successful candidate we seek will have experience leading large, complex technical organizations/programs with significant fiscal and programmatic responsibilities. Distinguished track record setting technical direction and goals, developing programs, implementing effective business practices and systems. A BS degree in a related technical discipline is required, while a graduate degree is preferred. The candidate must have the ability to obtain and maintain a DOE Q Clearance, which normally requires US citizenship.

To apply, please visit www.lanl.gov/jobs and reference job number 220100.

Los Alamos supports a drug-free workplace and is an Equal Opportunity Employer.



www.lanl.gov/jobs

area of specialization within Computer Science.

We are seeking applicants from all areas of Computer Science, spanning theoretical foundations, systems, software, and applications. We are also interested in applicants doing research at the frontiers of Computer Science with other disciplines, especially those with potential connections to Stanford's main multidisciplinary initiatives: Energy, Human Health, Environment and Sustainability, the Arts and Creativity, and the International Initiative.

Applicants must have completed (or be completing) a Ph.D., must have demonstrated the ability to pursue a program of research, and must have a strong commitment to graduate and undergraduate teaching. A successful candidate will be expected to teach courses at the graduate and undergraduate levels, and to build and lead a team of graduate students in Ph.D. research. Further information about the Computer Science Department can be found at <http://cs.stanford.edu>. The School of Engineering website may be found at <http://soe.stanford.edu>.

Applications should include a curriculum vita, brief statements of research and teaching interests, and the names of at least four references. Candidates are requested to ask references to send their letters directly to the search committee. Applications and letters should be sent to: Search Committee Chair, c/o Laura Kenny-Carlson, via electronic mail to search@cs.stanford.edu.

The review of applications will begin on Dec. 1, 2010, and applicants are strongly encouraged to submit applications by that date; however, applications will continue to be accepted at least until March 1, 2011.

Stanford University is an equal opportunity employer and is committed to increasing the diversity of its faculty. It welcomes nominations of and applications from women and members of minority groups, as well as others who would bring additional dimensions to the university's research and teaching missions.

Texas A&M University Department of Computer Science and Engineering Senior Faculty Position

In recognition of the increasing importance of computational sciences, the Department of Computer Science and Engineering at Texas A&M University (<http://www.cse.tamu.edu>) is recruiting for a senior faculty position in computational science as broadly defined. This position is one of three new senior hires dedicated to computational science that were created as part of an initiative led by the Institute for Applied Mathematics and Computational Science (<http://iamcs.tamu.edu>). There is considerable startup funding available.

Applications are invited for a **senior faculty position in computational sciences**, starting fall 2011, in the **Department of Computer Science and Engineering of the Dwight Look College of Engineering at Texas A&M University**.

The Department of Computer Science and Engineering has 39 tenured, tenure-track faculty and four senior lecturers. Texas A&M University CSE faculty members are well recognized for contributions to their fields. The department currently has one National Academy of Engineering member, seven IEEE Fellows, one ACM Fellow and over ten PYI/NYI/CAREER awardees. Additional information about the department can be



Worldwide Search for Talent

City University of Hong Kong is a dynamic, fast developing university distinguished by scholarship in research and professional education. As a publicly funded institution, the University is committed to nurturing and developing students' talent and creating applicable knowledge to support social and economic advancement. It is a forward looking university with over 20,000 students and more than 3,000 full-time faculty and staff from diverse international backgrounds and ethnicities. Currently, the University has six Colleges/Schools. Within the next few years, the University aims to recruit **200 more scholars** in various disciplines from all over the world, including **science, engineering, business, social sciences, humanities, law, creative media, energy, environment**, and other strategic growth areas.

Applications and nominations are invited for :

Head of Department of Computer Science

[Ref. A/620/32]

The Department of Computer Science is renowned for both its teaching and research, and a strong team of faculty/teaching staff who engages in internationally significant research in a wide range of areas including Multimedia and Image Computing, Pattern Recognition and Media Retrieval, Distributed Computing and Wireless Networking, Mobile and Pervasive Computing, Information Security, Internet Algorithms and Protocols, Bioinformatics, Databases, Artificial Intelligence and Knowledge Management, Software Engineering and Object Technology, and Web Services Computing.

The Department offers undergraduate programmes, taught master programmes and research degree programmes leading to MPhil and PhD awards. It aims at nurturing graduates with expert knowledge, technical skills and professional ethics that are essential in playing a key role in today's digital world; and engaging in a wide variety of researches with the highest potential for scientific, economic and societal impact.

Qualifications for Appointment

The Head of Department will provide strong academic leadership in the development of teaching and research within the Department, as well as providing effective managerial leadership. Candidates should possess a PhD with strong academic and professional qualifications, substantial relevant experience in tertiary education, and an internationally recognized record of research and scholarship. Candidates should also be able to foster and strengthen external research support from various organizations.

Salary and Conditions of Service

The successful candidate will be offered appointment to an academic rank commensurate with qualifications and experience. The appointee will be offered the headship appointment concurrently for an initial period of three years. Remuneration package will be very attractive, driven by market competitiveness and individual performance. Excellent fringe benefits include gratuity, leave, medical and dental schemes, and relocation assistance (where applicable).

Information and Application

Further information on the post and the University is available at <http://www.cityu.edu.hk>, or from the Human Resources Office, City University of Hong Kong, Tat Chee Avenue, Kowloon, Hong Kong [Fax : (852) 2788 1154 or (852) 3442 0311/email : cssearch@cityu.edu.hk]. Please send the nomination or application with a current curriculum vitae to the Human Resources Office by **7 November 2010**.

Please quote the reference of the post in the application and on the envelope. The University reserves the right to consider late applications and nominations, and not to fill the position. Personal data provided by applicants will be used for recruitment and other employment-related purposes. The University is an equal opportunity employer.

City University of Hong Kong was ranked the 124th among the world's top universities and the 15th in Asia according to the *Times Higher Education/Quacquarelli Symonds* surveys. <http://www.cityu.edu.hk>

found at <http://www.cse.tamu.edu>.

Texas A&M University CSE faculty applicants should apply online at http://www.cse.tamu.edu/dept_faculty. For questions concerning the position, contact: search@cse.tamu.edu.

Texas A&M University is an equal opportunity/affirmative action employer and actively seeks candidacy of women and minorities. Applications are welcome from dual career couples.

The Johns Hopkins University Department of Computer Science Tenure-track Faculty Positions

The Department of Computer Science at The Johns Hopkins University is seeking applications for tenure-track faculty positions. The search

is open to all areas of Computer Science, with a particular emphasis on candidates with research interests in machine learning, theoretical computer science, computational biology, computational aspects of biomedical informatics, or other data-intensive or health-related applications.

All applicants must have a Ph.D. in Computer Science or a related field and are expected to show evidence of an ability to establish a strong, independent, multidisciplinary, internationally recognized research program. Commitment to quality teaching at the undergraduate and graduate levels will be required of all candidates. Preference will be given to applications at the assistant professor level, but other levels of appointment will be considered based on area and qualifications. The Department is committed to building a diverse educational environment; women and

minorities are especially encouraged to apply.

A more extensive description of our search can be found at <http://www.cs.jhu.edu/Search2011>. More information on the department is available at <http://www.cs.jhu.edu>.

Applicants should apply using the online application which can be accessed from <http://www.cs.jhu.edu/apply>. Applications should be received by Dec 1, 2010 for full consideration. Questions should be directed to fsearch@cs.jhu.edu. The Johns Hopkins University is an EEO/AA employer.

Faculty Search

Johns Hopkins University
Department of Computer Science
Room 224 New Engineering Building
Baltimore, MD 21218-2682
Phone: 410-516-8775
Fax: 410-516-6134
fsearch@cs.jhu.edu
<http://www.cs.jhu.edu/apply>

Texas State University-San Marcos Department of Computer Science

Applications are invited for a tenure-track position at the rank of **Assistant Professor**. Applicants must have completed all requirements for a PhD with specialization in software engineering by September 1, 2011. Consult the department recruiting page at <http://www.cs.txstate.edu/recruitment/> for job duties, qualifications, application procedures, and information about the university and the department.

Texas State University-San Marcos will not discriminate against any person (or exclude any person from participating in or receiving the benefits of any of its activities or programs) on any basis prohibited by law, including race, color, age, national origin, religion, sex or disability, or on the basis of sexual orientation. Texas State University-San Marcos is a member of the Texas State University System.

Toyota Technological Institute at Chicago Computer Science Faculty Positions at All Levels

Toyota Technological Institute at Chicago (TTIC) is a philanthropically endowed degree-granting institute for computer science located on the University of Chicago campus. The Institute is expected to reach a steady-state of 12 traditional faculty (tenure and tenure track), and 12 limited term faculty. Applications are being accepted in all areas, but we are particularly interested in:

- ▶ Theoretical computer science
- ▶ Speech processing
- ▶ Machine learning
- ▶ Computational linguistics
- ▶ Computer vision
- ▶ Computational biology
- ▶ Scientific computing

Positions are available at all ranks, and we have a large number of limited term positions currently available.

For all positions we require a Ph.D. Degree or Ph.D. candidacy, with the degree conferred prior to date of hire. Submit your application electronically at: <http://ttic.uchicago.edu/facapp/>

*Toyota Technological Institute at Chicago
is an Equal Opportunity Employer*



The Institute of Information Science (IIS) at Academia Sinica, Taiwan, R.O.C. invites all qualified candidates to apply for the positions of junior and senior research fellows of all ranks (equivalent to the ranks of tenure track assistant, associate and full professors in a regular academic department without teaching responsibility) in all areas of Computer Science. In particular, candidates in the areas of computer systems, machine learning and natural language processing are strongly encouraged to apply.

Academia Sinica is a national academic research institution in Taiwan that conducts research on a broad spectrum of subjects in science and humanities. IIS is committed to high quality research in computer and information science and engineering. In addition to research funding supported by Academia Sinica, external funding through government agencies and industry-sponsored institutions is also available.

Full-time research fellows are free to set their own research directions. IIS currently has about 40 full-time research fellows and close to 300 full-time post doctoral fellows and research assistants. The areas of their current research include Systems Technology, Bioinformatics, Multimedia, Natural Language and Knowledge Processing, Network and Theoretical Computer Science and Parallel Processing.

All Candidates should have a Ph.D. degree in computer science or closely related fields with a strong research and publication record. Senior candidates must demonstrate strong leadership, and have an international reputation evidenced by publications, patents, industrial experiences, or other academic and scholarly achievements. Salary is commensurate with qualifications.

All candidates should send detailed curriculum vitae, and at least three letters of recommendation to

Dr. Pen-Chung Yew, Director
Institute of Information Science
Academia Sinica
Nankang 115, Taipei, Taiwan
TEL: 886-2-2788-3799 ext.1729
FAX: 886-2-2782-4814
E-mail: recruit@iis.sinica.edu.tw

Fluency in Chinese is an advantage, but not required. For additional information about IIS, please visit <http://www.iis.sinica.edu.tw>

Toyota Technological Institute Faculty Position Open

Toyota Technological Institute has an opening for a full professor or a tenure-track professor position in the Department of Advanced Science and Technology, Faculty of Engineering. For more information, please refer to the following website: http://www.toyota-ti.ac.jp/Jinji/home_E.htm

Research field: Intelligent information processing including learning theory and its application, information theory and its application, intelligent systems, computer vision, etc.

Qualifications: The successful candidate must have a Ph.D degree (or equivalent), a record of outstanding research achievements, and the ability to conduct strong research programs in the specified area. The candidate is expected to teach mathematics and programming of the introductory level and machine learning, information theory and signal processing at the advanced level. The supervision of undergraduate and graduate students in their research programs is also required.

Starting date: September 2011, or at the earliest convenience

Documents:

- (1) Curriculum vitae
- (2) List of publications
- (3) Copies of 5 representative publications
- (4) Description of major accomplishments and future plans for research activities and education (3 pages)
- (5) Names of two references with e-mail addresses and phone numbers
- (6) Application form available from our website

Deadline: December 20th, 2010

Inquiry: Committee chair,
Professor Tatsuo Narikiyo
(Tel) +81-52-809-1816,
(E-mail) n-tatsuo@toyota-ti.ac.jp

The above should be sent to:

Mr. Takashi Hirato
Administration Division
Toyota Technological Institute
2-12-1, Hisakata, Tempaku-ku
Nagoya, 468-8511 Japan
(Please write "Application for Intelligent Information Processing Laboratory" in red on the envelope.)

The University of Alabama at Birmingham

Department of Computer and
Information Sciences

Research Assistant Professor

The Department of Computer & Information Sciences at the University of Alabama at Birmingham (UAB) is seeking candidates for a non-tenure-track faculty position at the Research Assistant Professor level beginning November 1, 2010 or until job is filled. Candidates with expertise in Bioinformatics, Artificial Intelligence, and Data Mining who could interact with existing research groups in the School of Medicine and CIS to apply these techniques to the study of genetic diseases (in particular cystic fibrosis) are of interest. Also potential for multidisciplinary collaboration with research groups working in the areas of SNP analysis and the function of introns would be advantageous. The ideal candidate would also have a graduate

degree in microbiology, biochemistry or genetics and actual laboratory experience. Experience as an internal consultant in artificial intelligence/bioinformatics in either industry or academia would be a plus. For additional information about the department please visit <http://www.cis.uab.edu>.

Applicants should have demonstrated the potential to excel in one of these areas and in teaching at all levels of instruction. They should also be committed to professional service including departmental service. A Ph.D. in Computer Science or closely related field is required.

Applications should include a complete curriculum vita with a publication list, a statement of future research plans, a statement on teaching experience and philosophy, and minimally two letters of reference with at least one letter addressing teaching experience and ability. Applications and all other materials may be submitted via email to facapp@cis.uab.edu or via regular mail to:

Search Committee
Department of Computer and Information
Sciences
115A Campbell Hall
1300 University Blvd
Birmingham, AL 35294-1170

Interviewing for the position will begin as soon as qualified candidates are identified, and will continue until the position is filled.

The department and university are committed to building a culturally diverse workforce and strongly encourage applications from women and individuals from underrepresented groups. UAB has an active NSF-supported ADVANCE program

and a Spouse Relocation Program to assist in the needs of dual career couples. UAB is an Affirmative Action/Equal Employment Opportunity employer.

University of Chicago
Department of Computer Science
Professor, Associate Professor, Assistant
Professor, and Instructor

The Department of Computer Science at the University of Chicago invites applications from exceptionally qualified candidates in all areas of Computer Science for faculty positions at the ranks of Professor, Associate Professor, Assistant Professor, and Instructor. The University of Chicago has the highest standards for scholarship and faculty quality, and encourages collaboration across disciplines.

The Chicago metropolitan area provides a diverse and exciting environment. The local economy is vigorous, with international stature in banking, trade, commerce, manufacturing, and transportation, while the cultural scene includes diverse cultures, vibrant theater, world-renowned symphony, opera, jazz, and blues. The University is located in Hyde Park, a pleasant Chicago neighborhood on the Lake Michigan shore.

All applicants must apply through the University's Academic Jobs website, academiccareers.uchicago.edu/applicants/Central?quickFind=51071. A cover letter, curriculum vitae including a list of publications, a statement describing past and current research accomplishments and outlining future research plans, a description of teaching experience, and a list of references **must** be uploaded to be con-

ACCEPT THE NAVY CHALLENGE

Become a member of an elite research and development community involved in basic and applied scientific research and advanced technological development for tomorrow's Navy.

NAVAL RESEARCH LABORATORY

Senior Scientist for Advanced Computing Concepts
ST-1310, \$119,554 to \$179,700* per annum

*Rate limited to the rate for level III of the Executive Schedule (5U.S.C. 5304(g)(2))

Serves as the technical expert in the diverse areas of high performance computing and networking. The position provides expertise in scalable, massively parallel systems, the storage technologies required to service these systems, networking research, and application expertise for application to the memory – and speed-intensive Department of Navy computational problems.

Provides vision and technical direction to research efforts in massively parallel computing and high performance networking, including prototype systems.

As a distinguished scientist and recognized leader in his/her field the incumbent will be called upon to brief DoD senior officials regarding Laboratory research efforts in the above areas, to serve as an NRL liaison to the Navy and other national and international organizations, and to consult on important scientific and programmatic issues. Because of the sensitivity of some of the applications the incumbent must be eligible for TS-SCI security clearance.

Applicants should be recognized as national/international authorities in the above areas of research, and should have demonstrated the scientific vision and organizational skills necessary to bring long term, multi-faceted research programs to successful completion.

A resume or Optional Application for Federal Employment (OF-612) must be received by November 1, 2010. Apply to: Naval Research Laboratory, ATTN: Ginger Kisamore, Code 1810 Announcement #NWO-XXXX-00-K9734979-FL, 4555 Overlook Avenue, SW, Washington, DC 20375-5324 or apply online at <https://hro1.nrl.navy.mil/jobs/index.htm>. Faxed or emailed applications *Will Not* be accepted. Please contact Ginger Kisamore at ginger.kisamore@nrl.navy.mil for more information.



Navy is an Equal Opportunity Employer

sidered as an applicant. Candidates may also post a representative set of publications, to this website. The reference letters can be sent by mail or e-mail to: Chair, Department of Computer Science The University of Chicago 1100 E. 58th Street, Ryerson Hall Chicago, IL. 60637-1581

Or to: recommend-51071@mailman.cs.uchicago.edu (attachments can be in pdf, postscript or Microsoft Word).

Please note that at least three reference letters need to be mailed or e-mailed to the above addresses and one of them must address the candidate's teaching ability. Applicants must have completed all requirements for the PhD except the dissertation at time of application, and must have completed all requirements for the PhD at time of appointment. The PhD should be in Computer Science or a related field such as Mathematics or Statistics. To ensure full consideration of your application all materials [and letters] must be received by **November 19**. Screening will continue until all available positions are filled. The University of Chicago is an Affirmative Action/Equal Opportunity Employer.

**University of Massachusetts Lowell
Computer Science Department
Tenure-Track and Tenured Faculty Positions**

The University of Massachusetts Lowell is a comprehensive university with a national reputation

in science, engineering and technology, committed to educating students for lifelong success in a diverse world and conducting research and outreach activities that sustain the economic, environmental and social health of the region. In February 2009, a campus-wide strategic planning initiative was launched to reposition UMass Lowell as a world-class institution over the next decade. A major component of that initiative is to ensure that diversity and inclusion are in every aspect of our strategic plan. We seek a diverse talented candidate pool to be part of our mission and achievements.

UMass Lowell is located about 25 miles northwest of Boston in the high-tech corridor of Massachusetts. The Computer Science Department has 15 tenured and tenure-track faculty, serving about 220 BS students, 110 MS students, and 55 PhD students. It also offers bioinformatics options at all levels, and a PhD in computational mathematics.

The Computer Science faculty received approximately \$6M in the last two years in external research funding from the NSF, DOD, DOH, and corporations. The department has four NSF CAREER Award Recipients. See <http://www.cs.uml.edu> for more information.

The Computer Science Department at the University of Massachusetts Lowell invites applications for one or two Assistant Professor positions, and one or two positions at the rank of Associate or Full Professor. Positions will start in September 2011. Initial review of applications will begin immediately. The application deadline is **December 1, 2010**. Women and underrepresented minori-

ties are strongly encouraged to apply.

Assistant Professor. Applicants must hold a PhD in computer science or a closely related discipline, have two or more years of teaching and research experience as assistant professors or postdoctoral researchers, have participated in significant federal grant writing, and be committed to developing and sustaining an externally funded research program. We are especially seeking candidates with strong ongoing research who are PIs of funded projects from major US funding agencies. These are tenure-track positions.

Associate or Full Professor. Applicants must hold a PhD in computer science or a closely related discipline, have substantial teaching and research experience, have made significant contributions to their fields on strong ongoing research projects, be current PIs of substantial grants from major US funding agencies, and be committed to sustaining and strengthening an externally funded research program. These are tenured or tenure-track positions depending on qualifications.

All ranks: Outstanding candidates in any major computer science research area will be considered. In addition to developing/expanding a research program, the successful applicant will be encouraged to contribute to the collaborative research of the existing departmental groups. The successful candidate will be expected to teach undergraduate and graduate courses, including department core and specialty areas based on the candidate's expertise, and must have prior effective teaching experience.

How to apply:

1. Submit a cover letter, a current CV, research statement, teaching statement, and selected relevant research publications through our web site at <http://jobs.uml.edu> under "Faculty Positions". **You must apply using the online system. Make sure to apply to the correct rank.**
2. Arrange for at least three letters of recommendation to be included in your application.
3. Optional documents: If available, please include summaries of teaching evaluations.

The University of Massachusetts Lowell is committed to increasing diversity in its faculty, staff, and student populations, as well as curriculum and support programs, while promoting an inclusive environment. We seek candidates who can contribute to that goal and encourage you to apply and to identify your strengths in this area.

**York University
Faculty Applications, Assistant Professor level**

York University, Toronto, Canada: The Department of Computer Science and Engineering in collaboration with the Departments of Biology, and Science and Technology Studies invite faculty applications in the following areas: (i) Visual Neuroscience or Computational Neuroscience of Vision, and; (ii) Digital Media (Computer Graphics) with research interest in Technoscience - both at the Assistant Professor level in the tenure track stream. The deadline for applications is November 30, 2010 with a start date of July 1, 2011. For details, please visit <http://yorku.ca/acadjobs>. York University is an Affirmative Action Employer.



**Department Head
Department of Electrical
Engineering & Computer Science
South Dakota State University
Brookings, SD**

South Dakota State University invites applications and nominations for the position of Department Head of Electrical Engineering & Computer Science. SDSU, the state's land-grant and largest university, is a Carnegie RU/H (high research activity) institution with 12,400 students. The university is seeking an energetic academic leader with strategic vision, outstanding academic credentials and successful administrative experience. The Department Head, who reports to the Dean of Engineering, holds a 12-month position and oversees all of the department's administrative functions including academic, budget, facilities, research and outreach. In FY 2010 the department had 25 base-funded faculty and 390 students enrolled in undergraduate and graduate programs in electrical engineering, computer science and software engineering. The department is enjoying strong growth in enrollments and funded research, strong ties to industry and a beautiful new \$12 million-72,000 sq. ft. building.

The successful applicant must have an earned Ph.D. and distinguished record of performance consistent with appointment as a tenured full professor in a discipline appropriate to the department. He/she must also have a record of innovative and strategic leadership that would apply to a progressive and growing academic environment and a record of effective university administrative experience.

For detailed electronic application instructions, a full description of the position and information on the department, university and community, please visit <http://www.sdstate.edu/eecs/>. For the most complete consideration, applications should be received by Nov. 1, 2010. For questions on the electronic employment process, contact SDSU Human Resources at (605) 688-4128.

South Dakota State University is an AA/EEO employer.

[CONTINUED FROM P. 112] arms to the scientific and computer science community.

It's also a great testament to what can happen when scientists and computer scientists collaborate.

Yes. One of the tools we have produced in a project with the Berkeley Water Center is called SciScope. The researchers have been looking at the hydrology of the Russian River Valley in California, in which the patterns of use have completely changed over the last 50 years. Trees have been chopped down, rivers have been dammed, houses have been built, and all those sorts of things. The U.S. Geological Survey has stream data going back many years, but if you want to combine it with the rainfall data over the same period, that's held by National Oceanic and Atmospheric Administration, a different government agency.

So SciScope enables you to combine the two data sets.

You can add your own data and do new research. It's an example of what I call "scientific mashups," and it is, I think, increasingly how much research will be done in some fields. It's a little like Tim Berners-Lee's vision of the Semantic Web, but in a scientific context.

Astronomy is another field that has benefited from computer science.

The Sloan Digital Sky Survey changed everything, because it generated a high-resolution survey of 25% of the night sky. So, instead of an astronomer getting time on a telescope to look at a particular star system, going back to the university, analyzing the data, and publishing the results with one or two grad students, you've now got data on more than 300 million celestial objects available to study. In this case, the data is published before any detailed analysis has been done.

Gray was instrumental in building online databases to house the Sloan Digital Sky Survey data.

Jim and Alex Szalay also thought they could apply the same sort of infrastructure to a sensor network, so we built a sensor network in the grounds of Johns Hopkins University to investigate soil science. The exciting thing


"We are now seeing the emergence of a fourth paradigm for scientific research, namely data-intensive science."

is that a similar sensor network is now being deployed in Latin America, in the Atlantic rainforest near São Paulo.

What have these projects taught you about fostering meaningful collaboration between the scientific and computer science communities?

I've come to the conclusion that you cannot force scientists to adopt a technology no matter how useful you think it would be for them! You have to get as close to their way of working as possible and give them an immediate win. You can't say, "Go climb this cliff, and at the top there's a reward." So you need to form a partnership where there's an early win for the scientist and a win for you in that they're using at least some of your great research technology, suitably packaged to be usable by scientists.

What sort of reception has *The Fourth Paradigm* received?

It's been very complimentary, which is gratifying, and there's been a huge explosion on Twitter and in the blogosphere. We're working on ideas for a follow-up, and I'm working with the National Science Foundation's Advisory Committee on Cyberinfrastructure on a data task force. It would be premature to say we know exactly what people need, since that's what the scientific community has to tell us. We haven't got there yet, and that's one of the reasons why it's a very exciting time in science and computer science. 

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

© 2010 ACM 0001-0782/10/1000 \$10.00



ACM
Transactions on
Reconfigurable
Technology and
Systems

ACM Transactions on
Reconfigurable Technology
and Systems

SPECIAL SECTION ON THE 10TH INTERNATIONAL WORKSHOP ON FRBS

Article 1 11 pages	D. Burt M. Lee	Introduction
Article 2 10 pages	A. Saha M. Lee	Guest Editorial
Article 3 10 pages	S. Maitra M. Lee T. Saha T. Saha T. Saha T. Saha	Equipment of Shared Data Access in FRBS using Multiple Configurations
Article 4 10 pages	S. Maitra A. Saha	Statistical Analysis and Power Estimation-Based Modeling and Power Assignment in FRBS
Article 5 10 pages	T. Saha M. Lee M. Lee M. Lee	A Flexible Compiler with a Reconfigurable Processor

Continued on Back Cover

Association for Computing Machinery
International Computing and Communications Institute

◆ ◆ ◆ ◆ ◆

This quarterly publication is a peer-reviewed and archival journal that covers reconfigurable technology, systems, and applications on reconfigurable computers. Topics include all levels of reconfigurable system abstractions and all aspects of reconfigurable technology including platforms, programming environments and application successes.

◆ ◆ ◆ ◆ ◆

www.acm.org/trets
www.acm.org/subscribe

 Association for Computing Machinery

Q&A

Gray's Paradigm

Tony Hey talks about Jim Gray and his vision of a new era of collaborative, data-intensive science.

TONY HEY, VICE PRESIDENT of the External Research Division of Microsoft Research, has long straddled the scientific and computational worlds. Hey began his career as a particle physicist at the University of Southampton before changing fields and serving as head of its School of Electronics and Computer Science. Prior to his appointment at Microsoft, Hey served as director of the United Kingdom's e-Science Program, where he worked to develop technologies to enable collaborative, multidisciplinary, and data-intensive science. Here, he talks about a book of essays he co-authored, *The Fourth Paradigm*, which commemorates the work of his late colleague Jim Gray and points the way to a new era of scientific collaboration.

The title of your book, *The Fourth Paradigm*, refers to the idea that we need new tools to cope with the explosion of data in the experimental sciences.

Jim Gray's insight was that experimental science and theoretical science have been with us since Newton, and over the last 50 years, computational science has matured as a methodology for scientific research. Jim thought that we are now seeing the emergence of a fourth paradigm for scientific research, namely data-intensive science. For this, researchers need a different set of skills from those required for experimental, theoretical, and computational science.

Different skill sets such as?

For data-intensive science, researchers need a totally new set of skills such as an understanding of data mining,



Tony Hey speaking at the ninth annual Microsoft Research Faculty Summit, which brought together 400 academics from 150 universities across five continents.

data cleansing, data visualization, and how relational databases work. The new data-intensive research paradigm does not replace the other ones—it's quite clear that data-intensive science uses both theory and computation.

How did you come to be involved in this line of research?

I first met Jim Gray in 2001, when I was running the U.K.'s e-Science Program. In discussions with Jim over the next five years, I came to agree with his view that the computer science community can really make a difference to scientists who are trying to solve difficult problems.

And that's an idea you carry on in your work with Microsoft?

Indeed. Computer science has powerful technologies it can offer scientists, but also things it can learn from tackling some of the difficult scientific challenges. So I really have a wonderful job, working both with great computer scientists and with great scientists.

The essays in *The Fourth Paradigm* focus on new research in areas like environmental science, health, infrastructure, and communication.

There are important problems facing the world that we need to solve. The book is a call to [CONTINUED ON P. 111]

Introducing:

XRDS

The ACM Magazine for Students

XRDS delivers the tools, resources, knowledge, and connections that computer science students need to succeed in their academic and professional careers!

The All-New *XRDS: Crossroads* is the official magazine for ACM student members featuring:

- › Breaking ideas from top researchers and PhD students
- › Career advice from professors, HR managers, entrepreneurs, and others
- › Interviews and profiles of the biggest names in the field
- › First-hand stories from interns at internationally acclaimed research labs
- › Up-to-date information on the latest conferences, contests, and submission deadlines for grants, scholarships, fellowships, and more!



Also available

The All-New XRDS.acm.org

XRDS.acm.org is the new online hub of *XRDS* magazine where you can read the latest news and event announcements, comment on articles, plus share what's happening at your ACM chapter, and more. Get involved by visiting today!

XRDS.acm.org



Association for
Computing Machinery

Advancing Computing as a Science & Profession

INTERACT 2011

13th IFIP TC3 Conference In Human-Computer Interaction
Lisbon, Portugal

September, 5-9 2011



www.interact2011.org

Honorary Chairs:

Dun Norman, Larry Constantine,
Annelise Mark Pejtersen

General Conference Chairs:

Joaquim Jorge, Philippe Palanque

Full Research Papers Chairs:

Nicholas Graham, Nuno Nunes

Short Papers Chairs:

Daniel Gonçalves, Janet Wesson

Special Interest Group (SIGs) Chairs:

Gerrit van der Veer, Teresa Romão

Panels Chairs:

Regina Bernhaupt, Peter Forbrig,
Nuno Correia

Interactive Posters Chairs:

Monique Noirhomme-Fraiture,
Adérito Marcos

Demonstrations Chairs:

Verónica Orvalho, Greg Phillips

Doctoral Consortium Chairs:

Gitte Lindgaard, Manuel João Fonseca

Industrial Programme Chairs:

Antonio Câmara, Miguel Dias, Stacy
Hobson, Oscar Pastor, Virpi Roto

Workshops Chairs:

Julio Abascal, Nuno Guimarães

Tutorials Chairs:

José Gressat Campos, Paula Kotze

Student Design Competition Chairs:

Simone Diniz Junqueira Barbosa,
Luis Carrizo

Organizational Overviews Chairs:

Teresa Chambel, Mary Czerwinski

Publicity Chairs:

Paula Alexandra Silva, Tiago Guerreiro

Keynote speakers co-Chairs:

John Karat, Jean Vanderdonckt

Student Volunteers co-Chairs:

Xavier Ferre, Effie Law

Publications co-Chairs:

Pedro Campos, Marco Winckler

Website Chairs:

Gerhard Weber, Alfredo Ferreira

Local Organization Chairs:

Alfredo Ferreira, Pauline Jepp

Conference Theme

The conference theme, *Building Bridges*, recognizes the interdisciplinary and intercultural spirit of Human-Computer Interaction (HCI) research. The conference welcomes research that bridges disciplines, cultures and societies. Within the broad umbrella of HCI, we seek high quality contributions opening new and emerging HCI disciplines, bridging cultural differences, and tackling important social problems. INTERACT 2011 provides a forum for practitioners and researchers to discuss all aspects of HCI, including these challenges.

Call for Submissions

INTERACT 2011 solicits submissions in a broad range of categories, addressing all aspects of Human-Computer Interaction. Submissions are welcome in the form of long and short research papers; workshop, tutorial, panel and special interest group proposals; industrial reports; interactive experience posters; organizational overviews, and applications to join the doctoral consortium and the student design competition.

Submission Categories

Submissions will be peer-reviewed by an international panel of experts and are invited in the following track categories:

- Full Research Papers
- Short Papers
- Special Interest Groups
- Panels
- Interactive posters
- Demonstrations
- Doctoral Consortium
- Industrial Programme
- Workshops
- Tutorials
- Student Design Competition
- Organizational Overviews

Important Dates

Conference:
September 5th to 9th, 2011

Full Research Paper Submission:
January 10th, 2011 (abstract)
January 24th, 2011 (paper)

Tutorial and Workshop Submission:
March 21th, 2011

Submissions to Other Categories:
April 7th, 2011

Conference Language

INTERACT 2011 is an international conference whose official language is English. All submissions must be in English.

Follow us on:  tinyurl.com/facebook-interact2011

 twitter.com/interact2011

 SIGCHI



ifip

www.interact2011.org