COMMUNICATIONS OF THE ACM

CACM.ACM.ORG

04/2009 VOL.52 NO.04

A Direct Path to Dependable Software

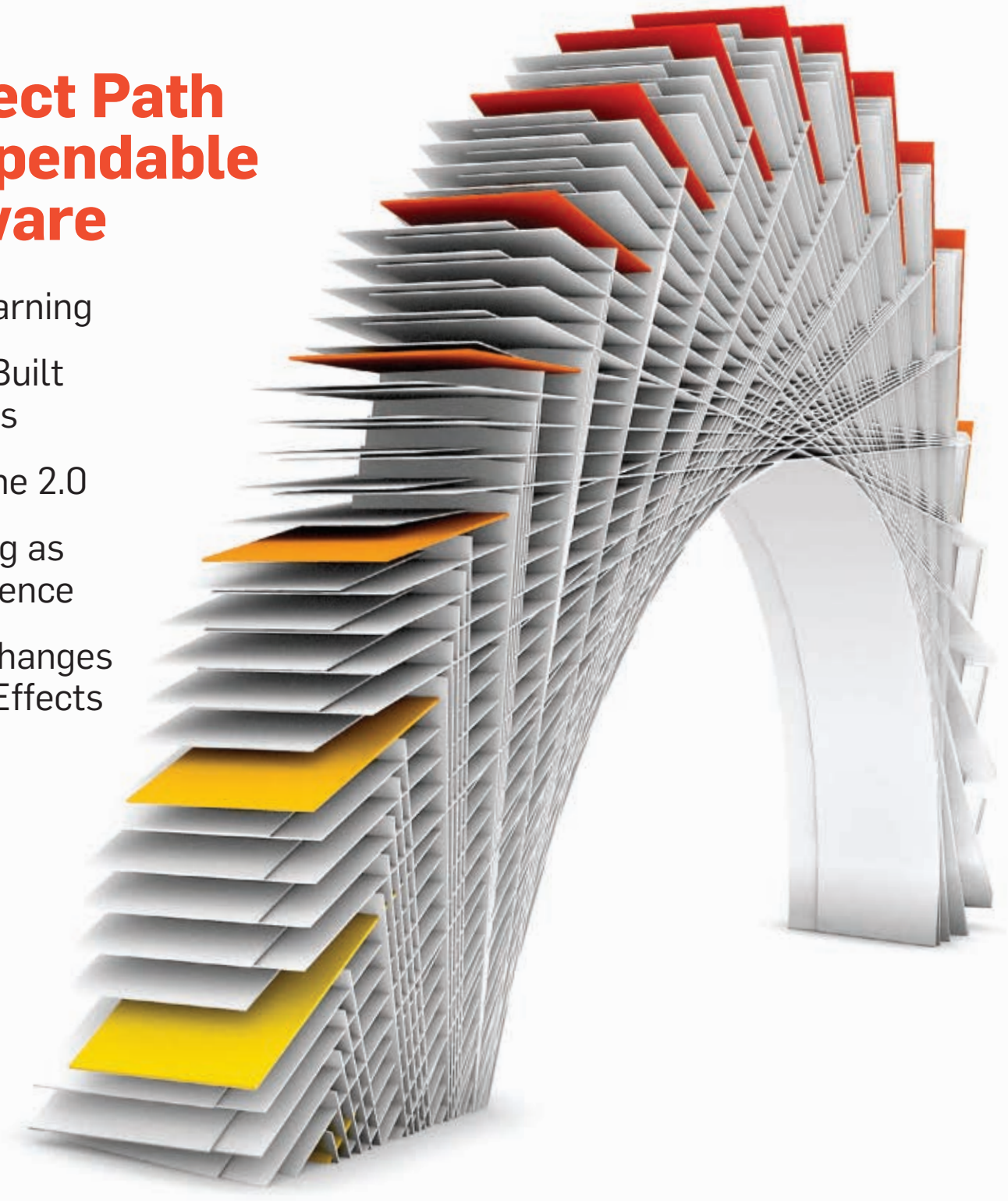
Active Learning

Purpose-Built Languages

Cybercrime 2.0

Computing as Social Science

System Changes and Side Effects



Association for Computing Machinery

RecSys'09: Third ACM Conference on Recommender Systems

New York City, NY, USA • October 22-25, 2009

Paper Submission: May 8, 2009

ORGANIZING COMMITTEE

General Chairs:
Lawrence Bergman, IBM Research, USA
Alex Tuzhilin, New York University, USA

Program Chairs:
Robin Burke, DePaul University, USA
Alexander Felfernig,
Graz University of Technology, Austria
Lars Schmidt-Thieme,
University of Hildesheim, Germany

Industry Chairs:
John Ciancutti, Netflix, USA
Paul Lamere, Sun Microsystems, USA

Workshop Chairs:
Joseph Konstan, University of Minnesota, USA
Sean McNee, FTI Technology, USA

Doctoral Symposium Chairs:
Michael O'Mahoney, University College Dublin, Ireland
Paul Resnick, University of Michigan, USA

Publicity Chair:
Zan Huang, Pennsylvania State University, USA

Asian Liaison:
Dong Zhang, Google Research, China

European Liaison:
Alexandros Nanopoulos,
University of Hildesheim, Germany

RecSys'09: The Third ACM Conference on Recommender Systems builds on the success of Recommenders 06 Summer School in Bilbao, Spain, RecSys'07 in Minneapolis, USA, and RecSys'08 in Lausanne, Switzerland. Many members of the practitioner and research communities valued the rich exchange of ideas made possible by the shared plenary sessions at these events. **RecSys'09** will promote the same close interaction among practitioners and researchers, reaching a wider range of participants including those from Europe and Asia. Published papers will go through a full peer review process. The conference proceedings are expected to be widely read and cited. In addition to a regular technical program, there will be tutorials covering the state-of-the-art of this domain, a doctoral consortium, an industrial program comprised of keynote speakers and practice/industry-paper tracks, and special-topic workshops.

TOPICS OF INTEREST INCLUDE (but are not limited to):

- Case studies of recommender system implementations
- Conversational recommender systems
- Context-aware and multidimensional recommender systems
- Evaluation of recommender systems
- Group recommenders
- The impact of recommenders in practice
- Innovative recommender applications
- Novel paradigms of recommender systems
- Personalization
- Recommendation algorithms
- Recommendation in social networks
- Recommender system interfaces
- Scalability issues
- Security and privacy
- Semantic web technologies for recommender systems
- Theoretical aspects of recommender systems
- User modeling and recommender systems
- User studies

More information at http://recsys.acm.org

Congratulations

2008 ACM Distinguished Members

ACM honors 37 new inductees as Distinguished Members in recognition of their contributions to both the practical and theoretical aspects of computing and information technology

2008 ACM Distinguished Engineers

David G. Belanger
AT&T

Brian A. Berenbach
Siemens Corporate Research

Krishnendu Chakrabarty
Duke University

Jen-Yao Chung
IBM T.J. Watson Research Center

Susan M. Dray
Dray & Associates, Inc.

Tim Duval
Health Net, Inc.

Jerrold M. Grochow
MIT

Mike A. Marin
IBM, Costa Mesa, CA

Dejan S. Milojicic
Hewlett-Packard Laboratories

Daniel S. Whelan
IBM, Costa Mesa, CA

2008 ACM Distinguished Scientists

Douglas C. Burger
Microsoft Research

Murray Campbell
IBM T.J. Watson Research Center

Yih-Farn Robin Chen
AT&T Labs – Research

James R. Cordy
Queen's University

Ernesto Damiani
University of Milan

Andreas Girgensohn
FX Palo Alto Laboratory

Robert J. Hall
AT&T Labs – Research

Hermann Kaindl
Vienna University of Technology

Craig A. Knoblock
University of Southern California

Donald H. Kraft
Louisiana State University

Arif A. Merchant
Hewlett Packard Laboratories

Marc A. Najork
Microsoft Research

Mitsunori Ogihara
University of Miami

Massoud Pedram
University of Southern California

Frederick E. Petry
Naval Research Laboratory

Vir V. Phoha
Louisiana Tech University

Vincenzo Piuri
University of Milan

Paul Walton Purdom
Indiana University

Dragomir R. Radev
University of Michigan

Louiqa Raschid
University of Maryland

Debanjan Saha
IBM

Jeffrey O. Shallit
University of Waterloo

Pradip K. Srimani
Clemson University

Mark A. Stalzer
California Institute of Technology

Frank Tip
IBM Research

Michael Waidner
IBM Corporation

Richard C. Waters
Mitubishi Electric Research Labs

Association for Computing Machinery

Advancing Computing as a Science & Profession

http://distinguished.acm.org

COMMUNICATIONS OF THE ACM

Departments

Last Byte

News

Rebooting Computing Summit

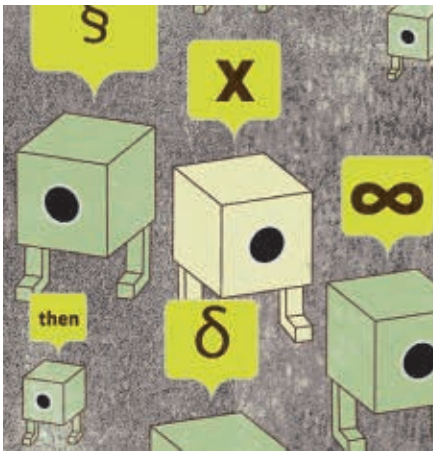
Viewpoints

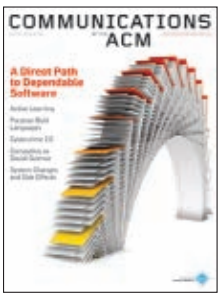
Association for Computing Machinery
Advancing Computing as a Science & Profession

PHOTOGRAPH COURTESY OF THE REBOOTING COMPUTING SUMMIT

04/2009
VOL. 52 NO. 4

Practice





About the Cover: Mikael Christensen, a Danish computer scientist and generative artist, created this 3D-bridge using open source software he wrote and calls Structure Synth. When he is not generating art, he is creating bioinformatics tools as one of the founders of Molegro, developers of novel high-quality drug discovery and data mining software. For more information about Christensen and to experience more of his artwork, see http://blog.hvidtfeldts.net/.

ILLUSTRATION BY JOHN HERSEY

Contributed Articles

Review Articles

Research Highlights

Virtual Extension

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

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

COMMUNICATIONS OF THE ACM

A monthly publication of ACM Media

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

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

BPA Audit Pending

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

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

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

COMMUNICATIONS OF THE ACM

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

POSTMASTER

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

Association for Computing Machinery

Printed in the U.S.A.

DOI:10.1145/1498765.1498767

Andrew McGettrick

Computing Education Matters

The student enrollment crisis in computer science has propelled the need to re-examine all aspects of computing education on a global scale. This disturbing drop has occurred at a time when there is a strong need to recruit more participants into the field and to engender an interest both in the discipline itself and in related innovation. For those of us in education, this downslide has been of great concern. Daunting challenges such as "transforming computing education" and "rebooting computing" (see story on page 19) are high on the agenda. ACM's Education Board and Education Council, charged with promoting computer science education in every possible way, have made enrollment a key focus of attention.

For background, the Education Board has existed within ACM for over three decades. Over the years, the Board has initiated important education activities regarding computer science curriculum developments as well as provided support and encouragement for projects such as Eric Roberts's noted work on the Java Task Force,[5] Peter Denning's work on Great Principles,[3] and Lillian Cassel's work on ontology.[2]

Over the last four years, the Board's activities were restructured and the ACM Education Council was born to bring together the educational and accreditation activities existing throughout ACM's various committees, task forces, and special interest groups. Part of the strategy for revamping the Education Board and the Education Council was to include greater industry representation. Due to this realignment, the work of the Education Board itself was reshaped with considerable emphasis on managing the work of the Education Council.

The Education Council meets about every eight months to keep members abreast of the educational concerns from industry, high-school teachers, as well as those involved in K–12 education. The Education Council also keeps track of the activities of professional bodies such as the National Science Foundation and NCWIT. Moreover, a vital role for the Education Council is to adopt an international perspective in identifying the concerns in computing education and to respond by undertaking activities that will ideally have a positive impact.

Some of the recent accomplishments of the Education Board and the Education Council include:

- The completion of a major undertaking in curriculum guidance in the form of the five volumes of CC2001: namely in Computer Science (2001 with an update in 2008); Information Systems (2002); Software Engineering (2004); Computer Engineering (2004); and Information Technology (2009). The board also finalized an Overview Report (2006) on this project (see ACM Educational Activities[1]).
- Producing and distributing approximately one million copies of a brochure promoting the many positive images of computing to middle- and high-school students.[4] The brochure and accompanying Web site were designed, with support from the Computer Science Teachers Association (CSTA), to increase the visibility of computer science in an encouraging way to a young audience.
- Supporting ACM's *Journal of Educational Resources in Computing* (JERIC) as it transformed into *Transactions on Computing Education*, with a first issue due this month.
- Creating a comprehensive chart of all the educational activities and initiatives within ACM and making it widely available.
- Supporting an initial computing education summit in China as well as a number of European conferences under the auspices of Informatics Education Europe.

Together, ACM's Education Board and Education Council have established an effective pattern of activities and accomplishments among their many programs and initiatives. Their primary activities of curricular guidance will continue and even expand. Working closely with CSTA and K–12 is vital to move educational initiatives in the upward direction. Above all, the Education Board must continue to ensure there is an international perspective and a leadership dimension to its activities. All of these programs and more will be summarized twice a year in *inroads*, the quarterly publication from ACM's special interest group on computer science education (SIGCSE).

While successes have been many, there are still many challenges ahead for the education community. Projects and initiatives designed to reverse declining enrollment in computing disciplines must proliferate and prevail if we are to succeed in stemming the enrollment downturn. One potential catalyst for the cause will be the adoption of new technological developments (for example, involving multi-core processors, IBM's racetrack memory, and vastly enhanced levels of interconnectivity) that are poised to transform the computing community and those drawn to it. As always, ACM will be at the forefront continually revitalizing its Education Board and the Education Council and seeking new and inspiring ways to address the challenges of the day. C

References

1. ACM Educational Activities; www.acm.org/education.
2. Cassel, L. Computing Ontology; http://what.csc.villanova.edu/twiki/bin/view/Main/OntologyProject.
3. Denning, P.J. Great Principles; cs.gmu.edu/pjd/GP.
4. Education Board and CSTA; computingcareers.acm.org.
5. Roberts, E. Java Task Force; jtf.acm.org.

Andrew McGettrick (andrew.mcgettrick@cis.strath.ac.uk) is a professor of computer science at the University of Strathclyde, Glasgow, Scotland, and chair of ACM's Education Board and Education Council.

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

Dear Colleague,

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

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

MEMBER BENEFITS INCLUDE:

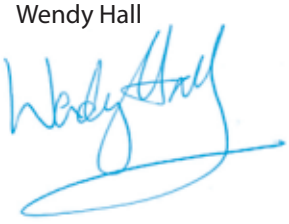
- Access to ACM's **Career & Job Center** offering a host of exclusive career-enhancing benefits
- **Free e-mentoring services** provided by MentorNet®
- **Full access to over 3,000 online courses** from SkillSoft®
- **Full access to 600 online books** from Safari® Books Online, featuring leading publishers, including O'Reilly (Professional Members only)
- **Full access to 500 online books** from Books24x7®
- A subscription to ACM's flagship monthly magazine, ***Communications of the ACM***
- Full member access to the new ***ACM Queue*** website featuring blogs, online discussions and debates, plus video and audio content
- The option to subscribe to the full **ACM Digital Library**
- The **Guide to Computing Literature**, with over one million searchable bibliographic citations
- The option to connect with the **best thinkers in computing** by joining **34 Special Interest Groups** or **hundreds of local chapters**
- **ACM's 40+ journals and magazines** at special member-only rates
- ***TechNews***, ACM's tri-weekly email digest delivering stories on the latest IT news
- ***CareerNews***, ACM's bi-monthly email digest providing career-related topics
- ***MemberNet***, ACM's e-newsletter, covering ACM people and activities
- **Email forwarding service & filtering service**, providing members with a free acm.org email address and **Postini** spam filtering
- And much, much more

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

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

Sincerely,

Wendy Hall



President
Association for Computing Machinery

Association for Computing Machinery

Advancing Computing as a Science & Profession

Association for Computing Machinery

Advancing Computing as a Science & Profession

membership application & *digital library* order form

Priority Code: ACACM28

You can join ACM in several easy ways:

Online	Phone	Fax
http://www.acm.org/join	*+1-800-342-6626 (US & Canada)* *+1-212-626-0500 (Global)*	*+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:10.1145/1498765.1498769

What Role for Computer Science in the War on Terror?

THE CONTRIBUTED ARTICLE "The Topology of Dark Networks" by Jennifer Xu and Hsinchun Chen (Oct. 2008) ignored sensitive cultural issues while addressing a subject that might by itself offend some people in Muslim societies, including those in the Middle East. The software system it described for fighting what some might call "Islamic terrorism" represents a highly charged political subject. A more appropriate place to publish would have been in a publication sponsored by, say, the U.S. Department of Defense, Central Intelligence Agency, or Federal Bureau of Investigation. ACM, which claims to be independent, with a clear mission to advance computer science while being open to members from around the world and free of geographic, ethnic, religious, or political affiliations, should stick to this mission and not involve itself in the so-called War on Terror.

Science is a universal language that should be used to bridge gaps between cultures, promote understanding and cooperation, and avoid worsening damage caused by politicians who push the world toward trouble. ACM should not take on such a sensitive subject that only increases tensions and does not make the world a better place.

This is my personal opinion. I would not seek to impose it on or cause offense to anyone.

Othman El Moulat, Rabat, Morocco

Xu Responds:

We apologize if our article appeared to be targeting particular groups. This was certainly not our intent. Our research tried to address the new Dark Network phenomenon using selected examples and available datasets. Our hope is to develop advanced, science-based, data-driven intelligence and security-informatics techniques that help analyze and understand illicit covert communication and interaction networks. We agree that computing research should not be used for political purposes. We also hope that our research supports the study and understanding of deeply complex social phenomena.

Jennifer Xu, Waltham, MA
Hsinchun Chen, Tucson, AZ

What Gates's Most Enduring Legacy Should Be

Michael Cusumano's Viewpoint column "Technology Strategy and Management" on "The Legacy of Bill Gates" (Jan. 2009) displayed a rather stunning values system by saying that "grow[ing] the PC software business... should be Gates' most enduring legacy." This is not a prediction of what will be Gates's most enduring legacy, though on this issue I would differ as well. Rather, it is a normative statement of what *should* be his most enduring legacy. Does Cusumano really hope that the massive changes now under way in international public health will not endure? His conclusion should not have been so surprising after he referred to Gates's philanthropy as "highly laudable" but only in the context of bemoaning what a distraction it had become from his business interests. I still found my jaw dropping at the word "should."

Max Hailperin, St. Peter, MN

NP-Completeness Not the Same as Separating P from NP

In his news story "The Limits of Computability" (Nov. 2008) David Lindley wrote: "Showing that a problem is NP-complete means proving that no known algorithm can solve it in polynomial time."

In fact, saying that a problem is NP-complete means only that it is "as hard as" any other problem in NP. Lindley apparently confused the definition of NP-completeness with the problem of separating P from NP. Such an error may be pardoned, even overlooked, in the science columns of a general-interest newspaper or magazine, but not in *Communications*.

Madhavan Mukund, Chennai, India

Lindley Responds:

Mukund is correct; this was a slip-up, though one that's easily rectified.

In its earlier paragraphs, the story defined an NP problem as one for which no polynomial-time solution is known, then explained the distinction between NP and NP-complete, but in introducing the unresolved question of whether P and NP are truly distinct, I should have referred to NP problems generally, not NP-complete problems in particular. With this in mind, the paragraph in question would read correctly.

David Lindley, Alexandria, VA

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.

DOI:10.1145/1498765.1498770 David Roman

An Ongoing Study in Usability

The new *Communications* Web site went live last month after several weeks of intense beta testing. While we gleaned many valuable insights and lessons in this process, several highlights and user comments do stand out:

▸ Testers spent an impressive 12 minutes on the beta site before it was even in full swing, making it nine seconds shy of Top 10 rankings tracked by Nielsen/NetRatings in March 2006.

▸ While *Communications*' site may illustrate some user preferences for print v. digital, print is still strong. At press time, Google Analytics usage statistics showed *Communications* magazine archive was among the top five destinations ranked by page views, just ahead of the February 2009 edition.

▸ The latest batch of mobile devices is putting the design model for *Communications*' site to the test. One user requested a single, narrow column layout for his small mobile screen; another commented on the scrolling required to view the site from his laptop.

▸ Even when beta-testers were prompted with such typical fighting words as "make bug-reporting a priority," the response was blissful. "I did not react immediately because I did not have any criticism," said one tester. "I must say, it is impressive."

ACM Member News

GRANDISON WINS AWARDS

Tyrone W. A. Grandison, leader of IBM Almaden Research Center's Intelligent Information Systems team and an ACM Senior Member, was named Pioneer of the Year by the National Society of Black Engineers. He was also named Modern-Day Technology Leader by the Black Engineer of the Year Global Competitiveness STEM Board. "I think [I won the awards] because I am willing to explore new areas every 12 to 15 months," Grandison said in an email interview. "My tendency is to look for emerging issues, define the pertinent problems, and actively seek to solve them."

KIESLER RECEIVES SIGCHI AWARD

Carnegie Mellon University professor Sara Kiesler won SIGCHI's Lifetime Achievement Award. "Sara's research in HCI has illuminated many of the most significant social impacts of computing, such as: 'flaming,' social equalization, open communication, electronic groups, information sharing, and distributed collaboration," noted SIGCHI in the award announcement. "She brought concepts from social psychology and HCI to robotics, helping to create the new interdisciplinary field of human-robot interaction."

SIGIR '09 INDUSTRY TRACK

Aiming to bridge the gap between research and practice, a full-day Industry Track will be held at SIGIR '09, which takes place in Boston, MA, from July 19–23. "The SIGIR '09 Industry Track brings together researchers and practitioners in the area that most defines our information age: information retrieval," said Daniel Tunkelang, the Industry Track chair, in an email interview. "This assembly of the leading lights from industry—Google's Matt Cutts, Microsoft's danah boyd, and the leading enterprise search vendors, Autonomy, Endeca, and FAST—offers an unprecedented opportunity for everyone to learn about the science and technology of real-world information retrieval in a vendor-neutral, analyst-neutral setting."

Science | DOI:10.1145/1498765.1498771

Graeme Stemp-Morlock

Learning More About Active Learning

Active learning algorithms are producing substantial savings in label complexity over passive learning approaches.

If your email address has ever landed on a spammer's list, you know what it's like for your inbox to be flooded with junk email day after day after day. To prevent this, spam filters were created, relying on a mixture of brute force computing with passive learning and refined processing with active learning. And while spam filters have become more sophisticated and your inbox is increasingly free of junk email, the theory behind active learning has lagged. In the last few years, however, the field has taken off.

"There's been surprisingly rapid progress," says Steve Hanneke, a Ph.D. student at Carnegie Mellon University. "If you look back five years, there was really very little known about what makes something an informative example, how important are they, and how much improvement we can expect. But we now have in the published literature a pretty clear picture of just how much improvement we can expect in active learning and what we mean by an informative example."

The difference between passive learning and active learning is about the teacher, and how much time the teacher wants to spend teaching. Passive learning requires large data sets, and the teacher has to label countless examples for the learner. Once every example is labeled, the data set is given to the learner, which then finds patterns that will allow it to sort future data correctly.

The obvious drawback is that passive learning requires a lot of time, and that's where active learning enters the picture.

In active learning, all of the examples are provided to the learner unlabeled. An algorithm analyses the unlabeled data set, and asks the teacher for labels. After the algorithm determines the basic shape of each label set, it asks the teacher to define the ambiguous examples in-between the various labels. By labeling only the most informative examples, the hope is that fewer labels

PHOTOGRAPH COURTESY OF STEVE HANNEKE

At Carnegie Mellon, Steve Hanneke's new work has the benefit of using established passive learning, alongside active learning with its guaranteed improvements, on the number of labels needed.

will be needed to tell the difference between, for instance, a spam email message offering you a free cruise and an email message from your parents telling you about their Alaskan cruise.

"By allowing the algorithm to interactively choose the 'most informative' samples to be labeled, the algorithm will be able to produce a much more accurate model using fewer labeled samples, meaning that we have to do a lot less tedious labeling by hand to get accurate prediction models," says Jenn Wortman, a Ph.D. student at the University of Pennsylvania. "It is possible to prove that in some special cases, active learning algorithms can produce models with the same accuracy as models built by passive learning algorithms while requiring exponentially fewer labeled examples."

Information Overload

Of course, deciding what makes an example informative or not is a huge challenge. Little progress had been made until 2005 when Sanjoy Dasgupta, a professor of computer science at the University of California, San Diego, published a paper, "Coarse Sample Complexity Bounds for Active Learning," which quantified how many labeled examples are needed to find a pattern with active learning over passive learning. The result was significantly less with active learning, but some basic assumptions were needed to see such dramatic outcomes.

One of the most problematic assumptions was that there could be no mislabeled examples causing noise, which would be unreasonable in real-world applications. The following year, however, Maria-Florina Balcan, Alina Beygelzimer, and John Langford published a paper, "Agnostic Active Learning," which described the first active learning algorithm that could work with noise but still provided improvement over passive learning. The algorithm, A^2, labeled examples from a region of disagreement at random, eliminating certain hypotheses until a pattern emerged that was as close as possible to a true understanding.

One of the major improvements of A^2 was finding a threshold function. With passive learning, every piece of data must be analyzed between two bounds before the threshold can be reliably established. With active learning, however, the algorithm can narrow down the threshold value in an almost binary search, resulting in exponentially fewer labels.

Additional research by Hanneke demonstrated that the algorithm performed well as long as the samples in the region of disagreement weren't too diverse and didn't differ in too many ways. Also, further refinements addressed the sampling bias created by labeling only the most informative examples. In essence, the labeled examples were not being chosen at ran-

Career

Computer Science Awards

The National Academy of Engineers was among several professional societies that recently honored a select group of researchers for their distinguished contributions to the field of computer science.

2009 EATCS AWARD

The European Association for Theoretical Computer Science (EATCS) honored Gérard Huet, director of research at the French National Institute for Research in Computer Science and Control, with the 2009 EATCS award, in recognition of his distinguished career in theoretical computer science.

TSUTOMU KANAI AWARD

Willy Zwaenepoel, a professor of computer science and director of the Computer Systems Laboratory at Rice University, won the IEEE Computer Society Tsutomu Kanai Award for major contributions to state-of-the-art distributed computing systems and their applications.

NAE MEMBERS

The National Academy of Engineering elected 65 new members and nine foreign associates for outstanding contributions to "engineering research, practice, or education, including, where appropriate, significant contributions to the engineering literature," and to the "pioneering of new and developing fields of technology, making major advancements in traditional fields of engineering, or developing/implementing innovative approaches to engineering education."

Fourteen computer scientists are among the new members and foreign associates. They are:

- **Paul M. Anderson, consultant, Power Math Associates;**
- **Sergey Brin, co-founder and president of technology, Google;**
- **William J. Dally, William R. and Inez Kerr Bell Professor of Computer Science, Stanford University;**
- **Jeffrey Dean, Google Fellow, Google;**
- **Jack B. Dennis, professor emeritus, Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology;**
- **Deborah L. Estrin, director, Center for Embedded Networked Sensing, University of California, Los Angeles;**
- **Sanjay Ghemawat, Google Fellow, Google;**
- **Paul C. Kocher, founder, president, and chief scientist, Cryptography Research Inc.;**
- **C. Mohan, IBM Fellow, IBM Almaden Research Center;**
- **Mendel Rosenblum, associate professor of computer science and of electrical engineering, Stanford University;**
- **Gurindar S. Sohi, John P. Morgridge Professor and E. David Cronon Professor of Computer Sciences, departments of computer sciences and electrical and computer engineering, University of Wisconsin, Madison;**
- **John A. Swanson, president, Swanson Analysis Services Inc.;**
- **William L. "Red" Whittaker, Fredkin Professor of Robotics, The Robotics Institute, Carnegie Mellon University;**
- **Peter T. Kirstein, professor, department of computer science, University College London.**

SLOAN RESEARCH FELLOWS

The Alfred P. Sloan Foundation awarded two-year fellowships to 118 researchers, including 16 computer scientists, "in recognition of distinguished performance and a unique potential to make substantial contributions to their field."

The computer scientists are: Scott Aaronson, MIT; Luis von Ahn, Carnegie Mellon University; Shuchi Chawla, University of Wisconsin, Madison; Kevin Fu, U. of Massachusetts–Amherst; Odest Chadwicke Jenkins, Brown University; David Kempe, University of Southern California; James Russell Lee, University of Washington; Zhuoqing Morley Mao, University of Michigan; Kamesh Munagala, Duke University; Tze Sing Eugene Ng, Rice University; Ryan William O'Donnell, Carnegie Mellon University; Fabio Pellacini, Dartmouth College; Ramesh Raskar, MIT; Alex C. Snoeren, University of California, San Diego; René Vidal, Johns Hopkins University; and Steve Zdancewic, University of Pennsylvania.

dom, and future data distributions might differ sharply from the labeled examples selected by the algorithm.

To correct the sampling bias, or generalization error, Daniel Hsu, a Ph.D. student at University of California, San Diego, and Dasgupta recommended a scheme where each chosen example has an importance weighting determined by its selection probability. More likely points have higher weightings, and less likely points have lower weightings. As a result, the algorithm could realize when it had selected an unlikely point that was very informative and correct its expectations for any future data.

Putting Theory Into Practice

With several solid years of theoretical work behind them, the active learning researchers are beginning to tackle an even larger challenge than developing the theoretical underpinnings of active learning. Now, they are trying to link theoretical algorithms with large-scale applications.

"There's a striking difference between what's known in practice and theory with active learning," says Hsu. "The theory is trying to catch up with what is known about what works well in practice, and to be able to say something mathematical about what works and what doesn't."

The University of Pennsylvania's Wortman agrees. "[Active learning] is used quite frequently in practice with impressive results, and it's a subject of increasing interest in the theory community, but there's still a huge gap between theory and practice. At a fundamental level, we don't really know 'why' the particular active learning algorithms that are used in practice work."

Toward that goal, some theorists are looking to find general methods of building algorithms that can be used in any situation. General purpose-built algorithms have existed for decades in passive learning. However, with active learning, an algorithm must almost be built from scratch for each new application.

At Carnegie Mellon, Hanneke is working on a type of "meta-algorithm" that can take established passive learning algorithms and churn out an active learning algorithm with passive learning algorithms as subroutines. His hope is that this new work, which he calls "activized learning," will have the benefit of using established passive learning, alongside active learning with its guaranteed improvements, on the number of labels needed.

Maria-Florina Balcan, a post-doctoral student with Microsoft Research New England, believes that active learning algorithms are too delicate for such a general method to exist. Instead, Balcan is broadening the type of interaction that active learning can handle beyond just labeling unlabeled examples. She points out that in other situations another interaction method might be more natural.

Take the example of trying to classify images of people by gender, says Balcan. If the interaction is changed, the teacher might be able complain to the learner that some classifiers are too general and need to be split, or that some are too similar and need to be merged. The teacher might suggest a new class to be added, or notice that the learning is making certain mistakes and suggest a new zone that will eliminate those classification errors.

"Much of the theoretical and practical work on active learning was developed on a very specific model and function of interaction," says Balcan. "But the interaction could be different. The main direction that I see for improvement in the future is extending the type of interaction between the learning algorithm and the user."

Given the progress made in the past several years, with more horizons opening up, the future of active learning appears to be very promising. "We now have active learning algorithms providing substantial guarantees on performance while relying on relatively minimal assumptions about the world," notes John Langford, doctor of learning at Yahoo! Research. "We have a reasonable understanding of where and when they provide performance improvements over passive learning. These algorithms are also practical, often yielding substantial savings in label complexity over passive learning approaches." C

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

Research & Development

U.K. Funding Fosters Innovation

The computational thinking that drives the field of computer science is a key tool for solving problems, designing systems, and understanding human behavior in many disciplines, according to a panel of international experts in Computer Science and Informatics (CS&I). The findings of the project, the Research Assessment Exercise 2008, confirmed the U.K. as a top-ranked research power among industrialized countries.

The survey reported an increased level in the influence of computer science on other disciplines, including bioinformatics, medicine, and e-health. It also found that more computer science research used mathematics to quantify the complexity and rigor of calculations. The analysis also found that research funding for CS&I for the period 2001 to 2008 more than doubled, from $376 million in 2001 to $763 million, leading participants to conclude that continued commitment to funding research and innovation is necessary to maintain global excellence in a battered economy.

The review of research in CS&I surveyed 81 colleges and universities and found the subject not only healthy and growing, but more rigorous, interdisciplinary, experimental, and user-oriented than ever.

"The vitality of the computing field, which is due in large measure to increased investment in research, is directly related to the degree of innovation that emerges from U.K. research institutions," says ACM President Dame Wendy Hall, who participated in the survey's computer science panel. "These innovations, in turn, foster research partnerships with startup companies as well as spinouts and collaborations with subject matter experts and multinational corporations. The resulting level of economic activity crosses into all industries, even creating new sectors that provide career opportunities in the computing and information technology field."

Technology | DOI:10.1145/1498765.1498772 Alex Wright

Our Sentiments, Exactly

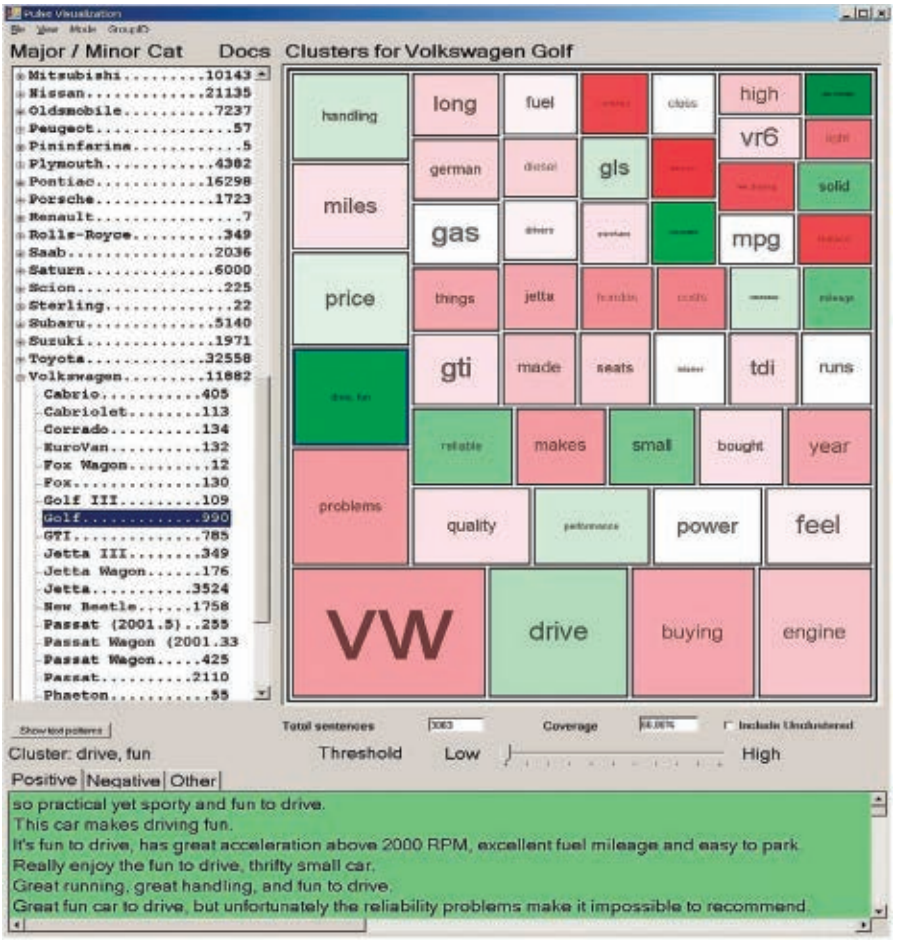
With sentiment analysis algorithms, companies can identify and assess the wide variety of opinions found online and create computational models of human opinion.

FACTS MAY BE stubborn things, as John Adams once put it, but at least they're easy to compute. From a data-processing perspective, opinions are much more stubborn.

In recent years, the Web has created a bull market in human opinion: movie reviews, product ratings, restaurant recommendations, and all kinds of other viewpoints expressed in articles, blogs, discussion groups, and elsewhere. As the Web accumulates more and more data, many of us rely on each other's opinions as a filter to help us make informed decisions. For many businesses, customer opinions have become a type of virtual currency that can make or break their products. As opinion data plays an increasingly important role on the Web, however, computer scientists are discovering the limitations of traditional text analytics algorithms for sorting opinions from raw facts.

The distinction between facts and opinions might seem clear enough on the surface, but in practice teasing them apart involves parsing many linguistic shades of gray. This is where the emerging field known as sentiment analysis comes in. Sometimes called opinion mining or subjectivity analysis, sentiment analysis is a new term that broadly refers to the identification and assessment of opinions, which for the purposes of computation might be defined as written expressions of subjective mental states.

Traditional text analytics algorithms work by scanning a body of text to extract and analyze keywords. That approach works well for identifying simple factual statements, but assessing opinions requires delving much deeper into the subtleties of human language. "Sentiments are very different from conventional facts," says analytics consultant Seth Grimes. While direct expressions of opinion are fairly easy to spot—for example, "I hated *Revenge of the Sith*"—most human sentiments fall somewhere along a continuum from objective fact to subjective experience. For example, "It's fifteen degrees outside" is an objective statement; "It's cold" reveals a somewhat more subjective point of view; while "I'm putting on two pairs of socks" constitutes a completely indirect expression of opinion disguised as a statement of fact.



An example of sentiment analysis from Michael Gamon in which topics from reviews of the Volkswagon Golf are depicted. The size of each topic box indicates the number of mentions of the topic, and the shading of each topic box indicates the average sentiment, ranging from negative (red) to neutral/none (white) to positive (green).

"We are dealing with sentiment that can be expressed in subtle ways," says Yahoo! researcher Bo Pang, co-author of the book *Opinion Mining and Sentiment Analysis*. To penetrate those subtleties, sentiment analysis algorithms assess written statements through a series of overlapping filters. They usually begin by attempting to determine the polarity of a particular sentiment—i.e., Is it positive or negative? Once that's established, they may try to determine the intensity of sentiment being expressed—i.e., How positive or negative is this statement? Next, an even more subtle layer of analysis might attempt to determine the degree of subjectivity—i.e., How partial or impartial is the point of view being expressed here? (This is often determined by looking at

SCREENSHOT COURTESY OF MICHAEL GAMON

the number of adjectives in a sentence.) Using these and other criteria, sentiment analysis algorithms can then begin to create computational models of human opinion.

Complicating matters even further are questions of context (who's speaking, and to whom?) and linguistic nuances like slang and ambiguity. A "bad motorcycle" might actually be a good one; whereas a "bad movie" is probably just plain bad. Sentiment analysis algorithms sometimes have to go beyond literal interpretations of a text to discern an author's original intent. Given the wide varieties of idiomatic writing on the Web, this is no small task. As Grimes notes, "You don't see 'Genistein inhibits protein histidine kinase...Not!' in a scientific paper."

Mining Collective Opinions

"With opinions, so much depends on the point of view of the user," says David Pierce, chief technology officer of Jodange, whose sentiment analysis software grew out of a research project by Claire Cardie at Cornell University and Jan Wiebe at the University of Pittsburgh. Drawing on a body of theory in linguistics, philosophy, and computational linguistics, their team developed an algorithm that tries to determine the context of any particular statement by isolating three key data points: the topic, the opinion holder, and the opinion itself. First, the algorithm employs an entity extraction routine that locates keywords to identify particular topics and opinion holders. Next, it layers that data onto a linguistic analysis of the opinion being expressed. The resulting unit of data is a triple consisting of opinion, opinion holder, and topic. These triples are then stored in a relational database, where they can be cross-referenced across multiple documents to create what Jodange vice president of product management and marketing Pia Chong calls a "walled garden of opinion."

By connecting opinions from multiple sources about a particular topic, the application can provide users with a bird's-eye view of a particular topic presented in a variety of different formats: straightforward lists, heat maps that show the concentration of opinions on particular topics, an opinion index that calculates positive or negative trends, or a so-called Doppler view that shows a graphical summary of opinion data. The company is currently working on a new predictive model that could use opinion data to predict future developments, such as the impact of written opinion on trends in a company's stock price.

A number of other companies are now developing their own variations of sentiment analysis software. Companies like Attensity, Clarabridge, Lexalytics Limited, SPSS, and TEMIS are developing their own proprietary versions of sentiment analysis software. All of these products employ some combination of keyword extraction and linguistic analysis to provide their customers with a particular understanding of collective opinion. Some of these products are targeted toward business applications, others toward consumer-facing Web applications.

For consumers, the most obvious applications for sentiment analysis involve enhancing search engines with more opinion data. "Sentiment analysis software could enable a much smoother user experience" for consumer research, says Pang. Microsoft's Product Search, which is part of Live Search, and Yelp's review highlights, which include phrases automatically extracted from user reviews, already rely on basic sentiment analysis to enhance their search results. Such interactions could eventually find their way into the general Web search experience. Pang suggests that such interactions could be fine-tuned for users at different stages of the research process, allowing them to narrow down from reviews of a product category to comparisons between products, then finally to in-depth product reviews.

Beyond the realm of consumer products, Pang also sees opportunities for sentiment analysis to shape the way people consume news. "When the media is having a field day, [users] might want to get a digest of different perspectives on the breaking news with analysis." Similar applications might eventually lead to new types of interfaces where readers could track the movement of opinion about particular stories over time.

For many businesses, online customer opinions have become a type of virtual currency that can make or break their products.

That kind of opinion-trending insight is particularly valuable to users working in business or government. These potential users might include business intelligence professionals, market researchers, or public relations specialists. Today, sentiment analysis vendors are already marketing their products to companies in the form of hosted services that provide opinion dashboards and other management tools. At this stage, sentiment analysis software is too new to have penetrated most IT firewalls. Eventually, however, companies may start exploring how to integrate sentiment analysis data with their core management systems. "Unified analysis is coming," says Grimes, "but it's not here yet."

Attensity is taking a step in that direction by marketing a suite of tools designed to help companies integrate sentiment analysis data with their internal business operations. In addition to providing sentiment analysis data, Attensity provides mechanisms for funneling that data into operational "queues" like marketing campaigns or call center scripts. "For example, if a valuable customer is upset they can route them to a special marketing campaign that compensates them through points or other things of value," explains Michelle de Haaff, Attensity's vice president of marketing and products.

As sentiment analysis finds its way into the business mainstream, vendors will likely continue to develop similar services that bring sentiment analysis into the IT mainstream. Once that integration starts to happen, companies will be able to feed opinion data into core business processes that can help them strengthen their customer relationships—and, ultimately, boost profits: a decidedly unsentimental goal. C

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

Society | DOI:10.1145/1498765.1498773 David Essex

Did Somebody Say Virtual Colonoscopy?

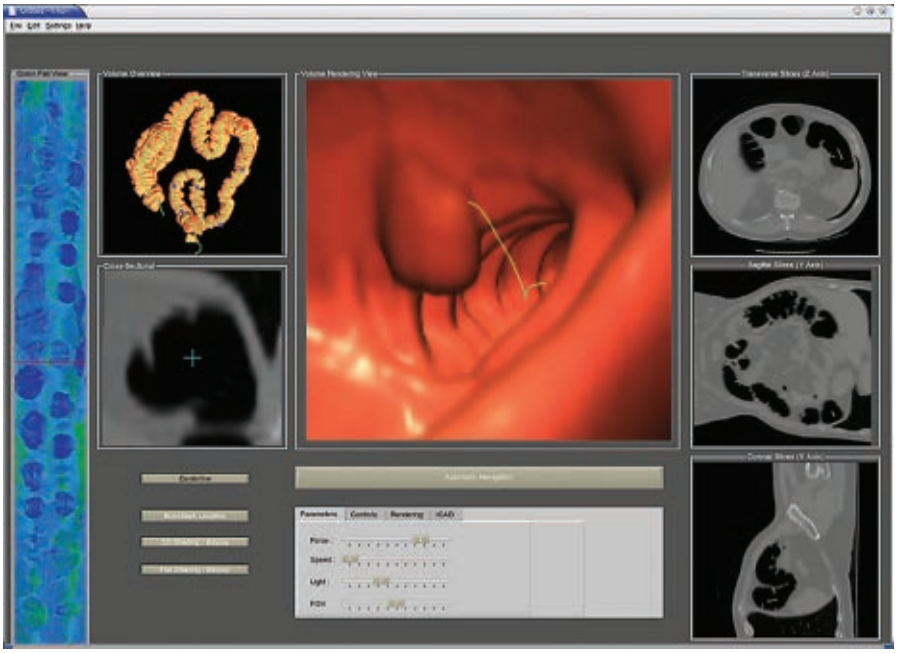
Doctors are saving lives with virtual, 3D exams that are less invasive than a conventional optical colonoscopy.

Most people over 50 years old ignore their doctor's advice and forgo the standard, invasive test that screens for colorectal cancer and precancerous growths. Fortunately, a computer-based, noninvasive alternative to the conventional optical colonoscopy, known as virtual colonoscopy, is changing people's attitudes—and could save tens of thousands of lives each year.

A virtual colonoscopy starts with computed tomography (CT), a common diagnostic technology that uses X-rays to record cross-sectional, 2D images of the body's interior. A 3D model is constructed by segmenting the colon from the rest of the abdomen and using an electronic cleansing algorithm to factor out fecal material. Next, doctors use visualization software to navigate a virtual fly-through of the colon. If a polyp or suspicious growth is found, doctors can perform a virtual biopsy and investigate further.

The case for a convenient mass-screening method is strong, says Arie Kaufman, chair of the computer science department at New York's Stony Brook University. Colorectal cancer is the third most common cancer and the second leading cause of cancer deaths in the U.S., with more than 140,000 new cases and more than 50,000 deaths a year. "If all patients 50 years of age and older will participate in these screening programs, over 92% of colorectal cancer will be prevented and over 600,000 lives could be saved worldwide every year," Kaufman says.

Virtual colonoscopies became possible in the mid-1990s, when Kaufman and others developed volume-rendering techniques that enabled 3D, virtual fly-throughs and associated tools, which were soon commercialized.



A screenshot of a user interface for virtual colonoscopy and computer-aided detection.

SCREENSHOT COURTESY OF ARIE E. KAUFMAN

Virtual colonoscopies have recently earned the imprimatur of the medical establishment, which prefers the formal term CT colonography. In a study at U.S. military hospitals, 1,233 symptomless subjects underwent a virtual colonoscopy, followed by a conventional optical colonoscopy, on the same day. The virtual colonoscopy results, reported in the *New England Journal of Medicine* in 2003, showed 94% sensitivity (real polyps found) and 96% specificity (false-positive rate) for polyps 8mm and larger—numbers comparable to those of optical colonoscopy, the gold standard among doctors. Subsequently, the U.S. Food and Drug Administration approved virtual colonoscopy for colon cancer screening.

Virtual colonoscopy outperforms optical colonoscopy in certain ways, advocates claim. A University of Wisconsin study, for example, found it better at finding 8mm and 10mm polyps. It also outperforms optical colonoscopy in finding polyps hidden in folds and around corners of the twisting tube of the colon, and in reliably reaching the farthest reaches, called the caecum. It can also do something optical colonoscopy, by its nature, cannot do: spot polyps on the colon's outer walls.

Also, because it is noninvasive, a virtual colonoscopy avoids the risk of the rare but deadly tears or holes that can occur during an optical colonoscopy (and which can require immediate surgery). "The examination is done on the data, rather than the patient," says Dr. C. Daniel Johnson, a principal investigator at the Mayo Clinic in Scottsdale, AZ, and the lead researcher on several studies.

Virtual colonoscopy's only significant health risk is a patient's exposure to radiation. This trade-off

is being addressed in a technique called low dose, which Kaufman's group is researching, with a grant from the National Institutes of Health (NIH). Another drawback of a virtual colonoscopy is it can't remove polyps; patients still need an optical colonoscopy for their surgical excision. However, a virtual colonoscopy study has shown that only about 7% of patients required such follow-up.

Insurers, however, have been slow to catch on. Medicare, for example, announced a tentative decision earlier this year to not pay for virtual colonoscopies. And some insurers have setup the payment codes that healthcare providers need to be reimbursed for the procedure, but the money has yet to materialize. However, Kaufman says the political will to mandate coverage is growing, and patients can pressure their insurer to pay for a virtual colonoscopy by refusing to undergo optical colonoscopy.

Early Detection

Screening is critical because a patient's successful outcome often hinges on the early detection of polyps. A virtual colonoscopy removes many of the uncomfortable hurdles. "Only 15%–19% of individuals eligible for screening currently undergo colon evaluation," says Hiroyuki Yoshida, an associate professor at Harvard Medical School and director of 3D Imaging Research at Massachusetts General Hospital. "The cathartic cleansing required for bowel preparation is the biggest barrier."

A virtual colonoscopy still requires preparation, so developing a laxative-free procedure will be indispensable to its practicality, Yoshida says. One hitch: eliminating laxatives leaves more fecal matter in the colon, which requires improvements in electronic cleansing. What's more, patients still must ingest an oral contrast agent, such as barium or iodine, and air or carbon dioxide must still be used to distend the colon. Yet, a laxative-free virtual colonoscopy could be ready for public use in the near future, Yoshida says.

Many of the challenging issues with virtual colonoscopies involve software applications. Computer-aided detection (CAD), the focus of Yoshida's research, brings much-needed automation to electronic cleansing and polyp detection. It holds great promise, says Kaufman. "In mammography, this has been entirely successful," he notes. "Basically, the computer is another set of eyes."

Virtual colonoscopies could become more available in remote places via telemedicine.

Skill in interpreting diagnostic images varies among radiologists, who can mistake the colon's normal valves and folds for polyps. Yoshida says CAD could make results more objective and consistent, and shorten radiologists' learning curve. It could also be useful in hospitals that lack expertise with virtual colonoscopies.

However, CAD presents its own challenges. "Sometimes CAD's weak spots are comparable to human viewers' weak spots," says Dr. Ronald Summers, a radiologist and senior investigator at NIH. One important challenge is detecting flat lesions, which are more difficult to detect with a virtual or optical colonoscopy, but constitute a higher risk for cancer.

Kaufman has co-developed a novel CAD technique, utilizing colon flattening and volume rendering, which has achieved perfect sensitivity and tolerable specificity, and could function as either a first or second reader.

Another problem is false positives: they require patients to undergo an optical colonoscopy. Of course, false positives aren't as deadly as false negatives, and doctors can dismiss false positives with a second read. Ideally, Kaufman says, CAD should be a first reader for radiologists, who can inspect the regions flagged by it. "I think the second read is the one the radiologist should use, because it has the highest sensitivity, but we're still working that out," Dr. Summers says.

There is also considerable debate

Technology

TechFest 2009

Microsoft Research unveiled more than 100 innovations at its annual TechFest showcase in late February in Redmond, WA. Among the emerging technologies projects were:

SECONDLIGHT
Used with Microsoft Surface, SecondLight can project images and detect gestures in mid-air above Surface's display (in addition to supporting its multitouch capabilities). The magic behind SecondLight involves Surface's LCD screen that can switch between opaque and translucent, alternating 60 times per second. A pair of recessed projectors alternate flickering 30 times per second, with one projector timed to illuminate the screen when it is opaque and the second projector timed to illuminate when it is transparent. When the screen is transparent, any object held above the screen will reveal what is being output by the second projector.

RENLIFANG
When you conduct a Web search for information about a specific person—say, an ex-girlfriend—you still need to examine all of the search results and piece together the information before being able to ascertain the parameters of her social universe. Renlifang is a Web entity-summarization system that automatically creates a biography page of the person; a social-network graph for the person; a shortest relationship path between the person and someone else; titles for the person found on the Web; and all of the structured information that Microsoft possesses on the person in its local database.

REAL-TIME STITCHING OF MOBILE-GENERATED VIDEOS
Microsoft demonstrated a real-time video-stitching system from multiple mobile phones that produces a wide field-of-view experience with high resolution. Possible applications include citizen journalism; virtual attendance of family events and gatherings; and emergencies in which citizens provide a real-time video feed for first responders before they reach the scene of the emergency.

on how to best combine 2D and 3D imagery. "We seem to be moving to a consensus that the sensitivity and specificity of 2D and 3D are comparable," Dr. Summers says. "The question is which one do you use as your primary." Yoshida thinks CAD could soon replace 3D visual fly-throughs for first reads, though expert examination of 2D images—what radiologists called "problem solving"—will still be needed.

2D is important in a virtual biopsy, which depends on flattening the images to simulate what pathologists do when dissecting a polyp. "They will slice it along its length and lay it flat and look at it," Dr. Summers says. "You do the same thing on the computer." Some experts, he notes, think such 2D dissection provides faster diagnosing than 3D fly-through.

Seeking Improvements

Medical and computer-science researchers are striving to make virtual colonoscopy technology more accurate, affordable, easier to use, and patient friendly.

A technique called dual-energy imaging, for instance, highlights polyps by blending images derived from different radiation doses to increase contrast, Yoshida says. And graphics processing unit-based rendering is being touted as a faster method of getting images to radiologists, as Kaufman's group has done. Also, Dr. Summers says his collaborators and him have figured out how to bolster CAD with wavelets on manifolds to reduce false positives by more precisely characterizing polyps. And machine learning and neural nets are the subject of ongoing research.

"The examination is done on the data, rather than the patient," says Dr. C. Daniel Johnson.

To increase virtual colonoscopies' usability, computer scientists are also focusing attention on the PCs that are used for analyzing images. One possibility is off-site image processing, which Yoshida says Massachusetts General Hospital is ready to implement.

Others hope to democratize virtual colonoscopies by getting the software to run effectively on desktop and laptop computers. For example, the Redmond, WA, company FiatLux Imaging employs the Direct3D technology in video games and in virtual colonoscopies. "It's usually required to run on very heavy-duty, expensive hardware," says Rosemary Fisher, FiatLux's clinical application specialist. "That is prohibitively expensive for small hospitals and clinics." Many of them lack colonography software and have little financial incentive to invest in it before insurers start uniformly reimbursing for virtual colonoscopies. But as spiral CT scanners become more broadly distributed, affordable volume-rendering software, such as FiatLux's Visualize, might make virtual colonoscopies more available in remote places via telemedicine.

The takeaway message is that virtual colonoscopies are poised to dramatically increase successful colon screening outcomes. Says Kaufman, "We're going to save 50,000 lives every year just in the U.S." C

David Essex is a freelance science writer based in Peterborough, NH.

Computer Graphics

Catmull Wins Second Oscar

ACM Fellow Ed Catmull, a computer scientist, co-founder of Pixar Animation Studios, and president of Walt Disney and Pixar Animation Studios, received the Gordon E. Sawyer Award from the Academy of Motion Picture Arts and Sciences in recognition of his lifetime of technical contributions and leadership in the field of computer graphics for the motion-picture industry. Catmull was presented with an Oscar statuette at the Scientific and Technical Awards Presentations last February at the Beverly Wilshire Hotel.

"Ed is one of the rare individuals who can bridge the space between science and art," said Academy President Sid Ganis. "His vision, ingenuity, and groundbreaking designs have made the impossible possible—for filmmakers and movie audiences around the world."

Catmull, who delivered a keynote address at SIGGRAPH 2008, has described ACM SIGGRAPH as his "home community." He is regarded as an innovator by the community for his key contributions to fundamental computer graphics concepts like z-buffer and subdivision surfaces, and has held several leadership positions in SIGGRAPH over three decades. In 1995, Catmull became an ACM Fellow, and was cited for "his many and noteworthy advances in computer graphics as an individual researcher, as an inspiring leader in the field, as a director of organizations, and as a mentor for many."

In the course of his career, Catmull founded the computer graphics laboratory at the New York Institute of Technology as well as the computer division of Lucasfilm Ltd., and Pixar Animation Studios.

In 2000, Catmull and his team received an Oscar for an Academy Award of Merit for their significant advancements to the field of motion picture rendering as illustrated in Pixar's *RenderMan*. He previously received two Scientific and Engineering Awards from the Academy. In 1992, he was part of a team recognized for the development of *RenderMan* software. In 1995, he was on a team honored for pioneering inventions in Digital Image Compositing. He also shared a Technical Achievement Award from the Academy in 2005.

PHOTOGRAPH BY AP PHOTO/MATT SAYLES

News | DOI:10.1145/1498765.1498774 Bob Violino

Time to Reboot

A diverse, international group of more than 200 attendees met at the Rebooting Computing Summit to address the problems confronting computer science.

THE CHALLENGES FACING the computing field are well known: enrollment in degree programs has steadily declined since 2001; women and minorities are underrepresented; many K–12 students have a negative perception of computing; and reports say the innovation rate in the field has decreased.

To address these formidable challenges, a group of more than 200 participants from many sectors touched by computing, including business, education, government, engineering, and science, held a three-day Rebooting Computing Summit at the Computer History Museum in Mountain View, CA, last January.

The meeting comes at a critical time for the computing field, says Peter Denning, chairman of the computer science department at the Naval Postgraduate School in Monterey, CA, and organizer of the invitational summit.

"According to the last figures I saw, the total number of computer science students in the pipeline and expected to graduate is about two-thirds of the number of jobs needing to be filled," Denning says. "These are rewarding jobs, demanding creativity."

In addition to the inadequate number of computer science students, another disturbing reality is that key meetings of the leaders in science don't regularly include computer scientists. "Computer science is often not at the table," Denning says, "and that hurts science badly."

Denning and a like-minded 18-member team decided that previous workshops and studies devoted to these issues hadn't produced enough impact. It was time to try something different, so they invited a diverse, international group representing all major sectors of computing to meet, share ideas and find common ground, and take action.

Attendees say the summit succeeded in generating excitement in a community that's been frustrated in its efforts to attract young people and collaborators, and report that they're eager to continue the momentum started at the event.

Conference organizer Peter Denning, left, and facilitator Ron Fry prepare before the start of the Rebooting Computing Summit, which was held at the Computer History Museum.

PHOTOGRAPH COURTESY OF THE REBOOTING COMPUTING SUMMIT

"The most exciting part of the meeting was getting together with people who all share a passion for computer science and a common goal of working to help revitalize the field," says Robb Cutler, past president of the Computer Science Teachers Association. The summit fostered connections among people with an interest in K–12 computer science programs, Cutler says.

Tim Bell, associate professor at the University of Canterbury in New Zealand, says the summit brought together many people in an environment that seeded a lot of cooperation. "Not only did I meet people interested in the same kind of project, but there was the energy and impetus to do something cooperative on a global scale," Bell says.

The summit's main achievement was the formation of 15 action groups that will carry out projects in the coming year. These groups include Image of Computing, Defining Computer Science, and K–8 FUNdamentals. Each group created a mission statement and a list of actions they plan to accomplish during the next year.

The Rebooting Computing Web site, rebootingcomputing.org, lists the groups, members, and contact information, and incorporates collaborative tools such as blogs, social networks, and wikis. Denning is encouraged by activity he's seen since the summit, such as the appearance online of several videos about the conference.

"The loss of attraction to [computing] comes from our being unable to communicate the magic and beauty of the field," Denning says. "We need to create an appreciation for the elegance and power of what computing can do." C

Bob Violino is a writer based in Massapequa Park, NY, who covers business and technology.

News | DOI:10.1145/1498765.1498775 Alan Joch

IT Ecosystem in Peril

Experts warn the U.S. may soon relinquish its leadership role in IT research and development.

The debate in Washington, D.C. over strengthening the nation's infrastructure shouldn't focus only on roads and bridges. To keep the U.S. economy healthy for years to come, policymakers must also bolster another vital infrastructure: the IT industry's research and development ecosystem.

That's one of the conclusions of *Assessing the Impacts of Changes in the Information Technology R&D Ecosystem: Retaining Leadership in an Increasingly Global Environment,* a 166-page report created by a National Research Council panel and released by the National Academy of Sciences in January.

The study's 12-member panel, consisting of IT leaders in academia, venture capital, and industry, examined the threats to U.S. prominence as a world IT leader. "The report says the U.S. risks ceding IT leadership to other nations within a generation, and I think that's absolutely true," says Ed Lazowska, a committee member and Bill & Melinda Gates chair in computer science and engineering at the University of Washington. "People should be seriously concerned about the entire ecosystem."

The response, according to the study, should be to build U.S. strengths in "conceptualizing idea-intensive new concepts, products, and services." For that it needs the "best-funded, most-creative" research institutions, world-leading technical and entrepreneurial talent, and advanced technology infrastructures, in particular for next-generation wireless broadband communications.

But some specific recommendations may be controversial during an economic downturn, including calls for additional government spending and increases in the number of H-1B visas for foreign nationals.

Reassess R&D Funding

Although the report doesn't recommend funding goals, Lazowska says the Obama administration should reassess R&D funding priorities. "We should be investing in research that's going to create the infrastructure we need five, 10, and 15 years from now," Lazowska says.

To do this, funding authorities should consider IT's contribution to the economy, says Randy H. Katz, co-chair of the study and professor of electrical engineering and computer science at the University of California, Berkeley. "Do a fair assessment of the impact of IT on the economy. By that [measure] a lot of the growth in the U.S. domestic product can be attributed to productivity growth related to the effective use of information technology," he says.

Recommendations for helping the nation remain a magnet for technical talent include strengthening computing education, and more controversially, expanding the number of H-1B visas issued to permit foreign nationals to study and work in the U.S.

Some U.S. citizens believe H-1B visas enable foreign nationals to take jobs from Americans and depress IT salaries. But the study argues for flexibility and recommends that the permits favor those with advanced degrees who subsequently work in the U.S. to create new products and companies, Katz explains.

The alternative is to allow international students to receive U.S. tax dollars for research and education "and then send those students back to their home countries to compete against us," warns Dan Reed, scalable and multicore computing strategist with Microsoft Research and a member of the study's review committee.

Finally, to fuel the economy's ability to encourage new companies, policymakers should reduce the "friction" caused by regulations such as the Bayh-Dole Act, which governs university intellectual property issues. "What's a modest burden for a major multinational can be a huge legal or regulatory burden for four guys in a garage," Reed says.

Katz calls entrepreneurs and venture capitalists "a very important and underexamined dimension" for IT R&D. "The takeaway I'd want for Congress is that one size does not necessarily fit all," Katz says. "Having some understanding [of this] is important in the way that they draft that legislation." C

Comparison of U.S., Japanese, and European Union estimated public funding of civilian information technology and communications research and development (in billions of dollars)

Year	United States	Japan	European Union
1999	1.2	1.9	2.7
2000	1.3	2.1	2.9
2001	1.6	2.3	3.0
2002	1.5	2.5	3.3
2003	1.7	2.6	3.3
2004	1.9	2.7	3.4
2005	1.8	2.7	3.5

Source: *Assessing the Impacts of Changes in the Information Technology R&D Ecosystem: Retaining Leadership in an Increasingly Global Environment,* National Research Council, 2009

Alan Joch is a business and technology writer based in Francestown, NH.

ACM CREATIVITY & COGNITION 2009

EVERYDAY CREATIVITY: SHARED LANGUAGES & COLLECTIVE ACTION

OCTOBER 27-30 2009

Berkeley Art Museum & UC Berkeley

CREATIVITY IS PRESENT IN ALL WE DO...

The 7th Creativity and Cognition Conference is to be held at the Berkeley Art Museum and the University of California, Berkeley (USA). The conference provides a forum for lively interdisciplinary debate exploring methods and tools to support creativity at the intersection of art and technology. We welcome submissions from academics and practitioners, makers and scientists, artists and theoreticians.

*FULL PAPERS *ART EXHIBITION *LIVE PERFORMANCES
*DEMONSTRATIONS *POSTERS *WORKSHOPS
*TUTORIALS *GRADUATE SYMPOSIUM

KEYNOTE SPEAKERS

MIHÁLY CSÍKSZENTMIHÁLYI
PROFESSOR OF PSYCHOLOGY & MANAGEMENT
Claremont Graduate University **[CA, USA]**

JOANN KUCHERA-MORIN
DIRECTOR, ALLOSPHERE RESEARCH LABORATORY
California Nanosystems Institute **[CA, USA]**

everyday creativity

SUBMISSION BY: APRIL 24 FOR MORE INFO SEE: WWW.CREATIVITYANDCOGNITION09.ORG

CONFERENCE COMMITTEE General Chair **Nick Bryan-Kinns [UK]** Program Chair **Mark Gross [USA]** Program Co-Chair **Ron Wakkary [Canada]** Program Co-Chair **Hilary Johnson [UK]** Program Co-Chair **Jack OX [USA]** Demonstrations & Posters Chair **Yukari Nagai [Japan]** Tutorials Chair **Yusuf Pisan [Australia]** Workshops Chair **Ryohei Nakatsu [Singapore]** Graduate Student Symposium Co-Chair **John C Thomas [USA]** Graduate Student Symposium Co-Chair **Celine Latulipe [USA]** Art Exhibition Chair **Jennifer Sheridan [UK]** Art Exhibition Executive Committee **Sara Diamond [Canada]** Treasurer **Ellen Do [USA]** Sponsorship **Doug Riecken [USA]** Publicity **David A Shamma [USA]** Local Inspiration **Richard Rinehart [USA]** Local Committee Chair **Daniela Rosner [USA]** Steering **Ernest Edmonds [Australia]**

PRINTED PROCEEDINGS PUBLISHED BY ACM PRESS & APPEAR IN THE ACM DIGITAL LIBRARY

SIGCHI special interest group computer human interaction

acm Association for Computing Machinery

DOI:10.1145/1498765.1498776

Richard Heeks

Emerging Markets
IT and the World's "Bottom Billion"

How can information technology be best applied to address problems and provide opportunities for inhabitants of the world's poorest countries?

While celebrating the emerging markets of Asia, India, and Latin America, let's spare a thought for the world's "bottom billion." These are the inhabitants of the Fourth World that sits beneath the Third World; dozens of countries that, in the words of economist Paul Collier "are falling behind, and often falling apart."[2]

As informatics professionals, why should we care about these countries? And how might IT best be used to help them?

The bottom billion—a population equivalent to that of the U.S. and Europe combined—lives overwhelmingly in sub-Saharan Africa or Central Asia. Life expectancy in these regions is just 50 years. One-in-seven children die before the age of five. They missed the globalization boat that sailed with many other developing countries in the 1980s and 1990s. While those other countries have grown steadily richer, the Fourth World of the bottom billion was actually poorer in 2000 than it was in 1970.[2] These countries are not emerging markets, they are fading markets: the whole of sub-Saharan Africa has an economy the size of Belgium's.

Should we be concerned?

Simple ethics says we should: developing an e-business solution to squeeze out a few extra ounces of profit or time-saving for the world's privileged population living in the global North pales in ethical importance compared to applying new technology to the mega-problems of the bottom billion. And self-interest says we should: the bottom billion—countries like Somalia, Afghanistan, and North Korea—are key sources of global instability and risk including drugs, piracy, and terror.

These countries are not emerging markets, they are fading markets: the whole of sub-Saharan Africa has an economy the size of Belgium's.

Not surprisingly, the bottom-billion nations have been among the least digital. But that is changing. Official figures may indicate an average of only three Internet users per hundred population[6] but that greatly underestimates the true reach. Information technology in the Fourth World is a communal, not individual, resource. As a result, many times more are casual users; and many times more again have indirect access to Internet-based data and applications through friends and relations. Internet connectivity is also growing fast: by 42% per annum in the bottom billion, compared to 18% in Europe.

Beyond the Internet, there is an even greater bottom-billion phenomenon: the cellphone. Ten years ago, Manhattan had more phone connections than all of Africa. Today, thanks to the cellphone, Africa has more phone connections than the U.S. and Canada combined. Approximately one-fifth of

bottom-billion citizens are subscribers (up from approximately 2% at the start of the century)[6] but, as with the Internet, the effective penetration—those who could access a cellphone from neighbors, relatives, or local call sellers if necessary—is higher; likely more than half the population.

Subscriber growth rates are also high—50% per year compared to less than 20% in Europe—and sometimes highest for the most-afflicted countries. Three examples of the worst of insecurity and instability in the bottom billion—Afghanistan, Democratic Republic of the Congo, and Somalia—have high investments and an average 100% annual growth rate.[7] Those rates are likely to sustain: in sub-Saharan Africa, for example, by 2012, it is estimated 90% of the population will be within cellphone coverage, up from 60% today.[3]

We are still a long way from North American rates of IT usage but it is well past time to move from talking about vague future possibilities and begin talking about actual current priorities.

The first priority should be engagement. I wonder how many *Communications* readers are working on an informatics project in a bottom-billion country. It is likely not a large number, because we tend to work where the money is rather than where the problems are.

In terms of IT, the three key priorities are mobiles, mobiles, and mobiles. As indicated earlier in this column, cellphones are now reaching far down into the bottom billion. At present, development solutions will need to be based around voice and text. But other possibilities are rapidly opening up.

One set of these possible solutions involves the integration of mobiles with other IT. This scenario could involve radio and television, converting these high-penetration but broadcast-only media into much more interactive forms, as currently being achieved in the growing number of "community radio" projects. Or we can think of integrating phones with telecenters and kiosks. Pilot projects already under way suggest this can multiply the impact of Web access many times.[4]

How will the Web and Internet reach the bottom billion? The GSM Association estimates that 80% of Internet delivery will occur through mobile devices.[3] However, mobiles are not the only way forward. WiMAX-plus-netbook systems can offer high-quality, low-cost Internet access. Even more intriguing is the potential use of "white space": the unused parts of the analog television broadcast spectrum. For the bottom billion, it will be many years before digital switchover and release of analog spectrum space, but new technology could be developed to identify and use existing white space spectrum gaps.[1] This reallocation of spectrum space could offer wide-reaching broadband Internet service at a fraction of the cost of other solutions.

In all these areas, though, we need more, better, and less expensive technological innovations that focus on the particular conditions and resource constraints of bottom-billion customers.

Beyond hardware and software, what application priorities do the bottom billion demand?

Analysis of the problems of the Fourth World may help provide an answer. The key problem is that of exclusion: the average bottom-billion citizen

ILLUSTRATION BY STUART BRADFORD

lives a life that is more 14th century than 21st century because their countries have been excluded from the benefits of globalization and because, within their countries, these citizens are excluded from the services, opportunities, and resources that we in the global North take for granted.

Social exclusion has prevented the bottom billion from accessing basics like health care, education, and government services. Many of these functions are information-based or information-enabled. IT therefore has a central role to play. For example, it can disintermediate the sometimes-corrupt gatekeepers who stand between poor citizens and the meeting of their social development needs.

Political exclusion has helped perpetuate bad governance in bottom-billion countries. Our Western mentalities have tended to simplistically condense the solution into elections. But Paul Collier argues these have been the least effective part of the governance fixes for the most-troubled nations. Instead, he says, we should be seeking greater transparency: helping citizens hold their governments to account.

Transparency is all about information flows, so IT is crucial, and we already have some pointers about new ways in which it can be used. In West Africa, the notion of government surveillance has been turned on its head to create citizen "sousveillance": monitoring democratic processes and reporting on them using short message service and taking cellphone pictures as evidence.

In East Africa, open information systems are being used to publicize how much government spending should be getting through to the "front line" of development. Combined with citizen report-back, in one case this raised the amount reaching schools from 20% of allocation to 90%.[2]

And IT is integral to a new form of openness and engagement: e-participatory budgeting, which provides online citizen discussion and decision making on how part of the government budget will be spent. Projects so far find thousands of people participating, and many more involved via friends and other IT-savvy intermediaries.

Perhaps most important of all is *economic exclusion*. It should be axiomatic that the main difference between the world's poorest and everyone else is... they have less money. We need to ensure, in harnessing IT for the bottom billion, that a strong priority is given to applications that help create wealth. To do this, IT must address the exclusion of poor individuals and poor nations from markets.

We tend to assume that IT has little or no role in these countries. In fact it does already and will in the future have much to offer.

IT can generate new market opportunities. Flying somewhat under the radar of government and donor agencies, IT can help directly create new microenterprises for the poor. Ongoing research at the University of Manchester suggests this is one of the fastest-growing sectors in the bottom billion: it involves those who set up their own Internet kiosks, those who stand on street corners selling cellphone calls, those who sell prepaid cellphone cards and phone covers, and many other business ideas.

IT can offer access to new sources of finance. Organizations like Kiva use Web microfinance portals to make a direct link between individual sponsors in the global North, and microentrepreneurs in the bottom billion. In a less organized way, individuals in poor communities are turning their cellphones into mobile wallets. Relatives overseas remit money in the form of airtime. This arrangement is increasingly accepted by storeowners, colleges, health centers, and other organizations as an alternative form of currency.

And IT can improve opportunities for trade. By offering access to prices in different consumer markets, IT can increase incomes for the poor. We now have evidence indicating that the more remote the farmer or microentrepreneur, the greater the benefit of IT.[8] As the IT base diffuses into the bottom billion, it also offers the prospect of digital trade; something especially valuable given that these nations are disproportionately landlocked. One small but quickly growing element of this is "IT social outsourcing": the offshoring of IT services work like digitization and data entry with a combined commercial and developmental intent. Evidence from initial projects suggests this can furnish not just new incomes but new skills and confidence to those involved.[5]

Perhaps I can best summarize all this by pointing to the *psychological exclusion* that we in the global North sometimes practice. We tend to exclude the bottom-billion countries from our worldviews and from our informatics work. We tend to assume that IT has little or no role in these countries. In fact it does already and will in the future have much to offer. We also tend to conceive of the bottom-billion citizens as non-users. In fact they are not only increasingly active users but they are in some sense innovators; constantly developing new applications and new business models with the technology but also looking for ideas and support from global partners like us. C

References

1. Bahl, V. White space networking and the commoditization of pervasive Internet access. Paper presented at Wireless4D 2008, (Karlstad, Sweden, December 2008).
2. Collier, P. *The Bottom Billion: Why the Poorest Countries are Failing and What Can Be Done About It*. Oxford University Press, Oxford, U.K., 2007.
3. Denton, A. Policy priorities to connect Africa. Paper presented at the 1st International Mobiles for Development conference, (Karlstad, Sweden, December 2008).
4. Donner, J. et al. Stages of design in technology for global development. *IEEE Computer 41*, 6 (June 2008), 34–41.
5. Heeks, R.B. and Arun, S. IT social outsourcing as a development tool. Development Informatics Group, IDPM, University of Manchester, U.K., 2007; http://www.womenictenterprise.org/IT%20SocialOutsourcing%20Kerala%20Paper.doc.
6. ICT Statistics Database, International Telecommunication Union, Geneva, 2009; http://www.itu.int/ITU-D/icteye/Indicators/Indicators.aspx.
7. Konkel, A. and Heeks, R.B. Challenging conventional views on mobile telecommunications investment: Evidence from conflict zones. Development Informatics Group, IDPM, University of Manchester, U.K., 2008; http://www.sed.manchester.ac.uk/idpm/research/publications/wp/di/short/di_sp09.pdf.
8. Muto, M. and Yamano, T. The impact of mobile phone coverage expansion on market participation: Panel data evidence from Uganda. *Journal of JBIC Institute 37* (2008), 48–63; http://www.jica.go.jp/jica-ri/publication/archives/jbic/report/review/pdf/37_05.pdf.

Richard Heeks (richard.heeks@manchester.ac.uk) is Professor of Development Informatics in the Institute for Development Policy and Management at the University of Manchester, U.K.

DOI:10.1145/1498765.1498777

George V. Neville-Neal

Kode Vicious

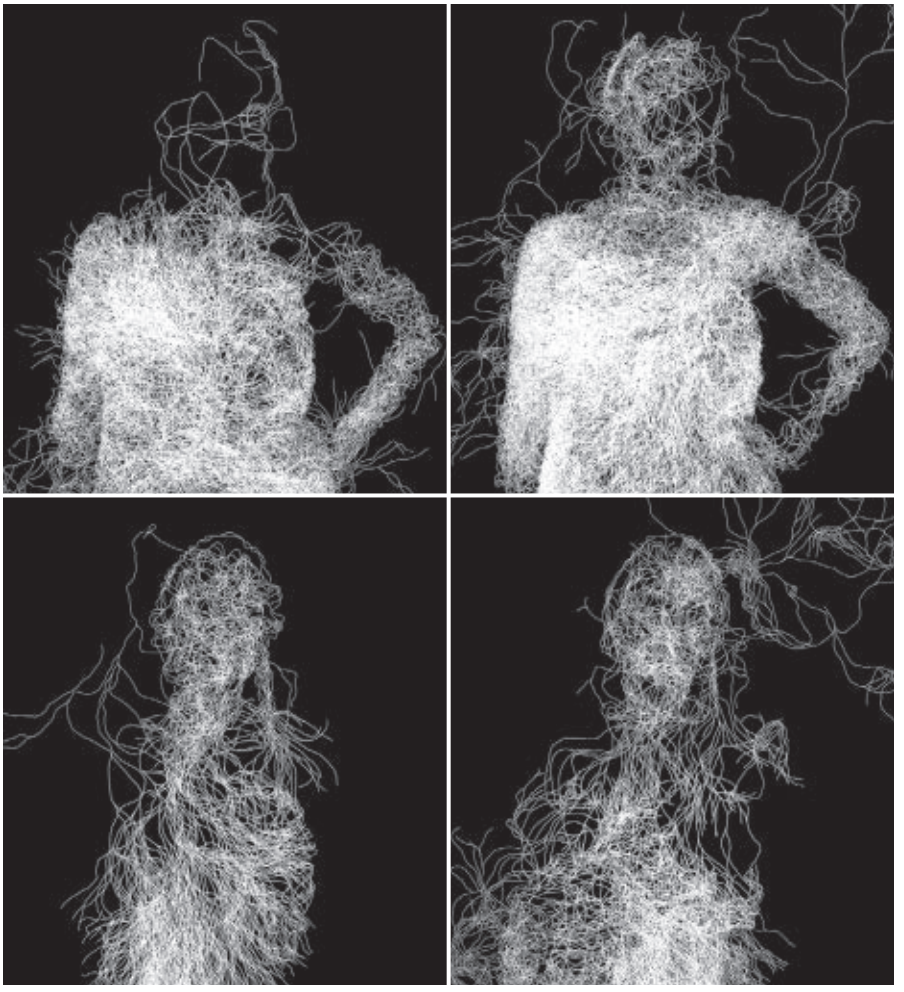
System Changes and Side Effects

Comparing the potential benefits of system changes that help and the detriments of changes made for the sake of change.

Dear KV,

I maintain a legacy system at work that uses Makefiles for building the code. Recently my company switched to a different system—SCons—for doing software builds and now I'm being asked to update our legacy code to build with this system. Personally, I don't see that we will get any benefit switching a legacy system's build strategy, but it's an order from management so I will likely have no choice. Why would anyone bother to switch a working build system to a newer one? Are build systems really that different? Is there really any innovation here that could matter?

All Built Up



Dear Built Up,

Innovation, as someone should have said, is in the eye, or perhaps hands, of the beholder. KV rarely condones change for its own sake: the change has to have some appreciable benefit to those who are working with the system. As this relates to build systems, there are few things more important to a programmer's work flow and therefore productivity than how their system is built. A poorly implemented build system will impede progress and lead to wasteful programmer downtime, mostly made up of programmers playing games in the hallway, buying questionable items on the Web, and learning yo-yo tricks. All fine pursuits of course—I learned at least five yo-yo tricks on one job when it took 20 minutes to build a piece of software, and the build locked up the machine we all used for our work. Of course such crippling times are in the past now, with the advent of speedy CPUs, memory, and disks there is no longer any wasted programmer time, and we can all depend on a quick compile/link phase. Uh, perhaps not.

There are several reasons why people switch build systems. I think the most common one is that people hate and fear Makefiles. Other than the misnamed autotools there is probably no

ILLUSTRATION BY RYAN ALEXANDER

more maligned component in a build system, and with good reason. While a simple Makefile is easy to read, a large software system requires a set of interlocking, and usually undocumented, Makefiles that can be as difficult to understand as modern architecture, and quite often is just as pretty to look at. Pull out just one piece and the whole edifice will come crashing down upon your head. The complexity of a system of Makefiles does not, normally, come from some perverse turn of mind of the authors, but is in fact due to three problems. The first is that dependency analysis, which is what the make program does, is a non-trivial exercise; the second is that Makefiles and build systems are generally not designed, they are accreted; and the third is that the build system is usually stretched to do all kinds of things that were not originally foreseen.

In the beginning was the Makefile, and it was good. It contained at most a few targets and a handful of source files. The rules to determine what should be built and when were straightforward and all was right with "make world." Over time all software systems grow and so the Makefile becomes the root of a tree of Makefiles that are spread across a set of directories. Next the include directive is used, and then chaos ensues. The tree that was rooted in the original Makefile becomes a set of creeping vines, with a strong resemblance to poison ivy because no one wants to touch it.

A typical build system not only takes source files and turns them into binary programs, but it may also be co-opted to produce documentation from the source code, perform pre-checkin testing, and so forth. Every time you add a new responsibility to the system you must tweak it in some way, which usually has unintended side effects. Eventually the Makefiles become so complex that they're impossible for new users to modify, and even experienced users make mistakes because modifications are likely to be infrequent. Just as with code, engineers rarely document their Makefiles, as a matter of fact they're likely never documented, even if the code is. The most common documentation for a build system consists of the rather unhelpful "type make<rtn>," which works in the typical case but gives no information as to how to debug a problem in the make system.

Every time you add a new responsibility to the system you must tweak it in some way, which usually has unintended side effects.

Speaking of debugging, this is one of the places where the make program fails miserably. While it does have a command line flag to say "don't do what I say, just show me what you'd do" as well as another command line flag to show debugging information, the output is usually so voluminous that it is unusable, and you wind up deleting and commenting lines in the Makefile until your bug is hidden, because removing lines doesn't actually make the bug go away. Debugging Makefiles can make you tear your hair out, and for what, you probably just wanted to add a build option or a new file to your system and now you've wasted two hours with your Makefiles. So, Makefiles are ugly and make people nervous and they tend to start scratching themselves absent-mindedly if you ask them to fix them.

If you think make is bad, well, SCons is certainly more interesting. It seems that what make lacked was a scripting language, since we all know that adding a language to a system will make it easier to use! SCons is written in Python and Python is the language in which its SConstruct files—the equivalent of Makefiles—are written. Much like make if you want to do something simple with SCons it's pretty easy to do so. If you thought that a complex set of Makefiles was difficult to understand, try reading a complex set of SConstruct files. You'll not only need to load up the mental context for your project, you'll also need to load your brain with Python. Now, don't get me wrong, Python is a fine thing, I code in it frequently, but I don't want to have to hack Python to get my software builds going. Having recently been exposed to SCons I can say that I don't prefer it over make. Although the system I was working with had supposedly been extended to build code with profiling turned on that extension was clearly not working. When I asked another engineer how he built the system with profiling his reply was, "I just hack it." "OK," I thought to myself, "let's see if I can just fix this." After all, I was going to need profiled code for quite a while and it seemed dumb of me to check out an extra copy of our tree just to have what should be possible with a compiler flag. Had I known what lay ahead, and were I a less stubborn engineer, I would have checked out another copy of the tree. I was determined to "do things right" and therein lies the road to madness. It took about four hours to load up the proper mental context, and to understand the interlocking SConstruct files. After many detours and chasing several wild geese, I was able to get my profile flags correctly into the build system. Strangely enough the number of lines I changed was less than 10, but finding *which* 10 lines was the interesting part. If I had not already been a Python programmer it would have probably taken days to get this to work.

Don't get me wrong, SCons certainly has some interesting features, including built-in support for object caching, but I would be hard-pressed to call it an improvement over make. My advice to you is to stick with what you have, that is if you can, in particular since you say the system is a legacy system. And my advice to those who wish to make my life easier by changing basic tools is to study the existing tools very carefully and to figure out which changes really help and which changes are just there for the sake of change.

KV

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.

DOI:10.1145/1498765.1498778

Michael Cusumano

Technology Strategy and Management

Strategies for Difficult (and Darwinian) Economic Times

How the axiom of survival of the fittest applies in the context of a global economic downturn.

CHARLES DARWIN HAS had considerable impact not only on the field of biology but also on theories of industry evolution and management. Within sociology departments and business schools, for example, during the 1970s and 1980s there emerged a strain of "population ecologists" that continues to influence much of the research on organizations and industries today. These scholars see a Darwinian process of survival that occurs at the "population" (the industry) level and has little to do with the actions (or inactions) of individual managers and firms. The argument, in simple form, is that most companies are unable to adapt to major change and that successful companies are mainly those whose structural characteristics happen to match well with demands of the new environment.[a] For example, most mainframe computer companies were unable to adapt to small machines and distributed computing, and so they disappeared. General Motors and other automobile makers that cannot efficiently operate at low production volumes and make money with small, fuel-efficient vehicles with tiny profit margins will face the same fate as vacuum tube producers that could not adapt to transistors.

Let me say up front that I do not completely agree with the population ecologist view. I have worked with many companies since the 1980s and believe the actions of managers had an important impact on performance and survival. I also teach in a business school where, we presume, it is worthwhile teaching

PHOTOGRAPH BY MARK MCLAUGHLIN

a The early most important works on this topic are Michael Hannan and John Freeman, "The Population Ecology of Organizations," *The American Journal of Sociology 82*, 5 (May 1977), 929–964; Howard Aldrich, *Organizations and Environments*, Stanford University Press, Palo Alto, CA, originally published 1979, classic edition 2008. See also Michael Hannan and John Freeman, *Organizational Ecology*, Harvard University Press, Cambridge, MA, 1989.

MBA candidates something about management. But it seems true that many organizations do not seem to be able to change, and that Darwinian processes take place as industries mature, companies fail, and industries consolidate over time. As I have written elsewhere, the U.S. has seen the number of publicly listed software product companies drop from over 400 in 1998 to less than 150 in 2006. The number of U.S.-listed IT services companies has experienced a similar consolidation.[b] If we look carefully at other industries—computer hardware, semiconductors, and telecommunications equipment, as well as automobiles—we see similar declines in the number of firms over time.[c]

Many of these industries will evolve and change, and even rebound as new competitors and technologies appear. U.S.-based companies like RCA and GE thought the consumer electronics and computer hardware businesses were mature by the early 1970s, and exited—missing enormous opportunities, ranging from VCRs to PCs and Internet services. So it is dangerous to assume that all industries will mature, consolidate, and decline. But what are managers to do when they face difficult economic times in their businesses—especially if, in the long run, theorists tell us the managers' individual actions may have little to do with the survival of their firms?

First, they should believe in Darwin—the fittest and the luckiest are most likely to survive. Government protection or subsidies are likely only to delay the inevitable. At the same time, however, managers need not believe in the extreme views of the population ecologists. Though change is difficult, something will separate the survivors from the failures. It may sometimes be luck or random processes (the market equivalent of genetic mutations). But managers cannot take that chance. They must assume their actions are relevant and may well make the difference between survival and failure. So what managers do or do not do is more, not less, important in difficult economic times, though the best managers are probably those who foresee and respond to the subtle hints of a changing environment before radical change and economic calamity actually set in.

The companies that survive for the very long term are likely to be the smartest as well as the financially fittest.

Second, once in the throes of an economic crisis, managers must do all they can "to save the mother ship" and make sure they will live to fight another day. Most companies have a set of core activities and people, and then peripheral activities and people. The extra weight may be fine to have around in good times and usually is justified as "diversification" or "future growth areas." But, in bad times, the non-core areas are likely to be cash drains and management distractions, and can easily turn into fatal liabilities. So, when sales start declining, managers should use this period as an opportunity to trim waste and wishful thinking and get back to the strongest foundations of their companies.

The following example is illustrative of the decisions that must be made under difficult economic circumstances. I consulted for a particular software vendor that vastly expanded its product lines during the Internet boom, growing extremely fast but accumulating hundreds of millions of dollars in losses that it viewed as an investment in the future. By the time the Internet bubble burst, the company's lineup had ballooned to some 144 distinct products, with a huge cash drain because thousands of salespeople, product managers, product engineers, quality staff, and field consultants were needed to design, deliver, and support all these products. I got the finance department to find out exactly which products were selling and which were not. We concluded that customers were really only buying 12 products. Revenues purported to come from the other products in fact were subsidies that came from licensing those 12 products in bundles with the other 132 products. My recommendation was to get rid of 80%–90% of the company's products and personnel, which I thought would ensure that at least the core of the company would survive. The management team resisted. Eventually, it took five years, a delisting from the stock market, near bankruptcy, and many senior-position departures before the company ultimately downscaled. It is a shadow of its former self but has survived to compete (or sell itself) another day.

In addition to scaling back to the strongest product or service lines, saving the mother ship means saving as many of the best people as you can. Simply cutting staff across the board makes no sense for most technology businesses because they depend so much on intellectual capital. Furthermore, if managers have to borrow money continually or get repeated cash infusions from their investors or their governments, then they are likely to have little or no negotiating power. If they do get additional money, it will be on Draconian terms—and managers should prefer to deal with Darwin rather than Draco (the ancient Greek lawmaker who favored very harsh punishments even for minor offenses).

Finally, I have a third recommendation for managers: Continue to look for opportunities to grow in areas of strength. As long as survival of the mother ship seems secure, then companies should use whatever resources they can afford to take advantage of the environment. Weak competitors will also be contracting but should be worse off, and their customers and talented employees are fair game. The companies that survive for the very long term are likely to be the smartest as well as the financially fittest: managers must be able to anticipate as well as exploit change in difficult economic times. C

b Michael Cusumano, "The Changing Software Business: Moving from Products to Services," *IEEE Computer 41*, 1 (Jan. 2008), 78–85.

c See Fernando Suarez and James Utterback, "Dominant Designs and the Survival of Firms," *Strategic Management Journal 16*, 6 (June 1995), 415–430.

Michael Cusumano (cusumano@mit.edu) is Sloan Management Review Distinguished Professor of Management and Engineering Systems at the MIT Sloan School of Management and School of Engineering in Cambridge, MA.

DOI:10.1145/1498765.1498779

Michael Buckley

Viewpoint
Computing as Social Science

Changing the way computer science is taught in college by encouraging students to develop solutions to socially relevant problems.

I HAVE BEEN concerned with the decline in computer science enrollments nationwide,[a] and it began to increasingly bother me as I prepared for the fall semester last year. With each new freshman class I worry that fewer students are choosing a career in computer science, and it's no stretch of the imagination to think it's an image problem. Not only are we not getting the word out to high schools, but I believe we're losing students in the first year of college as well. And it's because we don't offer students the idea that computer science is social, relevant, important, and caring, and thus we lose their interest. There might in fact be studies that show this, or perhaps not, but it's something I feel.

I began my last summer break as I do almost every year, looking through textbooks for something usable for a fall course in programming. I teach Java, which is more than adequate for new programmers to learn everything good and bad about addressing a computer. What they learn is that computers never do what we want, but only what we tell them to do. And what we tell them to do in a freshman programming course is too often dull. Write to the operator, print out the results of a calculation, order some list, and all too often the message, calculation, and list are irrelevant. How can I get them interested? I'm determined at the start of every semester to have a batch of eye-opening, to-the-point, significant examples, lessons, and problems.

To learn more about David's story, see http://www.sociallyrelevantcomputing.org/.

And so I look through the textbooks and the examples and sample programs, and I become aware of an obsession with animals. In the first 10 textbooks I've skimmed, I've learned how to count ducks, categorize puppies, separate cows from horses, manage a pet store, create a cyber-pet, add fish to a bowl, and so it goes. This can't possibly be the least bit interesting to a freshman who wants to learn computing.

I go back through the textbooks intentionally avoiding animal references and instead look for something else. I find games, plenty of them: Tetris, Othello, checkers, tic-tac-toe, even a good approximation of chess moves, which is wonderful if you come to college to play games. I'm not even sure what programming principle they're trying to teach, and in my best attempt at empathy with an incoming freshman, my eyes glaze over. I'm bored, and I have a vested interest. How can a student possibly find interest and relevance in this stuff? The texts rely solely on the student to be interested enough in programming to overcome the banality. We all know that practice is more fun than theory, but our attempts at practice aren't real.

I move on...let's see...a doughnut

a See http://www.cra.org/wp/index.php?p=105.

PHOTOGRAPH COURTESY OF JUDY SCHINDLER

counter to show iterations, a pizza maker to construct a list, a lemonade stand to demonstrate databases. If I was a student, beginning these important four years, and I was taught programming via doughnut machines, I would quit and go do something important. Major in some field that had an impact. Even the sterile environment of pure mathematics has me counting and measuring planets and populations. Sociology and psychology would have me charting behaviors. Chemistry and physics have me connected to the environment. Computing as portrayed in the literature has me running a pet store, playing games, or eating. That would seem to be it. There isn't a textbook out of the 60 I have on my shelf that makes me see computing as socially relevant.

And so the message is just not getting out there. Students' firsthand experience with computers—their music and their phones—is accepted and reinforced by the image we portray in school—one of unrelenting banality and geekdom—and potential computer science students do not see themselves as having a greater impact.

At the University of Buffalo we have two senior-level courses that require teams to create real systems for real clients. They are introduced to the wider community of people with disabilities and told to make a difference. That's it. Those are the instructions. Improve the quality of life of someone less able than you. If you can't figure it out, you fail. So don't fail.

A group of students tacked a sign up at a school for handicapped children that said "Student Inventors Available Free. Is there something you need? Call us." And they heard from a mother whose daughter could not use a computer because she had no fine motor skills. She could move her arms but not her fingers. So they made her a trackball out of a basketball, and wrote games that use the wide swing of her arms as she rotated the basketball. It's not perfect, but they were immersed and involved, and they visited with the family and they delivered a prototype. And no one will ever convince them that computer science is not social science, because outside the world of trivia we feed students in their freshman year, it certainly is.

Now people in the community call us. That's how my students met David, who was 43, suffered a stroke at age 27, and hadn't been able to speak since. He communicated with his nurses by pointing to a sheet of paper that was taped to his wheelchair. It had letters, words, and short phrases, and after much practice, a nurse or therapist could almost decipher what he wanted to say. So our students transferred that sheet of paper to a tablet PC, and when David touches a word, the computer speaks it. How difficult is that? Easier than counting doughnuts. The night they delivered that system, David called me at home with his new voice and thanked me. And said he waited 15 years to speak on the phone. And the pictures I saw later of the event clearly showed students crying.

There isn't a textbook out of the 60 I have on my shelf that makes me see computing as socially relevant.

Every once in a while, a student will say "I can't find a project," and I tell them to read the newspaper or consult other news sources. That itself sounds banal but it's not: right below the surface of a news item, there is most likely a problem to be solved. Find someone or something in trouble, and save it. That's how we found the number-one killer of firefighters on the job: it's not fire, smoke, or Dalmatian attack; it's heart attack. And so now we have a system that monitors vital signs and displays the statistics on a 3D model of a fire scene as the firefighters traverse it.[b]

We have remote-controlled wheelchairs, videoconferencing for homebound and hospital-bound children, a light-and-sound system (the students call DISCO) that teaches cause-and-effect to autistic children, and many more systems constantly evolving. All of this technology and creative energy is at our fingertips, but to sample our craft in the popular literature, you would think we were cyber pets on one end, artificial intelligence on the other, and nothing useful in between.

So back to the textbooks and the freshman year. In the senior-level courses, you can see the difference between simply relaying a difficult concept (teachers know when that lightbulb goes off in a student's head) and emotionalizing that concept (that's a whole different look behind their eyes, and probably why teachers become teachers). How do we get that same reaction? It's probably as much for me as for them that I want them to see computing as a craft for the greater good. I have to teach my students counting, so I will forego puppies and have them design a tamperproof voting system. When they learn two-dimensional arrays it will be to monitor the flow of pollution through Lake Erie. Databases? Not fish in a bowl but it will be drug interactions. My good friend Devika Subramanian at Rice University taught me how to use disaster evacuation planning to teach optimal paths and routing instead of using chess. I can't find any of these in a textbook that teaches CS1, so I'll have to invent them.

I know that writing a textbook is difficult. But so is teaching, and so is learning. C

b See http://www.sociallyrelevantcomputing.org/images/pic17.jpg.

Further Reading

1. Buckley, M. et al. Benefits of using socially relevant projects in computer science and engineering education. In *Proceedings of the Special Interest Group on Computer Science Education Conference*, 2004; http://www.sociallyrelevantcomputing.org/SIGCSE2004SociallyRelevantProjects.pdf.
2. Buckley, M., Schindler, K., Kershner, H., and Alphonce, C. Using socially relevant projects in a capstone design course in computer engineering. In *Proceedings of the American Society for Engineering Education Annual Conference*, 2004; http://portal.acm.org/citation.cfm?id=1028174.971463.
3. Nordlinger, N., Subramanian, D., and Buckley, M. Socially relevant computing. In *Proceedings of the Special Interest Group on Computer Science Education Conference*, 2008; http://www.cs.rice.edu/~devika/SIGCSEFinal.pdf.
4. Schindler, K., Buckley, M., Kershner, H., and Alphonce, C. Partnering with social service organizations to develop socially relevant projects in computer science and engineering. In *Proceedings of the International Conference on Engineering Education*, 2004; http://www.sociallyrelevantcomputing.org/ICEE2004.pdf.

Michael Buckley (mikeb@cse.buffalo.edu) is the director of the Center for Socially Relevant Computing at the University of Buffalo, NY.

DOI:10.1145/1498765.1498780 Bertrand Meyer, Christine Choppy, Jørgen Staunstrup, and Jan van Leeuwen

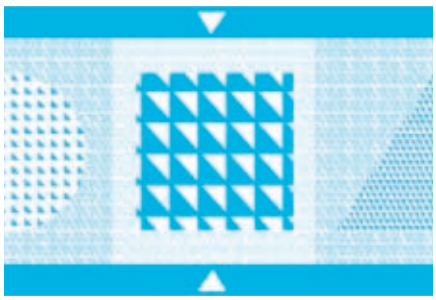
Viewpoint
Research Evaluation for Computer Science

Reassessing the assessment criteria and techniques traditionally used in evaluating computer science research effectiveness.

Academic culture is changing. The rest of the world, including university management, increasingly assesses scientists; we must demonstrate worth through indicators, often numeric. While the extent of the syndrome varies with countries and institutions, La Fontaine's words apply: "*not everyone will die, but everyone is hit*." Tempting as it may be to reject numerical evaluation, it will not go away. The problem for computer scientists is that assessment relies on often inappropriate and occasionally outlandish criteria. We should at least try to base it on metrics acceptable to the profession.

In discussions with computer scientists from around the world, this risk of deciding careers through distorted instruments comes out as a top concern. In the U.S. it is mitigated by the influence of the Computing Research Association's 1999 "best practices" report.[a] In many other countries, computer scientists must repeatedly explain the specificity of their discipline to colleagues from other areas, for example in hiring and promotion committees. Even in the U.S., the CRA report, which predates widespread use of citation databases and indexes, is no longer sufficient.

Informatics Europe, the association of European CS departments,[b] has undertaken a study of the issue, of which this Viewpoint column is a preliminary result. Its views commit the authors only. For ease of use the conclusions are summarized through 10 concrete recommendations.



Our focus is evaluation of individuals rather than departments or laboratories. The process often involves many criteria, whose importance varies with institutions: grants, number of Ph.D.s and where they went, community recognition such as keynotes at prestigious conferences, best paper and other awards, editorial board memberships. We mostly consider a particular criterion that always plays an important role: publications.

Research Evaluation

Research is a competitive endeavor. Researchers are accustomed to constant assessment: any work submitted—even, sometimes, invited—is peer-reviewed; rejection is frequent, even for senior scientists. Once published, a researcher's work will be regularly assessed against that of others. Researchers themselves referee papers for publication, participate in promotion committees, evaluate proposals for funding agencies, answer institutions' requests for evaluation letters. The research management edifice relies on assessment of researchers by researchers.

Criteria must be fair (to the extent possible for an activity circumscribed by the frailty of human judgment); openly specified; accepted by the target scientific community. While other disciplines often participate in evaluations, it is not acceptable to impose criteria from one discipline on another.

Computer Science

Computer science concerns itself with the representation and processing of information using algorithmic techniques. (In Europe the more common term is *Informatics*, covering a slightly broader scope.) CS research includes two main flavors, not mutually exclusive: Theory, developing models of computations, programs, languages; Systems, building software artifacts and assessing their properties. In addition, domain-specific research addresses specifics of information and computing for particular application areas.

CS research often combines aspects of engineering and natural sciences as well as mathematics. This diversity is

a For this and other references, and the source of the data behind the results, see an expanded version of this column at http://se.ethz.ch/~meyer/publications/cacm/research_evaluation.pdf.

b See http://www.informatics-europe.org.

part of the discipline's attraction, but also complicates evaluation.

Across these variants, CS research exhibits distinctive characteristics, captured by seminal concepts: algorithm, computability, complexity, specification/implementation duality, recursion, fixpoint, scale, function/data duality, static/dynamic duality, modeling, interaction...Not all scientists from other disciplines realize the existence of this corpus. Computer scientists are responsible for enforcing its role as basis for evaluation:

1. Computer science is an original discipline combining science and engineering. Researcher evaluation must be adapted to its specificity.

The CS Publication Culture

In the computer science publication culture, prestigious conferences are a favorite tool for presenting original research—unlike disciplines where the prestige goes to journals and conferences are for raw initial results. Acceptance rates at selective CS conferences hover between 10% and 20%; in 2007–2008:

- ICSE (software engineering): 13%
- OOPSLA (object technology): 19%
- POPL (programming languages): 18%

Journals have their role, often to publish deeper versions of papers already presented at conferences. While many researchers use this opportunity, others have a successful career based largely on conference papers. It is important not to use journals as the only yardsticks for computer scientists.

Books, which some disciplines do not consider important scientific contributions, can be a primary vehicle in CS. Asked to name the most influential publication ever, many computer scientists will cite Knuth's *The Art of Computer Programming*. Seminal concepts such as Design Patterns first became known through books.

The research management edifice relies on assessment of researchers by researchers.

2. A distinctive feature of CS publication is the importance of selective conferences and books. Journals do not necessarily carry more prestige.

Publications are not the only scientific contributions. Sometimes the best way to demonstrate value is through software or other artifacts. The Google success story involves a fixpoint algorithm: Page Rank, which determines the popularity of a Web page from the number of links to it. Before Google was commercial it was research, whose outcome included a paper on Page Rank and the Google site. The site had—beyond its future commercial value—a research value that the paper could not convey: demonstrating scalability. Had the authors continued as researchers and come up for evaluation, the software would have been as significant as the paper.

Assessing such contributions is delicate: a million downloads do not prove scientific value. Publication, with its peer review, provides more easily decodable evaluation grids. In assessing CS and especially Systems research, however, publications do not suffice:

3. To assess impact, artifacts such as software can be as important as publications.

Another issue is assessing individual contributions to multi-author work. Disciplines have different practices (2007–2008):

- *Nature* over a year: maximum coauthors per article 22, average 7.3
- *American Mathematical Monthly*: 6, 2
- OOSPLA and POPL: 7, 2.7

Disciplines where many coauthors are the norm use elaborate name-ordering conventions to reflect individual contributions. This is not the standard culture in CS (except for such common practices as listing a Ph.D. student first in a joint paper with the advisor.

4. The order in which a CS publication lists authors is generally not significant. In the absence of specific indications, it should not serve as a factor in researcher evaluation.

Bibliometry

In assessment discussions, numbers typically beat no numbers; hence the temptation to reduce evaluations to

such factors as publication counts, measuring output, and citation counts, measuring impact (and derived measures such as indexes, discussed next).

While numeric criteria trigger strong reactions,[c] alternatives have problems too: peer review is strongly dependent on evaluators' choice and availability (the most competent are often the busiest), can be biased, and does not scale up. The solution is in combining techniques, subject to human interpretation:

5. Numerical measurements such as publication-related counts must never be used as the sole evaluation instrument. They must be filtered through human interpretation, particularly to avoid errors, and complemented by peer review and assessment of outputs other than publications.

Measures should not address volume but impact. Publication counts only assess activity. Giving them any other value encourages "write-only" journals, speakers-only conferences, and Stakhanovist research profiles favoring quantity over quality.

6. Publication counts are not adequate indicators of research value. They measure productivity, but neither impact nor quality.

Citation counts assess impact. They rely on databases such as ISI, CiteSeer, ACM Digital Library, Google Scholar. They, too, have limitations:

- Focus. Publication quality is just one aspect of research quality, impact one aspect of publication quality, citations one aspect of impact.
- Identity. Misspellings and mangling of authors' names lose citations. Names with special characters are particularly at risk. If your name is Krötenfänger, do not expect your publications to be counted correctly.
- Distortions. Article introductions heavily cite surveys. The milestone article that introduced NP-completeness has far fewer citations than a later tutorial.
- Misinterpretation. Citation may imply criticism rather than appreciation. Many program verification articles cite a famous protocol paper—to show that their tools catch an equally famous error in the protocol.
- Time. Citation counts favor older contributions.
- Size. Citation counts are absolute; impact is relative to each community's size.
- Networking. Authors form mutual citation societies.
- Bias. Some authors hope (unethically) to maximize chances of acceptance by citing program committee members.

An issue of concern to computer scientists is the tendency to use publication databases that do not adequately cover CS.

The last two examples illustrate the occasionally perverse effects of assessment techniques on research work itself.

The most serious problem is data quality; no process can be better than its data. Transparency is essential, as well as error-reporting mechanisms and prompt response (as with ACM and DBLP):

7. Any evaluation criterion, especially quantitative, must be based on clear, published criteria.

This remains wishful thinking for major databases. The methods by which Google Scholar and ISI select documents and citations are not published or subject to debate.

Publication patterns vary across disciplines, reinforcing the comment that we should not judge one by the rules of another:

8. Numerical indicators must not serve for comparisons across disciplines.

This rule also applies to the issue (not otherwise addressed here) of evaluating laboratories or departments rather than individuals.

CS Coverage in Major Databases

An issue of concern to computer scientists is the tendency to use publica-

c D. Parnas, "Stop the Numbers Game," *Commun. ACM 50*, 11 (Nov. 2007), 19–21; available at http://tinyurl.com/2z652a. Parnas mostly discusses counting publications, but deals briefly with citation counts.

Calendar of Events

April 13–16
Computation and Control,
San Francisco, CA,
Sponsored: SIGBED,
Contact: Paulo Tabuada,
Email: tabauda@ee.ucla.edu

April 15–18
The 8th International Conference on Information Processing in Sensor Networks,
San Francisco, CA,
Sponsored: SIGBED,
Contact: Rajesh Gupta,
Email: rgupta@ucsd.edu

April 20–24
Design, Automation and Test in Europe,
Nice, France,
Sponsored: SIGDA,
Contact: Benini Luca,
Email: lbenini@deis.unibo.it

April 20–24
The 18th International World Wide Web Conference,
Madrid, Spain,
Contact: Gonzalo Leon,
Email: gonzalo.leon@pucp.edu.pe

April 26–30
International Conference on the Foundations of Digital Games,
Port Canaveral, FL,
Contact: Emmet James Whitehead, Jr.,
Email: ejw@cs.ucsc.edu

May 1–2
Western Canadian Conference on Computing Education,
Burnaby, BC Canada,
Contact: Diana Cukierman,
Email: Diana@cs.sfu.ca

May 5–8
14th International Conference on Animation, Effects, Games and Digital Media,
Stuttgard, Germany,
Contact: Thomas Haegele,
Email: thomas.haegele@filmakademie.de

May 10–13
ACM 2009 International Conference on Supporting Group Work,
Sanibel Island, FL,
Sponsored: SIGCHI,
Contact: Erling Carl Havn,
Email: havn@man.dtu.dk

tion databases that do not adequately cover CS, such as Thomson Scientific's ISI Web of Science.

The principal problem is what ISI counts. Many CS conferences and most books are not listed; conversely, some publications are included indiscriminately. The results make computer scientists cringe.[d] Niklaus Wirth, Turing Award winner, appears for minor papers from indexed publications, not his seminal 1970 Pascal report. Knuth's milestone book series, with an astounding 15,000 citations in Google Scholar, does not figure. Neither do Knuth's three articles most frequently cited according to Google.

Evidence of ISI's shortcomings for CS is "internal coverage": the percentage of citations of a publication in the same database. ISI's internal coverage, over 80% for physics or chemistry, is only 38% for CS.

Another example is Springer's *Lecture Notes in Computer Science*, which ISI classified until 2006 as a journal. A great resource, LNCS provides fast publication of conference proceedings and reports. Lumping all into a single "journal" category was absurd, especially since ISI omits top non-LNCS conferences:

- The International Conference on Software Engineering (ICSE), the top conference in a field that has its own ISI category, is not indexed.
- An LNCS-published workshop at ICSE, where authors would typically try out ideas not yet ready for ICSE submission, was indexed.

ISI indexes SIGPLAN Notices, an unrefereed publication devoting ordinary issues to notes and letters and special issues to proceedings of such conferences as POPL. POPL papers appear in ISI—on the same footing as a reader's note in a regular issue.

The database has little understanding of CS. Its 50 most cited CS references include "*Chemometrics in food science,*" from a "*Chemometrics and Intelligent Laboratory Systems*" journal. Many CS entries are not recognizable as milestone contributions. The cruelest comparison is with CiteSeer, whose Most Cited list includes many publications familiar to all computer scientists; it has not a single entry in common with the ISI list.

Our focus is evaluation of individuals rather than departments or laboratories.

ISI's "highly cited researchers" list includes many prestigious computer scientists but leaves out such iconic names as Wirth, Parnas, Knuth and all the 10 2000–2006 Turing Award winners except one. Since ISI's process provides no clear role for community assessment, the situation is unlikely to improve.

The inevitable deficiencies of alternatives pale in consideration:

9. In assessing publications and citations, ISI Web of Science is inadequate for most of CS and must not be used. Alternatives include Google Scholar, CiteSeer, and (potentially) ACM's Digital Library.

Anyone in charge of assessment should know that attempts to use ISI for CS will cause massive opposition and may lead to outright rejection of *any* numerical criteria, including more reasonable ones.

Assessment Formulae

A recent trend is to rely on numerical measures of impact, derived from citation databases, especially the *h-index*, the highest n such that C (n) $\geq n$, where C (n) is the citation count of the author's n-th ranked publication. Variants exist:

- The *individual h-index* divides the h-index by the number of authors, better reflecting individual contributions.
- The *g-index*, highest n such that the top n publications received (together) at least n^2 citations, corrects another h-index deficiency: not recognizing extremely influential publications. (If your second most cited work has 100 citations, the h-index does not care whether the first has 101 or 15,000.)

The "Publish or Perish" site[e] computes these indexes from Google Scholar data. Such indexes cannot be more credible than the underlying databases; results should always be checked manually for context and possible distortions. It would be as counterproductive to reject these techniques as to use them blindly to get definitive researcher assessments. There is no substitute for a careful process involving complementary sources such as peer review.

Assessing Assessment

Scientists are taught rigor: submit any hypothesis to scrutiny, any experiment to duplication, any theorem to independent proof. They naturally assume that processes affecting their careers will be subjected to similar standards. Just as they do not expect, in arguing with a Ph.D. student, to impose a scientifically flawed view on the sole basis of seniority, so will they not let management impose a flawed evaluation mechanism on the sole basis of authority:

10. Assessment criteria must themselves undergo assessment and revision.

Openness and self-improvement are the price to pay to ensure a successful process, endorsed by the community. This observation is representative of our more general conclusion. Negative reactions to new assessment techniques deserve consideration. They are not rejections of assessment per se but calls for a professional, rational approach. The bad news is that there is no easy formula; no tool will deliver a magic number defining the measure of a researcher. The good news is that we have ever more instruments at our disposal, which taken together can help form a truthful picture of CS research effectiveness. Their use should undergo the same scrutiny that we apply to our work as scientists. C

d All ISI searches as of mid-2008.

e See http://www.harzing.com/resources.htm#/pop.htm.

Bertrand Meyer (Bertrand.Meyer@inf.ethz.ch) is a professor of software engineering at ETH Zurich, the Swiss Federal Institute of Technology, and chief architect of Eiffel Software, Santa Barbara, CA.

Christine Choppy (Christine.Choppy@lipn.univ-paris13.fr) is a professor of computer science at Université Paris XIII and member of LIPN (Laboratoire d'Informatique de Paris Nord), France.

Jørgen Staunstrup (jst@itu.dk) is provost of the IT University in Copenhagen, Denmark.

Jan van Leeuwen (jan@cs.uu.nl) is a professor of computer science at Utrecht University, The Netherlands.

ACM Digital Library

www.acm.org/dl

The Ultimate Online INFORMATION TECHNOLOGY Resource!

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

The **ACM Digital Library** interface includes:

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

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

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

Available only to ACM Members.
Join ACM online at **www.acm.org/joinacm**

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

Phone: 1.800.342.6626 (U.S. and Canada)
+1.212.626.0500 (Global)
Fax: +1.212.944.1318
Hours: 8:30 a.m.-4:30 p.m., Eastern Time

Email: acmhelp@acm.org
Join URL: www.acm.org/joinacm
Mail: ACM Member Services
General Post Office
PO Box 30777
New York, NY 10087-0777 USA

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

Association for Computing Machinery

Advancing Computing as a Science & Profession

AD29

DOI:10.1145/1498765.1498781

The ecosystem of purpose-built languages is a key part of systems development.

BY MIKE SHAPIRO

Purpose-Built Languages

In my college computer science lab, two eternal debates flourished during breaks from long nights of coding and debugging: "emacs versus vi?" and "what is the best programming language?" Later, as I began my career in industry, I noticed that the debate over programming languages was also going on in the hallways of Silicon Valley campuses. It was the 1990s, and at Sun Microsystems many of us were watching Java claim significant mindshare among developers, particularly those previously developing in C or C++.

I have always found the notion of *best* language to be too subjective and too dependent on the nature of the programming task at hand. Over my career, however, I have spent significant time pondering two related questions that I think are more fundamental. First, is software engineering at large being done on *fewer* languages over time? That is, is the set of computer languages *converging*? Second, what makes a particular language "better" or useful or more rapidly adopted for a particular task?

In examining these questions I have found it particularly interesting to look not at the battle of the heavyweights, but rather at their less well-studied offshoots, the *purpose-built* languages. These languages sprout like weeds alongside the road of mainstream language development, and they exhibit properties and a history that lead one to reconsider instinctive answers to the fundamental language questions. Considering purpose-built languages, programming language development is not converging at all, and utility seems to have little to do with traditional notions of structure or properties that are empirically "better" from a language-design perspective. Purpose-built languages even defy a strict definition worthy of a prescriptive compiler grammarian: they somehow seem "smaller" than a full-fledged programming language; they are not always Turing-complete; they can lack formal grammars (and parsers); they are sometimes stand-alone but often a part of a more complex environment or containing program; they are often but not always interpreted; they are typically designed for a single purpose but often (accidentally) jump from one type of use to another. And some are even nameless.

Most significantly, purpose-built languages have often formed an essential part of the development of larger software systems such as operating systems, whether as a part of developer tools or as glue between distinct pieces of a larger environment. So it is particularly interesting to unearth some of these lesser-known creations and look at their connections to our larger language insights. In my career, while working on several commercial operating systems and large software components, I have come to conclude that not only are new languages developing all the time, but they are also often integral to the growth and maintenance of larger-scale software systems.

The Unix environment, with its philosophy of little tools that can be easily connected, was an ideal greenhouse for the growth of purpose-built languages. A cursory scan of Unix manuals from the early 1980s shows more than 20 little languages of vari-

ILLUSTRATION BY JOHN HERSEY

≤
δ
≈
≥
then
+
∞
=
then
|...|
≈
if
≠
||...||
Σ
÷
δ
≥
√
if
<|>
else
∂
≈
else
else
if
|...|
Δ
then
else
ß
{ : }
≤
∞
≠
§
x
≠
||...||
∞
+1
Σ
then
δ

Figure 1: Little languages in Unix, early 1980s.

Debuggers
adb
dbx
stabs

Shells/Utilities
sh
csh
awk
bc, dc
ed, ex, vi
sed

Text Formatting
eqn
pic
nroff, troff

Programming Tools
lex
make
m4
ratfor
yacc

Games
rogue, trek

Libraries
regex

Configuration
sendmail.cf

ous forms in active use, as shown in Figure 1.

These languages vary from complete programming languages (sh) to preprocessors (yacc) to command-line syntax (adb) to representations of state machines or data structures (regular expressions, debugger "stabs"). Twenty years later, when Sun Microsystems released the modern Unix system Solaris 10, almost all of the new significant operating-system features involved the introduction of new purpose-built languages: the DTrace debugging software introduced the D language for tracing queries; the Fault Management system included a language for describing fault propagations; the Zones and Service Management features included XML configuration grammars and new command-line interpreters.

The history of one of these little Unix languages, that of the adb debugger, is particularly illustrative of the accidental evolution and stickiness of something small but useful in a larger system.

Evolution Trumps Intelligent Design

The early development of Unix occurred on DEC PDP systems, which had a very simple debugger available known as ODT, or Octal Debugging Technique. (This terrific name conjures thoughts of a secret kung fu maneuver used to render the PDP's 12-bit registers paralyzed.) The ODT program supported an incredibly primitive syntax: an octal physical memory address was specified at the start of each command and suffixed with a single character (say, B for breakpoint) or a slash (/) to read and optionally to write the content of that memory location, as shown in Figure 2.

Thus, a little language was born. The ODT syntax clearly inspired the form of the first debugger for the new Unix system being developed on the PDP, which was simply called db. At the time of Unix v3 in 1971, the db command syntax borrowed the basic ODT model and began extending it with additional character suffixes to define addressing modes and formatting options, as shown in Figure 3.

By 1980, db had been replaced by adb, which was included with the AT&T SVR3 Unix distribution. The syntax had evolved to add new debugging commands over the intervening years and now supported not just simple addresses but arithmetic expressions (`123+456 /` was now legal). Also, a character after "/" now indicated a data format, and a character after "$" or ":" now indicated an action. The adb syntax is shown in Figure 4.

The addition of "$<" to read an external file of commands was particularly interesting, because it spawned the development of primitive adb programs or macros that executed a series of commands to display the contents of a C data structure at a particular memory address. That is, to display a kernel proc structure, you would take its address and then type "`$<proc`" to execute a predefined series of commands to display each memory of the C data structure for a process. The content of the proc macro in SunOS 4 from 1984 is shown in Figure 5. To make this output understandable, the "/" command could now be suffixed with quoted string labels, newlines (`n`) and tabs (`16t`) to be included among the decoded data.

Figure 2: ODT-8 debugger syntax, circa 1967.

`400 /`	display the content of memory location 400
`400 / 6046`	observe the value is 6046 at that location
`400 / 6046 2345`	rewrite the value to 2345
`3000B`	set a breakpoint at 3000

ODT-8 Octal Debugging Technique Manual, DEC-D8-COCO-D, Dec. 1967

Figure 3: Unix V3 "db" syntax, circa 1971

address `/`	Print a word in octal
address `\`	Print a byte in octal
address `` ` ``	Print a byte in ASCII
address `?`	Disassemble an instruction
`$`	Print the machine registers

Unix Version 3 Manual Pages, DB(1), Nov. 3, 1971.

Figure 4: AT&T SVR3 "adb" syntax, circa 1980.

expression `/o`	Print a two-byte integer in octal at the specified address
expression `/x`	Print a two-byte integer in hexadecimal at the specified address
expression `/d`	Print a two-byte integer in decimal at the specified address
$a	Print an Algol stack backtrace
$c	Print a C stack backtrace
$r	Print the machine registers
expression `:b`	Set a breakpoint
`$<` *filename*	Read and execute the command found in the specified file

Kernighan, B. Ratfor: A preprocessor for a Rational Fortran. *Software—Practice and Experience*. Oct. 1975.

The "." variable evaluates to the input address used when applying the macro, and the "+" variable evaluates to that input address incremented by the byte count of all preceding format characters. The macros were then maintained with the kernel source code.

More than a decade later, in 1997, I was working at Sun on what would become Solaris 7. This release was our first 64-bit kernel, but the kernel-debugging tool of choice was still adb just as it was in 1984, and our source base now contained hundreds of useful macro files. Unfortunately, the implementation of adb was essentially impossible to port cleanly from 32-bit to 64-bit to debug the new kernel, so it seemed the time was ripe for the development of a new clean code base with many more modern debugger features.

As I considered how best to approach the problem, I was struck by the fact that despite its brittle, unstructured code base, the key feature of adb was that its syntax was imbued deeply in the minds and behaviors of all of our most experienced and effective engineers. (As someone aptly put it at the time: "It's in the fingers.") So I set out to build a new modular debugger (mdb) that would support an API for advanced kernel debugging and other modern features, yet would remain precisely backward-compatible with existing syntax and macros. Sophisticated new features were added after a new prefix ("`::`") so they would not break the existing syntax (for example, "`::findleaks`" to check for kernel memory leaks). The entire syntax was at last properly encoded as a yacc parser. Macro files were phased out in favor of compiler-generated debug information, but the "`$<`" syntax was left as an alias. Another decade later, mdb remains the standard tool for postmortem debugging of the OpenSolaris kernel and has been extended by hundreds of programmers.

The debugger tale illustrates that a little purpose-built language can evolve essentially at random, have no clear design, no consistent grammar or parser, and no name, and yet endure and grow in shipping operating systems for more than 40 years. In the same time period, many mainstream languages came and went into the great beyond (Algol, Ada, Pascal, Cobol, and so on). Fundamentally, this debugger has survived for one reason: it concisely encoded the exact task its users performed and thereby connected to those users. Take an address, dump out its content, find the next address, follow it to the next location of interest, dump out its content, and

Figure 5: Debugging the "proc" structure on SunOS 4, circa 1984.

```
address $<proc
./"link"16t"rlink"16t"nxt"16t"prev"nXXXX
+/"as"16t"segu"16t"stack"16t"uarea"nXXXX
+/"upri"
+/"upri"8t"pri"8t"cpu"8t"stat"8t"time"8t"nice"nbbbbbb
+/"slp"8t"cursig"16t"sig"bbX
+/"mask"16t"ignore"16t"catch"nXXX
+/"flag"16t"uid"8t"suid"8t"pgrp"nXddd
+/"pid"8t"ppid"8t"xstat"8t"ticks"nddxd
+/"cred"16t"ru"16t"tsize"nXXX
+/"dsize"16t"ssize"16t"rssize"nXXX
+/"maxrss"16t"swrss"16t"wchan"nXXX
+/16+"%cpu"16t"pptr"16t"tptr"nXXX
+/"real itimer"n4D
+/"idhash"16t"swlocks"ndd
+/"aio forw"16t"aio back"8t"aio count"8t"threadcnt"nXXXX
```

Figure 6: Fortran and Ratfor, circa 1975.

```
      if (a .gt. b) goto 10          if (a <= b) {
                sw = 0                         sw = 0;
                write(6, 1) a, b                       write(6, 1) a, b
                goto 20                        } else {
   10           sw = 1                         sw = 1;
                write(6, 1) b, a                       write(6, 1) b, a
   20           …                              }
```

Figure 7: C and Algol mutant from early "adb".

```
#define  IF if(
#define  THEN ){
#define  ELSE } else {
#define  FI }
#define  ANDF &&
…
localsym(cframe)
L _ INT cframe;
{
        INT symflg;
        WHILE nextsym() ANDF localok
              ANDF  symbol.symc[0]!='~'
              ANDF (symflg=symbol.symf)!=037
        DO IF symflg>=2 ANDF symflg<=4
              THEN  localval=symbol.symv;
                    return(TRUE);
              ELIF  symflg==1
              THEN  localval=leng(shorten(cframe)+symbol.symv);
                    return(TRUE);
              ELIF  symflg==20 ANDF lastframe
              THEN  localval=leng(lastframe+2*symbol.symv-10);
                    return(TRUE);
              FI
        OD
        return(FALSE);
}
```

so on. For purpose-built languages, a deep connection to a task and the user community for that task is often worth more than clever design or elegant syntax.

Mutation and Hybridization

Mutation, some accidental and some intentional, often plays a critical role in the development of purpose-built systems languages. One common form of mutation involves adding a subset of the syntax of one language (for example, expressions or regular expressions) to another language. This type of mutation can be implemented using a preprocessor that converts one high-level form to another or intermingles preprocessed syntax with the target syntax of a destination language. Mutations may diverge far enough that a new hybrid language is formed. The parser tools yacc and bison are the most well-known examples of a complete hybrid language: a grammar is declared as a set of parsing rules intermingled with C code that is executed in response to the rules; the utilities then emit a finished C program that includes the rule code and the code to execute a parsing state-machine on the grammar.

Another example of this type of mutation in early Unix was the Ratfor (Rational Fortran) preprocessor developed by Kernighan. Ratfor permitted the author to write Fortran code with C expressions and logical blocks, and the result was translated into Fortran syntax with line numbers and goto statements, as shown in Figure 6.

An even stranger mutant language was a hybrid of C and Algol syntax developed using the C preprocessor and used in the code for, what else, adb. Apparently, Steve Bourne, the author of the Algol-like Unix sh syntax, was determined that some of Algol's genome would carry on in the species. Some sample code is shown in Figure 7.

Alas, a later version of the code was run through the preprocessor and then checked in so as to ease maintenance. Many future languages have included more clearly designed cross-breeding to ease the transition from one environment to another. Following the widespread adoption of C, its expression syntax found its way into an incredible number of new languages, little and big, including Awk, C++, Java, JavaScript, D, Ruby, and many others. Similarly, following the success of Perl, many other scripting languages adopted its useful extensions to regular expression syntax as a new canonical form. Core concepts such as expression syntax often form the bulk of a small language, and borrowing from a well-established model permits rapid language implementation and rapid adoption by users.

Take an address, dump out its content, find the next address, follow it to the next location of interest, dump out its content, and so on. For purpose-built languages, a deep connection to a task is often worth more than clever design or elegant syntax.

Symbiosis

In the development of a larger software system, little languages often live in symbiotic partnership with the mainstream development language or with the software system itself. The adb macro language described earlier would likely not have survived outside of the source-code base of its Unix parent. The macro language of your favorite spreadsheet is another example: it exists to provide a convenient way to manipulate the user-visible abstractions of the containing software application.

In the operating-system world, my favorite little-known example of symbiosis is the union of Forth and SPARC assembly language created at Sun as part of the work on the OpenBoot firmware. The idea was to create a small interpreter used as the boot environment on SPARC workstations. Forth was chosen for the boot and hardware bring-up environment for new hardware because the language kernel was tiny and could be brought up immediately on a new processor and platform. Then, using the Forth dictionaries, new commands could be defined on the fly in the interpreter for debugging. Since Forth permits its dictionaries to override the definition of words (tokens) in the interpreter, someone developed the creative idea of using the interpreter as a macro assembler for the hardware. A set of dictionaries was created that redefined each of the opcodes in SPARC ("ld," "move," "add," and so on) with Forth code that would compute the binary representation of the assembled instructions and store them into memory. Therefore, entire low-level functions could be written in what appeared to be assembly language, prefixed with Forth headers, and

typed into the tiny interpreter, which would then assemble the object code in memory as it parsed the tokens and executed the resulting routine.

In recent years, Web browsers have become fertile ground for mutation and symbiosis. Two central figures in modern Web development are interpreted JavaScript and XML. (XML itself is the syntax for a variety of other languages and an abundant source of hybrid languages and mutations.) In the common Ajax programming model, JavaScript objects can be serialized to XML form, and XML encodings can be used to pass remote procedure calls back to a server. In one such encoding, XML-RPC, a standard extension called multicall is provided for the browser client to issue multiple procedure calls from the client to the server in a single transfer. An example of a single call to a method x.foo and then a series of calls to the same method using multicall is shown in Figure 8.

While implementing Ajax user-interface code for a new line of storage products, the Sun Fishworks team wanted to develop a way to minimize unnecessary client-server interactions. The first concept developed was the notion of a `multicall` invocation whose parameter was the result of another call. In the following example, the method `x.foo` is called on the result of `x.bar` in a single XML-RPC interaction as shown in Figure 9.

The trick here is that the new structure member `methodParams` indicates that the next members are not static parameters but more methods to be called recursively, with the result pushed onto a stack. Once a stack had been born, it was only natural to start throwing in operators from a stack-based language, forming an entirely new interpreted language that itself is declared as data in JavaScript, sent to the server by the existing XML-RPC serialization, and executed by extensions to our XML-RPC interpreter engine. A few of the operators that we implemented at Sun are shown in Figure 10.

This example illustrates that the symbiotic relationship between JavaScript and XML essentially allows our language to exist without requiring its own lexer or parser, and fundamentally serves the purpose of offloading performance-critical code from JavaScript to our server and minimizing round-trips. In the videogame industry, a similar symbiosis (without the hybrid syntax) has developed between Lua and C/C++. The Lua scripting language provides a popular form for writing non-performance-critical code in videogame engines, and the Lua interpreter design makes it easy to bridge to C code.

Once two or more languages are interacting in a large software system, it becomes only natural for an ecosystem of tools (likely incorporating little languages with hybrid syntax) to spring up around them to ease the maintenance, development, and debugging of the entire system. If a rich ecosystem grows around the languages of a complete software system, both little and big, purpose-built and general-purpose, the longer the overall environment will thrive and its constituents survive. Therefore, as we build our towers of software abstraction ever higher, we should expect to see and know more languages, not fewer. C

Figure 8: JavaScript, XML-RPC, and Multicall.

```
x.foo({ bar: 123, baz: 456 });

system.multicall(
        { methodName: 'x.foo',
                params: [ { bar: 123, baz: 456 } ] },
        { methodName: 'x.foo',
                params: [ { bar: 789, baz: 654 } ] },
        { methodName: 'x.foo',
                params: [ { bar: 222, baz: 333 } ] }
)
```

Figure 9: Multicall with parameterized results.

```
system.multicall(
        { methodName: 'x.foo', methodParams: [
                { methodName: 'x.bar', params: [ 1, 2, 3 ] }
        ] },
        ...
)
```

Figure 10: Multicall with stack operators.

```
system.multicall(
        { foreach: [ [ 2, 4, 6 ], [
                { methodName: 'x.foo', params: [] },
                { push: [] },
                { div: [ { pop: [] }, 2 ] }
        ] ] }
        ...
)
```

Mike Shapiro (mws@sun.com) is a Distinguished Engineer at Sun Microsystems and is currently leading Sun's Fishworks advanced engineering team in San Francisco. He previously worked in Sun kernel engineering where he developed a variety of technologies for Solaris including pgrep, pkill, mdb, dumpadm, libproc, CTF, fmd, the DTrace D language and compiler, smbios, and a variety of features related to CPU, memory, I/O, and software fault handling and diagnosis.

A previous version of this article appeared in *ACM Queue*, January 2009.

DOI:10.1145/1498765.1498782

Web-based malware attacks are more insidious than ever. What can be done to stem the tide?

BY NIELS PROVOS, MOHEEB ABU RAJAB, AND PANAYIOTIS MAVROMMATIS

Cybercrime 2.0: When the Cloud Turns Dark

AS THE WEB has become vital for our day-to-day transactions, it has also become an attractive avenue for cyber crime. Financially motivated, the crime we see on the Web today is quite different from the more traditional network attacks. A few years ago adversaries heavily relied on remotely exploiting servers identified by scanning the Internet for vulnerable network services. Autonomously spreading computer worms such as Code Red and SQL Slammer were examples of such scanning attacks. Their huge scale put even the Internet at large at risk; for example, SQL Slammer generated traffic sufficient to melt down backbones. As a result, academia and industry alike developed effective ways to fortify the network perimeter against such attacks. Unfortunately, adversaries similarly changed tactics moving away from noisy scanning to more stealthy attacks.

PHOTOGRAPH BY NEIL CROSBY

Not only did they change their tactics, but also their motivation. Previously, large-scale events such as network worms were mostly exhibitions of technical superiority. Today, adversaries are primarily motivated by economic incentives to not only exploit and seize control of compromised systems for as long as possible but to turn their assets into revenue.

The Web offers adversaries a powerful infrastructure to compromise computer systems and monetize the resulting computing resources as well as any information that can be stolen from them. Adversaries achieve this by employing the Web to serve malicious Web content capable of compromising users' computers and running arbitrary code on them. This has largely been enabled due to the increased complexity of Web browsers and the resulting vulnerabilities that come with complex software. For example, a modern Web browser provides a powerful computing platform with access to different scripting languages, (for example, Javascript) as well as external plugins that may not follow the same security policies applied by the browser (for example, Flash, Java). While these capabilities enable sophisticated Web applications, they also allow adversaries to collect information about the target system and deliver exploits specifically tailored to a user's computer. Web attacks render perimeter defenses that disallow incoming connections useless against exploitation as adversaries use the browser to initiate out-bound connections to download attack payloads. This type of traffic looks almost identical to the users' normal browsing traffic and is not usually blocked by network firewalls.

To prevent Web-based malware from infecting users, Google has developed an infrastructure to identify malicious Web pages. The data resulting from this infrastructure is used to secure Web search results as well as protect browsers such as Firefox and Chrome. In this article, we discuss interesting Web attack trends as well as

some of the open challenges associated with this rising threat.

Web Attacks

As Web browsers have become more capable and the Web richer in features, it is difficult for the average user to understand what happens when visiting a Web page. In most applications visiting a Web page causes the browser to pull content from a number of different providers, for example, to show third-party ads, interactive maps, or display online videos. The shear number of possibilities to design Web pages and make them attractive to users is staggering. Overall, these features increase the complexity of the components that constitute a modern Web browser. Unfortunately, each browser component may introduce new vulnerabilities an adversary can leverage to gain control over a user's computer. Over the past few years we have seen an increasing number of browser vulnerabilities,[5,8] some of which have not had official fixes for weeks.

For an adversary to exploit a vulnerability, it requires the user visit a Web page that contains malicious content. One way to attract user traffic is to send spam email messages that advertise links to malicious Web pages. However, this delivery mechanism has some drawbacks. For the exploit to be delivered, the user must open the spam email and then click on the embedded link. The ubiquitous Web infrastructure provides a better solution to this bottleneck. While it is easy to exploit a Web browser, it is even easier to exploit Web servers. The relative simplicity of setting up and deploying Web servers has resulted in a large number of Web applications with remotely exploitable vulnerabilities. Unfortunately, these vulnerabilities are rarely patched, and therefore, remote exploitation of Web servers is increasing. To exploit users, adversaries just need to compromise a Web server and inject malicious content, for example, via an IFRAME pointing to an exploit server. Any visitor to such a compromised Web server becomes a target of exploitation. If the visitor's system is vulnerable, the exploit causes the browser to download and execute arbitrary payloads. We call this process "drive-by download." Depending on the popularity of the compromised Web site, an adversary may get access to a large user population. Last year, Web sites with millions of visitors were compromised that way.

Many drive-by downloads can be detected automatically via client honeypots. However, when adversaries use social engineering to trick the users into installing malicious software, automated detection is significantly complicated.

Taking Over Web Servers. Turning Web servers into infection vectors is unfortunately fairly straightforward. Over the last couple years, we have observed a number of different attacks against Web servers and Web applications, ranging from simple password guessing to more advanced attacks that can infect thousands of servers at once. In general, these attacks aim at altering Web site content to redirect visitors to servers controlled by the adversary. Here, we expand on some examples of recent dominant server attacks.

SQL Injection Attacks. SQL injection is an exploitation technique commonly used against Web servers that run vulnerable database applications. The vulnerability happens when user input is not properly sanitized (for example, by filtering escape characters and string literals) therefore causing well crafted user input to be interpreted as code and executed on the server. SQL injection has been commonly used to perpetrate unauthorized operations on a vulnerable database server such as harvesting users' information and manipulating the contents of the database. In Web applications running a SQL database to manage users' authentication, adversaries use SQL injection to bypass login and gain unauthorized access to user accounts or, even worse, to gain administrative access to the Web application. Other variants of these attacks allow the adversary to directly alter the contents of the server's database and inject the adversary's own content.

Last year, a major SQL injection attack was launched by the Asprox botnet.[15] In this attack several thousand bots were equipped with an SQL injection kit that starts by sending specially crafted queries to Google searching for servers that run ASP.net, and then launches SQL injection attacks against the Web sites returned from those queries. In these attacks the bot sends an encoded SQL query containing the exploit payload (similar to the format shown here) to the target Web server.

http://www.victim-site.com/asp_application.asp?arg=<encoded sql query>

The vulnerable server decodes and executes the query payload which, in the

Asprox case, yields SQL code similar to the snippet shown in Figure 1. The decoded payload searches the Web server folders for unicode and ASCII files and injects an IFRAME or a script tag in them. The injected content redirects the Web site users to Web servers controlled by the adversary and therefore subjects them to direct exploitation.

We monitored the Asprox botnet over the past eight months, and observed bots getting instructions to refresh their lists of the domains to inject. Overall, we have seen 340 different injected domains. Our analysis of the successful injections revealed that approximately six million URLs belonging to 153,000 different Web sites were victims of SQL injection attacks by the Asprox botnet. While the Asprox botnet is no longer active, several victim sites are still redirecting users to the malicious domains. Because bots inject code in a non-coordinated manner, many Web sites end up getting multiple injections of malicious scripts over time.

Redirections via `.htaccess`. Even when the Web pages on a server are harmless and unmodified, a Web server may still direct users to malicious content. Recently, adversaries compromised Apache-based Web servers and altered the configuration rules in the `.htaccess` file. This configuration file can be used for access control, but also allows for selective redirection of URLs to other destinations. In our analysis of Web servers, we have found several incidents where adversaries installed `.htaccess` configuration files to redirect visitors to malware distribution sites, for example, to fake anti-virus sites as we discuss later.

One interesting aspect of `.htaccess` redirections is the attempt to hide the compromise from the site owner. For example, redirection can be conditional based on how a visitor reached the compromised Web server as determined by the HTTP Referer header of the incoming request. In the incidents we observed, the `.htaccess` rules were configured so that visitors arriving via search engines were redirected to a malware site. However, when the site owner typed the URL directly into the browser's location bar, the site would load normally as the Referer header was not set.

Figure 2 shows an example of a compromised `.htaccess` file. In this example, users visiting the compromised site via any of the listed search engines will be redirected to http://89.28.13.204/in.html?s=xx. Notice that the initial redirect is usually to an IP address that acts as a staging server and redirects users to a continuously changing set of domains. The staging server manages which users get redirected where. For example, the staging server may check whether the user has already visited the redirector and return an empty payload on any subsequent visit. We assume this is meant to make analysis and reproduction of the redirection chain more difficult. Adversaries also frequently rewrite the `.htaccess` file to point to different IP addresses. Removing the `.htaccess` without patching the original vulnerability or changing the server credentials will not solve the problem. Many Web masters attempted to delete the `.htaccess` and found a new one on their servers the next day.

Figure 1. A decoded snippet of the SQL injection query sent by Asprox bots.[13]

```
DECLARE @T VARCHAR(255),@C VARCHAR(255)
DECLARE Table _ Cursor CURSOR FOR  SELECT a.name,b.name
FROM sysobjects a,syscolumns b
WHERE a.id=b.id AND a.xtype='u'
AND (b.xtype=99 OR b.xtype=35
OR b.xtype=231 OR b.xtype=167)
OPEN Table _ Cursor FETCH NEXT FROM Table _ Cursor INTO @T,@C
WHILE(@@FETCH _ STATUS=0)
BEGIN EXEC('UPDATE ['+@T+']
SET ['+@C+']=RTRIM(CONVERT(VARCHAR(4000),['+@C+']))+'''''')
FETCH NEXT FROM Table _ Cursor INTO @T,@C
END CLOSE Table _ Cursor
DEALLOCATE Table _ Cursor
```

Taking Over Web Users. Once the adversaries have turned a Web server into an infection vector, visitors to that site are subjected to various exploitation attempts. In general, client exploits fall under two main categories: automated drive-by downloads and social engineering attacks.

Drive-by downloads. In this category, adversaries attempt to exploit flaws in either the browser, the operating system, or the browser's external plugins. A successful exploit causes malware to be delivered and executed on the user's machine without her knowledge or concent. For example, a popular exploit we encountered takes advantage of a vulnerability in Microsoft Data Access Components (MDACS) that allows arbitrary code execution on a user's computer.[7] A 20-line Javascript code snippet was enough to exploit this vulnerability and initiate a drive-by download.

Another popular exploit is due to a vulnerability in Microsoft Windows WebViewFolderIcon. The exploit Javascript uses a technique called heap spraying that creates a large number of Javascript string objects on the heap. Each Javascript string contains x86 machine code (shellcode) necessary to download and execute a binary on the exploited system. By spraying the heap, an adversary attempts to create a copy of the shellcode at a known location in memory and then redirects program execution to it.

Social engineering attacks. When drive-by downloads fail to compromise a user's machine, adversaries often employ social

Figure 2. A snippet from the .htacsess file of a compromised Apache server.[11]

```
RewriteEngine On
RewriteCond %{HTTP _ REFERER} .*google.*$ [NC,OR]
RewriteCond %{HTTP _ REFERER} .*aol.*$ [NC,OR]
RewriteCond %{HTTP _ REFERER} .*msn.*$ [NC,OR]
RewriteCond %{HTTP _ REFERER} .*altavista.*$ [NC,OR]
RewriteCond %{HTTP _ REFERER} .*ask.*$ [NC,OR]
RewriteCond %{HTTP _ REFERER} .*yahoo.*$ [NC]
RewriteRule .* http://89.28.13.204/in.html?s=xx [R,L]
```

engineering techniques to trick users into installing and running malware by themselves. Unfortunately, the Web is rich with deceptive content that lures users into downloading malware.

One common class of attacks includes images that resemble popular video players, along with a false warning that the computer is missing essential codecs for displaying the video, or that a newer version of the video player plugin is required to view it. Instead, the provided link is for downloading a trojan that, once installed, gives the adversary full control over the user's machine.

A more recent trick involves fake security scans. A specially crafted Web site displays fake virus scanning dialogs, along with animated progress bars and a list of infections presumably found on the computer. All the warnings are false and are meant to scare the user into believing their machine is infected. The Web site then offers a download as solution, which could be another trojan, or ask the user for a registration fee to perform an unnecessary clean-up of their machine.

We have observed a steady increase in fake anti-virus attacks: From July to October 2008, we measured an average of 60 different domains serving fake security products, infecting an average of 1,500 Web sites. In November and December 2008, the number of domains increased to 475, infecting over 85,000 URLs. At that time the Federal Trade Commission reported that more than one million consumers were tricked into buying these products, and a U.S. district court issued a halt and an asset freeze on some of the companies behind these fake products.[3] This does not appear to have been sufficient to stop the scheme. In January 2009, we observed over 450 different domains serving fake security products, and the number of infected URLs had increased to 148,000.

Malware activities on the user's machine. Whether a user was compromised by a social engineering attack or a successful exploit and drive-by download, once the adversaries have control over a user's machine, they usually attempt to turn their work into profit.

In prior work,[10] we analyzed the behavior of Web malware installed by drive-by downloads. In many cases, malware was equipped with key-loggers to spy on the user's activity. Often, a back door was installed, allowing the adversary to access the machine directly at a later point in time. More sophisticated malware turned the machine into a bot listening to remote commands and executing various tasks on demand. For example, common uses of botnets include sending spam email or harvesting passwords or credit cards. Botnets afford the adversary a degree of anonymity since spam email appears to be sent from a set of continuously changing IP addresses making it harder to blacklist them.

To help improve the safety of the Internet, we have developed an extensive infrastructure for identifying URLs that trigger drive-by downloads. Our analysis starts by inspecting pages in Google's large Web repository. While exhaustive inspection of each page is prohibitively expensive as the repository contains billions of pages, we have developed a lightweight system to identify candidate pages more likely to be malicious. The candidate pages are then subjected to more detailed analysis in a virtual machine allowing us to determine if visiting a page results in malicious changes to the machine itself. The lightweight analysis uses a machine-learning framework that can detect 90% of all malicious pages with a false positive rate of only 10^{-3}. At this false positive rate, the filter reduces the workload of the virtual machines from billions of pages to only millions. The URLs that are determined to be malicious are further processed into host-suffix path-prefix patterns. Since 2006, our system has been used to protect Google's search. Our data is also published via Google's Safe Browsing API to browsers such a Firefox, Chrome, and Safari. These browsers employ our data to prevent users from visiting harmful pages.

Challenges

Despite our efforts to make the Web safer for users, there are still a number of fundamental challenges requiring future work, including:

Securing Web Services. Establishing a presence on the Web, ranging from simple HTML pages to advanced Web applications, has become an easy process that enables even people with little technical knowledge to set up a Web service. However, maintaining such a service and keeping it secure is still difficult. Many Web application frameworks require programmers to follow strict security practices, such as sanitizing and escaping user input. Unfortunately, as this burden is put onto the programmer, many Web applications suffer from vulnerabilities that can be remotely exploited.[12, 14] For example, SQL injection attacks are enabled by a programmer neglecting to escape external input.

Popular Web applications such as bulletin boards or blogs release security updates frequently, but many administrators neglect to update their installations. Even the Web server software itself, such as Apache or IIS, is often out-of-date. In previous work,[10] we found over 38% of Apache installations and 40% of PHP installations in compromised sites to be insecure and out-of-date.

To avoid the compromising of Web applications, it is important to develop mechanisms to keep Web servers and Web applications automatically patched. Some Web applications already notify Web masters about security updates, but the process of actually installing security patches is often still manual and complicated.

It is difficult to be completely safe against drive-by downloads. All that is required for an adversary to gain control over your system is a single vulnerability. Any piece of software that is exposed to Web content and not up-to-date can become the weakest link.

Many browser plugins and add-ons, such as toolbars, do not provide automatic updates. Furthermore, system updates often require a restart after installation discouraging users from applying the security patches on time.

Even if a system was fully patched, the window of vulnerability for some software is often very large. According to Krebs, major browsers were unsafe for as long as 284 days in 2006, and for at least 98 days criminals actively used vulnerabilities for which no patches were available to steal personal and financial data from users.[5, 6] Although progress on providing fault isolation in browsers that may prevent vulnerabilities from being exploited has been made,[1, 4] a completely secure browser still needs to be developed.

Detecting Social Engineering At-

tacks. Many drive-by downloads can be detected automatically via client honeypots. However, when adversaries use social engineering to trick the users into installing malicious software, automated detection is significantly complicated. Although, user interactions can be simulated by the client honeypot, a fundamental problem is the user's expectation about the functionality of a downloaded application compared to what it actually does. In the video case described earlier, the user expected to watch a video. After downloading and installing such a trojan, nothing usually happens. This could warn the user that something is amiss and might result in the user trying to fix their system. However, there is no reason why the installed software could not also play a video leaving the user with no reasons to suspect that she was infected.

Similarly, in addition to extorting the user for money, some of the fake anti-virus software does actually have some detection capability for old malware. The question then is how to determine if a piece of software functions as advertised. In general, this problem is undecidable. For example, the popular Google toolbar allows a user to opt into receiving the pagerank of a visited page. This works by sending the current URL to Google and then returning the associated pagerank and displaying it in the browser. This functionality was desired by the user and a legitimate feature. However, a similar piece of software might not disclose its functionality and send all visited URLs to some ominous third party. In that case, we would label the software spyware.

Automated analysis[2,9] is more difficult when malicious activity is triggered only under certain conditions. For example, some banking trojans watch the URL in the browser window and overlay a fake input field only for specific banking Web sites. Automated tools may discover the overlay functionality, but if the trojan was to compare against one-way hashes of URLs determining which banks were targeted could be rather difficult.

Conclusion

Without doubt, Web-based malware is a security concern for many users. Unfortunately, the root cause that allows the Web to be leveraged for malware delivery is an inherent lack of security in its design—neither Web applications nor the Internet infrastructure supporting these applications were designed with a well-thought-out security model. Browsers evolved in complexity to support a wide range of applications and inherited some of these weaknesses and added more of their own. While some of the solutions in this space are promising and may help reduce the magnitude of the problem, safe browsing will continue to be a far sought-after goal that deserves serious attention from academia and industry alike. C

Whether a user was compromised by a social engineering attack, or a successful exploit and drive-by download, once the adversaries have control over a user's machine, they usually attempt to turn their work into profit.

References

1. Barth, A., Jackson, C., and Reis, C. *The Security Architecture of the Chromium Browser;* http://crypto.stanford.edu/websec/chromium/chromium-security-architecture.odf.
2. Brumley, D., Hartwig, C., Kang, M., Liang, Z., Newsome, J., Song, D., and Yin, H. BitScope: Automatically dissecting malicious binaries. Technical Report, Technical Report CMU-CS-07-133, School of Computer Science, Carnegie Mellon University, March 2007.
3. Court halts bogus computer scans (Dec. 2008); www.ftc.gov/opa/2008/12/winsoftware.shtm
4. Grier, C., Tang, S., and King, S. Secure Web browsing with the OP Web browser. *Security and Privacy*, 2008. IEEE Symposium, 2008, 402–416.
5. Krebs, B. Internet Explorer unsafe for 284 days in 2006. *Washington Post Online* Blog, Jan. 2007.
6. Krebs, B. Blogfight: IE vs. Firefox security. *Washington Post Online* Blog, Jan. 2009.
7. Microsoft. Microsoft Security Bulletin MS06-014: Vulnerability in the Microsoft Data Access Components (MDACS) Function Could Allow Code Execution. May, 2006.
8. Microsoft. Microsoft Security Advisory (935423): Vulnerability in Windows Animated Cursor Handling, Mar. 2007.
9. Moser, A., Kruegel, C., and Kirda, E. Exploring multiple execution paths for malware analysis. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2007, 231–245.
10. Polychronakis, M., Mavrommatis, P., and Provos, N. Ghost Turns Zombie: Exploring the Life Cycle of Web-based Malware. In *Proceedings of the 1st USENIX Workshop on Large-Scale Exploits and Emergent Threats* (Apr. 2008).
11. Provos, N. Using htaccess To Distribute Malware. Dec. 2008; www.provos.org/index.php?/archives/55-Using-htaccess-To-Distribute-Malware.html.
12. Provos, N., Mavrommatis, P., Rajab, M.A., and Monrose, F. All your IFRAMEs point to us. *USENIX Security Symposium*, 2008, 1–16.
13. Raz, R. Asprox silent defacement. *Chapters in Web Security*, Dec. 2008; http://chaptersinWebsecurity.blogspot.com/2008/07/asprox-silent-defacement.html.
14. Small, S., Mason, J., Monrose, F., Provos, N., and Stubblefield, A. To catch a predator: A natural language approach for eliciting malicious payloads. *USENIX Security Symposium*, 2008, 171–184.
15. Stewart, J. Danmec/Asprox SQL injection attack tool analysis. *Secure Works Online*, May 2008; www.secureworks.com/research/threats/danmecasprox.

Niels Provos (niels@google.com) joined Google in 2003 and is currently a principle software engineer in the Infrastructure Security Group. His areas of interest include computer and network security as well as large-scale distributed systems. He is serving on the USENIX Board of Directors.

Moheeb Abu Rajab (moheeb@google.com) joined Google in 2008 and is currently a software engineer in the Infrastructure Security Group. His areas of interest include computer and network security.

Panayiotis Mavrommatis (Panayiotis@google.com) joined Google in 2006 and is currently working as a senior software engineer in the Security Group.

DOI:10.1145/1498765.1498783

Dynamic languages offer a taste of object-relational mapping that eases application code.

BY CHRIS RICHARDSON

ORM in Dynamic Languages

A major component of most enterprise applications is the code that transfers objects in and out of a relational database. The easiest solution is often to use an ORM (object-relational mapping) framework, which allows the developer to declaratively define the mapping between the object model and database schema and express database-access operations in terms of objects. This high-level approach significantly reduces the amount of database-access code that needs to be written and boosts developer productivity.

Several ORM frameworks are in use today. For example, the Hibernate,[2] TopLink,[11] and OpenJPA[1] frameworks are popular with Java developers, and NHibernate[10] is used by many .NET developers. Two newer ORM frameworks that have recently received a lot of attention from enterprise developers are Active Record[15] for Ruby[14] and GORM (Grails Object Relational Mapping)[12] for Groovy.[7] These new frameworks differ from traditional ORM frameworks in that they are written in dynamic languages that allow new program elements to be created at runtime. Active Record and GORM use these dynamic capabilities in ways that can significantly simplify an application.

This article looks at how GORM works. It compares and contrasts GORM with Hibernate, focusing on three areas: defining object-relational mapping; performing basic save, load, and delete operations on persistent objects; and executing queries. It describes how GORM leverages the dynamic features of Groovy to provide a different flavor of ORM that has some limitations but for many applications is much easier to use.

Groovy, Grails, and GORM

GORM is the persistence component of Grails, which is an open source framework that aims to simplify Web development. Grails is written in Groovy, a dynamic, object-oriented language that runs on the JVM (Java Virtual Machine). Because Groovy interoperates seamlessly with Java, Grails can leverage several mature Java frameworks. In particular, GORM uses Hibernate, a popular and robust ORM framework.

GORM, however, is much more than a simple wrapper around the Hibernate framework. Instead, it provides a very different kind of API. GORM is different in two ways. First, the dynamic features of the Groovy language enable GORM to do things that are impossible in a static language. Second, the pervasive use of CoC (Convention over Configuration) in Grails reduces the amount of configuration required to use GORM. Let's look at each of these reasons in more detail.

Dynamic Groovy. GORM relies heavily on the dynamic capabilities of the Groovy language. In particular, it makes extensive use of Groovy's ability to define methods and properties at runtime. In a static language such as Java, a property access or a method invocation is resolved at compile time. In comparison, Groovy does not resolve property accesses and method invocations until runtime. A Groovy application can dynamically define methods and properties.

ILLUSTRATION BY DAVE BOLLINGER

Figure 1.

```
groovy:000> String.metaClass.doubleString = { -> delegate + delegate }
===> groovy.lang.ExpandoMetaClass$ExpandoMetaProperty@14a18d
groovy:000> "ACM Queue".doubleString()
===> ACM QueueACM Queue
```

Figure 2.

Using Annotations for ORM

```
@Entity
class Customer {
…
@Id
@GeneratedValue
private long id;

@Version
private long version ;

private String name;

@OneToMany
private Set<Account> accounts;
…
}
```

Using XML for ORM

```
<hibernate-mapping
 default-access="field">

<class name="Customer">
 <id name="id">
  <generator class="native" />
 </id>

<version name="version"/>

<property name="name"/>

<set name="accounts">
 <key/>
 <one-to-many class="Account"/>
</set>
…
</class>
…
</hibernate-mapping>
```

Groovy provides a couple of different ways to add methods and properties to a class at runtime. The simplest approach is to define `propertyMissing()` or `methodMissing()` methods. The `propertyMissing()` method is called by the Groovy runtime when the application attempts to access an undefined property. Similarly, the `methodMissing()` method is called when the application calls an undefined method. These methods enable an object to behave as if the property or method existed.

The second and more sophisticated approach is to use the wonderfully named `ExpandoMetaClass`. Every Groovy class has a `metaClass` property that returns an `ExpandoMetaClass`. An application can add methods or properties to a class by manipulating this metaclass. For example, Figure 1 is a code snippet that adds a method to the String class that concatenates a string with itself.

This code snippet obtains the `String` metaclass and assigns to its `doubleString` property a closure (a kind of anonymous method) that implements the new method.

Groovy applications often use `methodMissing()` and `ExpandoMetaClass` together. The first time an undefined method is invoked, `missingMethod()` defines the method using the `ExpandoMetaClass`. The next time around, the newly defined method is called directly, thereby bypassing the relatively expensive `missingMethod()` mechanism.

Later you will see how Grails uses `methodMissing()` and `ExpandoMetaClass` to inject persistence-related methods and properties into domain classes at runtime, thereby simplifying application code.

Convention over configuration. The second key idea in GORM is CoC. Its premise is that a framework should have sensible defaults and should not require developers explicitly to configure every facet; instead, only the exceptional cases should require configuration. CoC was first popularized by the Rails and Grails frameworks, but mainstream Java EE frameworks including Spring[16] have begun to adopt the concept. Today, developers expect modern Java EE frameworks to require much less configuration than older frameworks.

CoC is used throughout Grails. For example, built-in defaults determine how to map an HTTP request to a handler class. Similarly, GORM has rules for defining which classes to persist and how to include defaults for column and table names. Because of CoC, a typical Grails application contains significantly less configuration code and metadata than an application using a traditional framework.

Now that we have looked at the key underpinnings of GORM, let's learn how to use it.

GORM Mapping

A key part of using an ORM framework is specifying how the object model maps to the database. The developer must specify how classes map to tables, attributes map to columns, and relationships map to either foreign keys or join tables. This section looks at how this works using a traditional ORM framework and then how it is accomplished in Grails.

Mapping with XML and annotations. The persistent state of a Java class is either its fields or its properties. A field is the Java equivalent of an instance variable. A property is defined by getter and setter methods that follow the JavaBeans[6] naming conventions. For example, `getFoo()` and `setFoo()` define the property called `foo`. The getter and setter methods often provide access to a field of the same name as the property, although they are not required to do so.

A Hibernate application can map the fields or properties of domain classes to the database schema using either XML or annotations. Figure 2 shows an annotation example on the left and an XML example on the right. Both examples persist the fields of the `Customer` class, but an application can persist properties either by annotating the getter methods or by omitting the default-access attribute from the XML document.

XML and annotations produce equivalent metadata. They both specify that the `Customer` class is persistent. They also specify that Hibernate should generate an object's primary key using whatever mechanism is appropriate for the underlying database and store

it in the id field. The version field is configured to store a Hibernate-maintained version number. They both persist the name field and specify that the accounts field represents a one-to-many relationship.

XML and annotations both have defaults for table and column names. The table name defaults to the name of the class and the column name defaults to the name of the property. You can override these defaults using extra annotations or XML attributes and elements. For example, you can specify the table name using the @Table annotation or the name attribute of the <class> element.

Each approach has benefits and drawbacks. One advantage that XML has over annotations is that it separates the O/R mapping from the Java code, which decouples the domain classes from Hibernate. One problem with this separation is that it can be more difficult to keep the mapping and code in sync. XML also tends to be more verbose than annotations. Moreover, the XML mapping must explicitly list all of the persistent properties of a class, whereas fields of certain basic types such as Customer.name are automatically persistent when using annotations.

Another problem is that regardless of whether you are using XML or annotations, you often need to add fields to store the primary key and a version number. The primary-key field is usually required by Hibernate or by a domain object's clients. The version number is used for optimistic locking. The trouble with these fields, however, is that typically the application's business logic does not require them. They must be added to every domain class solely to support persistence.

O/R mapping in GORM. Grails relies heavily on Convention over Configuration when defining ORM. It automatically treats classes in the grails app/domain directory as being persistent. GORM automatically persists the properties of each class. It defaults table and column names from the class and property names. GORM also adds primary-key and version-number properties to each class.

The following is an example domain class. The Customer class has a field called name. Also, because this field has default visibility, Groovy automatically defines the name property by defining getName() and setName() methods.

```
class Customer {
    String name
}
```

GORM automatically maps the Customer class to the customer table and maps the name property to the name column. GORM adds an id property to the class and maps it to a primary-key column called id. It also adds a version property and maps it to a version column. Unlike a traditional ORM framework, GORM requires very little configuration, provided that the database schema matches the defaults.

Another nice feature of GORM is that it will maintain creation and last updated times for domain model classes. You simply have to define lastUpdated and dateCreated properties on your classes, and GORM will automatically update them. In comparison, you must write code to do this when using vanilla Hibernate.

GORM also makes it easy to map relationships by using static properties to supply metadata in a similar fashion to annotations in other languages. For example, the static property hasMany defines the one-to-many relationships for a domain class. The value of the hasMany property is a map. Each map entry defines a one-to-many relationship: its key is the name of the property that stores the collection, and its value is the class of the collection elements. For each one-to-many relationship GORM adds a property to store the collection of objects, as well as methods for maintaining the relationship.

The following is an example of how to map a one-to-many relationship between the Customer class and the Account class.

```
class Customer {
    static hasMany =
    [accounts : Account]
}

class Account {
    static belongsTo =
    Customer
    Customer customer
}
```

The collection of accounts is stored in a property called accounts, which GORM adds to the Customer class at runtime. The relationship is mapped using a foreign key called customer_id in the account table. The belongsTo property specifies that a Customer owns the account and it should be deleted if the customer is deleted.

GORM also dynamically defines a couple of methods for managing this relationship. The addToAccounts() method adds an account to the collection, and the removeFromAccounts() method removes an account. These methods also maintain the inverse relationship from Account to Customer. By automatically defining these methods, which would otherwise have to be written by hand, GORM simplifies the code and makes it less error prone.

Configuring the mapping. CoC reduces the amount of configuration that is required. Sometimes, however, you need to specify some aspects of the ORM. For example, table or column

Figure 3.

(a)

```
Long pk = …
Session session = sessionFactory.getCurrentSession()
Account account = (Account)session.get(Account.class, pk)
```

(b)

```
interface AccountDao {
  Account get(long accountId);
  …
}
class AccountDaoImpl implements AccountDao {
  public Account get(long accountId) {
   Session session = sessionFactory.getCurrentSession()
   return (Account)session.get(Account.class, pk)
}
}
```

names might not match the defaults, or perhaps a class has derived properties that should not be persisted. To support these requirements, GORM lets you specify various aspects of the ORM. Rather than using a different configuration language such as XML or annotations, however, GORM uses snippets of Groovy code in the domain classes.

Here is an example of how to override the default table and column names and specify that a property should not be persisted.

```
class Customer {
   static transients =
   ["networth"]

   static mapping = {
      id column: 'customer_id'
      table 'crc_customer'
       columns {
        name column:
        'customer_name'
      }
   }


   def getNetworth() {
      def networth = 0
      accounts.each
       {networth + it.balance}
      networth
   }
...
}
```

In this example, the `transients` property, which is a list of property names, specifies that the `networth` property, which calculates the total balance of the customer's accounts and is defined by the `getNetworth()` method, is not persistent. The `mapping` property maps the `Customer` class to the `crc_customer` table; the `id` property to the `customer_id` column; and the `name` property to the `customer_name` property.

The value of the `mapping` property is a Groovy closure object, which is a kind of anonymous method. Although it might not be immediately apparent, the body of the mapping closure is a sequence of method calls. For example, `"id column: 'customer_id'"` is a call to an id method with a map parameter containing a single entry that has `column:` as the key and `'customer_id'` as the value.

The mapping closure is an example of a DSL (domain-specific language),[4] which is a mini-language for representing information about a domain. DSLs are used by Grails for a variety of configuration tasks. Groovy applications often define one or more DSLs as well. Several features of the Groovy language make it easy to write DSLs, including closures, literal lists and maps, and a flexible syntax that does not, for example, require parentheses around method arguments. They enable a developer to write highly readable and concise DSLs without having to go outside of the language and use mechanisms such as XML.

Manipulating Persistent Objects

Applications must save, load, and delete persistent objects. A traditional ORM framework provides an API object that has methods for manipulating persistent data. GORM, however, takes a very different and simpler approach that leverages Groovy's ability to define new methods at runtime.

When using a traditional ORM framework, the application manipulates persistent data by invoking methods on an API object. For example, a Hibernate application uses a Session object, which represents a connection to the database to save, load, and delete persistent objects. Note that usually an application needs only to save newly created objects. Most ORM frameworks, including Hibernate, track changes to persistent objects and automatically update the database.

Figure 3a shows a code snippet that illustrates how an application can load an account with the specified primary key. This code snippet obtains the current Session and calls `get()` to load the specified account.

An application's business logic could use the `Session` directly. Doing so, however, would violate the Separation of Concerns principle.[3] The application code would be a mix of business logic and persistence logic, which makes it more complex and much more difficult to test. It also tightly couples the business logic to the ORM framework, which is undesirable given the furious rate at which Java EE frameworks evolve.

Figure 4.

(a)

```
methclass
AccountDaoImpl .. {
public List<Account> findByBalanceLessThan(double threshold) {
  Session session = Session.currentSession();
  Query query = session.createQuery("from Account where balance < ?")
  query.setParameter(1, threshold);
  return (List<Account>) query.list();
}
```

(b)

```
class AccountDaoImpl .. {
 public List<Account> findByBalanceLessThan(double threshold) {
  Criteria c = session.createCriteria(Account.class)
  c.add(Restrictions.lt("balance", threshold))
  return c.list();
}
```

Figure 5.

(a)

```
def accounts = Account.findAllByBalanceLessThan(threshold)
```

(b)

```
List accounts = Account.findAll("from Account where balance < ?",
[threshold])
```

(c)

```
def c = Account.createCriteria()
def results = c.list {
   lt("balance", threshold)
}
```

A better approach is to use the DAO (data-access object) pattern,[8] which encapsulates the data-access logic within a DAO class. A DAO defines methods for persisting, loading, and deleting objects. It also defines finder methods, which execute queries and are discussed in more detail later. The DAO methods are invoked by the business logic and call the ORM framework to access the database.

Figure 3b shows an example of a Hibernate DAO for the `Account` domain class. This DAO consists of the `AccountDao` interface, which defines the public methods, and an `AccountDaoImpl` class, which implements the interface and calls Hibernate to access the database.

The DAO pattern simplifies the business logic and decouples it from the ORM framework, but it has some drawbacks. The first problem is that many DAOs consist of cookie-cutter code that is tedious to develop and maintain. This has caused some developers to abandon the DAO pattern and write business logic that directly calls the ORM framework, despite the drawbacks of doing so.

One way to reduce the amount of cookie-cutter code is to use a generic DAO.[9] This consists of a superinterface, which defines the CRUD (create, read, update, delete) operations, and a superclass, which implements them. The superinterface and the superclass are parameterized by the entity class, which makes them strongly typed. Application DAOs extend the generic DAO interface and implementation class. Using a generic DAO eliminates some but not all of the cookie-cutter code, so it's only a partial solution.

Another problem with using DAOs is that some application classes might not be able to reference them. Modern Java EE applications resolve inter-component references using a mechanism known as dependency injection.[5] When the application starts up, an assembler instantiates each application component and injects it with references to the required components. Resolving inter-component references in this way simplifies the components and promotes loose coupling.

One limitation of dependency injection, however, is that it does not easily allow noncomponents such as domain objects to obtain references to components such as DAOs. Domain objects are instantiated by the application rather than by the component assembler. It's tricky, although not impossible,[13] for the component assembler to intercept the instantiation of such objects and inject dependencies. As a result, business logic residing in domain objects cannot always reference components such as DAOs.

Despite limitations, developers of a wide range of applications will find GORM extremely useful. Developers can use GORM independently of Grails, but it is targeted at Web application developers who can benefit from the rapid development capabilities of the Grails framework.

There are a couple of ways to work around this limitation. Components such as services, which can use dependency injection, pass DAOs as method parameters to domain classes, which cannot. This works well in some situations, but in more complex cases the code becomes cluttered with extra parameters. Another workaround is to move the code that needs to use the DAOs into components where it can use dependency injection. The trouble with moving business logic out of the entities is that it degrades the design and results in an anemic domain model.

Dynamic persistence methods in GORM. GORM provides a different style of persistence API. Rather than providing an API object, it injects methods for saving, loading, and deleting persistent objects into domain classes. This mechanism decouples the business logic from the underlying ORM framework without having to use DAOs. It also eliminates the need for application code to obtain references to the ORM framework API objects or DAOs.

GORM injects several methods into domain classes, including `save()`, which saves a newly created object; `get()`, which loads an object by its primary key; and `delete()`, which deletes an object. Here is an example that uses these methods:

```
Customer c = new
 Customer("John Doe")

if (!c.save())
   fail "save failed"

Customer c2 = Customer.get(c.id)

c2.delete()

assertNull Customer.get(c.id)
```

This example creates a `Customer` object and saves it in the database by

calling `save()`. It then loads the customer by calling `Customer.get()`. Finally, it deletes the customer by calling `delete()`. Note that none of these methods is defined in the source code for the `Customer` class. GORM implements them using the `missingMethod()`/`ExpandoMetaClass` mechanism described earlier.

GORM's dynamically defined persistence methods eliminate a lot of DAO code while decoupling application code from the ORM framework. GORM sidesteps the problem of how noncomponents obtain references to DAOs. Code anywhere in a GORM application can perform data-access operations. Of course, whether that is always appropriate is another issue since, as I discuss later, it can result in database-access code being scattered throughout the application.

One significant limitation of GORM is that it does not support multiple databases. A Hibernate application explicitly uses a particular session and can thereby select which database to access. A GORM application uses the persistence methods that are injected into domain classes and cannot select which database to use. Moreover, as of the time of writing, the mechanism used for configuring GORM does not support multiple databases. This limitation might prevent many applications from using GORM, including those that horizontally scale by using multiple databases.

Executing Queries

An application may not know the primary keys of the objects it needs to load. Instead, it must execute a query that retrieves objects based on the values of their attributes. When using a traditional ORM framework, an application executes queries by invoking methods on API objects provided by the framework. This code is usually encapsulated by DAOs to decouple the application from the ORM framework. As with persistence methods, GORM takes a different approach that often simplifies application code.

Hibernate provides several ways to execute queries. An application can, for example, use the `Query` interface to execute queries written in HQL (Hibernate Query Language), which is a powerful object-oriented, textual query language. Figure 4a is a DAO finder that retrieves accounts with balances less than some minimum.

GORM injects persistence-related methods into domain classes at runtime. It eliminates a significant amount of data-access methods and classes, while still decoupling the business logic from the ORM framework.

This method obtains a `Session` and creates a `Query` object. It then sets the query's parameter and executes the query, which returns a list of `Account` objects.

A Hibernate application can also use the Criteria Query API to execute queries. This API provides methods for building a query programmatically. It is especially useful when an application needs to build a query dynamically since it eliminates the need to concatenate query string fragments. (Figure 4b is an example of a criteria query that finds accounts with low balances.) This code snippet creates a `Criteria` object for the `Account` class. It then adds a restriction and executes the query.

One problem with the DAO finders is that most have the same structure as the example: create a query, set the parameters, and execute the query. The only variables are the query and the parameters. As with the persistence methods, these cookie-cutter methods and the DAOs that contain them are tedious to develop, test, and maintain.

Dynamic GORM Finders

GORM has a dynamic finder mechanism that eliminates the need to write simple queries and DAO finder methods. It uses Groovy's dynamic capabilities to add finder methods to domain classes. For example, an application can find accounts with low balances, as shown in Figure 5a. Provided that the method name follows certain naming conventions, the `missingMethod()`/`ExpandoMetaClass` mechanism intercepts the call to the method and defines a method that parses the method name to build a query and executes it.

GORM dynamic finders support a rich query language. Finder method names can use comparison operators such as equals, less than, and greater than. They can also use the `and`, `or`, and `not` logical operators. Even though the query language is limited to the properties of a single class—no joins—many queries can be expressed as dynamic finders. A GORM application contains much less data-access code and has far fewer explicit dependencies on the Hibernate framework. In addition, because the finder meth-

ods are readily available on the domain classes, GORM avoids the problem of needing to resolve inter-component references.

One potential drawback of these finder methods is that the method name is the definition of the query. It is not always possible to define an intentional revealing name for a query that encapsulates the actual implementation. As a result, evolving business requirements can cause the names of finder methods to change, which increases the cost of maintaining the application.

For applications that need to execute more elaborate queries, GORM provides a couple of different options. An application can execute HQL queries directly. For example, an application can execute an HQL query to retrieve accounts with low balances, as shown in Figure 5b. This code snippet invokes the `findAll()` method, which GORM injects into each domain class. It takes an HQL query and a list of parameters as arguments.

One nice feature of this API is that it allows an application to execute an HQL query without explicitly invoking the Hibernate API. The application does not have to solve the problem of obtaining a reference to a DAO or other component. One drawback, however, is that knowledge of HQL is hardwired into the application.

The other option, which is especially useful when constructing queries dynamically, is to use GORM criteria queries, which wrap the Hibernate Criteria API described earlier. As with the other APIs, GORM dynamically injects a `createCriteria()` method into domain classes. This method allows an application to construct and execute a query without having an explicit dependency on the Hibernate API.

Figure 5c is the GORM criteria query version of the query that retrieves accounts with low balances. The `createCriteria()` method returns an object for building queries. The application executes the query by calling `list()`, which takes a Groovy closure as an argument and returns a list of matching objects. The closure argument contains method calls such as `lt()` that add restrictions to the query.

Applications can use these APIs to execute queries that are not supported by dynamic finders. One potential downside, which could be considered to be a weakness of GORM, is the potential lack of modularity and violation of the Separation of Concerns principle. There is a risk of scattering the data-access operations for a domain class throughout the application. Some data-access methods are defined by the domain class, but the rest are intermingled with the application's business logic, which could be considered to be a lack of modularity. Ideally, such data-access logic should be encapsulated within DAOs but, unfortunately, GORM does not explicitly support them.

Summary

GORM provides an innovative style of O/R mapping that simplifies application code. One of the key ways it does this is by leveraging the dynamic features of the Groovy language. GORM injects persistence-related methods into domain classes at runtime. It eliminates a significant amount of data-access methods and classes, while still decoupling the business logic from the ORM framework.

GORM's extensive use of CoC simplifies application code. Provided that GORM's defaults for table and column names match the schema, a class can be mapped to the database schema with little or no configuration. GORM also injects every domain class with primary-key and version-number fields, which further reduces the amount of coding required.

GORM has some limitations. It does not easily support multiple databases. Dynamic finder methods cannot have an intentional revealing name that encapsulates the query. GORM lacks support for DAO classes, even though complex applications might benefit from the improved modularity that they offer. Applications that work with a legacy schema will not be able to take advantage of CoC since they require explicit configuration of ORM.

Despite these limitations, developers of a wide range of applications will find GORM extremely useful. Developers can use GORM independently of Grails but it is targeted at Web application developers who can benefit from the rapid development capabilities of the Grails framework. In addition, GORM is best used when developing applications that access a single database or when using database middleware that makes multiple databases appear as a single database. Developers will get the most benefit from GORM when they have control over the database schema and can leverage GORM's CoC features.

Acknowledgments

I would like to thank the following reviewers for the helpful feedback on drafts of this article: Ajay Govindarajan, Azad Bolour, Dmitriy Volk, Brad Neighbors, and Scott Davis. I would also like to thank the members of the SF Bay Groovy and Grails meet-up and the anonymous *ACM Queue* reviewers who provided feedback on this article.

References

1. Apache OpenJPA; http://openjpa.apache.org/.
2. Bauer, C., and Gavin, K.. *Java Persistence with Hibernate.* Manning Publications, 2006.
3. Dijkstra. E.W. On the role of scientific thought. In *Selected Writings on Computing: A Personal Perspective.* Springer-Verlag, 1982, 60–66.
4. Domain-specific language; www.martinfowler.com/bliki/DomainSpecificLanguage.html.
5. Fowler, M. Inversion of control containers and the dependency injection pattern; www.martinfowler.com/articles/injection.html (2004).
6. Java SE Desktop Technology; http://java.sun.com/javase/technologies/desktop/javabeans/index.jsp.
7. Koenig, D., Glover, A., King, P., Laforge, G., and Skeet, J. *Groovy in Action.* Manning Publications, 2007.
8. Marinescu, F. *EJB Design Patterns: Advanced Patterns, Processes, and Idioms.* Manning Publications, 2007.
9. Mellqvist, P. Don't repeat the DAO! 2006; www.ibm.com/developerworks/java/library/j-genericdao.html.
10. NHibernate; http://sourceforge.net/projects/nhibernate/.
11. Oracle TopLink; www.oracle.com/technology/products/ias/toplink/index.html.
12. Rocher, G. *The Definitive Guide to Grails.* Apress, 2006.
13. The Spring Framework Reference Documentation. See @Configurable; http://static.springframework.org/spring/docs/2.5.x/reference/index.html.
14. Thomas, D., Fowler, C., and Hunt, A. *Programming Ruby: The Pragmatic Programmers' Guide.* Pragmatic Bookshelf, 2004.
15. Thomas, D., Hansson, D., Breedt, L., Clark, M., Fuchs, T., and Schwarz, A. *Agile Web Development with Rails.* Pragmatic Bookshelf, 2005.
16. Walls, C., Breidenbach, R. *Spring in Action, second edition.* Manning Publications, 2007.

Chris Richardson is a developer and architect with more than 20 years of experience. He is the author of *POJOs in Action* (Manning Publications, 2006), which describes how to build enterprise Java applications with POJOs and lightweight frameworks. He runs a consulting and training company that specializes in helping companies reduce development costs and increase developer effectiveness. He has been a technical leader at Insignia, BEA, and elsewhere. Richardson is the founder of Cloud Tools, which is a source project for deploying Java applications on Amazon EC2, and of Cloud Foundry, a startup that provides outsourced datacenter management for Java applications on the Cloud. He has a computer science degree from the University of Cambridge in England and lives in Oakland, CA; www.chrisrichardson.net.

A previous version of this article appeared in *ACM Queue*, May/June 2008.

DOI:10.1145/1498765.1498784

Comprehensive knowledge bases would tap the Web's deepest information sources and relationships to address questions beyond today's keyword-based search engines.

BY GERHARD WEIKUM, GJERGJI KASNECI, MAYA RAMANATH, AND FABIAN SUCHANEK

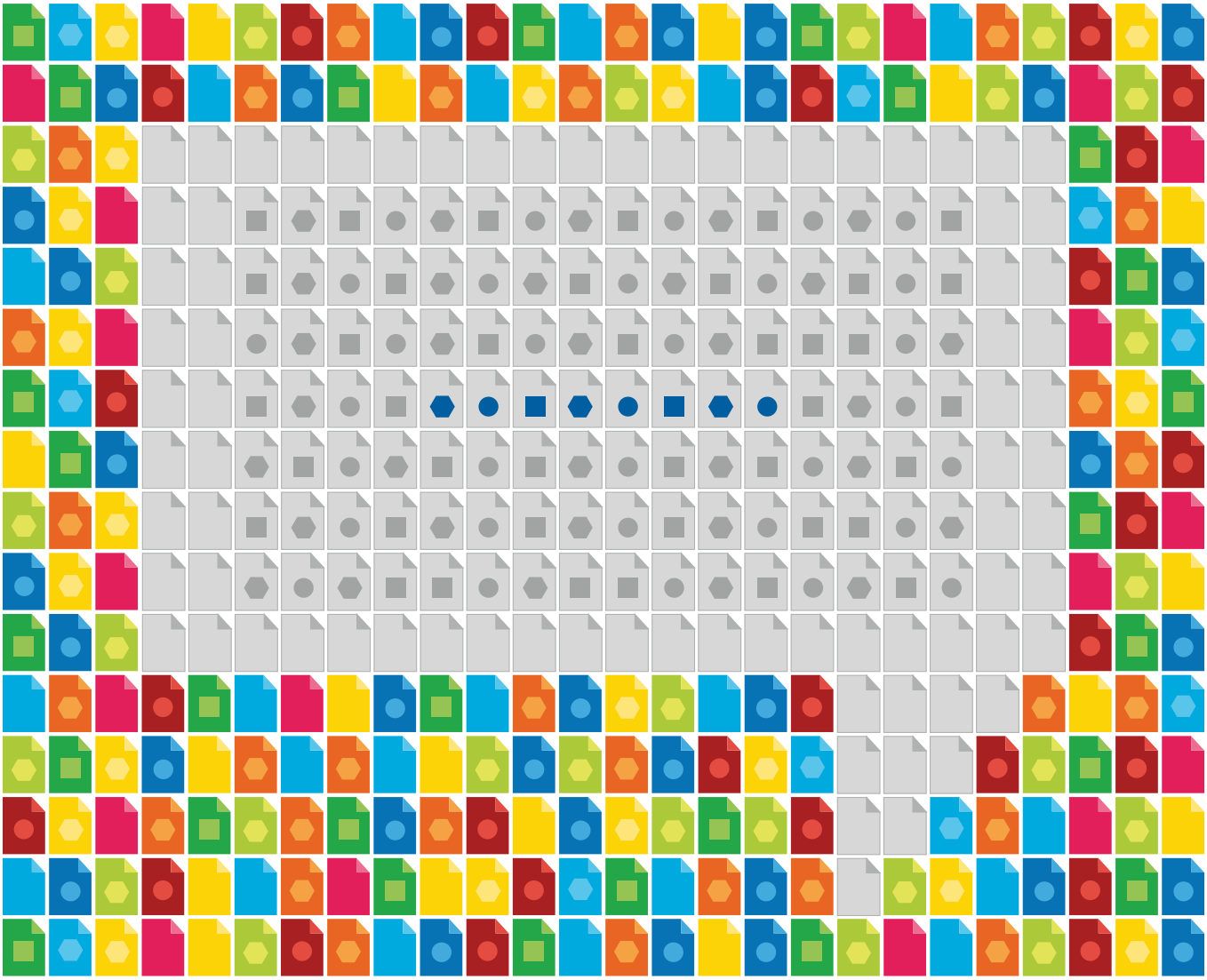
Database and Information-Retrieval Methods for Knowledge Discovery

OUR AIM HERE is to advocate for the integration of database systems (DB) and information-retrieval (IR) methods to address applications that are emerging from the ongoing explosion and diversification of digital information. One grand goal of such an endeavor is the automatic building and maintenance of a comprehensive knowledge base of facts from encyclopedic sources and the scientific literature. Facts should be represented in terms of typed entities and relationships and allow expressive queries that return ranked results with precision in an efficient and scalable manner. We thus explore how DB and IR methods might contribute toward this ambitious goal.

DB and IR are separate fields in computer science due to historical accident. Both investigate concepts, models, and computational methods for managing large amounts of complex information, though each began almost 40 years ago with very different application areas as motivations and technology drivers; for DB it was accounting systems (such as online reservations and banking), and for IR it was library systems (such as bibliographic catalogs and patent collections). Moreover, these two directions and their related research communities emphasized very different aspects of information management; for DB it was data consistency, precise query processing, and efficiency, and for IR it was text understanding, statistical ranking models, and user satisfaction.

There were attempts at integration (late 1990s), most notably the probabilistic datalog and probabilistic relational-algebra models,[13,14] the proximal node model,[19] and the WHIRL approach to similarity joins.[9] But it is only in the past few years that mission-critical applications have emerged with a compelling need for integrated DB and IR methods and platforms. From an IR perspective, digital libraries of all kinds are becoming rich information repositories, with documents augmented by metadata and annotations captured in semistructured data formats (such as XML); enterprise search on intranet data represents a variant of this theme.

From a DB point of view, application domains (such as customer support, product and market research, and health-care management) reflect data growth in terms of both structured and unstructured information. Web 2.0 applications (such as social networks) require support for structured and textual data, as well as ranking and recommendation in the presence



of uncertain information of highly diverse quality (see Figure 1). The Figure categorizes information systems along two dimensions: how the data is to be managed and how the data is to be searched. The first divides the world of digital data into structured data (such as like schema-oriented records with numerical, categorical, and short-string attributes) and unstructured data (such as natural-language text and multimodal information, including speech and video) and loose collections of heterogeneous records. The second dimension distinguishes sophisticated query languages that express logical conditions from simple keyword search as the prevalent way of posing queries to search engines. Since the late 1960s DB and IR systems have resided in two totally separate quadrants in the Figure, while it seemed as though the other two were useless or unoccupied.

Since the late 1990s, DB and IR researchers have explored these previously blank quadrants (middle of the Figure). IR-style keyword search over structured data (such as relational databases) makes sense when the structural data description—the schema—is so complex that information needs cannot be concisely or conveniently expressed in a structured query. As an example of this difficulty, consider a social-network database with tables of users, friends, and posted items (such as photos, videos, and recommended books or songs), as well as ratings and comments. Assume a user wants to find the connections shared by Alon, Raghu, and Surajit with respect to the Semantic Web. Answers might be that the three co-authored a book on the Semantic Web, two edited a book, one commented on it, or the three are friends and one posted a video called "Semantic Web Saga." With structured querying, where each value (such as "Alon") refers to a particular attribute (such as User.Name and Friend.Name), the combinatorial options lead to very complex queries with many joins and unions. Much simpler is to state five keywords—"Alon, Raghu, Surajit, Semantic, Web"—and let the system compute the most meaningful answers in a relational graph. This relaxed attitude toward the schema (which value should occur in which attribute) naturally entails IR-style ranking.

Conversely, linguistic and learning-based information-extraction techniques have been applied in order to augment textual sources with structured records and enable expressive DB-style querying over originally unstructured data. Consider an information request about "the life of the scientist Max Planck" to be evaluated over an XML-based digital library, perhaps an extended form of Wikipedia. A simple approach would be to formulate a keyword query like "life scientist

Max Planck." Unfortunately, the results would be dominated by information about the Max-Planck Institutes (approximately 80 in Germany) in the area of life sciences. Structured query languages (such as SQL and XPath Full-Text) allow professional users to specify more precisely what they are interested in, possibly in the form of attribute name-value conditions (such as Name = "Max Planck") and XML structure-and-content conditions (such as

//Article [Person ftcontains "Max Planck"]
[Category ftcontains "science"] //Biography).

These search predicates yield much more precise answers but may require approximate matching (to counter overspecified queries) and result ranking[2,25] and related references.

Meanwhile, the initially pure quadrants for DB and IR systems have been substantially enhanced by new methods for digital libraries, enterprise search and analytics, text extensions for database engines, and ranking capabilities for SQL and XQuery. The boundaries between quadrants are blurring; the DB and IR fields are increasingly fertilizing each other. The "Future" part of Figure 1 envisions convergence toward an integrated solution, though only the future alone can reveal if this goal is feasible and how it might be achieved.

Many applications must be able to manage both structured and unstructured data. Consider a health-care scenario involving, say, relational tables with the following schemas (attribute names and types, unique keys underlined):

Disease (DId int; Name char[50]; Category int; Pathogen char[50]; ...)
Patient (PId int; ... ; Age int; TreatedDId int; ResponsibleHId int; Timestamp date; Report longtext; ...)
Hospital (HId int; Address char[200]; ...)

Some of the information, especially foreign-key references between relations (such as a patient record referring to a disease identifier), is suitable for structured queries. But long text fields, often containing valuable latent information, are amenable to only keyword and text-similarity search. Moreover, some of the attributes (such as Category) may refer to external taxonomies and ontologies (such as the Unified Medical Language System).

To illustrate the nature of querying such data, consider an information request (such as "Find young patients in central Europe who have been reported, in the past two weeks, to have symptoms of tropical virus diseases and an indication of anomalies"). Computing relevant answers requires evaluating structured predicates (such as range conditions on Age and joins with additional ontology tables for identifying the values of Disease—Name, Category, Pathogen—referring to tropical virus diseases). This computation also involves fuzzy predicates on Report to test the "anomaly" condition. Moreover, hospital Address must be matched against geographic taxonomies with some inherent vagueness (such as about, say, whether England and Italy are part of central Europe). Finally, with such fuzzy matching and similarity tests it is absolutely necessary for the query engine to be able to provide meaningful rankings of the results. For example, a case with strong evidence of anomalies that may be at the border or just outside of central Europe or happened just outside the two-week time window should rank higher than a case that satisfies the structured (and spatio-temporal) conditions very well, though little evidence in the Report suggests it was particularly anomalous.

Structured and unstructured search conditions are combined in a single query, and the query results must be ranked. The queries must be evaluated over very large data sets that exhibit high update rates as new cases of diseases are added. A programmer can build such an application through two separate platforms—a DB system for the structured data and an IR search engine for the textual and fuzzy-matching issues. But this widely adopted approach is a challenge to application developers, as many tasks are not covered by the underlying platforms and must be addressed in the application code. An integrated DB/IR platform would greatly simplify development of the application and largely reduce the cost of maintaining and adapting it to future needs.

Figure 1: Past, present, and future of DB and IR methods.

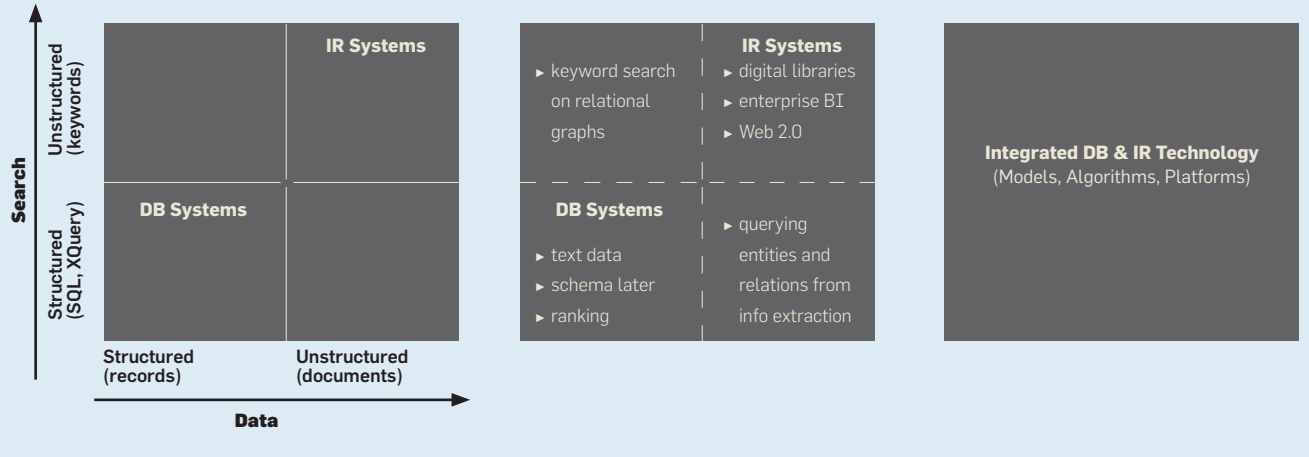
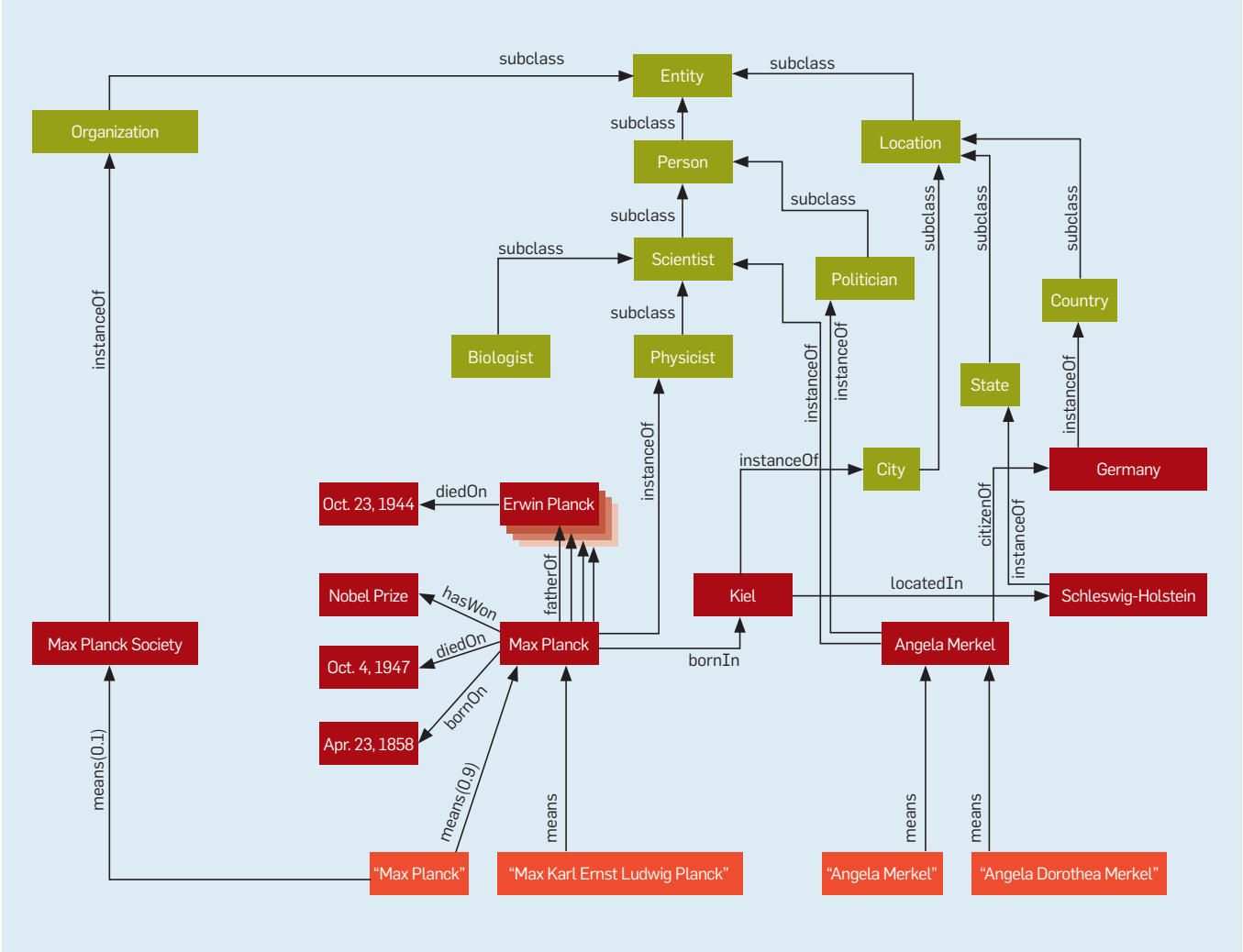


Figure 2: Excerpt from the YAGO knowledge base.



Abstracting from this application-centric discussion, we have identified several compelling motivations for bringing IR concepts to DB systems and vice versa, leading to the following DB and IR concepts and methods a developer would find useful:

Approximate matching and record linkage. Adding text-matching functionality to DB systems often entails approximate matching (such as due to spelling variants) and when text fields refer to named entities lead to record linkage for matching entities. For example, the strings "William J. Clinton" and "Bill Clinton" likely denote the same person, and the names "M-31" and "NGC 224" should be reconciled to denote the Andromeda galaxy. Approximate matching by similarity measures requires IR-style ranking.

Too-many-answers ranking. Preference search of, say, travel portals and product catalogs often poses a too-many-answers problem. Narrowing the query conditions may overshoot by producing too few or even no results; interactive reformulation and browsing is time-consuming and might irritate users. Large result sets inevitably require ranking based on data and/or workload statistics, as well as on user profiles;

Schema relaxation and heterogeneity. In the DB world, the norm is that applications access multiple databases, often with a run-time choice of the data sources. Even if each source contains structured data records and comes with an explicit schema, there is no unified global schema unless a breakthrough could be achieved to magically perform perfect on-the-fly data integration. So the application program must be able to cope with the heterogeneity of the underlying schema names, XML tags, and Resource Description Framework (RDF) properties, and queries must be schema-agnostic or at least tolerant to schema relaxation;

Information extraction and uncertain data. Textual information contains named entities and relationships in natural-language sentences that can be made explicit through information-extraction techniques (pattern matching, statistical learning, and natural-language processing). However, this approach can lead to large knowledge bases with facts that exhibit uncertainty; querying extracted facts thus entails ranking.

Entity search and ranking. Recognizing entities in text sources allows entity-search queries about, say, electronics products, travel destinations, and movie stars, boosting search capabilities on intranets, portals, news feeds, and the business- and entertainment-oriented parts of the Web. Extracting binary relations between entities, as well as place and time attributes,

could pave the way toward semantic IR on digital libraries (such as PubMed), news, and blogs and also aid natural-language question answering and searching the deep, or hidden, Web.

Harvesting, Searching, Ranking the Web

The Web has the potential for being the world's most comprehensive knowledge base, but we are still far from exploiting it. Valuable scientific and cultural content is all mixed up with huge amounts of noisy, low-quality, unstructured text and media. The challenge is how to extract the important facts from the Web and organize them into an explicit knowledge base that captures entities and semantic relationships among them. Imagine a formally structured Wikipedia with the same scale and richness as Wikipedia itself but that offers a precise and concise representation of knowledge that enables expressive and precise querying.

Figure 2 outlines what such a knowledge base might look like, depicting an excerpt from our own Yet Another Great Ontology (YAGO) knowledge base,[24] a typed entity-relationship graph that can be represented in the RDF or Owl-Lite data models. Building and maintaining it in a largely automated manner is not only difficult but an opportunity for computer science to contribute toward high-value assets for science, culture, and society. DB and IR methods could indeed have the potential to play major roles in this endeavor.

With a knowledge base that sublimates valuable content from the Web, we could address difficult questions beyond the capabilities of today's keyword-based search engines. For example, a user might ask for a list of drugs that inhibit proteases and obtain a fairly comprehensive list of drugs for this HIV-relevant family of enzymes. Such advanced information requests are posed by knowledge workers, including scientists, students, journalists, historians, and market researchers. Although it is possible to find relevant answers, the process is laborious and time-consuming, as it often requires rephrasing queries and browsing through many potentially promising but ultimately useless result pages. The following example questions illustrate this complexity:

Which German Nobel laureate survived both world wars and outlived all four of his children? The answer is Max Planck. The bits and pieces needed to answer are not difficult to locate: lists of Nobel prize winners, birth and death dates of the relevant people, the names of family members extracted from biographies, and dates associated with the various children. Gathering and connecting these facts is straightforward for a human but could take them days of manually inspecting Web pages.

Which politicians are also accomplished scientists? Today's search engines fail on such questions because they match words and return pages rather than identify entities (such as persons) and test their relationships. Moreover, the question entails a difficult ranking problem. Wikipedia alone contains hundreds of names listed in the categories "Politicians" and "Scientists." An insightful answer must rank important people first, say, the German chancellor Angela Merkel, who has a doctoral degree in physical chemistry, and Benjamin Franklin, who made scientific discoveries and was a founding father of the U.S.

How are Max Planck, Angela Merkel, Jim Gray, and the Dalai Lama related? All four have doctoral degrees from German universities (honorary doctorates for Gray and the Dalai Lama). Discovering interesting facts about multiple entities and their connections on the Web is virtually impossible due to the sheer amount of interconnected pages about these four famous people.

Note that even though the questions are asked in natural language, they would remain equally difficult to answer even if expressed in a formal language. Conversely, a rich knowledge base of entities and relationships would enable much more effective natural-language question answering.

Information organization and search on the Web are being augmented with increasingly sophisticated structure, context awareness, and semantic flavor in the form of faceted search, vertical-domain search, entity search, and deep-Web search. All major search engines recognize a large fraction of worldwide product names, have built-in knowledge about geographic locations, and return high-precision results for popular queries about consumer interests, travel, and entertainment. Information-extraction and entity-search methods are clearly at work. But these efforts focus only on specific domains. Generalizing the approach toward a universal methodology for knowledge harvesting requires bolder steps, and three major research avenues promise to contribute to this goal:

Semantic-Web-style knowledge repositories (such as ontologies and taxonomies). Included are general-purpose ontologies and thesauri (such as SUMO, OpenCyc, and WordNet), as well as domain-specific ontologies and terminological taxonomies (such as GeneOntology and UMLS in the biomedical domain);

Large-scale information extraction (IE) from text sources in the spirit of a Statistical Web. IE methods—entity recognition and learning relational patterns—are increasingly scalable and less dependent on human supervision[1, 10, 21]; and

Social tagging and Web 2.0 communities that constitute the social Web. Human contributions are abundant in the form of semantically annotated Web pages, phrases in pages, images, and videos, together providing "wisdom of the crowds." Freebase and other such endeavors collect structured data records from human communities. Wikipedia is another example of the Social Web paradigm, including semistructured data (such as infoboxes) that can be augmented with explicit facts.[4, 24, 27]

Research projects often combine elements of the semantic, statistical, and social approaches. Here, we discuss several interesting projects, highlighting YAGO results:

Libra. Aiming to support entity search on the Web, the Microsoft Research Lab in Beijing has developed comprehensive technology for information extraction, including pattern-matching algorithms tailored to typical Web-page layouts and trained learning of patterns using advanced models (such as hierarchical conditional random fields[28]). A particularly fruitful focus is to extract entities and their attributes from product-related pages with HTML tables and lists. These methods and tools are being used to build and maintain several

vertical-domain portals, including product search and the Libra portal for scholarly search on extracted records about authors, papers, conferences, and communities.

Once the facts are gathered and organized into searchable form, a typical IR issue arises concerning how a system should rank the results of an entity-centric query. To this end, an advanced statistical language model (LM) has been extended from the form of document-oriented bags of words to the form of structured records.[20]

Libra is an example of the Statistical-Web approach.

Cimple/DBLife. The Cimple project,[11,22] being carried out jointly by the University of Wisconsin and Yahoo! Research, is similar to Libra, aiming to generate and maintain community-specific portals with structured information gathered from Web sources. However, it applies a number of methods to achieve this goal, as we illustrate by discussing its flagship application: the DBLife portal.

DBLife features automatically compiled "super-homepages" of researchers with bibliographic data, as well as facts about community services (such as PC work), colloquium lectures, and more. For gathering and reconciling these facts, Cimple has a suite of DB-style extractors based on pattern matching and dictionary lookups. The extractors are combined into execution plans and periodically applied to a carefully selected set of relevant Web sources, including prominent sites like DBLP and the Dbworld archive and important conference and university pages that are selected semi-automatically. While the overall approach makes use of IR concepts like *tf*idf*-based ranking and Web-graph link analysis, Cimple emphasizes a more DB-oriented toolkit for declarative extraction programs, using Datalog as a query-language framework and DB rewriting techniques for query optimization.[17]

Cimple leans more toward the Semantic-Web approach and less toward a Statistical-Web approach. In addition, it contains Social-Web elements, most notably, a Wiki-based mechanism for users to provide feedback about incorrect facts they identify on community portals.

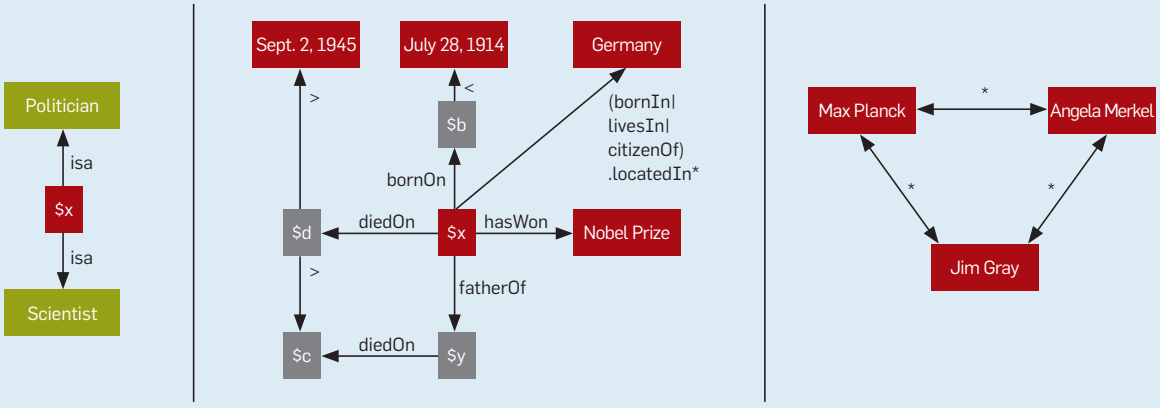
KnowItAll/TextRunner. Both Libra and Cimple operate on the basis of one page at a time, then aim to extract as many facts as possible from the given page. A dual view is to focus on one or more entity types or relationship types, aiming to populate them by inspecting many pages and exploiting their redundancies. For example, a user might want to find all cities on planet Earth, along with all scientists, guitar players, and other unary relations (entity types). For binary relations, a user might consider gathering all CEOs of all companies, all (city, river) pairs where a city is located on a river, or the answers to questions like: Who discovered what? and Which enzyme triggers which biochemical process?

The challenge is how to extract the important facts from the Web and organize them into an explicit knowledge base that captures entities and semantic relationships among them.

The KnowItAll project[5,6,12] at the University of Washington in Seattle has pursued this goal, using techniques that combine pattern matching, linguistic analysis, and statistical learning. KnowItAll starts with a set of seeds: the instances of the relation of interest (such as a set of cities or a set of (city, river) pairs).[12] This is the only "training input" needed by KnowItAll, which automatically finds sentences on the Web with the seeds, extracts linguistic patterns surrounding the seeds, performs statistical analyses to identify strong patterns, and finally identifies the most useful patterns to obtain extraction rules. For example, the phrase templates "*located in downtown $x*" and "*$x is located on the banks of $y*" may be determined to be good rules for extracting cities and (city, river) pairs, respectively. Now these rules can be applied to newly seen Web pages, yielding facts or fact candidates, some in turn considered as new, additional seeds. Needed are statistical inferences to identify good rules and assess the confidence in the harvested facts.

The TextRunner tool[5] pays special attention to scalability and simplifies the entire fact-gathering pipeline. It has a completely unsupervised bootstrapping phase for identifying simple patterns, just enough to identify, with high confidence, noun phrases and verbal patterns. When TextRunner sees a new Web page, it aggressively extracts all potentially meaningful instances of all possible binary relation types from the page text; Banko et al.[5] refers to this processing mode as open information extraction, or "machine reading."

Figure 3: Example queries for the YAGO knowledge base.



KnowItAll and TextRunner are examples of Statistical-Web methods for large-scale knowledge acquisition.

YAGO for Large-Scale Semantic Knowledge

Our YAGO project[23,24] shares the KnowItAll and TextRunner goal of large-scale knowledge harvesting but emphasizes high accuracy and consistency rather than high recall (coverage). YAGO is best characterized as a Semantic-Web approach, gathering its knowledge by (primarily) integrating information from Wikipedia and WordNet. It also employs text-mining-based techniques. YAGO contains close to two million entities and about 20 million facts about them, where facts are instances of binary relations. Extensive sampling has shown that YAGO accuracy is at least 95%, and many of its errors (false positives) are due to incorrect entries in Wikipedia itself. YAGO is publicly available at www.mpi-inf.mpg.de/yago/.

Two Wikipedia assets—infoboxes and the category system—are almost structured data. Infoboxes are collections of attribute name-value pairs often based on templates and reused for important types of entities (such as countries, companies, scientists, music bands, and sports teams). For example, the infobox for Max Planck delivers such data as *birth_date = April 23, 1858, birth_place = Kiel, death_date = October 4, 1947, nationality = Germany, and alma_mater = Ludwig-Maximilians-Universität München*. As for the category system, the Max Planck article is manually placed in such categories as *German_Nobel_laureates, Nobel_laureates_in_physics, quantum_physics*, and *University_of_Munich_alumni*. All give YAGO clues about instanceOf relations, so it can infer that the entity Max Planck is an instance of the classes GermanNobelLaureates, NobelLaureatesInPhysics, and UniversityOfMunichAlumni. But YAGO must be careful, as the placement in category *quantum_physics* does not mean that Max Planck is an instance of QuantumPhysics. The YAGO extractors employ linguistic processing (noun phrase parsing) and mapping rules to achieve high accuracy in harvesting the categories information.

These examples of YAGO information extraction indicate that relying solely on Wikipedia infoboxes and categories may result in a large but incoherent collection of facts. For example, we may know that Max Planck is an instance of GermanNobelLaureates but be unable to automatically infer that he is also an instance of Germans and of Nobel Laureates. Likewise, the fact that he was a physicist does not automatically tell us he was a scientist. To address these shortcomings, YAGO makes intensive use of the WordNet thesaurus (lightweight ontology), integrating the facts it harvests from Wikipedia with the taxonomic backbone provided by WordNet.

While WordNet knows many abstract classes and the "is-a" and "part-of" relationships among them, it has only sparse information about individual entities that would populate its classes. The wealth of entities in Wikipedia complements WordNet nicely; conversely, the rigor and extensive coverage of WordNet's taxonomy compensate for the gaps and noise in the Wikipedia category system. Each individual entity YAGO discovers must be mapped into at least one existing YAGO class. If this fails, the entity and its related facts are not admitted into the knowledge base. Analogously, classes derived from Wikipedia category names (such as GermanNobelLaureates) must be mapped with a subclass relationship to one or more superclasses (such as NobelLaureates and Germans). These procedures ensure that YAGO maintains a consistent knowledge base, where consistency eliminates dangling entities and classes and guarantees that the subclass relation is acyclic.

Kylin/KOG. The "Intelligence in Wikipedia" project also extracts information from Wikipedia through its tools Kylin[27] and Kylin Ontology Generator (KOG).[26] Whenever an infobox type includes an attribute in some articles but the attribute has no value for a given article, Kylin analyzes the full text of the article to derive the most likely value. Like KnowItAll and TextRunner (but unlike Libra, Cimple, and YAGO), Kylin pursues open extraction by considering all potentially significant attributes, even if they occur only sparsely in the entire Wikipedia corpus. KOG builds on Kylin's output, unifies attribute names, derives type signatures, and (like YAGO) maps these entities onto the WordNet taxonomy through statistical relational learning.[15] KOG goes beyond YAGO by discovering new relationship types. It builds on the class system of both YAGO and DBpedia,[4] along with the entities in

each class, to train its learning algorithms for generating the subsumption graph among classes.

The Kylin/KOG project combines all three knowledge-gathering paradigms: Semantic-Web-oriented by being targeted at infoboxes; Social-Web-based by leveraging the input of the large Wikipedia community; and Statistical-Web-style through learning methods.

Searching and Ranking YAGO with NAGA

The query language we designed for YAGO adopts concepts from the standardized SPARQL Protocol and RDF Query Language for RDF data but extends them through more expressive pattern matching and ranking.[3,18] The prototype system that implements these features is called NAGA (for Not Another Google Answer, www.mpi-inf.mpg.de/yago/). Viewing the knowledge base as a graph, users and programmers alike can construct a query with the help of subgraph templates; Figure 3 outlines three examples related to the question scenarios discussed earlier.

The leftmost query in the Figure about politicians who are also scientists shows two nodes matched by the desired results and one node (labeled *$x*) denoting a variable for which the query must find all bindings. The edge labels denote relationships and need to be matched by the results. Here, "isa" is shorthand notation for a composition of two connected edges that correspond to the relationships *instanceOf* between an entity and a class and *subclass* between two classes. This way the user also finds people who belong to the classes "mayor" and "physicist."

The query in the middle of Figure 3 (a simpler variant of the German-Nobel-laureate question) generalizes the point about labels referring to compositions of relations. The label (*bornIn*|*livesIn*|*citizenOf*).*locatedIn** is a regular expression that allows users to avoid overspecifying their information demand. We may be generous when we call a person German, and the locatedIn relationship often reflects geographical hierarchies (such as with cities, counties, states, and countries).

The rightmost query of Figure 3 is a broad relatedness query that looks for commonalities or other connections among several entities. Here again, users or programmers would use regular expressions as edge labels in the query's graph template.

NAGA queries often return too many results and so must rank these results. For example, a query like "*What is known about Einstein?*," which may be phrased as a single-edge graph pattern *isa(Einstein, $y)*, returns dozens if not hundreds of results, including many uninteresting ones like *isa(Einstein, Entity)*, *isa(Einstein, Organism)*, and *isa(Einstein, Colleague)*. Ranking models for such results is much more difficult than for traditional search engines, as the system must consider the graph structure in both queries and results. Three general criteria must be accommodated:

Informativeness. Users prefer informative answers—salient or interesting facts, as opposed to overly generic facts or facts that are trivially known already. In the example query about Einstein the user would prefer the answers *isa(Einstein, Physicist)* or *isa(Einstein, NobelLaureate)* over the near-trivial results or a nontrivial but still less important fact like *isa(Einstein, Vegetarian)*. However, when asking a different query about noteworthy vegetarians, the latter fact should be one of the highest-ranked results. Informativeness is a query-dependent (and potentially user-and-situation-dependent) notion;

Confidence. Users may occasionally find uncertain, dubious, or false statements in the YAGO knowledge base. Each fact is annotated with a confidence value derived from the fact's original data sources and the extraction methods YAGO used. The quality of the sources tapped by YAGO can be quantified in terms of authority and trust measures in the spirit of Google's PageRank, and the quality of different extractors can be empirically assessed. Various ways are available for combining these measures into a single confidence value,[7,8] and high-confidence answers are preferred; and

Compactness. Whenever a query returns paths or graphs rather than individual nodes, we are interested in compact graphs and short paths. For example, a query about how Einstein and Bohr are related should return a short answer path that says something like "both are physicists," rather than a convoluted answer like "Einstein was a vegetarian like Tom Cruise who was born in the same year Bohr died."

A good ranking function is needed to combine all three criteria. Here, we sketch our approach for informativeness. We developed for NAGA a new kind of statistical LM for graph-structured data and queries. The parameters of the model are estimated from corpus or workload statistics. Consider the simple queries *isa(Einstein; $y)* about Einstein and *bornIn($y; Frankfurt)* about people born in Frankfurt. For the Einstein query YAGO estimates conditional co-occurrence probabilities

$$\frac{P[\text{Einstein} \wedge \text{Physicist}]}{P[\text{Einstein}]}$$

and

$$\frac{P[\text{Einstein} \wedge \text{Vegetarian}]}{P[\text{Einstein}]}$$

to compare and rank two possible answers. For the Frankfurt query YAGO computes and compares

$$\frac{P[\text{Goethe} \wedge \text{born}\hat{}\text{Frankfurt}]}{P[\text{born} \wedge \text{Frankfurt}]}$$

against

$$\frac{P[\text{Weikum} \wedge \text{born}\hat{}\text{Frankfurt}]}{P[\text{born} \wedge \text{Frankfurt}]}$$

clearly favoring Goethe as a top result.

This LM-based ranking allows NAGA to rank politicians who are also scientists with high informativeness, with Benjamin Franklin, Angela Merkel, and other prominent figures showing up in the top ranks. So while the YAGO knowledge base is primarily a Semantic-Web endeavor, the ranking for its search engine is built on Statistical-Web assets.

Despite good progress, these approaches face three notable challenges:

Scalable harvesting. Most new knowledge is produced in textual form (such as in scientific publications). Methods for natural-language IE face inherent trade-offs regarding training effort vs. easy deployment and precision vs. recall of the results. Scaling up the IE machinery for higher throughput without sacrificing quality is a formidable problem. For example, can IE tools process all blog postings on the planet at the same rate they are produced, without missing relevant facts or producing too many false positives?;

Expressive ranking. The LM-based ranking models pursued by Libra and

NAGA should be extended to better capture the context of the user and the data. User context requires personalized and task-specific LMs that consider current location, time, short-term history, and intention in the user's digital traces. Data context calls for LMs for entity-relationship graphs, aiming to better model complex patterns beyond single facts (edges) and consider types; and

Efficient search. Evaluating complex query predicates over graphs is computationally difficult. Moreover, the need for ranking suggests that the system should avoid materializing overly large numbers of results and better aim for solely computing the top-k results in a more efficient way.[16]

On a grander scale is the question of which is the most appropriate paradigm. The three avenues toward comprehensive knowledge harvesting—Semantic, Statistical, and Social Web—are by no means mutually exclusive. The projects outlined here combine aspects of several of these directions. Deeper understanding of feedback between and synergies from the three paradigms is an overriding theme of great potential value to researchers. Semantic-Web sources can be powerful bootstrap tools for large-scale Statistical-Web mining. Statistical-Web tools may produce many false hypotheses, but they can be assessed by Social-Web platforms with large communities of users that engage in human-computing tasks. Social-Web endeavors in turn are often grassroots catalysts for developing high-value knowledge repositories that eventually become Semantic-Web assets; examples are Wikipedia and derived knowledge bases (such as YAGO and DBpedia).

Conclusion

We have presented motivations for and approaches toward integrating the historically separated DB and IR methodologies. While deep DB/IR integration may be wishful thinking, at least for the time being, we observe strong trends toward adopting IR concepts in the DB world and vice versa. In addition to applications that must be able to manage structured and unstructured data or highly heterogeneous information sources, we also see increasing interest and success in extracting entities and relationships from text sources. The envisioned path toward automatically building and growing comprehensive knowledge bases with expressive search and ranking capabilities may take a long time to mature. In any case, it is an exciting and rewarding challenge that should appeal to and benefit from innovation in several research communities, most notably DB and IR.

Acknowledgments

Our work on knowledge harvesting is supported by the Excellence Cluster "Multimodal Computing and Interaction" (www.mmci.uni-saarland.de) funded by the German Science Foundation. C

References

1. Agichtein, E. Scaling information extraction to large document collections. *IEEE Data Engineering Bulletin 28*, 4 (Dec. 2005), 3–10.
2. Amer-Yahia, S, and Lalmas, M. XML search: Languages, INEX, and scoring. *ACM SIGMOD Record 35*, 4 (Mar. 2006), 16-23.
3. Anyanwu, K., Maduko, A., and Sheth, A. SPARQ2L: Towards support for subgraph extraction queries in RDF databases. In *Proceedings of the 16th International Conference on World Wide Web* (Banff, Canada, May 8–12). ACM Press, New York, 2007, 797–806.
4. Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., and Ives, Z. DBpedia: A nucleus for a Web of open data., In *Proceedings of the Sixth International Semantic Web Conference* (Pusan, Korea, Nov. 11–15). Springer, Berlin/Heidelberg, 2007, 722–735.
5. Banko, M., Cafarella, M., Soderland, S., Broadhead, M., and Etzioni, O. Open information extraction from the Web. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence* (Hyderabad, India, Jan. 6–12, 2007), 2670–2676; www.ijcai.org.
6. Cafarella, M., Re, C., Suciu, D., and Etzioni, O. Structured querying of Web text data: A technical challenge. In *Proceedings of the Third Biennial Conference on Innovative Data Systems Research* (Asilomar, CA, Jan. 7–10, 2007), 225–234; www.crdrdb.org.
7. Chakrabarti, S. Dynamic personalized PageRank in entity-relation graphs. In *Proceedings of the 16th International Conference on World Wide Web* (Banff, Canada, May 8–12). ACM Press, New York, 2007, 571–580.
8. Cheng, T., Yan, X., and Chang, K. Entity Rank, Searching entities directly and holistically. In *Proceedings of the 33rd International Conference on Very Large Data* Bases (Vienna, Austria, Sept. 23–27). ACM Press, New York, 2007, 387–398.
9. Cohen, W. Integration of heterogeneous databases without common domains using queries based on textual similarity. In *Proceedings of the ACM SIGMOD International Conference on Management of Data* (Seattle, June 2–4). ACM Press, New York, 1998, 201–212.
10. Cunningham, H. An introduction to information extraction. In *Encyclopedia of Language and Linguistics, Second Edition*, K. Brown et al., Eds., Elsevier, Amsterdam, 2005.
11. DeRose, P., Shen, W., Chen, F., Doan, A.-H., and Ramakrishnan, R. Building structured Web community portals: A top-down, compositional, and incremental approach. In *Proceedings of the 33rd International Conference on Very Large Data Bases* (Vienna, Austria, Sept. 23–27). ACM Press, New York, 2007, 399–410.
12. Etzioni, O., Cafarella, M., Downey, D., Popescu, A.-M., Shaked, T., Soderland, S., Weld, D., and Yates, A. Unsupervised named-entity extraction from the Web: An experimental study. *Artificial Intelligence 165*, 1 (June 2005), 91–134.
13. Fuhr, N. and Rölleke, T. A probabilistic relational algebra for the integration of information retrieval and database systems. *ACM Transactions on Information Systems 15*, 1 (Jan. 1997), 32–66.
14. Fuhr, N. Probabilistic datalog: A logic for powerful retrieval methods. In *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (Seattle, July 9–13). ACM Press, New York 1995, 282–290.
15. Getoor, L. and Taskar, B., Eds. *Introduction to Statistical Relational Learning.* MIT Press, Cambridge, MA, 2007.
16. Ilyas, I., Beskales, G., and Soliman, M. A survey of top-k query-processing techniques in relational database systems. *ACM Computing Surveys 40*, 1 (Oct. 2008), 1–58.
17. Ipeirotis, P., Agichtein, E., Jain, P., and Gravano, L. Towards a query optimizer for text-centric tasks. *ACM Transactions on Database Systems 32*, 4 (Nov. 2007).
18. Kasneci, G., Suchanek, F., Ifrim, G., Ramanath, M., and Weikum, G. NAGA: Searching and ranking knowledge. In *Proceedings of the 24th International Conference on Data Engineering* (Cancun, Mexico, Apr. 7–12). IEEE Computer Society, Washington, D.C., 2008, 953–62.
19. Navarro, G. and Baeza-Yates, R. Proximal nodes: A model to query document databases by content and structure. *ACM Transactions on Information Systems 15*, 4 (1997), 400–435.
20. Nie, Z., Ma, Y., Shi, S., Wen, J.-R., and Ma, W.-Y. Web object retrieval. In *Proceedings of the 16th International Conference on World Wide Web* (Banff, Canada, May 8–12). ACM Press, New York, 2007, 81–90.
21. Sarawagi, S. Information extraction. *Foundations and Trends in Databases 1*, 3 (2008), 261–377.
22. Shen, W., Doan, A.H., Naughton, J., and Ramakrishnan, R. Declarative information extraction using datalog with embedded extraction predicates. In *Proceedings of the 33rd International Conference on Very Large Databases* (Vienna, Austria, Sept. 23–27). ACM Press, New York, 2007, 1033–1044.
23. Suchanek, F., Kasneci, G., and Weikum, G. YAGO: A large ontology from Wikipedia and WordNet. *Journal of Web Semantics 6*, 3 (2008), 203–217.
24. Suchanek, F., Kasneci, G., and Weikum, G. YAGO: A core of semantic knowledge. In *Proceedings of the 16th International Conference on World Wide Web* (Banff, Canada, May 8–12). ACM Press, New York, 2007, 697–706.
25. Theobald, M., Bast, H., Majumdar, D., Schenkel, R., and Weikum, G. TopX: Efficient and versatile top-k query processing for semistructured data. *VLDB Journal 17*, 1 (Jan. 2008), 81–115.
26. Wu, F. and Weld, D. Automatically refining the Wikipedia infobox ontology. In *Proceedings of the 17th International Conference on World Wide Web* (Beijing, Apr. 21–25). ACM Press, New York, 2008, 635–644.
27. Wu, F. and Weld, D. Autonomously semantifying Wikipedia. In *Proceedings of the 16th ACM Conference on Information and Knowledge Management* (Lisbon, Nov. 6–10). ACM Press, New York, 2007, 41–50.
28. Zhu, J., Nie, Z., Wen, J.-R., Zhang, Bo, and Ma, W.-Y. Simultaneous record detection and attribute labeling in Web data extraction. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Philadelphia, PA, Aug. 20–23). ACM Press, New York, 2006, 494–503.

Gerhard Weikum (weikum@mpi-inf.mpg.de) is a scientific director leading the research group on databases and information systems at the Max Planck Institute for Informatics, Saarbruecken, Germany.

Gjergji Kasneci (kasneci@mpi-inf.mpg.de) is a doctoral student at the Max Planck Institute for Informatics, Saarbruecken, Germany.

Maya Ramanath (ramanath@mpi-sb.mpg.de) is a researcher at the Max Planck Institute for Informatics, Saarbruecken, Germany.

Fabian Suchanek (suchanek@mpi-inf.mpg.de) is a researcher at the Max Planck Institute for Informatics, Saarbruecken, Germany.

DOI:10.1145/1498765.1498785

The Roofline model offers insight on how to improve the performance of software and hardware.

BY SAMUEL WILLIAMS, ANDREW WATERMAN, AND DAVID PATTERSON

Roofline: An Insightful Visual Performance Model for Multicore Architectures

CONVENTIONAL WISDOM IN computer architecture produced similar designs. Nearly every desktop and server computer uses caches, pipelining, superscalar instruction issue, and out-of-order execution. Although the instruction sets varied, the microprocessors were all from the same school of design. The relatively recent switch to multicore means that microprocessors will become more diverse, since no conventional wisdom has yet emerged concerning their design. For example, some offer many simple processors vs. fewer complex processors, some depend on multithreading, and some even replace caches with explicitly addressed local stores. Manufacturers will likely offer multiple products with differing numbers of cores to cover multiple price-performance points, since Moore's Law will permit the doubling of the number of cores per chip every two years.[4] While diversity may be understandable in this time of uncertainty, it exacerbates the

already difficult jobs of programmers, compiler writers, and even architects. Hence, an easy-to-understand model that offers performance guidelines would be especially valuable.

Such a model need not be perfect, just insightful. The 3Cs (compulsory, capacity, and conflict misses) model for caches is an analogy.[19] It is not perfect, as it ignores potentially important factors like block size, block-allocation policy, and block-replacement policy. It also has quirks; for example, a miss might be labeled "capacity" in one design and "conflict" in another cache of the same size. Yet the 3Cs model has been popular for nearly 20 years precisely because it offers insight into the behavior of programs, helping programmers, compiler writers, and architects improve their respective designs.

Here, we propose one such model we call Roofline, demonstrating it on four diverse multicore computers using four key floating-point kernels.

Performance Models

Stochastic analytical models[4,24] and statistical performance models[7,25] can accurately predict program performance on multiprocessors but rarely provide insight into how to improve the performance of programs, compilers, and computers[1] and can be difficult to use by nonexperts.[25]

An alternative, simpler approach is "bound and bottleneck analysis." Rather than try to predict performance, it provides "valuable insight into the primary factors affecting the performance of computer systems. In particular, the critical influence of the system bottleneck is highlighted and quantified."[20]

The best-known example of a performance bound is surely Amdahl's Law,[3] which says the performance gain of a parallel computer is limited by the serial portion of a parallel program and was recently applied to heterogeneous multicore computers.[4,18]

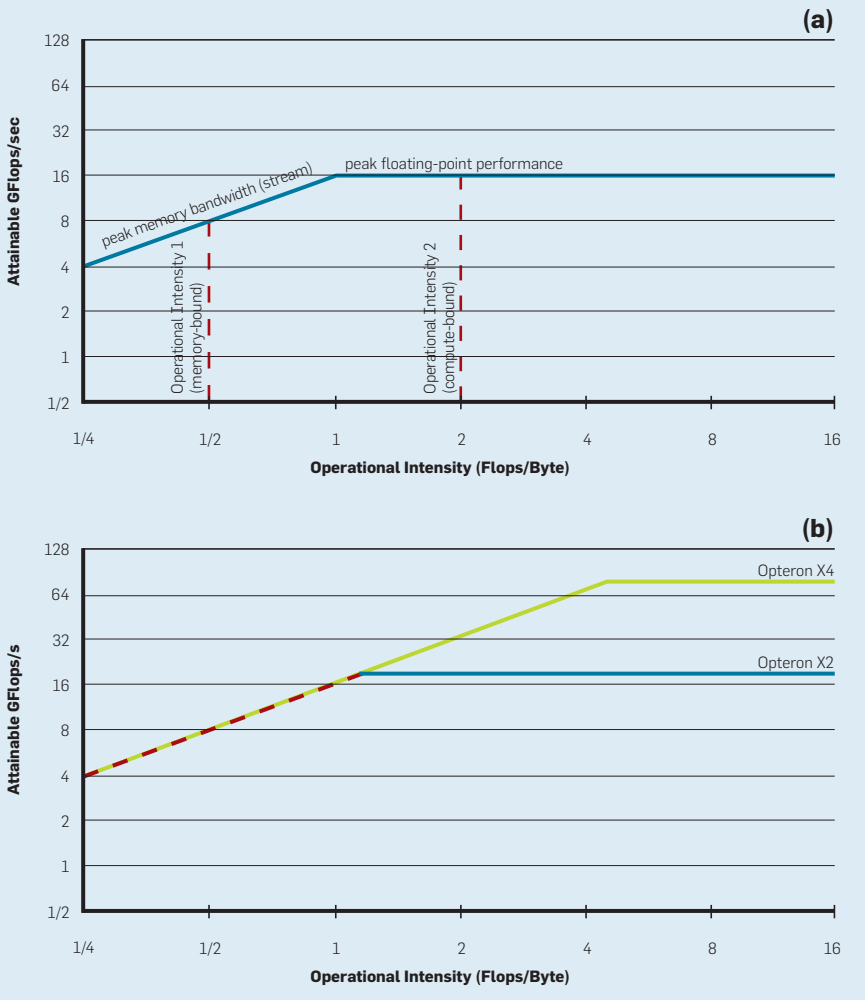
Roofline Model

For the foreseeable future, off-chip memory bandwidth will often be the constraining resource in system performance.[23] Hence, we want a model that relates processor performance to off-chip memory traffic. Toward this goal, we use the term "operational intensity" to mean operations per byte of DRAM traffic, defining total bytes accessed as those bytes that go to the main memory after they have been filtered by the cache hierarchy. That is, we measure traffic between the caches and memory rather than between the processor and the caches. Thus, operational intensity predicts the DRAM bandwidth needed by a kernel on a particular computer.

We say "operational intensity" instead of, say, "arithmetic intensity"[16] or "machine balance"[8,9] for two reasons: First, arithmetic intensity and machine balance measure traffic between the processor and the cache, whereas efficiency-level programmers want to measure traffic between the caches and DRAM. This subtle change allows them to include memory optimizations of a computer into our bound-and-bottleneck model. Second, we think the model will work with kernels where the operations are not arithmetic, as discussed later, so we needed a more general term than "arithmetic."

The proposed Roofline model ties together floating-point performance, operational intensity, and memory performance in a 2D graph. Peak floating-point performance can be found through hardware specifications or microbenchmarks. The working sets of the kernels we consider here do not fit fully in on-chip caches, so peak memory performance is defined by the memory system behind the caches. Although one can find memory performance through the STREAM benchmark,[22] for this work we wrote a series of progressively optimized microbenchmarks designed to determine sustainable DRAM bandwidth. They include all techniques to get the best memory performance, including

Figure 1: Roofline model for (a) AMD Opteron X2 and (b) Opteron X2 vs. Opteron X4.



prefetching and data alignment. (See Section A.1 in the online Appendix[a] for more detail of how to measure processor and memory performance and operational intensity.)

Figure 1a outlines the model for a 2.2GHz AMD Opteron X2 model 2214 in a dual-socket system. The graph is on a log-log scale. The *y*-axis is attainable floating-point performance. The *x*-axis is operational intensity, varying from 0.25 Flops/DRAM byte-accessed to 16 Flops/DRAM byte-accessed. The system being modeled has peak double precision floating-point performance of 17.6 GFlops/sec and peak memory bandwidth of 15GB/sec from our benchmark. This latter measure is the steady-state bandwidth potential of the memory in a computer, not the pin bandwidth of the DRAM chips.

One can plot a horizontal line showing peak floating-point performance of the computer. The actual floating-point performance of a floating-point kernel can be no higher than the horizontal line, since this line is the hardware limit.

How might we plot peak memory performance? Since the *x*-axis is Flops per Byte and the *y*-axis is GFlops/sec, gigabytes per second (GB/sec)—or (GFlops/sec)/(Flops/Byte)—is just a line of unit slope in Figure 1. Hence, we can plot a second line that bounds the maximum floating-point performance that the memory system of the computer can support for a given operational intensity. This formula drives the two performance limits in the graph in Figure 1a:

$$\text{Attainable GFlops/sec} = min \left\{ \begin{array}{l} \text{Peak Floating-Point Performance} \\ \text{Peak Memory Bandwidth} \times \text{Operational Intensity} \end{array} \right.$$

The two lines intersect at the point of peak computational performance and peak memory bandwidth. Note that these limits are created once per multicore computer, not once per kernel.

For a given kernel, we can find a point on the *x*-axis based on its operational intensity. If we draw a vertical line (the pink dashed line in the figures) through that point, the performance of the kernel on that computer must lie somewhere along that line.

The Roofline sets an upper bound on performance of a kernel depending on the kernel's operational intensity. If we think of operational intensity as a column that hits the roof, either it hits the flat part of the roof, meaning performance is compute-bound, or performance is ultimately memory-bound.

The horizontal and diagonal lines give this bound model its name. The Roofline sets an upper bound on performance of a kernel depending on the kernel's operational intensity. If we think of operational intensity as a column that hits the roof, either it hits the flat part of the roof, meaning performance is compute-bound, or it hits the slanted part of the roof, meaning performance is ultimately memory-bound. In Figure 1a, a kernel with operational intensity 2.0 Flops/Byte is compute-bound and a kernel with operational intensity 1.0 Flops/Byte is memory-bound. Given a Roofline, you can use it repeatedly on different kernels, since the Roofline doesn't vary.

Note that the ridge point (where the diagonal and horizontal roofs meet) offers insight into the computer's overall performance. The *x*-coordinate of the ridge point is the minimum operational intensity required to achieve maximum performance. If the ridge point is far to the right, then only kernels with very high operational intensity can achieve the maximum performance of that computer. If it is far to the left, then almost any kernel can potentially hit maximum performance. As we explain later, the ridge point suggests the level of difficulty for programmers and compiler writers to achieve peak performance.

To illustrate, we compare the Opteron X2 with two cores in Figure 1a to its successor, the Opteron X4 with four cores. To simplify board design, they share the same socket. Hence, they have the same DRAM channels and can thus have the same peak memory bandwidth, although prefetching is better in the X4. In addition to doubling the number of cores, the X4 also has twice the peak floating-point performance per core; X4 cores can issue two floating-point SSE2 instructions per clock cycle, whereas X2 cores can issue two instructions every other clock. As the clock rate is slightly faster—2.2GHz for X2 vs. 2.3GHz for X4—the X4 is able to achieve slightly more than four times the peak floating-point performance of the X2 with the same memory bandwidth.

Figure 1b compares the Roofline models for these two systems. As expected, the ridge point shifts right

a Please go to doi.acm.org/10.1145/1498765.1498785#supp

from 1.0 Flops/Byte in the Opteron X2 to 4.4 in the Opteron X4. Hence, to realize a performance gain using the X4, kernels need an operational intensity greater than 1.0 Flops/Byte.

Adding Ceilings to the Model

The Roofline model provides an upper bound to performance. Suppose a program performs far below its Roofline. What optimizations should one implement and in what order? Another advantage of bound-and-bottleneck analysis is that "a number of alternatives can be treated together, with a single bounding analysis providing useful information about them all."[20]

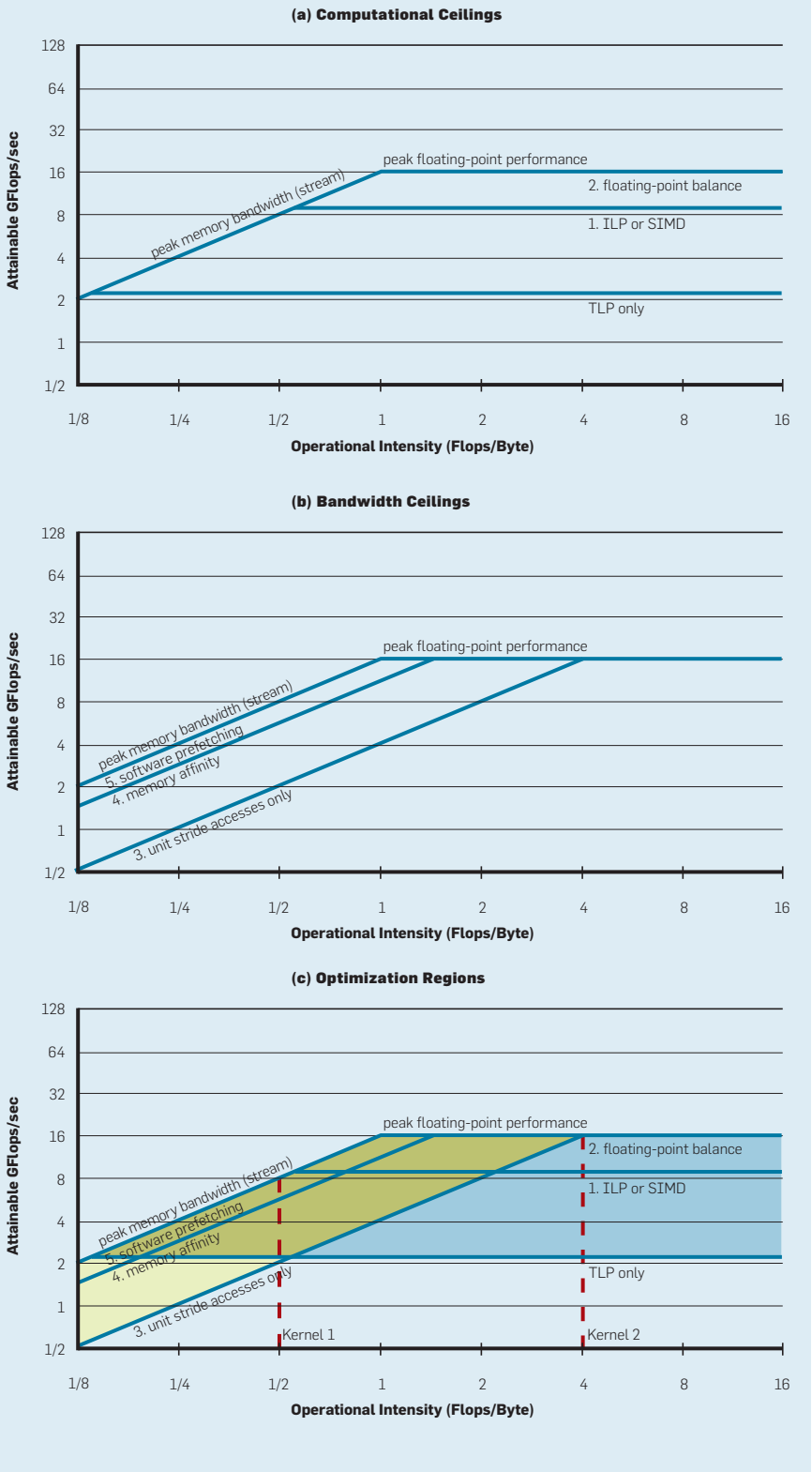
We leverage this insight to add multiple ceilings to the Roofline model to guide which optimizations to implement. It is similar to the guidelines loop balance gives the compiler. We can think of each optimization as a "performance ceiling" below the appropriate Roofline, meaning you cannot break through a ceiling without first performing the associated optimization.

For example, to reduce computational bottlenecks on the Opteron X2, almost any kernel can be helped with two optimizations:

Improve instruction-level parallelism (ILP) and apply SIMD. For superscalar architectures, the highest performance comes when fetching, executing, and committing the maximum number of instructions per clock cycle. The goal is to improve the code from the compiler to increase ILP. The highest performance comes from completely covering the functional unit latency. One way to hide instruction latency is by unrolling loops. For x86-based architectures, another way is using floating-point SIMD instructions whenever possible, since a SIMD instruction operates on pairs of adjacent operands; and

Balance floating-point operation mix. The best performance requires that a significant fraction of the instruction mix be floating-point operations (discussed later). Peak floating-point performance typically also requires an equal number of simultaneous floating-point additions and multiplications, since many computers have multiply-add instructions or an equal number of adders and multipliers.

Figure 2: Roofline model with ceilings for Opteron X2.



Memory bottlenecks can be reduced with the help of three optimizations:

Restructure loops for unit stride accesses. Optimizing for unit-stride memory accesses engages hardware prefetching, significantly increasing memory bandwidth;

Ensure memory affinity. Most microprocessors today include a memory controller on the same chip with the

Table 1: Characteristics of four recent multicore computers.

MPU Type	Intel Xeon (Clovertown, e5345)	AMD Opteron X4 (Barcelona, 2356)	Sun UltraSPARC T2+ (Niagara 2, 5120)	IBM Cell (QS20)
ISA	x86/64	x86/64	SPARC	Cell SPEs
Total Threads	8	8	128	16
Total Cores	8	8	16	16
Total Sockets	2	2	2	2
GHz	2.33	2.30	1.17	3.20
Peak GFlops/sec	75	74	19	29
Peak DRAM GB/sec	21.3r, 10.6w	2 × 10.6	2 × 21.3r, 2 × 10.6w	2 × 25.6
Stream GB/sec	5.9	16.6	26.0	47.0
DRAM Type	FBDIMM	DDR2	FBDIMM	XDR

processors. If the system has two multicore chips, then some addresses go to the DRAM local to one multicore chip, and the rest go over a chip interconnect to access the DRAM local to another chip. The latter lowers performance. This optimization allocates data and the threads tasked to that data to the same memory-processor pair, so the processors rarely have to access the memory attached to other chips; and

Use software prefetching. The highest performance usually requires keeping many memory operations in flight, which is easier to do via prefetching than by waiting until the data is actually requested by the program. On some computers, software prefetching delivers more bandwidth than hardware prefetching alone.

Like the computational Roofline, computational ceilings can come from an optimization manual,[2] though it's easy to imagine collecting the necessary parameters from simple microbenchmarks. The memory ceilings require running experiments on each computer to determine the gap between them (see online Appendix A.1). The good news is that like the Roofline, the ceilings must be measured only once per multicore computer.

Figure 2 adds ceilings to the Roofline model in Figure 1a; Figure 2a shows the computational ceilings and Figure 2b the memory bandwidth ceilings. Although the higher ceilings are not labeled with lower optimizations, these lower optimizations are implied; to break through a ceiling, the programmer must have already broken through all the ones below. Figure 2a shows the computational "ceilings" of 8.8 GFlops/sec if the floating-point operation mix is imbalanced and 2.2 GFlops/sec if the optimizations to increase ILP or SIMD are also missing. Figure 2b shows the memory bandwidth ceilings of 11 GB/sec without software prefetching, 4.8 GB/sec without memory affinity optimizations, and 2.7 GB/sec with only unit stride optimizations.

Figure 2c combines Figures 2a and 2b into a single graph. The operational intensity of a kernel determines the optimization region, and thus which optimizations to try. The middle of Figure 2c shows that computational optimizations and memory bandwidth optimizations overlap; we picked the colors to highlight this overlap. For example, Kernel 2 falls in the blue trapezoid on the right, suggesting the programmer should work only on the computational optimizations. If a kernel fell in the yellow triangle on the lower left, the model would suggest trying just memory optimizations. Kernel 1 falls in the green (= yellow + blue) parallelogram in the middle, suggesting the programmer try both types of optimization. Note that the Kernel 1 vertical line falls below the floating-point imbalance optimization, so optimization 2 may be skipped.

The ceilings of the Roofline model suggest which optimizations the programmer should perform. The height of the gap between a ceiling and the next higher ceiling is the potential reward for trying this optimization. Thus, Figure 2 suggests that optimization 1, which improves ILP/SIMD, has a large potential benefit for optimizing computation on that computer, and optimization 4, which improves memory affinity, has a large potential benefit for improving memory bandwidth on that computer.

The order of the ceilings suggests the optimization order, so we rank the ceilings from bottom to top; those most likely to be realized by a compiler or with little effort by a programmer are at the bottom and those that are difficult for a programmer to implement or inherently lacking in a kernel are at the top. The one quirky ceiling is floating-point balance, since the actual mix depends on the kernel. For most kernels, achieving parity between multiplies and additions is difficult, but for a few kernels, parity is natural. One example is sparse matrix-vector multiplication; for this domain, we would place floating-point mix as the lowest ceiling, since it is inherent. Like the 3Cs model, as long as the Roofline model delivers on insight, it need not be perfect.

Tying the 3Cs to Operational Intensity

Operational intensity tells programmers which ceilings need the most attention. Thus far, we have assumed

that the operational intensity is fixed, though this is not always the case; for example, for some kernels, the operational intensity increases with problem size (such as for Dense Matrix and FFT problems).

Caches filter the number of accesses that go to memory, so optimizations that improve cache performance increase operational intensity. Thus, we may couple the 3Cs model to the Roofline model. Compulsory misses set the minimum memory traffic and hence the highest possible operational intensity. Memory traffic from conflict and capacity misses can considerably lower the operational intensity of a kernel, so we should try to eliminate such misses.

For example, we can reduce traffic from conflict misses by padding arrays to change cache line addressing. A second example is that some computers have a non-allocating store instruction, so stores go directly to memory and do not affect caches. This approach prevents loading a cache block with data to be overwritten, thereby reducing memory traffic. It also prevents displacing useful items in the cache with data that will not be read, thereby saving conflict misses.

This shift of operational intensity to the right could put a kernel in a different optimization region. Generally, we advise improving operational intensity of the kernel before implementing other optimizations.

Demonstrating the Model

To demonstrate the Roofline model's utility, we now construct Roofline models for four recent multicore computers and then optimize four floating-point kernels. We'll then show that the ceilings and rooflines bound the observed performance for all computers and kernels.

Four diverse multicore computers. Given the lack of conventional wisdom concerning multicore architecture, it's not surprising that there are as many different designs as there are chips. Table 1 lists the key characteristics of the four multicore computers, all dual-socket systems, that we discuss here.

The Intel Xeon uses relatively sophisticated processors, capable of executing two SIMD instructions per clock cycle that can each perform two double-precision floating-point operations. It is the only one of the four machines with a front-side bus connecting to a common north bridge chip and memory controller. The other three have the memory controller on chip.

The Opteron X4 also uses sophisticated cores with high peak floating-point performance but is the only computer of the four with on-chip L3 caches. The two sockets communicate over separate, dedicated hypertransport links, making it possible to build a "glueless" multi-chip system.

The Sun UltraSPARC T2+ uses relatively simple processors at a modest clock rate compared to the other three, allowing it to have twice as many cores per chip. It is also highly multithreaded, with eight hardware-supported threads per core. It has the highest memory bandwidth of the four, as each chip has two dual-channel memory controllers that can drive four sets of DDR2/FBDIMMs.

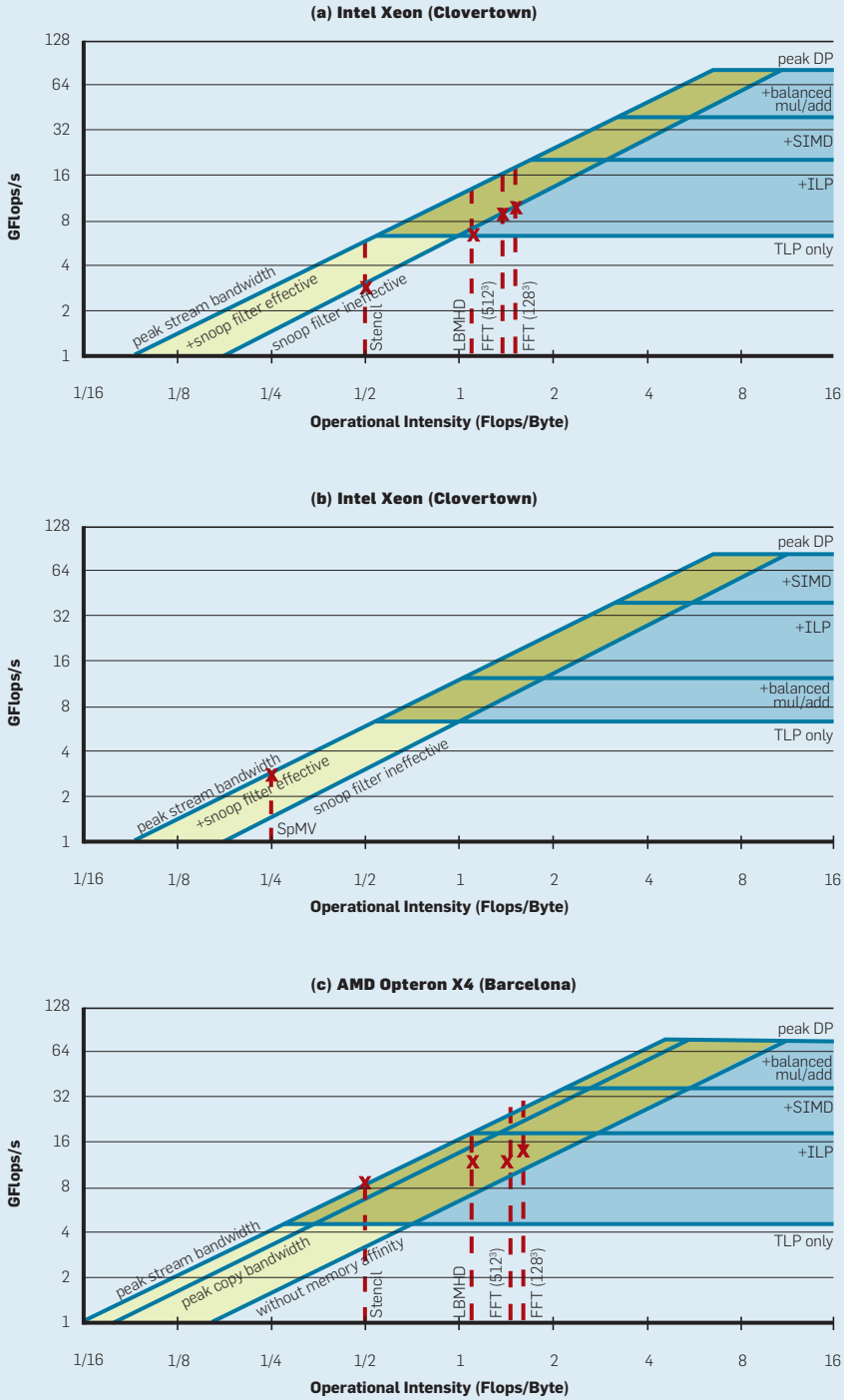
The clock rate of the IBM Cell QS20 is the highest of the four multicores at 3.2GHz. It is also the most unusual of the four, with a heterogeneous design, a relatively simple PowerPC core, and eight synergistic processing elements (SPEs) with their own unique SIMD-style instruction set. Each SPE also has its own local memory, instead of a cache. An SPE must transfer data from main memory into the local memory to operate on it and then back to main memory when the computation is completed. It uses Direct Memory Access, which has some similarity to software prefetching. The lack of caches means porting programs to Cell is more challenging.

Four diverse floating-point kernels. Rather than pick programs from a standard parallel benchmark suite (such as Parsec[5] and Splash-2[30]), we were inspired by the work of Phil Colella,[11] an expert in scientific computing at Lawrence Berkeley National Laboratory, who identified seven numerical methods he believes will be important for computational science and engineering for at least the next decade. Because he identified seven, they are called the Seven Dwarfs and are specified at a high level of abstraction to allow reasoning about their behavior across a broad range of implementations. The widely read "Berkeley View" report[4] found that if the data types were changed from floating point to integer, the same Seven Dwarfs would also be found in many other programs. Note that the claim is not that the Dwarfs are easy to parallelize but that they will be important to computing in most current and future applications; designers are thus advised to make sure they run well on the systems they create, whether or

Table 2: Characteristics of four floating-point kernels.

Name	Operational Intensity	Description
SpMV[29]	0.17 to 0.25	Sparse Matrix-Vector multiply: y = A*x where A is a sparse matrix and x, y are dense vectors; multiplies and adds equal.
LBMHD[28]	0.70 to 1.07	Lattice-Boltzmann Magnetohydro-dynamics is a structured grid code with a series of time steps.
Stencil[12]	0.33 to 0.50	A multigrid kernel that updates seven nearby points in a 3D stencil for a 256^3 problem.
3D FFT	1.09 to 1.64	3D Fast Fourier Transform (2 sizes: 128^3 and 512^3).

Figure 3a–3c: Roofline model for Intel Xeon, AMD Opteron X4, and IBM Cell.



not those systems are parallel.

One advantage of using these higher-level descriptions of programs is that we are not tied to code that might have been originally written to optimize an old computer to evaluate future systems. Another advantage of the restricted number is that efficiency-level programmers can create autotuners for each kernel that would search the alternatives to produce the best code for that multicore computer, including extensive cache optimizations.[13]

Table 2 lists the four kernels from among the Seven Dwarfs we use to demonstrate the Roofline model on the four multicore computers listed in Table 1; the autotuners discussed in this section are from three sources:[12, 28, 29]

For these kernels, there is sufficient parallelism to utilize all the cores and threads and keep them load balanced; see online Appendix A.2 for how to handle cases when load is not balanced.

Roofline models and results. Figure 3 shows the Roofline models for Xeon, X4, and Cell. The pink vertical dashed lines indicate the operational intensity and the red X marks performance achieved for that particular kernel. However, achieving balance is difficult for the others. Hence, each computer in Figure 3 has two graphs: the left one has multiply-add balance as the top ceiling and is used for Lattice-Boltzmann Magnetohydrodynamics (LBMHD), Stencil, and 3D FFT; the right one has multiply-add as the bottom ceiling and is used for SpMV. Since the T2+ lacks a fused multiply-add instruction nor can it simultaneously issue multiplies and adds, Figure 4 shows a single roofline for the four kernels on the T2+ without the multiply-add balance ceiling.

The Intel Xeon has the highest peak double-precision performance of the four multicores. However, the Roofline model in Figure 3a shows this level of performance can be achieved only with operational intensities of at least 6.7 Flops/Byte; in other words Clovertown requires 55 floating-point operations for every double-precision operand (8B) going to DRAM to achieve peak performance. This high ratio is due in part to the limitation of the front-side bus, which also carries the coherency traffic that can consume up to half the bus bandwidth. Intel includes a snoop filter to prevent unnecessary coherency traffic on the bus. If the working set is small enough for the hardware to filter, the snoop filter nearly doubles the delivered memory bandwidth.

The Opteron X4 has a memory controller on chip, its own path to 667MHz DDR2 DRAM, and separate paths for coherency. Figure 3 shows that the ridge point in the Roofline model is to the left of the Xeon, at an operational intensity of 4.4 Flops/Byte. The Sun T2+ has the highest memory bandwidth so the ridge point is an exceptionally low operational intensity of just 0.33 Flops/Byte. It keeps multiple memory transfers in flight by using many threads. The IBM Cell ridge

point of operational intensity is 0.65 Flops/Byte.

Here, we demonstrate the Roofline model on four diverse mutlicore architectures running four kernels representative of some of the Seven Dwarfs:

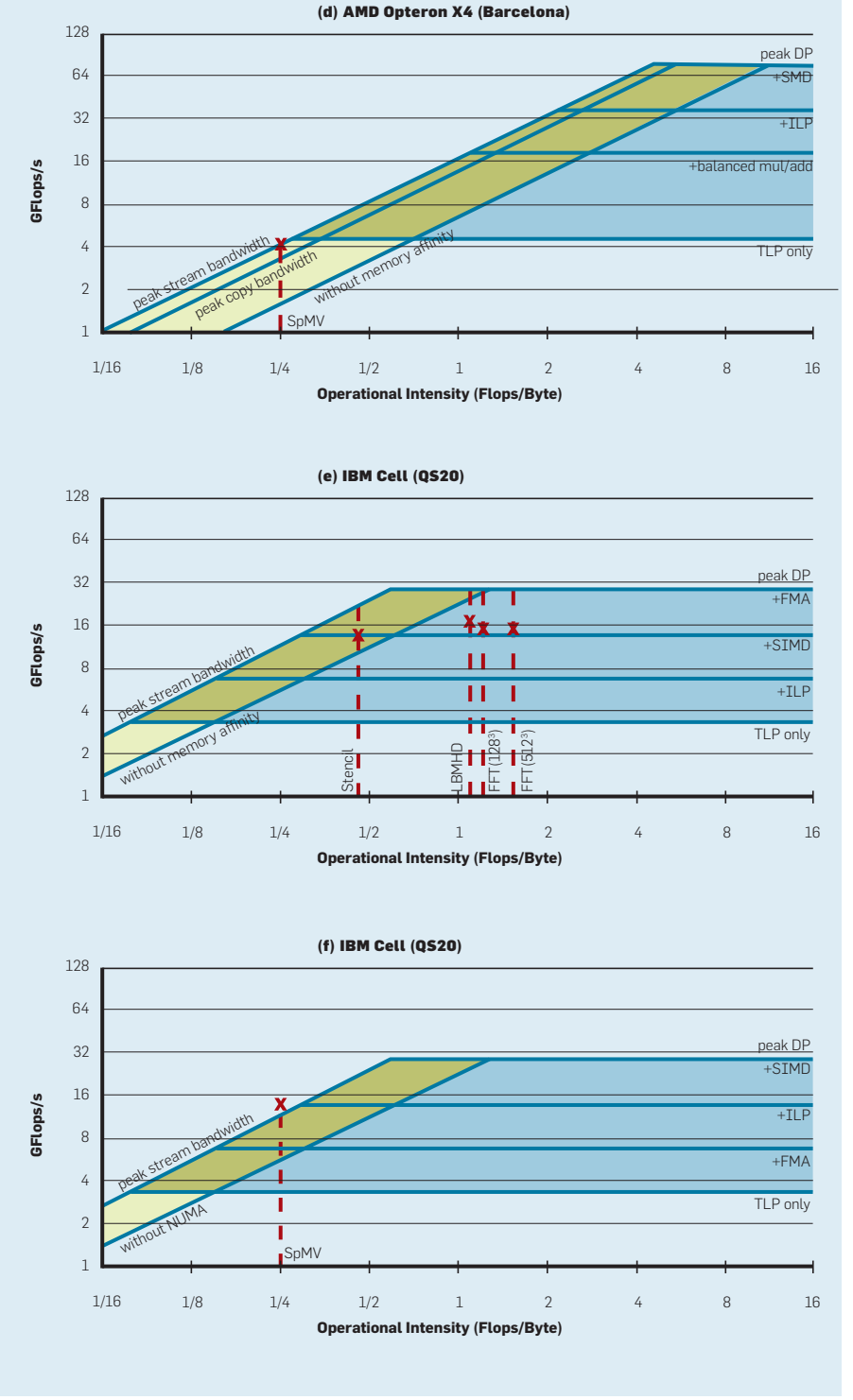
Sparse matrix-vector multiplication. The first example kernel of the sparse matrix computational dwarf is Sparse Matrix-Vector multiply (SpMV); the computation is $y = A^*x$, where A is a sparse matrix and x and y are dense vectors. SpMV is popular in scientific computing, economic modeling, and information retrieval. Alas, conventional implementations often run at less than 10% of peak floating-point performance in uniprocessors. One reason is the irregular accesses to memory, which might be expected from sparse matrices. The operational intensity varies from 0.17 Flops/Byte before a register blocking optimization to 0.25 Flops/Byte afterward[26] (see online Appendix A.1).

Given that the operational intensity of SpMV was below the ridge point of all four multicores in Figure 3, most optimizations involve the memory system. Table 3 summarizes the optimizations used by SpMV and the rest of the kernels. Many are associated with the ceilings in Figure 3, and the height of the ceilings suggests the potential benefit of these optimizations.

Lattice-Boltzmann Magnetohydrodynamics. Like SpMV, LBMHD tends to achieve a small fraction of peak performance on uniprocessors due to the complexity of the data structures and the irregularity of memory access patterns. The Flop-to-Byte ratio is 0.70 vs. 0.25 or less in SpMV. By using the no-allocate store optimization, a programmer can improve the operational intensity of LBMHD to 1.07 Flops/Byte. Both x86 multicores offer this cache optimization, but Cell does not have this problem since it uses DMA. Hence, T2+ is the only one of the four computers with the lower intensity of 0.70 Flops/Byte.

Figures 3 and 4 show that the operational intensity of LBMHD is high enough that both computational and memory bandwidth optimizations make sense on all multicores, except the T2+ where the Roofline ridge point is below that of LBMHD. The T2+ reaches its performance ceiling using only the computational optimizations.

Figure 3d–3f: Roofline model for Intel Xeon, AMD Opteron X4, and IBM Cell.



Stencil. In general, a stencil on a structured grid is defined as a function that updates a point based on the values of its neighbors. The stencil structure remains constant as it moves from one point in space to the next. For this work, we use the stencil derived from the explicit heat equation, a partial differential equation on a uniform 2563 3D grid.[12] The stencil's neighbors are the nearest six points along each axis, as well as the center point itself. This stencil performs eight floating-point operations for every 24B of compulsory memory traffic on write-allocate

Figure 4: Roofline model for Sun UltraSPARC T2+.

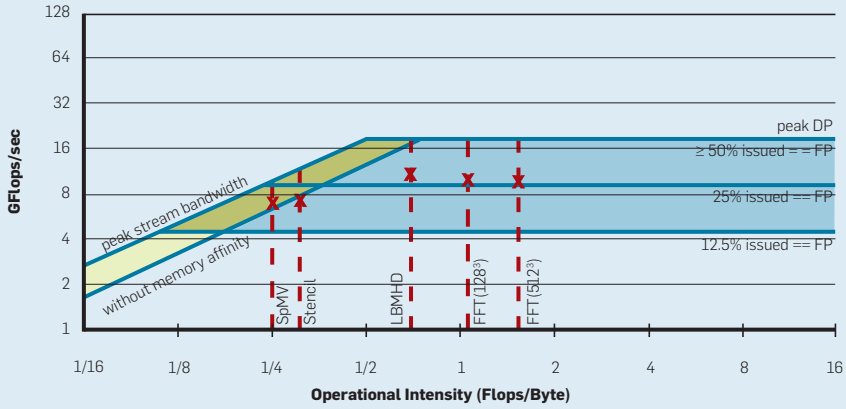
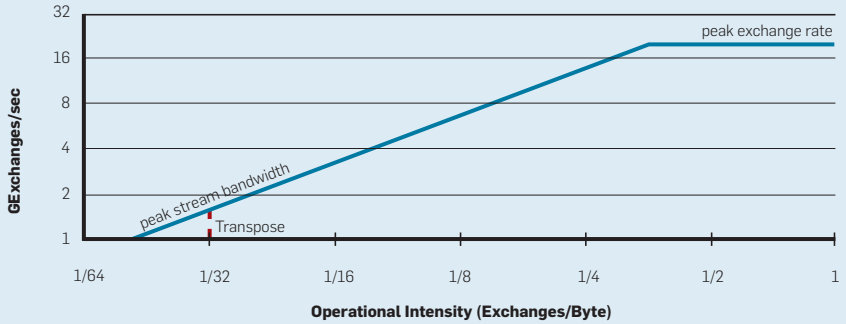


Figure 5: Roofline for transpose phase of 3D FFT for the Cell.



architectures, yielding an operational intensity of 0.33 Flops/Byte.

3D FFT. This fast Fourier transform is the classic divide-and-conquer algorithm that recursively breaks down a discrete Fourier transform into many smaller ones. The FFT is ubiquitous in many domains, including image processing and data compression. An efficient approach for 3D FFT is to perform 1D transforms along each dimension to maintain unit-stride accesses. We computed the 1D FFTs on Xeon, X4, and T2+ using an autotuned library (FFTW).[15] For Cell, we implemented a radix-2 FFT.

FFT differs from SpMV, LBMHD, and Stencil in that its operational intensity is a function of problem size. For the 128^3- and 512^3-point transforms we examine, the operational intensities are 1.09 and 1.41 Flops/Byte, respectively; Cell's 1GB main memory is too small to hold 512^3 points, so we estimate this result. On Xeon and X4, an entire 128×128 plane fits in cache, increasing temporal locality and improving the intensity to 1.64 for the 128^3-point transform.

Productivity vs. performance. In addition to performance, productivity (or the programming difficulty of achieving good performance) is another important issue for the parallel computing revolution.[4] One question is whether a low ridge point gives insight into productivity.

The Sun T2+ (with the lowest ridge point of the four computers) was the easiest to program due to its large memory bandwidth and easy-to-understand cores. The advice for these kernels on T2+ is simply to try to get good-performing code from the compiler, then use as many threads as possible. The downside is that the L2 cache is only 16-way set associative, which can lead to conflict misses when 64 threads access the cache, as it did for the Stencil kernel.

In contrast, the computer with the highest ridge point had the lowest unoptimized performance. The Intel Xeon was difficult to program because it was difficult to understand the memory behavior of the dual front-side buses, how hardware prefetching worked, and the difficulty of getting good SIMD code from the compiler. The C code for both it and the Opteron X4 are liberally sprinkled with intrinsic statements involving SIMD instructions to get good performance.

With a ridge point close to the Xeon, the Opteron X4 required about as much effort, since it benefited from the most types of optimization. However, its memory behavior was easier to understand than the memory behavior of the Xeon.

The IBM Cell (with a ridge point almost as low as the Sun T2+) involved two types of challenges. First, it was difficult for the compiler to exploit the SIMD instructions of Cell's SPE, so at times we needed to help the compiler by inserting intrinsic statements with assembly language instructions into the C code. This comment reflects the immaturity of the IBM compiler, as well as the difficulty of compiling for these SIMD instructions. Second, the memory system is more challenging. Since each SPE has local memory in a separate address space, we could not simply port the code and start running on the SPE. We needed to change the program to issue DMA commands to transfer data back and forth between local store and memory. The good news is that DMA played the role of software prefetch in caches. DMA for a local store is easier to program, achieve good memory performance, and overlap with computation than scheduling prefetches for caches.

To demonstrate the utility of the Roofline Model, Table 4 lists the upper and lower bounding ceilings and the GFlops/sec and GB/sec per kernel-computer pair; recall that operational intensity is the ratio between the two rates. The ceilings listed are the ceilings sandwiching actual performance. All 16 combinations of kernel and computer validate this bound-and-bottleneck model since Roofline's upper and lower ceilings bound performance and the kernels were optimized, as the lower ceilings suggest. The metric that limits performance is in bold; 15 of 16 ceilings are memory-bound for Xeon

Table 3: Kernel optimizations.[12, 28, 29]

Memory affinity. Reduce accesses to DRAM memory attached to the other socket.
Long unit-stride accesses. Change loop structures to generate long unit-stride accesses to engage the prefetchers; also reduces TLB misses.
Software prefetching. Software and hardware prefetching both used to get the most from memory systems.
Reduce conflict misses. Pad arrays to improve cache-hit rates.
Unroll and reorder loops. To expose sufficient parallelism and improve cache utilization, unroll and reorder loops to group statements with similar addresses; improves code quality, reduces register pressure, facilitates SIMD.
"SIMD-ize" code. The x86 compilers didn't generate good SSE code, so we made a code generator to produce SSE intrinsics.
Compress data structures (SpMV only). Since bandwidth limits performance, we used smaller data structures: 16b vs. 32b index and smaller representations of non-zero subblocks.[27]

and X4, while the bottleneck is almost evenly split for T2+ and Cell. For FFT, the surrounding ceilings are memory-bound for Xeon and X4, but compute-bound for T2+ and Cell.

Fallacies About Roofline

We have presented this material in several venues, prompting a number of misconceptions we address here:

Fallacy: The model does not account for all features of modern processors (such as caches and prefetching). The definition of operational intensity we use here does indeed factor-in caches; memory accesses are measured between the caches and memory, not between the processor and caches. In our discussion of performance models, we showed that the memory bandwidth measures of the computer include prefetching and any other optimization (such as blocking) that can improve memory performance. Similarly, some of the optimizations in Table 3 explicitly involve memory. Moreover, in our discussion on tying the 3Cs to operational intensity, we demonstrated the optimizations' effect on increasing operational intensity by reducing capacity and conflict misses.

Fallacy: Doubling cache size increases operational intensity. Autotuning three of the four kernels gets very close to the compulsory memory traffic; the resultant working set is sometimes only a small fraction of the cache. Increasing cache size helps only with capacity misses and possibly conflict misses, so a larger cache has no effect on the operational intensity for the three kernels. However, for 128^3 3D FFT, a larger cache could capture a whole plane of a 3D cube, improving operational intensity by reducing capacity and conflict misses.

Fallacy: The model doesn't account for the long memory latency. The ceilings for no software prefetching in Figures 3 and 4 are at lower memory bandwidth precisely because they cannot hide the long memory latency.

Fallacy: The model ignores integer units in floating-point programs, possibly limiting performance. For the example kernels we've outlined here, the amount of integer code and integer performance can affect performance. For example, the Sun UltraSPARC T2+ fetches two instructions per core per clock cycle and doesn't implement the SIMD instructions of the x86 that can operate on two double-precision floating-point operands at a time. Relative to other processors, the T2+ expends a larger fraction of its instruction issue bandwidth on integer instructions and executes them at a lower rate, hurting overall performance.

Fallacy: The model has nothing to do with multicore. Little's Law[17, 20, 21] dictates that considerable concurrency is necessary to really push the limits of the memory system. This concurrency is more easily satisfied in a multicore than in a uniprocessor. While the bandwidth orientation of the Roofline model certainly works for uniprocessors, it is even more helpful for multicores.

Fallacy: You need to recalculate the Roofline model for every kernel. The Roofline needs to be calculated for given performance metrics and computers just once; it then guides the implementation for any program for which that metric is the critical performance metric. The kernels we've explored here use floating-point operations and main memory traffic. The ceilings are measured once but can be reordered depending on whether or not multiplies and adds are naturally balanced in the kernel (see the earlier discussion on adding ceilings to the model).

Note that the heights of the ceilings we discuss here document the maximum potential gain of a code performing this optimization. An interesting future direction is to use performance counters to adjust the height of the ceilings and the order of the ceilings for a particular kernel to show the actual benefits of each optimization and the recommended order to try them (see online Appendix A.3).

Fallacy: The model is limited to easily optimized kernels that never hit in the cache. These kernels do indeed hit in the cache; for example, the cache-hit rates of our three multicores with on-chip caches are at least 94% for Stencil and 98% for FFT. Moreover, if the Seven Dwarfs were easy to optimize, it would bode well for the future of multicores. However, our experience is that it is not easy to create the fastest version of these numerical methods on the divergent multicore architectures discussed here. Indeed, three of the results were judged significant enough to be accepted for publication at major conferences.[12, 28, 29]

Fallacy: The model is limited to floating-point programs. Our focus here has been on floating-point programs, so the two axes of the model are floating-point operations per second and the floating-point operational intensity of accesses to main memory. However, the Roofline model can work for other kernels where performance is a function of different performance metrics. A concrete example is the transpose phase of 3D FFT, which performs no floating-point operations at all. Figure 5 shows a Roofline model for just this phase on Cell, with exchanges replacing Flops in the model. One exchange involves reading and writing 16B, so its operational intensity is 1/32 pairwise Exchanges/Byte. Despite the computational metric being memory exchanges, there is still a computational

Table 4: Achieved performance and nearest Roofline ceilings, with metric limiting performance in bold (3D FFT is 128^3)

		Upper Ceiling			Achieved Performance			Lower Ceiling		
	Kernel	Type	Name	Value	Compute	Memory	O.I.	Type	Name	Value
Intel Xeon	SpMV	Memory	Stream BW	11.2GB/sec	2.8GFlops/sec	**11.1GB/sec**	0.25	Memory	Snoop filter	5.9GByte/sec
	LBMHD	Memory	Snoop filter	5.9GB/sec	5.6GFlops/sec	**5.3GB/sec**	1.07	Memory	(none)	0.0GByte/sec
	Stencil	Memory	Snoop filter	5.9GB/sec	2.5GFlops/sec	**5.1GB/sec**	0.50	Memory	(none)	0.0GByte/sec
	3D FFT	Memory	Snoop filter	5.9GB/sec	9.7GFlops/sec	**5.9GB/sec**	1.64	Compute	TLP only	6.2GFlops/sec
AMD X4	SpMV	Memory	Stream BW	17.6GB/sec	4.2GFlops/sec	**16.8GB/sec**	0.25	Memory	Copy BW	13.9GByte/sec
	LBMHD	Memory	Copy BW	13.9GB/sec	11.4GFlops/sec	**10.7GB/sec**	1.07	Memory	No Affinity	7.0GByte/sec
	Stencil	Memory	Stream BW	17.6GB/sec	8.0GFlops/sec	**16.0GB/sec**	0.50	Memory	Copy BW	13.9GByte/sec
	3D FFT	Memory	Copy BW	13.9GB/sec	14.0GFlops/sec	**8.6GB/sec**	1.64	Memory	No Affinity	7.0GByte/sec
Sun T2+	SpMV	Memory	Stream BW	36.7GB/sec	7.3GFlops/sec	**29.1GB/sec**	0.25	Memory	No Affinity	19.8GByte/sec
	LBMHD	Memory	No Affinity	19.8GB/sec	10.5GFlops/sec	**15.0GB/sec**	0.70	Compute	25% issued FP	9.3GFlops/sec
	Stencil	Compute	25% issued FP	9.3GFlopss/sec	**6.8GFlops/sec**	20.3GB/sec	0.33	Memory	No Affinity	19.8GByte/sec
	3D FFT	Compute	Peak DP	19.8GFlops/sec	**9.2GFlops/sec**	10.0GB/sec	1.09	Compute	25% issued FP	9.3GFlops/sec
IBM Cell	SpMV	Memory	Stream BW	47.6GB/sec	11.8GFlops/sec	**47.1GB/sec**	0.25	Memory	FMA	7.3GFlops/sec
	LBMHD	Memory	No Affinity	23.8GB/sec	16.7GFlops/sec	**15.6GB/sec**	1.07	Memory	Without FMA	14.6GFlops/sec
	Stencil	Compute	Without FMA	14.6GFlopss/sec	**14.2GFlops/sec**	30.2GB/sec	0.47	Memory	No Affinity	23.8GByte/sec
	3D FFT	Compute	Peak DP	29.3GFlops/sec	**15.7GFlops/sec**	14.4GB/sec	1.09	Compute	SIMD	14.6GFlops/sec

horizontal Roofline, since local stores and caches could affect the number of exchanges that go to DRAM.

Fallacy: The Roofline model must use DRAM bandwidth. If the working set fits in the L2 cache, the diagonal Roofline could be L2 cache bandwidth instead of DRAM bandwidth, and the operational intensity on the *x*-axis would be based on Flops per L2 cache Byte accessed. The diagonal memory performance line would move up, and the ridge point would surely move to the left. For example, Jike Chong of the University of California, Berkeley, ported two financial partial differential equation (PDE) solvers to four other multicore computers: the Intel Penryn and Larrabee and NVIDIA G80 and GTX280.[10] He used the Roofline model to keep track of all four of their peak arithmetic throughput and L1, L2, and DRAM bandwidths. By analyzing an algorithm's working set and operational intensity, he was able to use the Roofline model to quickly estimate the needs for algorithmic improvement. Specifically, for the option-pricing problem with an implicit PDE solver, the working set is small enough to fit into L1, and the L1 bandwidth is sufficient to support peak arithmetic throughput; the Roofline model thus indicates that no optimization is necessary. For option pricing with an explicit PDE formulation, the working set is too large to fit into cache, and the Roofline model helps indicate the extent cache blocking is necessary to produce peak arithmetic performance.

Conclusion

The sea change from sequential computing to parallel computing is increasing the diversity of computers that programmers must confront when building correct, efficient, scalable, portable software.[4] Here, we've described a simple, visual computational model we call the Roofline Model to help identify which systems would be a good match for important kernels or conversely to determine how to change kernel code or hardware to run desired kernels well. For floating-point kernels that do not fit completely in caches, we've shown how operational

intensity—the number of floating-point operations per byte transferred from DRAM—is an important parameter for both the kernels and the multicore computers.

We applied Roofline to four kernels from among the Seven Dwarfs[4,11] to four recent multicore designs: AMD Opteron X4, Intel Xeon, IBM Cell, and Sun T2+. The ridge point—the minimum operational intensity to achieve maximum performance—proved to be a better predictor of performance than clock rate or peak performance. Cell offered the highest attained performance (GFlops/sec) on these kernels, but T2+ was the easiest computer on which to achieve its highest performance. One reason is because the ridge point of the Roofline Model for T2+ was the lowest.

Just as the graphical Roofline Model offers insights into the difficulty of achieving the peak performance of a computer, it also makes obvious when a computer is imbalanced. The operational ridge points for the two x86 computers were 4.4 and 6.7—meaning a 35 to 55 Flops/Byte operand that accesses DRAM—yet the operational intensities for the 16 combinations of kernels and computers in Table 4 ranged from 0.25 to just 1.64, with a median of 0.60 Flops/Byte. Architects should keep the ridge point in mind if they want programs to reach peak performance on their new designs.

We measured the roofline and ceilings using microbenchmarks but could have used performance counters (see online Appendix A.1 and A.3). There may indeed be a synergistic relationship between performance counters and the Roofline Model. The requirements for automatic creation of a Roofline model could guide the designer as to which metrics should be collected when faced with literally hundreds of candidates but only a limited hardware budget.[6]

Roofline offers insights into other types of multicore systems (such as vector processors and graphical processing units); other kernels (such as sort and ray tracing); other computational metrics (such as pair-wise sorts per second and frames per second); and other traffic metrics (such as L3 cache bandwidth and I/O bandwidth). Alas, there are many more opportunities for Roofline-oriented research than we can pursue. We thus invite others to join us in the exploration of the effectiveness of the Roofline Model.

Acknowledgments

This research was sponsored in part by the Universal Parallel Computing Research Center funded by Intel and Microsoft and in part by the Office of Advanced Scientific Computing Research in the U.S. Department of Energy Office of Science under contract number DE-AC02-05CH11231. We'd like to thank FZ-Jülich and Georgia Tech for access to Cell blades and Joseph Gebis, Leonid Oliker, John Shalf, Katherine Yelick, and the rest of the Par Lab for feedback on Roofline, and to Jike Chong, Kaushik Datta, Mark Hoemmen, Matt Johnson, Jae Lee, Rajesh Nishtala, Heidi Pan, David Wessel, Mark Hill, and the anonymous reviewers for insightful feedback on early drafts. C

References

1. Adve, V. *Analyzing the Behavior and Performance of Parallel Programs.* Ph.D. thesis, University of Wisconsin, 1993; www.cs.wisc.edu/techreports/1993/TR1201.pdf.
2. AMD. *Software Optimization Guide for AMD Family 10h Processors, Publication 40546*, Apr. 2008; www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/40546.pdf.
3. Amdahl, G. Validity of the single processor approach to achieving large-scale computing capabilities. In *Proceedings of the AFIPS Conference*, 1967, 483–485.
4. Asanovic, K., Bodik, R., Catanzaro, B., Gebis, J., Keutzer, K., Patterson, D., Plishker, W., Shalf, J., Williams, S., and Yelick, K. *The Landscape of Parallel Computing Research: A View from Berkeley.* Technical Report UCB/EECS-2006-183. EECS, University of California, Berkeley, Dec. 2006.
5. Bienia, C., Kumar, S., Singh, J., and Li, K. *The PARSEC Benchmark Suite: Characterization and Architectural Implications.* Technical Report TR-81 1-008. Princeton University, Jan. 2008.
6. Bird, S., Waterman, A., Klues, K., Datta, K., Liu, R., Nishtala, R., Williams, S., Asanovi, K., Demmel, J., Patterson, D., and Yelick, K. A case for sensible performance counters. Submitted to the First USENIX Workshop on Hot Topics in Parallelism (Berkeley CA, Mar. 30–31, 2009); www.usenix.org/events/hotpar09/.
7. Boyd, E., Azeem, W., Lee, H., Shih, T., Hung, S., and Davidson, E. A hierarchical approach to modeling and improving the performance of scientific applications on the KSR1. In *Proceedings of the 1994 International Conference on Parallel Processing*, 1994, 188–192.
8. Callahan, D., Cocke, J., and Kennedy, K. Estimating interlock and improving balance for pipelined machines. *Journal of Parallel Distributed Computing 5* (1988), 334–358.
9. Carr, S. and Kennedy, K. Improving the ratio of memory operations to floating-point operations in loops. *ACM Transactions on Programming Languages and Systems 16*, 4 (Nov. 1994).
10. Chong, J. Private communication on financial PDE solvers, 2008.
11. Colella, P. *Defining Software Requirements for Scientific Computing.* Presentation, 2004.
12. Datta, K., Murphy, M., Volkov, V., Williams, S., Carter, J., Oliker, L., Patterson, D., Shalf, J., and Yelick, K. Stencil computation optimization and autotuning on state-of-the-art multicore architectures. In *Proceedings of the 2008 ACM/IEEE SC08 Conference* (Austin, TX, Nov. 15–21). IEEE Press, Piscataway, NJ, 2008, 1-12.
13. Demmel, J., Dongarra, J., Eijkhout, V., Fuentes, E., Petitet, A., Vuduc, R., Whaley, R., and Yelick, K. Self-adapting linear algebra algorithms and software. *Proceedings of the IEEE: Special Issue on Program Generation, Optimization, and Adaptation 93*, 2 (2005).
14. Dubois, M. and Briggs, F.A. Performance of synchronized iterative processes in multiprocessor systems. *IEEE Transactions on Software Engineering SE-8*, 4 (July 1982), 419–431.
15. Frigo, M. and Johnson, S. The design and implementation of FFTW3. *Proceedings of the IEEE: Special Issue on Program Generation, Optimization, and Platform Adaptation 93*, 2 (2005).
16. Harris, M. Mapping computational concepts to GPUs. In *ACM SIGGRAPH Courses*, Chapter 31 (Los Angeles, July 31-Aug. 4). ACM Press, New York, 2005.
17. Hennessy, J. and Patterson, D. *Computer Architecture: A Quantitative Approach, Fourth Edition.* Morgan Kaufmann Publishers, Boston, MA, 2007.
18. Hill, M. and Marty, M. Amdahl's Law in the multicore era. *IEEE Computer* (July 2008), 33–38.
19. Hill, M. and Smith, A. Evaluating associativity in CPU caches. *IEEE Transactions on Computers 38*, 12 (Dec. 1989), 1612–1630.
20. Lazowska, E., Zahorjan, J., Graham, S., and Sevcik, K. *Quantitative System Performance: Computer System Analysis Using Queueing Network Models.* Prentice Hall, Upper Saddle River, NJ, 1984.
21. Little, J.D.C. A proof of the queueing formula L = λ W. *Operations Research 9*, 3 (1961), 383–387.
22. McCalpin, J. *STREAM: Sustainable Memory Bandwidth in High-Performance Computers*, 1995; www.cs.virginia.edu/stream.
23. Patterson, D. Latency lags bandwidth. *Commun. ACM 47*, 10 (Oct. 2004).
24. Thomasian, A. and Bay, P. Analytic queueing network models for parallel processing of task systems. *IEEE Transactions on Computers C-35*, 12 (Dec. 1986), 1045–1054.
25. Tikir, M., Carrington, L., Strohmaier, E., and Snavely, A. A genetic algorithms approach to modeling the performance of memory-bound computations. In *Proceedings of the SC07 Conference* (Reno, NV, Nov. 10–16). ACM Press, New York, 2007.
26. Vuduc, R., Demmel, J., Yelick, K., Kamil, S., Nishtala, R., and Lee, B. Performance optimizations and bounds for sparse matrix-vector multiply. In *Proceedings of the ACM/IEEE SC02 Conference* (Baltimore, MD, Nov. 16–22). IEEE Computer Society Press, Los Alamitos, CA, 2002.
27. Williams, S. *Autotuning Performance on Multicore Computers.* Ph.D. Thesis. University of California, Berkeley, Dec. 2008; www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-164.html.
28. Williams, S., Carter, J., Oliker, L., Shalf, J., and Yelick, K. Lattice Boltzmann simulation optimization on leading multicore platforms. In *Proceedings of the IEEE International Symposium on Parallel and Distributed Processing Symposium* (Miami, FL, Apr. 14–18, 2008), 1–14.
29. Williams, S., Oliker, L., Vuduc, F., Shalf, J., Yelick, K., and Demmel, J. Optimization of sparse matrix-vector multiplication on emerging multicore platforms. In *Proceedings of the ACM/IEEE SC07 Conference* (Reno, NV, Nov. 10–16). ACM Press, New York, 2007.
30. Woo, S., Ohara, M., Torrie, E., Singh, J.-P., and Gupta, A.The SPLASH-2 programs: Characterization and methodological considerations. In *Proceedings of the 22nd Annual International Symposium on Computer Architecture*. ACM Press, New York, 1995, 24–37.

Samuel Williams (SWWilliams@lbl.gov) is a research scientist at Lawrence Berkeley National Laboratory, Berkeley, CA.

Andrew Waterman (waterman@eecs.berkeley.edu) is a graduate student researcher in the Parallel Computing Laboratory of the University of California, Berkeley.

David Patterson (pattrsn@eecs.berkeley.edu) is Director of the Parallel Computing Laboratory of the University of California, Berkeley, and a past president of ACM.

The Best Place to Find the Perfect Job... Is Just a Click Away!

No need to get lost on commercial job boards.
The ACM Career & Job Center is tailored specifically for you.

JOBSEEKERS

- Manage your job search
- Access hundreds of corporate job postings
- Post an anonymous resume
- Advanced Job Alert system

EMPLOYERS

- Quickly post job openings
- Manage your online recruiting efforts
- Advanced resume searching capabilities
- Reach targeted & qualified candidates

NEVER LET A JOB OPPORTUNITY PASS YOU BY!

START YOUR JOB SEARCH TODAY!

http://www.acm.org/careercenter

Association for Computing Machinery

Advancing Computing as a Science & Profession

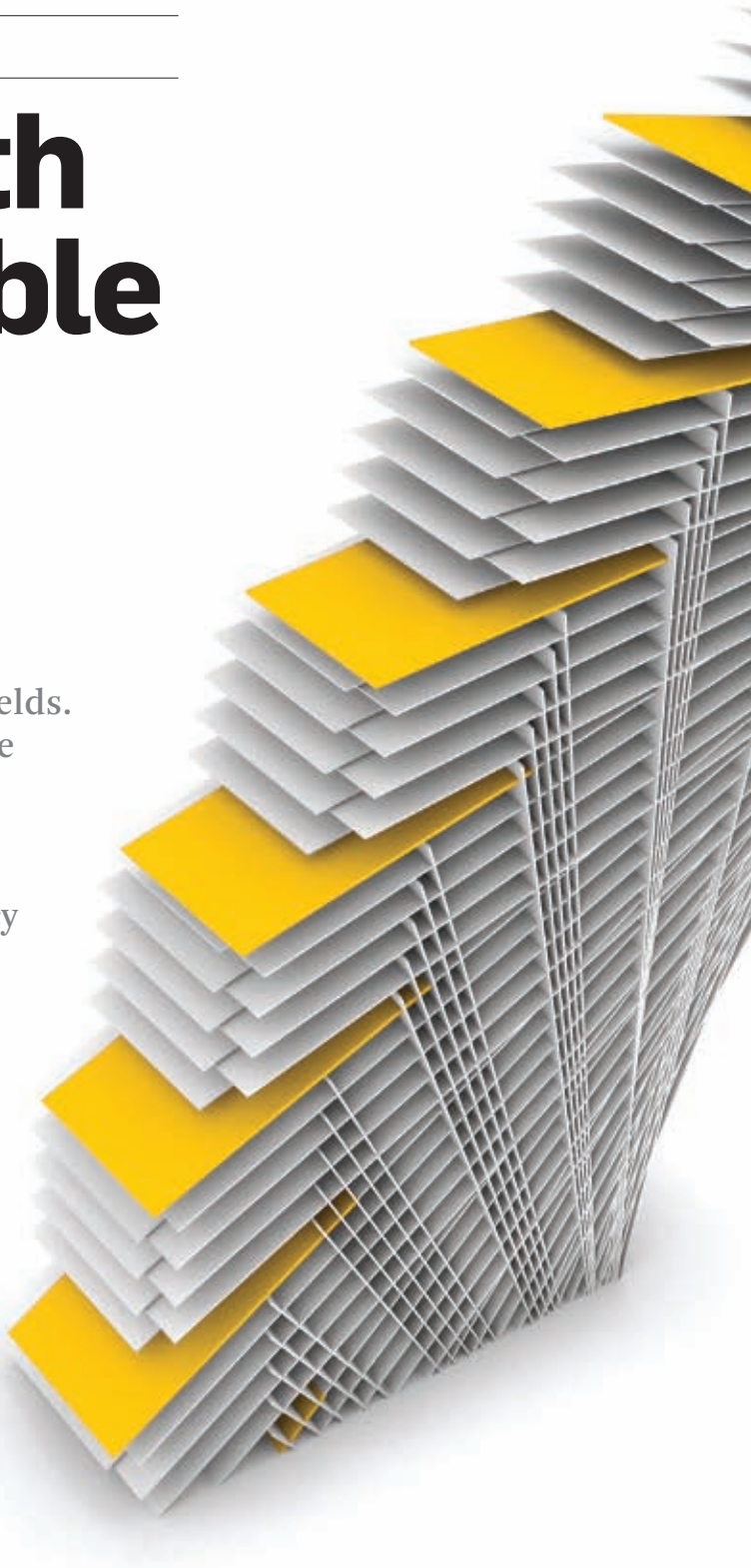
POWERED BY JOBTARGET

DOI:10.1145/1498765.1498787

Who could fault an approach that offers greater credibility at reduced cost?

BY DANIEL JACKSON

A Direct Path to Dependable Software



SOFTWARE PLAYS A fundamental role in our society, bringing enormous benefits to all fields. But because many of our current systems are highly centralized and tightly coupled,[33] we are also susceptible to massive and coordinated failure.

A Chicago hospital lost its entire pharmacy database one night, and it was only able to reconstruct medication records for its patients by collecting paper printouts from nurses' stations. In their report on this incident,[6] Richard Cook and Michael O'Connor concluded: "Accidents are signals sent from deep within the system about the sorts of vulnerability and potential for disaster that lie within." Similar signals have been asent, for example, in the fields of electronic voting,[24] air traffic control,[13] nuclear power,[25] and energy distribution.[34]

The growing tendency to embed

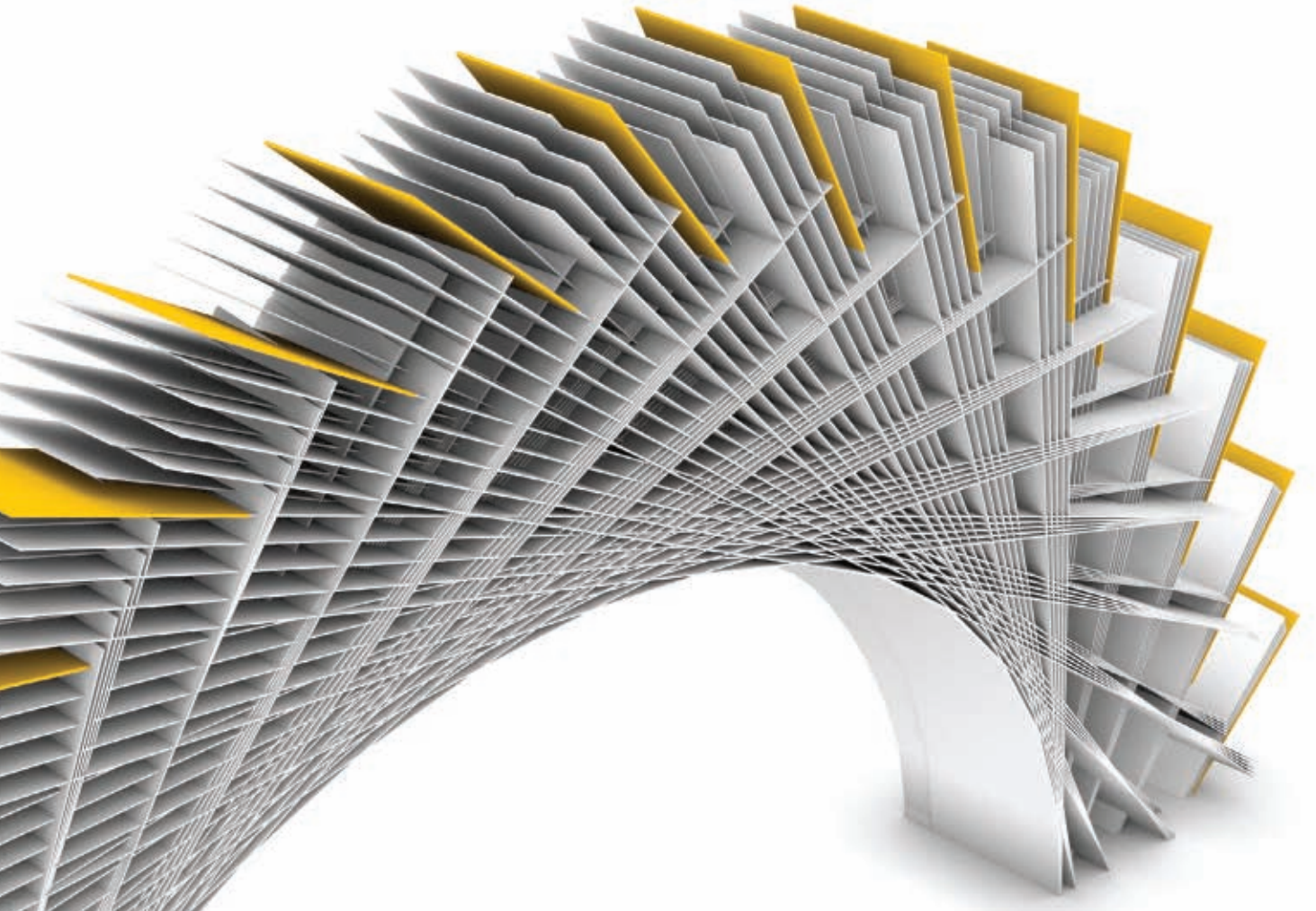


ILLUSTRATION BY MIKAEL CHRISTENSEN

software in powerful and invasive physical devices brings greater risk, especially in medicine, where software can save lives but also kill.[36] Software problems led to the recall of 200,000 implanted pacemakers and defibrillators between 1990 and 2000.[30] In the 20 years prior to 2005, the U.S. Food and Drug Administration (FDA) recorded 30,000 deaths and 600,000 injuries from medical-device failures.[9] How many of these incidents can be attributed to software is unclear, though separate studies have found that about 8% of medical-device recalls are software-related. Moreover, few of the device failures that occur—perhaps only 1 in 40—are actually reported,[12] so the actual incidence of injuries is likely to be higher.

What would it take to make software more dependable? Until now, most approaches have been indirect, involving practices—processes, tools, or techniques—believed to yield dependable software. The case for dependability has thus rested on the extent to which the developers adhered to these practices. This article argues that developers should instead produce *direct* evidence of their software's dependability. The potential advantages of this approach are greater credibility (as the claim is not contingent on the effectiveness of the practices) and reduced cost (because development resources can be focused where they have the most impact).

The Need for a Direct Approach

A dependable system is one you can *depend* on—that is, you can place your trust in it. A rational person or organization only does this with *evidence* that the system's benefits far outweigh its risks. Without such evidence, a system cannot be depended on, in much the same way that a download from an unknown Web site cannot be said to be "safe" just because it happens not to harbor a virus.

Perhaps in the future we will know enough about software-development practices that the very use of a particular technique will constitute evidence of the resulting software's quality. Today, however, we are far from that goal. Although individual companies can predict defect rates within product families based on historical data, industrywide data collection and analysis barely exist.

Contrast software systems with cars, for example. In the U.S., the National Highway Traffic Safety Administration (NHTSA) maintains several databases that include records of all fatal accidents—approximately 40,000 a year—and data about how particular models fare in crashes. Researchers can use this data to correlate risk with design features. NHTSA also receives data from auto companies regarding warranty claims, defects, and customer complaints. Similarly, the National Transportation Safety Board (NTSB), best known for its work in aviation, analyzes highway accidents and issues reports on, among other things, the efficacy of safety devices such as seatbelts and airbags.

The software industry has no comparable mechanism, and as a society we have almost no data on the causes or effects of software failure. Producers of software therefore cannot benefit from industry data that would improve their designs and development strategies, and consumers cannot use such

data to make informed purchasing decisions.

Actually, where data is collected, it is often suppressed; many companies withhold even basic information about the number and severity of defects in their products, even when issuing patches that purport to resolve them. And no government agency is charged with investigating software failures or even recording software-related fatal accidents. When an accident report does implicate software, it rarely includes enough information to allow any general lessons to be learned.

Over the past few decades we have developed approaches and technologies that can dramatically improve the quality of software. They include better platforms (safe programming languages, operating systems with address-space separation, virtual machines), better development infrastructure (configuration control, bug tracking, traceability), better processes (spiral and agile models, prototyping), and better tools (integrated environments, static analyzers, model checkers). Moreover, we have made progress in understanding the fundamentals, for example, of problem structuring, design modeling, software architecture, verification, and testing. All of these advances can be misused, however, and none of them guarantees success. The field of empirical software development is attempting to fill the gap and provide scientific measures of efficacy, but there is still no evidence compelling enough that simply using a given approach establishes with confidence the quality of the resulting system.

Many certification standards were devised with the good intent of enforcing best practices, but they have had the opposite effect. Instead of encouraging the selection of the best tool for the job, and directing attention to the most critical aspects of a system and its development, they impose burdensome demands to apply the same—often outdated—techniques uniformly, resulting in voluminous documentation of questionable value. The Common Criteria security certification that Microsoft obtains for its operating systems, for example, costs more than its internally devised mitigations but is believed by the company to be far less effective.

Government agencies are often in the unfortunate position of having to evaluate complex software systems solely on the basis of evidence that some process, however arbitrary, was adhered to and some amount of testing, whether conclusive or not, was performed. Not surprisingly, certified systems sometimes fail catastrophically. A particularly tragic example was the failure of an FDA-certified radiation-therapy machine in Panama in 2001;[20] fatal overdoses resulted from poorly engineered software in an incident reminiscent of the Therac failures of 15 years earlier. Even the most highly regarded standards demand expensive practices whose value is hard to assess. DO178B, for example, the safety standard used in the U.S. for avionics systems, requires a level of test coverage known as MCDC that is extremely costly and whose benefits studies have yet to substantiate.[14]

A very different approach, sometimes called "goal-based" or "case-based" certification, is now gaining currency. Instead of particular practices being mandated, the developer is instead called upon to provide *direct* evidence that the particular system satisfies its claimed dependability goals. In the U.K., the Ministry of Defence has dramatically simplified its procurement standards for software under this approach, with contractors providing "software reliability cases" to justify the system. Even in the early stages, a reliability case is required to defend the proposed architecture and to show that the contractor is capable of making a case for the development itself.[31]

This direct approach has not yet been adopted by American certifiers and procurers. Recently, however, a number of government agencies, spearheaded by the High Confidence Software and Systems Coordinating Group, funded a National Academies study to address widespread concerns about the costs and effectiveness of existing approaches to software dependability.[22] The direct approach recommended by the study is the basis of this article.

Why Testing Isn't Good Enough

Testing is a crucial tool in the software developer's repertoire, and the use of automated tests—especially "regression tests" for catching defects introduced by modifications—is a mark of competence. More extensive testing can only improve the quality of software, and many researchers are recognizing the potential for harnessing computational resources to increase the power of testing yet further. At the same time, however, despite the famous pronouncement of Edsger Dijkstra that testing can be used to show the presence of errors but not their absence, there is a widespread folk belief that testing is sufficient evidence for dependability.

Everything we know about testing indicates that this belief is false. Although some small components can be tested exhaustively, the state space of an entire system is usually so huge that the proportion of scenarios executed in a typical test is vanishingly small. Software is not continuous, so a successful test for one input says nothing about the system's response to a similar but distinct input. In practice, it is very difficult even to achieve full code coverage—that is, with every statement of the code being executed.

An alternative approach is to generate tests in a distribution that matches the expected usage profile, adjusted for risk so that most of the testing effort is spent in the most critical areas of functionality. But the number of tests required to obtain high confidence (even with some dubious statistical assumptions) is far larger than one might imagine. For example, to claim a failure rate of one input in a thousand to a 99% confidence, about 5,000 test cases are needed, assuming no bugs are found.[28] If testing reveals 10 bugs, nearer to 20,000 subsequent tests without failure are needed. Contrary to the intuition of many programmers, finding bugs should not increase confidence that fewer bugs remain; indeed, it is evidence that there are more bugs to be found.

Thus while testing may provide adequate confidence that a program is good for a noncritical application, it becomes increasingly difficult and expensive as higher levels of assurance are demanded; under such circumstances, testing cannot deliver the confidence required at a reasonable cost. Most systems will be tested more thoroughly by their users than by their developers, and will often be executing in uncharted territory, exploring combinations of

state components that were not tested and perhaps not even considered during design.

Nevertheless, most certification regimes still rely primarily on testing. Developers and certifiers sometimes talk self-assuredly about achieving "five nines" of dependability, meaning that the system is expected to survive 100,000 commands or hours before failing. The mismatch between such claims and the reality of software failures led one procurer to quip "It's amazing how quickly 10^5 hours comes around."

A Direct Approach

The direct approach, by definition, is straightforward. The desired dependability goal is explicitly articulated as a collection of *claims* that the system has some critical *properties*. An argument, or *dependability* case, is constructed that substantiates the claims. The remainder of this article develops these notions and outlines some of their implications, but first we turn to the fundamental questions of what constitutes a system and what it means for the system to be dependable.

What is a system? An engineered product that is introduced to solve a particular problem and that consists of software, the hardware platform on which the software runs, the peripheral devices through which the product interacts with the environment, and any other components that contribute to achieving the product's goals (including human operators and users) is considered a *system*. In many cases, the system's designers must assume that its operators behave in a certain way. An air traffic management system, for example, cannot prevent a midair collision if a pilot is determined to hit another aircraft; eliminating this assumption would require a separation of aircraft that would not be economically feasible. When a system's dependability is contingent on assumptions about its operators, they should be viewed as a component of the system and the design of operating procedures regarded as an essential part of the overall design.

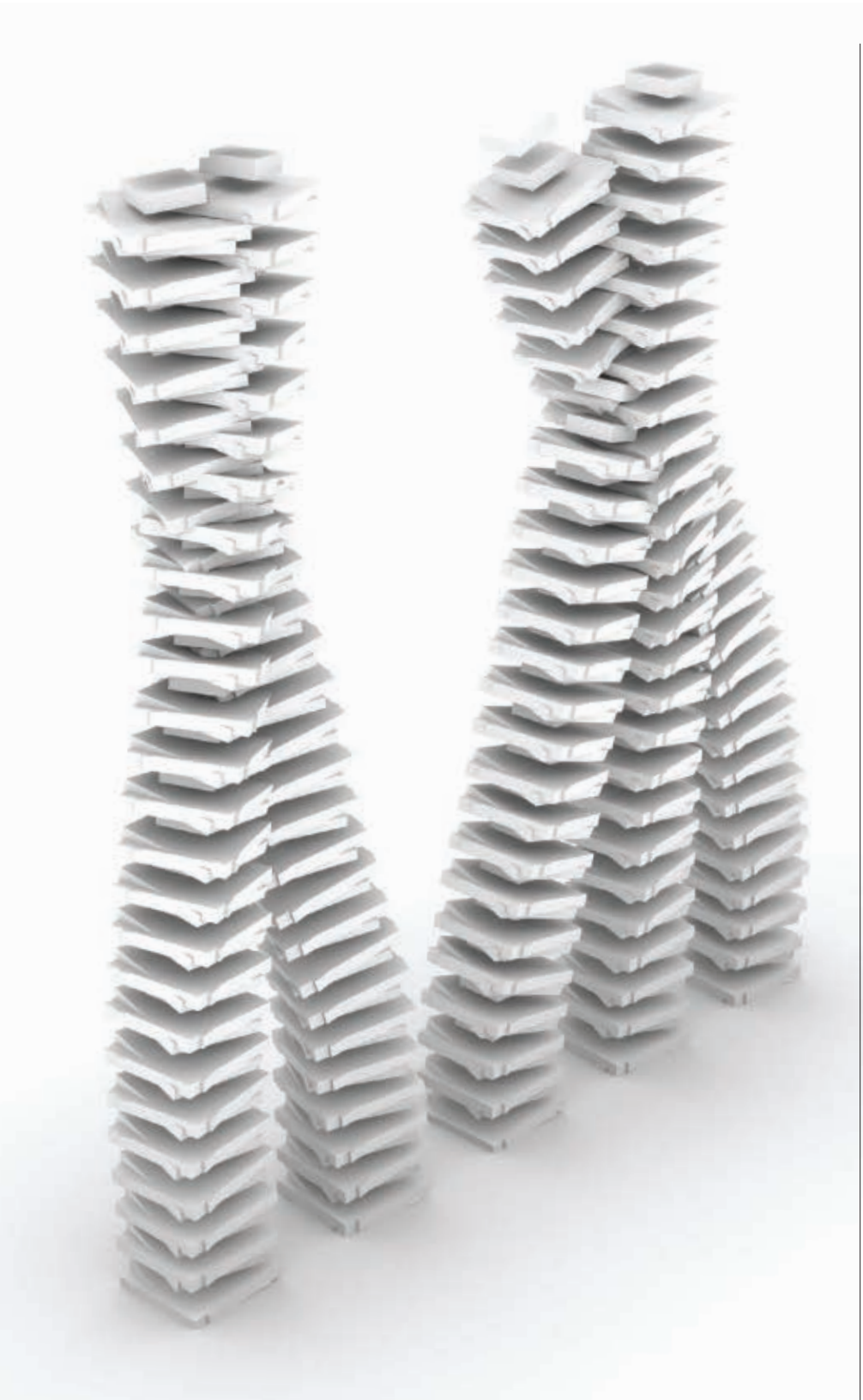
What does "dependable" mean? A system is *dependable* if can be depended on—that is, trusted—to perform a particular task. As noted earlier, such trust is only rational when evidence of the system's ability to act without exhibiting certain failures has been assessed. So a system cannot be dependable without evidence, and dependability is thus not merely the absence of defects or the failures that may result from them but the presence of concrete information suggesting that such failures will not occur.

As in all engineering enterprises, dependability is a trade-off between benefits and risks, with the level of assurance (and the quality and cost of the evidence) being chosen to match the risk at hand.

As in all engineering enterprises, dependability is a trade-off between benefits and risks, with the level of assurance (and the quality and cost of the evidence) being chosen to match the risk at hand. Our society is not willing to tolerate the failure of a nuclear power plant, air traffic control center, or energy distribution network, so for such systems we will be willing to absorb larger development and certification costs. Criticality depends, of course, on the context of use. A spreadsheet program becomes critical if it is used, say, for calculating radiotherapy doses. And there are systems, such as GPS satellites and cellphone networks, on which so many applications depend that widespread failure could be catastrophic.

Dependability is not a metric that can be measured on a simple numeric scale, because different kinds of failures have very different consequences. The cost of preventing all failures will usually be prohibitive, so a dependable system will not offer uniform levels of confidence across all functions. In fact, a large variance is likely to be a characteristic of a dependable system. Thus a dependable radiotherapy system may become unavailable but cannot be allowed to overdose a patient; a dependable e-commerce site may display advertisements incorrectly, give bad search results, and perhaps lose shopping-cart items over time, but it must never bill the wrong amount or leak customers' credit card details; a dependable file synchronizer may report spurious conflicts but should never silently overwrite newer versions of files.

Together, these considerations imply that the first steps in developing a dependable system involve drawing its boundaries—deciding which components in addition to the software, physical and human, will be relied on; identifying the critical properties; and determining what level of confidence is required.



Properties and where they reside. So far I have talked loosely about a dependable system performing some functions or tasks. But for articulating claims about a system's desired behavior, this level of granularity is too coarse. It is preferable instead to focus on critical *properties*. Some will be associated with individual functions, but more often a property will crosscut several functions.

For dependability, focusing on properties is generally better than focusing on functions because the properties are what matter. Moreover, they can usually be separated more cleanly from one another, and they retain their meaning as the set of functions offered by the system changes over its lifetime. A critical property of a crime database, for example, may be that every access to the database by a user is logged in some file. Identifying instead some critical subset of logging functions would be inferior, as the full correctness of these functions would likely be neither necessary nor sufficient for establishing the logging property. Common Criteria, a certification scheme for security, makes this mistake; it focuses attention on the security functions alone, despite the fact that many attacks succeed precisely because they exploit loopholes in other functions that were not thought to be security-related.

Some software systems provide an entirely virtual service, but most interact with the physical world. When the purpose of a system is to produce, control, or monitor particular physical phenomena, they should form the vocabulary for expressing critical properties. This might seem obvious, but there is long tradition of writing requirements in terms of interfaces closer to the software, perhaps because it's easier or because of a division of labor that isolates software engineers from system-level concerns. In a radiotherapy application, for example, a critical property is not that the emitted beam has a bounded intensity, or that the right signal is conveyed to the beam-generating device, or that the beam settings are computed correctly in the code. It is that the patient does not receive an excessive dose.

There is a chain of events connecting the ultimate physical effects of the system at one end back through the signals of the peripherals in the middle to the instructions executed in the code at the other end. The more the critical property is formulated using phenomena closer to the software and further away from the ultimate effects in the real world, the more its correlation to the fundamental concerns of the users is weakened.

An infamous accident illustrates the potentially dire consequences of this too-close-to-the-software tendency. An Airbus A320 landing at Warsaw Airport in 1993 was equipped with an interlock intended to prevent the pilot from activating reverse thrust while airborne.

ILLUSTRATION BY MIKAEL CHRISTENSEN

Unfortunately, the software had been designed to meet a requirement that reverse thrust be disabled unless wheel pulses were being received (indicating that the wheels were turning and thus in contact with the ground). Because of rain on the runway, the aircraft aquaplaned when it touched down, and the wheels did not turn, so the software dutifully disabled reverse thrust and the aircraft overran the runway. Had the critical property been expressed in terms of being on the ground rather than receiving wheel pulses, the invalid assumption that they were equivalent may have been scrutinized more carefully and the flaw detected. (This account is simplified; for the full incident report see Ladkin[26]).

This view of requirements is due to Michael Jackson[23] and has been adopted by Praxis in its REVEAL requirements engineering method.[17] More specialized variants of the idea have appeared before, most notably in David Parnas's Four Variable Model.[32]

The dependability case. The evidence for dependability takes the form of a dependability case—an argument that the software, in concert with other components, establishes the critical properties. What exactly comprises the case—such as how detailed it should be and what mix of formal and informal arguments is appropriate —will vary between developments, but certain features are essential.

First, the case should be *auditable* so that it can be evaluated by a third-party certifier, independent both of developer and customer. The effort of checking that the case is sound should be much less than the effort of building the case in the first place. In this respect, a dependability case may be like a formal proof: hard to construct but easy to check. To evaluate a case, a certifier should not need any expert knowledge of the developed system or of the particular application, although it would be reasonable to assume expertise in software engineering and familiarity with the domain area.

Second, the case should be *complete*. This means that the argument that the critical properties apply should contain no holes to be filled by the certifier. Any assumptions that are not justified should be noted so that it is clear to the certifier who will be responsible for discharging them. For example, the dependability case may assume that a compiler generates code correctly; or that an operating system or middleware platform transports messages reliably, relying on representations by the producers of these components that they provide the required properties; or that users obey some protocol, relying on the organization that fields the system to train them appropriately. For a product that is not designed with a particular customer in mind, the assumptions become disclaimers, for example, that an infusion pump may fail under water or that a file synchronizer will work only if applications do not subvert file modification dates. Assumptions made to simplify the case, and that are no more easily substantiated by others, are suspect. Suppose an analysis of a program written in C, for example, contains an assumption that array accesses are within bounds. If this assumption cannot readily be checked, the results of the analysis cannot be trusted.

Third, the case should be *sound*. It should not, for example, claim full correctness of a procedure on the basis of nonexhaustive testing; or make unwarranted assumptions that certain components fail independently; or reason, in a program written in a language with a weak memory model that the value read from a shared variable is the value that was last written to it.

Implications

On the face of it, these recommendations—that developers express the critical properties and make an explicit argument that the system satisfies them—are hardly remarkable. If followed, however, they would have profound implications for how software is procured, developed, and certified.

Dependability case as product. In theory, one could construct a dependability case *ex post facto*, when the entire development had been completed. In practice, however, this would be near impossible and, in any case, undesirable. Constructing the case is easier and more effective if done hand-in-hand with other development activities, when the rationale for development decisions is fresh and readily available. But there is a far more important reason to consider the dependability case from the very outset of development. By focusing on the case, the developer can make decisions that ease its construction, most notably by designing the system so that critical properties are easier to establish. Decoupling and simplicity, discussed later, offer perhaps the greatest opportunities here.

This is the key respect in which the direct approach to dependability demands a sea change in attitude. Rather than just setting in place some practices or disciplines that are intended to improve dependability, the developers are called upon, every step of the way, to consider their decisions in the light of the system's dependability and to view the evidence that these decisions are sound as a work product that is as integral to the final system as the code itself.

Procurement. A change to a direct approach affects not only developers but also procurers, and the goals set at the start must be realistic in terms both of their achievement and demonstration.

The Federal Aviation Administration specified three seconds of downtime per year for the infamous Advanced Automation System for air-traffic control (which was ultimately canceled after an expenditure of several billion dollars), even though it would have taken 10 years just to obtain the data for substantiating such a requirement.[5] It was later revised to five minutes.

More fundamentally, however, our society as a whole needs to recognize that the enormous benefits of software inevitably bring risks and that functionality and dependability are inherently in conflict. If we want more dependable software, we will need to stop evaluating software on the basis of its feature set alone. At the same time, we should be more demanding, and less tolerant of poor-quality software. Too often, the users of software have been taught to blame themselves for its failures and to absorb the costs of workarounds.

After the failure of the USS *Yorktown*'s onboard computer system, in which the ship's entire control and navigation network went down after an officer calibrating a fuel valve entered a zero into a database application (in an attempt to overwrite a bad value that the system had produced), blame was initially placed on software. After

an investigation, however, the Navy cited human error. The ship's commanding officer reported that 'Managers are now aware of the problem of entering zero into database fields and are trained to bypass a bad data field and change the value if such a problem were to occur again'.

With the indirect approach to certification, it has been traditional that procurers give detailed prescriptions for how the software should and should not be developed and which technologies should be used. By contrast, the direct approach frees the developer to use the best available means to achieve the desired goal; constraints on the development are evaluated by the objective measure of whether they improve dependability or not.

Structuring requirements. How requirements are approached sets the tone of the development that follows. A cursory nod to analyzing the problem can result in functionality so unrelated to the users' needs that the developers become mired in endless cycles of refactoring and modification. On the other hand, a massive and windy document can overwhelm the designers with irrelevant details and tie their hands with premature design decisions. Ironically, what a requirements document says can do as much damage as what it fails to say.

In the context of dependability, the approach to requirements is especially important. The standard criteria of course apply: that the developers listen carefully to the stakeholders to understand not merely the functions they say they want but also, more deeply, the purposes they believe these functions will enable them to accomplish; that the requirements be expressed precisely and succinctly; and that great care be taken to avoid making irrevocable decisions when they could be postponed and made later on the basis of much fuller information.

Two other criteria take on greater significance, however. Deciding which requirements are critical (and how critical they are) is the first and most vital design step, determining in large part the cost of the system and the contexts in which it will usable. The architecture of the system will likely be based on these requirements, because (as explained later) the most feasible way to offer high dependability at reasonable cost is to exploit modularity to establish critical properties locally.

Second, it is important to clearly record any assumptions made about the software's operating environment (including the behavior of human operators). These assumptions will be an integral part of the dependability case, influence the design, and become criteria for evaluating the contexts in which the software can be deployed.

These two criteria are hardly new, but they are not always followed. Perhaps under pressure from customers to provide an extensive catalog of features, analysts often express requirements as a long list of functions. In a radiotherapy application, for example, the analyst—aware of the risk of delivering incorrect doses or of unauthorized access—might include sections describing the various functions or use cases associated with selecting doses or logging in but might neglect the more important task of describing the most critical properties explicitly—such as that the delivered dose corresponds to the prescribed dose and that access be restricted to certain staff.

Sometimes developers appreciate the value of prioritization but have been shy to make the bold decisions necessary for downgrading (or eliminating) noncritical requirements. Rather than asking "What are the critical properties?" we might instead ask "What properties are *not* critical?" If we have trouble answering this latter question, the properties that *are* critical have probably not been identified correctly.

Decoupling and simplicity. How much the cost of developing a system will increase if a particular critical property is to be assured depends on how much of the system is involved. If the critical property is not localized, the entire codebase must be examined to determine whether or not it holds—in essence, all of the code becomes critical. But if the property is *localized* to a single component, attention can be focused on that component alone, and the rest of the codebase can be treated as noncritical. Put another way, the cost of making a system dependable should vary not with the size of the whole system but with the extent and complexity of the critical properties.

Decoupling is the key to achieving locality. Two components are decoupled if the behavior of one is unaffected by that of the other. Maximizing decoupling is a guiding principle of software design in general, but it is fundamental to dependability. It is addressed first during requirements analysis by defining functions and services so that they are self-contained and independent of one another. Then, during design, decoupling is addressed by allocating functionality to particular components, which allows key invariants to be localized and the minimization of communication; and by crafting interfaces that do not expose the internals of a service to its clients and do not connect clients to each other unnecessarily. Finally, the decoupling introduced in the design must be realized in the code by using appropriate language features to protect against errors that might compromise it.

The design of an e-commerce system, for example, might enforce a rigorous separation between billing and other subsystems; in that way, more complex (and less dependable) code can be written for less critical features (such as product search) without compromising essential financial properties.

Decoupling is an important way of securing simplicity, and its benefits, in system design. As Tony Hoare famously said in his Turing Award lecture (discussing the design of Ada): "[T]here are two ways of constructing a software design: One way is to make it so simple there are obviously no deficiencies; and the other way is to make it so complicated that there are no obvious deficiencies." Many practitioners are resistant to the claim that simplicity is possible (and some even to the claim that it is desirable). They tend to think that the advocates of simplicity do not recognize the inherent complexity of the problems solved by computer systems or that they imagine that simplicity is easily achieved.

Simplicity is *not* easy to achieve, and, as Alan Perlis noted in one of his famous aphorisms, it tends to follow complexity rather than precede it. The designer of a system that will work in a complicated and critical domain faces difficult problems. The question is not whether complexity can be eliminated but whether it can be tamed so that the

resulting system is as simple as possible under the circumstances. The cost of simplicity may be high, but the cost of lowering the floodgates to complexity is higher. Edsger Dijkstra explained: "The opportunity for simplification is very encouraging, because in all examples that come to mind the simple and elegant systems tend to be easier and faster to design and get right, more efficient in execution, and much more reliable than the contrived contraptions that have to be debugged into some degree of acceptability."[8]

Process and culture. While process may not be sufficient for dependability, it is certainly necessary. A rigorous process will be needed to ensure that attention is paid to the dependability case and to preserve the chain of evidence as it is constructed. In the extreme, if there is no credible process, a certifier has no reason to believe that the deployed software even corresponds to the software that was certified. For example, in a well-known incident in 2003 an electronic voting system was certified for use in an election but a different version of the system was installed in voting booths.[39]

A rigorous process need not be a burdensome one. Because every engineer involved in a project is expected to be familiar with the process, it should be described by a brief and easily understood handbook, tailored if necessary to that project. Rather than interfering with the technical work, the process should eliminate a mass of small decisions, thereby freeing the engineers to concentrate on the more creative aspects of the project. Standards for machine-processable artifacts—especially code—should be designed to maximize opportunities for automation. For example, a coding standard that mandates how variables are named and how specification comments are laid out can make it possible to extract all kinds of cross-referencing and summarization information with lexical tools alone. Bill Griswold has observed that the more the programmer embeds semantic information in the naming conventions of the code, the more readily it can be exploited.[15]

One of the paradoxes of software certification is that burdensome processes (such as requiring MCDC test coverage) do seem to be correlated with more dependable software, even though there is little compelling evidence that the processes themselves achieve their stated aims. This may be explained by a social effect. The companies that adhere to the strictest processes tend to attract and reward employees who are meticulous and risk-averse. Having a strong "safety culture" can be the major factor in determining how safe the products are, and in fact one study of formal methods found that their success may be due more to the culture surrounding them than to anything more direct.[35] Efforts to build and maintain a strong safety culture can pay dividends. Richard Feynman, in his dissenting report following the *Challenger* inquiry,[10] was effusive in his praise of the constructively adversarial attitude among NASA's software engineers, to which he ascribed the high dependability of their software.

The question is not whether complexity can be eliminated but whether it can be tamed so that the resulting system is as simple as possible under the circumstances. The cost of simplicity may be high, but the cost of lowering the floodgates to complexity is higher.

Advances in software-verification technology may tempt us to imagine (in a Leibnizian fantasy) that one day we will be able to check the dependability of software simply by running it through a machine. But dependability cases will always contain informal elements that cannot be verified mechanistically; truth will have to be assessed by an impartial review not only of the case itself but also of the credibility of the organization that produced it. An entirely product-based certification approach thus makes no more sense than one based entirely on process. Given that the organization that produces the software and the software itself are intertwined, attempts at improving dependability, and efforts to measure it, must take both into account.

Robust foundations. Just as a skyscraper cannot easily be built on sand, a robust software system cannot be built on a foundation of weak tools and platforms. Fifty years after the invention of static typing and automatic memory management, the decision to use an unsafe programming language such as C or C++ (which provide neither) requires serious justification, and for a critical system the benefits that are obtained in compensation for the loss of safety have to be extraordinarily compelling. Arguments against safety based on performance are usually overstated. And with Java and C# now widely known and available, and equipped

with impressive libraries, there is no longer a reason to consider safe languages as boutique technologies.

The value of static typing is often misunderstood. It is not that type errors don't occur during execution. They do, because most statically typed languages are sufficiently complex that some checks are inevitably postponed to runtime. Similarly, the value of strong typing is not that runtime type errors are more acceptable than other kinds of failure. An unanticipated failure is never good news. Moreover, runtime type checking can make things worse: the Ariane 5 rocket might not have been lost had an arithmetic overflow in an irrelevant module not been propagated to the top level.

Strong typing has two primary benefits. First, it prevents a module from writing to regions of memory that it cannot name (through local and global variables and sequences of field accesses). This means that a syntactic dependence analysis can determine the potential couplings between modules and can be used to establish that one module is decoupled from another, thus making it possible to ignore the latter when analyzing the behavior of the former. Second, strong typing makes runtime failures happen earlier, as soon as a type error occurs, rather than later when the failure is likely to be harder to diagnose and may have done more damage. Static typing provides the important additional advantage of catching many type errors at compile time. This is extremely valuable, because type errors are often symptoms of serious mistakes and structural flaws.

Complexity in a programming language can compromise dependability because it increases the chance that the program will behave differently from what the programmer envisaged. It is important to avoid obscure mechanisms, especially those that have a platform-dependent interpretation. Coding standards can be very helpful in taming dangerous language features.[18,19]

Progress in language design has produced major improvements, but old lessons are easily forgotten. The original design of Java, for example, lacked iterators (as a control construct) and generics, and it did not unify primitive types and objects, even though these features had been part of CLU in 1975.[27] When it was realized that these features were essential, it was too late to incorporate them cleanly. It may be curmudgeonly to complain about Java, especially because it brings so many good ideas to mainstream programming—in particular, strong typing—that might otherwise have languished in obscurity. And, to be fair, Java incorporates features of older languages in a more complex setting; subtyping, in particular, makes it much harder to incorporate other features (such as generics). Nevertheless, it does seem sad that the languages adopted by industry often lack the robustness and clarity of their academic predecessors.

It is important to realize that arguments that are not mechanically checked are likely to be flawed, so their credibility must suffer and confidence in any dependability claims that rely on them must be reduced accordingly.

It is important to recognize that dependability was not the primary goal in the design of most programming languages. Java was designed for platform independence, and its virtual machine includes a class loader that absorbs much of the complexity of installation variability. As a result, however, a simple call to a constructor in the source code sets in motion a formidable amount of machinery that could compromise the system's dependability.

The choice of computing platform—such as operating system, middleware, and database—must also be carefully considered. A platform that has been widely adopted for general applications usually has the advantage of lower cost and a larger pool of candidate developers. But commodity platforms are not usually designed for critical applications. Thus when high dependability is required, enthusiasm for their use should be tempered by the risks involved.

The (in)significance of code. A dependability argument is a chain with a variety of links. One link may argue that a software component has some property, another that a peripheral behaves in a certain way, yet another that a human operator obeys some protocol, and together they might establish the end-to-end dependability requirement. But the overall argument is only as strong as the chain's weakest link.

Many software engineers and researchers are surprised to learn that the correctness of the code is rarely that weakest link. In an analysis of fatal accidents that were attributed to software problems, Donald MacKenzie found that coding errors were cited as

causes only 3% of the time.[29] Problems with requirements and usability dwarf the problems of bugs in code, suggesting that the emphasis on coding practices and tools, both in academia and industry, may be mistaken. Exploiting tools to check arguments at the design and requirements level may be more important; and it is often more feasible, as artifacts at the higher level are much smaller.[21]

Nevertheless, the correctness of code is a vital link in the dependability chain. Even if the low incidence of failures due to bugs reflects success in improving code quality, the cost is still unacceptable,[11] especially when very high assurance is required. Note too that in the arena of security, code vulnerabilities are responsible for a much higher proportion of failures than in the safety arena.

Testing and analysis. Testing is a crucial part of any software-development process, and its effectiveness can be amplified by liberal use of runtime assertions, by formulating tests early on, by creating tests in response to bug reports, and by integrating testing into the build so that tests are run frequently and automatically. But as discussed above, testing cannot generally deliver the high levels of confidence that are required for critical systems. Thus analysis is needed to fill the gap.

Analysis might involve any of a variety of techniques, depending on the kind of property being checked and the level of confidence required. In the last decade, dramatic advances have been made in analyses that establish properties of code fully automatically through the use of theorem proving, static analysis, model checking, and model finding.

How well these techniques will work and how widely they will be adopted remain to be seen. But a number of industrial successes demonstrate that the approaches are at least feasible and, in the right context, effective. Microsoft, for example, now includes a sophisticated verification component in its driver development toolkit;[3] Praxis has achieved extraordinarily low defect rates using a variety of formal methods;[16] and Airbus has used static analysis to show the absence of low-level runtime errors in the A340 and A380 flight-control software.[7]

Until these approaches are more widely adopted, many development teams will choose to rely instead on manual code review. In any case, it is important to realize that arguments that are not mechanically checked are likely to be flawed, so their credibility must suffer and confidence in any dependability claims that rely on them must be reduced accordingly.

The credibility of tools. Tools are enormously valuable, but the glamour of automation can sometimes overwhelm our better judgment. A symptom of this is our tendency to invest terms used to describe tools with more significance than their simple meaning. For example, inventors of program analyses have long classified their creations as "sound" or "unsound." A sound analysis establishes a property with perfect reliability. That is, if the analysis does not report a bug, then there is no possible execution that can violate the property. This notion helpfully distinguishes verifiers from bug finders—a class of tools that are very effective at catching defects, especially in low-quality software, but that usually cannot contribute evidence of dependability because they tend to be heuristic and therefore unsound.

But the assumption that sound tools are inherently more credible is dangerous. Alex Aiken found that an unsound tool uncovered errors in a codebase that a prior analysis, using a sound tool, had failed to catch. The much higher volume of false alarms produced by the sound tool overwhelmed its users and made the real defects harder to identify.[1] In recent years, developers of analysis tools have come to realize that the inclusion of false positives is just as problematic as the exclusion of true positives and that more sophisticated measures are needed.

Even if an analysis establishes a property with complete assurance, the question of whether the property itself is sufficient still remains. For example, eliminating arithmetic overflows and array bounds errors from a program is certainly progress. But knowing that such faults are absent may not help the dependability case unless there is either: a chain of reasoning connecting this knowledge to assertions about end-to-end properties; or some strong statistical evidence that the absence of these faults is correlated with the absence of other faults.

Among analysis tools, mathematical proof is generally believed to offer the highest level of confidence. An analysis substantiated with a proof can be certified independently by examining the proof in isolation, thereby mitigating the concern that the tool that produced the proof might have been faulty.

Proof is not foolproof, however. When a bug was reported in his own code (part of the Sun Java library), Joshua Bloch found[4] that the binary search algorithm—proved correct many years before (by, amongst others Jon Bentley in his *Communications* column) and upon which a generation of programmers had relied—harbored a subtle flaw. The problem arose when the sum of the low and high bounds exceeded the largest representable integer. Of course, the proof wasn't wrong in a technical sense; there was an assumption that no integer overflow would occur (which was reasonable when Bentley wrote his column, given that computer memories back then were not large enough to hold such a large array). In practice, however, such assumptions will always pose a risk, as they are often hidden in the very tools we use to reason about systems and we may not be aware of them until they are exposed.

Closing Thoughts

The central message of this article is that it is not rational to believe that a software system is dependable without good reason. Thus any approach that promises to develop dependable software must provide such reason. A clear and explicit articulation is needed of what "dependable" means for the system at hand, and an argument must be made that takes into account not only the correctness of the code but also the behavior of all the other components of the system, including human operators.

Is this approach practical? The cost of constructing a dependability case, after all, may be high. On the other hand, such construction should focus resources, from the very start of the development, where they bring the greatest return, and the effort invested in obtaining a decoupled design may reduce the cost of maintenance later. The experience of Praxis shows that many of the approaches that the indus-

try regards as too costly (such as formal specification and static analysis of code) can actually reduce overall cost.[2]

Similarly, even though the augmentation of testing with more ambitious analysis tools will require greater expertise than is available to many teams today, this avenue does not necessarily increase the cost either. When low levels of confidence suffice, testing may be the most cost-effective way to establish dependability. As the required level of confidence rises, though, testing soon becomes prohibitively expensive, and the use of more sophisticated methods is likely to be more economical. Invariants may be harder to write than test cases, but a single invariant defines an infinite number of test cases, so a decision to write one (and use a tool that checks all the cases it defines) will pay off very soon.

Efforts to make software more dependable or secure are inherently conservative and therefore risk retarding progress, and many practitioners understandably see certification schemes and standards as millstones around their necks. But because a direct approach based on dependability cases gives developers an incentive to use whatever development methods and tools are most economic and effective, the approach therefore rewards innovation.

Acknowledgments

The key ideas in this article come from a National Academies study[22] that I chaired. I am very grateful to the members of my committee—Joshua Bloch, Michael DeWalt, Reed Gardner, Peter Lee, Steven Lipner, Charles Perrow, Jon Pincus, John Rushby, Lui Sha, Martyn Thomas, Scott Wallsten, and David Woods; to our study director Lynnette Millett; to Jon Eisenberg, director of the Academies' Computer Science and Telecommunications Board; and to our sponsors, especially Helen Gill, who was instrumental in making the case for the study. John Rushby and Martyn Thomas deserve recognition for having been long and eloquent advocates of the direct approach. Many of the opinions expressed in this article, however, are my own and have not been approved by the committee or by the Academies.

Thanks too to Butler Lampson, Shari Lawrence Pfleeger, and Derek Rayside, who gave extensive and helpful suggestions on an initial draft of the article; to the anonymous reviewers; and to Hari Balakrishnan, Bill Maisel and Andrew Myers who gave valuable feedback and shared their expertise on particular topics.

A version of this article with a fuller list of references is available at http://sdg.csail.mit.edu/publications.html.

References

1. Aiken, A. and Xie, Y. Context- and path-sensitive memory leak detection. *Proceedings of the 5th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering* (Sept. 2005).
2. Amey, P. Correctness by construction: Better can also be cheaper. *CrossTalk*: The Journal of Defense Software Engineering (Mar. 2002); www.praxis-his.com/pdfs/c_by_c_better_cheaper.pdf.
3. Ball, T. and Rajamani, S. The SLAM project: Debugging system software via static analysis. In *Proceedings of the 29th ACM Symposium on Principles of Programming Languages* (Portland, Oregon, Jan. 16–18), 2002.
4. Bloch, J. Extra, extra—read all about it: Nearly all binary searches and mergesorts are broken; googleresearch.blogspot.com/2006/06/extra-extra-read-all-about-it-nearly.html.
5. Cone, E. The ugly history of tool development at the FAA. *Baseline Magazine 4*, 9 (Apr. 8, 2002).
6. Cook, R. and O'Connor, M. Thinking about accidents and systems. In *Medication Safety: A Guide to Health Care Facilities*, H.R. Manasse and K.K. Thompson, Eds. American Society of Health-System Pharmacists, Washington, DC, 2005; www.ctlab.org/documents/ASHP_chapter.pdf.
7. Cousot, P. Proving the absence of run-time errors in safety-critical avionics code. In *Proceedings of the Seventh ACM & IEEE International Conference on Embedded Software*. (Salzburg, Austria, Sept. 30–Oct. 3), ACM Press, New York, 2007.
8. Dijkstra, E.W. The tide, not the waves. In *Beyond Calculation: The Next Fifty Years of Computing*, Denning, P. and Metcalfe, R., Eds. Copernicus (Springer-Verlag), 1997.
9. FDA. *Ensuring the safety of marketed medical devices: CDRH's medical device post-market safety program*, 2006.
10. Feynman, R.P. Appendix F: Personal observations on the reliability of the shuttle. In *Report of the Presidential Commission on the Space Shuttle Challenger Accident*, 1986; science.ksc.nasa.gov/shuttle/missions/51-l/docs/rogers-commission/Appendix-F.txt.
11. Gallaher, M. and Kropp, B. *Economic Impacts of Inadequate Infrastructure for Software Testing*, National Institute of Standards and Technology, 2002.
12. GAO. *Medical Devices: Early Warning of Problems Is Hampered by Severe Under-reporting.* Publication PEMD-87-1, U.S. Government Printing Office, 1986.
13. Geppert, L. Lost radio contact leaves pilots on their own. *IEEE Spectrum 41*, 11 (Nov. 2004); www.spectrum.ieee.org/nov04/4015.
14. German, A. and Mooney, G. Air vehicle software static code analysis—Lessons learnt. In *Proceedings of the Ninth Safety-Critical Systems Symposium*, F. Redmill and T. Anderson, Eds. Springer-Verlag, Bristol, U.K., 2001.
15. Griswold, W. Coping with crosscutting software changes using information transparency. In *Reflection 2001: The Third International Conference on Metalevel Architectures and Separation of Crosscutting Concerns* (Kyoto, Sept. 25–28, 2001).
16. Hall, A. Using formal methods to develop an ATC information system. *IEEE Software 13*, 2 (Mar. 1996).
17. Hammond, J., Rawlings, R., and Hall, A. Will it work? In *Proceedings of the 5th International Symposium on Requirements Engineering* (Toronto, Aug. 27–31, 2001).
18. Hatton, L. and Safer C. *Developing Software in High-Integrity and Safety-Critical Systems.* McGraw-Hill, 1995.
19. Holzmann, G. The power of ten: Rules for developing safety critical code. *IEEE Computer 39*, 6, (June 2006).
20. IAEA. *Investigation of an Accidental Exposure of Radiotherapy Patients in Panama: Report of a Team of Experts.* (Vienna, Austria, May 26–June 1, 2001); www-pub.iaea.org/MTCD/publications/PDF/Pub1114_scr.pdf.
21. Jackson, D. Dependable software by design. *Scientific American* (June 2006); www.sciam.com/article.cfm?id=dependable-software-by-de&colID=1.
22. Jackson, D., Thomas, M., and Millett, L., Eds. *Software For Dependable Systems: Sufficient Evidence?* National Research Council. National Academies Press, 2007; books.nap.edu/openbook.php?isbn=0309103940.
23. Jackson, M. Problem Frames: Analysing and Structuring Software Development Problems. Addison-Wesley, 2001.
24. Gross, G. E-voting vendor: Programming errors caused dropped votes. *Network World* (Aug. 22, 2008); www.networkworld.com/news/2008/082208-e-voting-vendor-programming-errors-caused.html.
25. Krebs, B. Cyber incident blamed for nuclear power plant shutdown. *Washington Post* (June 5, 2008); www.washingtonpost.com/wp-dyn/content/article/2008/06/05/AR2008060501958_pf.html.
26. Ladkin, P.,Transcriber. *Transcription of Report on the Accident of Airbus A320-211 Aircraft in Warsaw on Sept. 14, 1993.* Main Commission Aircraft Accident Investigation Warsaw; www.rvs.uni-bielefeld.de/publications/Incidents/DOCS/ComAndRep/Warsaw/warsaw-report.html.
27. Liskov, B. A history of CLU. *ACM SIGPLAN Notices 28*, 3 (Mar. 1993).
28. Littlewood, B. and Wright, D. Some conservative stopping rules for the operational testing of safety-critical software. *IEEE Transactions on Software Engineering 23*, 11 (Nov. 1997).
29. MacKenzie, D. *Mechanizing Proof: Computing, Risk, and Trust*. MIT Press, 2001.
30. Maisel, W., Sweeney, M., Stevenson, W., Ellison, K., and Epstein, L. Recalls and safety alerts involving pacemakers and implantable cardioverter-defibrillator generators. *Journal of the American Medical Association 286*, 7 (Aug. 15, 2001).
31. Ministry of Defence. *Defence Standard 00-42: Reliability And Maintainability Assurance Guides, Part 2: Software*, 1997.
32. Parnas, D. and Madey, J. Functional documentation for computer systems. *Science of Computer Programming 25*, 1 (1995).
33. Perrow, C. *Normal Accidents*, Princeton University Press, 1999.
34. Perrow, C. *The Next Catastrophe: Reducing our Vulnerabilities to Natural, Industrial, and Terrorist Disasters.* Princeton University Press, 2004.
35. Pfleeger, S. and Hatton, L. Investigating the influence of formal methods. *Computer 30*, 2 (Feb. 1997).
36. Rockoff, J. Flaws in medical coding can kill: Spread of computers creates new dangers, FDA officials warn. *Baltimore Sun* (June 30, 2008); http://pqasb.pqarchiver.com/baltsun/access/1502776681.html?dids=1502776681:1502776681&FMT=ABS&FMTS=ABS:FT&type=current&date=Jun+30%2C+2008&author=Jonathan+D.+Rockoff&pub=The+Sun&desc=FLAWS+IN+MEDICAL+CODING+CAN+KILL.
37. Salvadori, M. *Why Buildings Stand Up: The Strength of Architecture.* Norton, 1980. See also Levy, M. and Salvadori, M. *Why Buildings Fall Down: How Structures Fail.* Norton, 1992.
38. Slabodkin, G. Navy: Calibration flaw crashed Yorktown LAN. *Government Computing News* (Nov. 9, 1998); www.gcn.com/print/17_30/33914-1.html.
39. Zetter, K. E-voting undermined by sloppiness. *Wired* (December 17, 2003); www.wired.com/politics/security/news/2003/12/61637.

Daniel Jackson (dnj@mit.edu) is a professor of computer science at the Massachusetts Institute of Technology and a principal investigator at MIT's Computer Science and Artificial Intelligence Lab, Cambridge, MA.

research highlights

DOI:10.1145/1498765.1498788

Technical Perspective
Disk Array Models for Automating Storage Management

By Arif Merchant

LARGE DISK ARRAYS are everywhere, even if we, as end users of computer services, rarely notice them. When we shop at an Internet retailer, the product and account data come from a disk array in a data center. Our email, banking, payroll, insurance, and tax data all reside on disk arrays. The hardware used is typically diverse, obtained from multiple vendors at different times. Depending upon an application's requirements for throughput, response time, availability, and reliability, its data may be distributed across disk arrays and even across multiple data centers and made resilient to failures through replication or the addition of error correction codes.

Managing disk array storage is extremely complex, involving tasks such as initially placing the data, arranging for data backup, prioritizing data access so that each application can receive the performance it requires, periodically migrating data from one disk array to another, monitoring performance, and diagnosing any performance problems found. Some partially automated tools can assist the operator, but in the end, storage management is a manually intensive process. As a result, management decisions are geared toward simplifying implementation, rather than optimizing application performance.

The key to reducing costs and improving the performance and dependability of storage is to automate the management tasks because computers can keep track of complex environments and intricate decisions better than human beings. However, the management system must understand the behavior of the storage system it manages. For example, when a new database server is to be installed, where should its data be placed? Would one of the existing disk arrays in the data center suffice, or is a new one needed? An automated management system can consider numerous options to make an informed decision, but it must be able to predict the performance impact of each option. In other words, the management system needs a model to answer the question "How will my applications' storage performance change if I take this option?"

Building accurate performance models of storage systems has long been a stumbling block to designing automated storage management systems, because one needs to be able to build models, quickly and easily, for the multitude of disk arrays in use, and for a wide variety of workloads. While models of basic disk drives for simple workloads are known, most data centers use disk arrays, which are much more complex because they aggregate a number of disks with cache and control firmware. Earlier disk array performance models either were hand-built for each disk array model and required extensive tuning for good accuracy, or were based on benchmark measurements of a few workloads on the device. In either case, the models were only accurate for workloads similar to those used to build the models.

To address this problem, Michael Mesnier, Matthew Wachs, Raja Sambasivan, Alice Zheng, and Gregory Ganger have proposed a new approach, called *relative fitness modeling*. Rather than directly building a performance model for each disk array, the authors suggest it is easier to characterize the difference in performance between disk arrays. These models, built by measuring the differences in performance between a given pair of arrays for a representative set of workloads, are shown empirically to apply to a larger set of workloads. Then, if we have a relative fitness model for the differences between two arrays, and we know how a given workload performs on the first array, we can predict the performance of the workload on the second. This scenario is common. For example, a user may have measured the average I/O response time of an application on an existing array; if the disk array vendor can provide a relative fitness model of the differences between the user's existing disk array and a newer one, the I/O response time of the application on the new array can be predicted.

The relative fitness method is an important step in modeling the performance of disk arrays, but many challenges persist. In particular, disk array models must be able to predict accurately the performance of an arbitrary combination of workloads, given the increasing trend of storage consolidation in the data center (that is, storing multiple application data sets on the same disk array), and this problem remains open. The success of the relative fitness method gives us hope that similar techniques can be used to predict the performance of workload combinations; this is an active area of work for storage systems researchers. C

Reference

1. Wilkes, J. Data services – from data to containers (Invited talk). In *Proceedings of the FAST '03 Conference on File and Storage Technologies*. (San Francisco, CA, Mar.-Apr. 2003); www.usenix.org/events/fast03/tech/fast03_keynote.pdf.

Arif Merchant (arif_merchant@hp.com) is a Principal Research Scientist at HP Labs, Palo Alto, CA.

DOI:10.1145/1498765.1498789

Relative Fitness Modeling

By Michael P. Mesnier, Matthew Wachs, Raja R. Sambasivan, Alice X. Zheng, and Gregory R. Ganger

Abstract

Relative fitness is a new approach to modeling the performance of storage devices (e.g., disks and RAID arrays). In contrast to a conventional model, which predicts the performance of an application's I/O on a given device, a relative fitness model predicts performance *differences* between devices. The result is significantly more accurate predictions.

1. INTRODUCTION

Relative fitness: the fitness of a genotype compared with another in the same gene system.

Managing storage within a data center can be surprisingly complex and costly. Large data centers have numerous storage devices of varying capability, and one must decide which application data sets (e.g., database tables, web server content) to store on which devices. Sadly, the state-of-the-art in Information Technology (IT) requires much of this to be done manually. At best, this results in an overworked system administrator. However, it can also lead to suboptimal performance and wasted resources.

Many researchers believe that automated storage management[2, 5] is one way to offer some relief to administrators. In particular, application workloads can be automatically assigned to storage devices. Doing so requires accurate predictions as to how a workload will perform on a given device, and a *model* of a storage device can be used to make these predictions. Specifically, one trains a model to predict the performance of a device as a function of the I/O characteristics of a given workload.[1, 7, 11, 13] Common I/O characteristics include an application's read/write ratio, I/O pattern (random or sequential), and I/O request size.

Though it sounds simple, such modeling has not been realized in practice, primarily because of the difficulty of obtaining workload characteristics that are good predictors of performance, yet also suitable for use in a model. For example, the I/O request size of an application is often approximated with an average, as opposed to the actual distribution (e.g., bimodal). Although such approximations reduce modeling complexity, they can lead to inaccurate predictions.

This article describes a new modeling approach called *relative fitness* modeling.[9, 10] A relative fitness model uses observations (performance and resource utilization) from one storage device to predict the performance of another, thereby reducing the dependence on workload characteristics. Figure 1 illustrates relative fitness modeling for two hypothetical devices A and B.

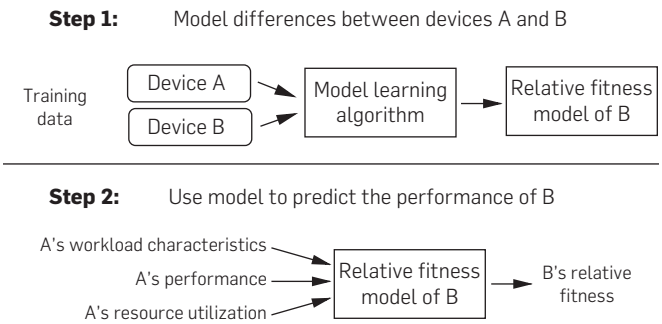
The insight behind relative fitness modeling is best obtained through analogy. When predicting your grade in a college course (a useful prediction during enrollment), it is helpful to know the grade received by a peer (his performance) and the number of hours he worked each week to achieve that grade (his resource utilization). Naturally, our own performance for a certain task is a complex function of the characteristics of the task and our ability. However, we have learned to make predictions relative to the experiences of others with similar abilities, because it is easier.

Applying the analogy, two storage devices may behave similarly enough to be reasonable predictors for each other. For example, they may have similar RAID levels, caching algorithms, or hardware platforms. As such, their performance may be related. Even dissimilar devices may be related in some ways (e.g., for a given workload type, one usually performs well and the other poorly). The objective of relative fitness modeling is to learn such relationships.

2. BACKGROUND

Storage performance modeling is a heavily researched area, including analytical models,[11] statistical or probabilistic models,[1, 7] and machine learning models.[10, 13] Models are either *white-box* or *black-box*. White-box models use knowledge of the internals of a storage device (e.g., drives, controllers, and caches), and black-box models do not. Given the complexity of modern-day storage devices,[12] black-box approaches are becoming increasingly attractive.

Figure 1: Using sample workloads, a model learns to predict how the performance of a workload changes between two devices (A and B). To predict the performance of a new workload on B, the workload characteristics, performance, and resource utilization (as measure on device A) are input into the model of B. The prediction is a performance scaling factor, which we refer to as B's "relative fitness."



A previous version of this research paper was published in the *Proceedings of the International Conference on Measurement and Modeling of Computer Systems* (San Diego, CA, June 2007), ACM, NY.

Perhaps the simplest of all black-box models is a numeric average. For example, the fuel efficiency of a car (average miles per gallon) and a soccer player's performance (average goals per game) are both black-box models. Of course, such models can be easily extended with workload characteristics (e.g., highway or city, home game or away), and an average can be maintained for each type of workload.

Table 1 shows a simple black-box model of a storage device (a table of performance averages), and Figure 2 shows the same information in a regression tree.[3] Both models are indexed using one workload characteristic (the average request size of the I/O that is issued to the storage device by the application), and both models must be *trained* with sample workloads in order to learn performance averages for various request sizes. Some form of interpolation is required when an exact match is not found in the model. For example, to predict the performance of a workload with 3KB requests, using Table 1, one might average the 2 and 4KB performance and predict 37MB/s. Of course, storage researchers have explored a number of workload characteristics in addition to request size, including the read/write ratio, multiprogramming level (queue depth), I/O inter-arrival delay, and spatial locality. More complex characteristics (e.g., I/O burstiness, spatio-temporal correlation) have also been investigated.

More formally, a model of a storage device i (white-box or black-box) can be expressed as a function F_i. During training, the inputs are the workload characteristics $\mathbf{WC}_i$ of an application running on device i and the output is a performance metric P_i (bandwidth, throughput, or latency):

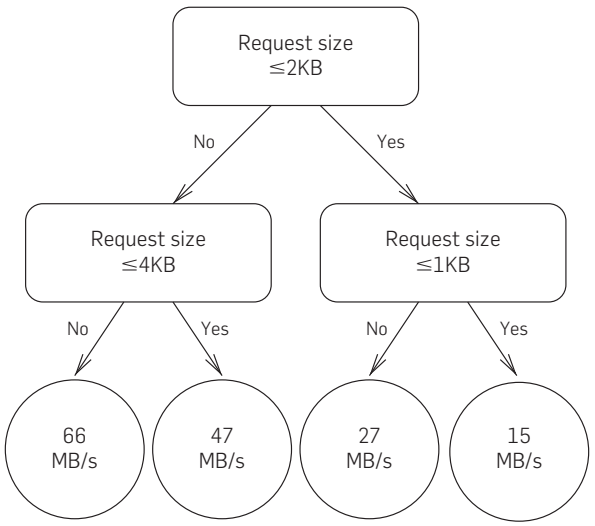
$$\mathrm{P}_i = F_i(\mathbf{WC}_i). \quad (1)$$

We refer to Equation 1 as an *absolute model*, to signify that the inputs $\mathbf{WC}_i$ are absolute, and not relative to some other device. However, in practice, one does not possess $\mathbf{WC}_i$, as this would require running the workload on device i in order to obtain them. Because running the workload to obtain $\mathbf{WC}_i$ obviates the need for predicting the performance of device i, one instead uses the characteristics $\mathbf{WC}_j$ obtained from some other storage device j. That is, the model assumes that the characteristics of a workload are static and will not differ across storage devices. More precisely, the model assumes that $\mathbf{WC}_i$ and $\mathbf{WC}_j$ are equivalent. However, this is not always a safe assumption.

Table 1: A table-based model that records the performance of a disk drive for sequentially-read data.

Request Size	Bandwidth
1KB	15MB/s
2KB	27MB/s
4KB	47MB/s
8KB	66MB/s

Figure 2: A regression tree that learns the performance of a disk drive for sequentially read data.



2.1. The challenges with absolute models

The primary challenges with absolute models relate to workload characterization, which has been an open problem for decades.[4] First, one must discover the performance-affecting characteristics of a workload. This can be challenging given the heterogeneity of storage devices.[8] For example, a storage array with a large cache may be less sensitive to the spatial access pattern than an array with little cache, so models of the devices would likely focus on different workload characteristics when predicting performance.

Second, one must manage the trade-off between expressiveness and conciseness. Most models expect numbers as input, and it can be challenging to describe complex workloads with just a few numbers. In effect, workload characterization compresses the I/O stream to just a few distinguishing features. The challenge is to compress the stream without losing too much information.

Third, and more fundamentally, an absolute model does not capture the connection between a workload and the storage device on which it executes. While the assumption of static workload characteristics (i.e., $\mathbf{WC}_i = \mathbf{WC}_j$) is safe for *open* workloads, where the workload characteristics are independent of the I/O service time, it is not safe for *closed* workloads. The most obvious change for a closed workload is the I/O arrival rate: if a storage device completes the I/O faster, then an application is likely to issue I/O faster. And other characteristics can change, such as the average request size, access pattern, read/write ratio, and queue depth. Such effects occur when file systems, page caches, and other OS middleware reside between an application and the storage device. Although the application may issue the same I/O, the characteristics of the I/O as seen by the storage device could change due to write reordering, aggregation and coalescing, caching, prefetching, and other interactions between an operating system and a storage device. For example, a slower device can result in a workload with larger inter-arrival times and larger write requests (due to request coalescing) when compared to a faster device.

Collectively, these challenges motivate the work presented in this article. Rather than attempt to solve the difficult problem of identifying workload characteristics that are expressive, yet concise and static across devices, we choose to use performance and resource utilization. That is, we use

the performance and utilization of device j to predict the performance of a different device i. Of course, such *relative* models must be built between each pair of devices, as performance and resource utilization are device-specific.

3. RELATIVE FITNESS MODELING

Relative fitness begins with an absolute model (Equation 1). Recall that a workload is running on device j, $\mathbf{WC}_j$ can be measured on device j, and we want to predict the performance of moving the workload to a different device i. The first objective of relative fitness is to capture the changes in workload characteristics from device j to i, that is, to predict $\mathbf{WC}_i$ given $\mathbf{WC}_j$. Such change is dependent on the devices, so we define a function $G_{j \to i}$ that predicts the workload characteristics of device i given j:

$$\mathbf{WC}_i = G_{j \to i}(\mathbf{WC}_j).$$

We can now apply G in the context of an absolute model F_i:

$$\mathrm{P}_i = F_i(G_{j \to i}(\mathbf{WC}_j)).$$

However, rather than learn two functions, the composition of F and G can be expressed as a single composite function $RM_{j \to i}$ which we call a *relative model*:

$$\mathrm{P}_i = RM_{j \to i}(\mathbf{WC}_j). \quad (2)$$

With each model now involving an origin j and target i, we can use the performance of device j ($\mathbf{Perf}_j$) and its resource utilization ($\mathbf{Util}_j$) to help predict the performance of device i. $\mathbf{Perf}_j$ is a vector of performance metrics such as bandwidth, throughput, and latency. $\mathbf{Util}_j$ is a vector of values such as the device's cache utilization, the hit/miss ratio, its network bandwidth, and its CPU utilization:

$$\mathrm{P}_i = RM_{j \to i}(\mathbf{WC}_j, \mathbf{Perf}_j, \mathbf{Util}_j). \quad (3)$$

In other words, one can now describe a workload relative to some other device. Recalling the analogy, if you want to predict your grade in a course that a colleague has already taken, you could simply have the colleague tell you his grade and the number of hours he worked each week. Other details of the course (workload characteristics) could be useful, but this information may not be as critical.

Next, rather than predict performance P_i, one can predict the performance ratio $\frac{\mathrm{P}_i}{\mathrm{P}_j}$, which may be a simpler function to model (e.g., perhaps device i is twice as fast as device j). We call such a model a *relative fitness model*:

$$\frac{\mathrm{P}_i}{\mathrm{P}_j} = RF_{j \to i}(\mathbf{WC}_j, \mathbf{Perf}_j, \mathbf{Util}_j). \quad (4)$$

To use the relative fitness model, one solves for P_i:

$$\mathrm{P}_i = RF_{j \to i}(\mathbf{WC}_j, \mathbf{Perf}_j, \mathbf{Util}_j) \times \mathrm{P}_j.$$

3.1. Model training

Training a relative fitness model requires workload samples from two devices i and j. Each workload sample can be described with three vectors: workload characteristics (**WC**), performance (**Perf**), and resource utilization (**Util**). During training, the goal is to learn relationships between the predictor variables ($\mathbf{WC}_j$, $\mathbf{Perf}_j$, and $\mathbf{Util}_j$) and the predicted relative fitness value $\frac{\mathrm{P}_i}{\mathrm{P}_j}$ (for some P in **Perf**).

Table 2(c) shows the format of the training data for a relative fitness model. For comparison, Table 2(b) shows that of a relative model which trains to predict performance (not a ratio), and Table 2(a) shows that of an absolute model which only requires samples from one storage device.

Given sufficient training data, one can construct a relative fitness model using a variety of learning algorithms. The problem falls under the general scope of *supervised learning*, where one has access to a set of predictor variables (**WC**, **Perf**, and **Util**), as well as the desired response (the relative fitness value). It is as though an oracle (or supervisor) gives the true output value for each sample, and the algorithms need only learn the mapping between input and output.

The domain of supervised learning problems can be further subdivided into classification (discrete-valued predictions) and regression (continuous-valued predictions). Relative fitness values are continuous, and there are many regression models in statistical literature. We choose to use classification and regression tree (CART) models, for their simplicity, flexibility, and interpretability.[3]

3.2. Summary and modeling cost

Whereas conventional absolute modeling constructs one model per device and assumes that the workload characteristics are static across devices, relative fitness modeling constructs two models for each pair of devices ($i \to j$ and $j \to i$) and implicitly models the changing workload characteristics. In addition, the relative approaches use performance and resource utilization when making predictions, thereby relaxing the dependency on expressive workload characteristics.

Of course, the cost of the relative approach is the additional model construction: $O(n^2)$ versus $O(n)$, where n is the number of storage devices. However, in our evaluation, model construction takes at most a few seconds. Moreover, models can be built and maintained by each storage device. That is, each device can

Table 2: Training data formats for the various models. The last column in each table is the variable that we train a model to predict. All other columns are predictor variables.

Sample	Predictor Variables	Predicted Variables
(a) Absolute model		
1	$\mathbf{WC}_{i,1}$	$\mathrm{P}_{i,1}$
2	$\mathbf{WC}_{i,2}$	$\mathrm{P}_{i,2}$
N	$\mathbf{WC}_{i,n}$	$\mathrm{P}_{i,n}$
(b) Relative model		
1	$\mathbf{WC}_{j,1}$ $\mathbf{Perf}_{j,1}$ $\mathbf{Util}_{j,1}$	$\mathrm{P}_{i,1}$
2	$\mathbf{WC}_{j,2}$ $\mathbf{Perf}_{j,2}$ $\mathbf{Util}_{j,2}$	$\mathrm{P}_{i,2}$
N	$\mathbf{WC}_{j,n}$ $\mathbf{Perf}_{j,n}$ $\mathbf{Util}_{j,n}$	$\mathrm{P}_{i,n}$
(c) Relative fitness model		
1	$\mathbf{WC}_{j,1}$ $\mathbf{Perf}_{j,1}$ $\mathbf{Util}_{j,1}$	$\mathrm{P}_{i,1}/\mathrm{P}_{j,1}$
2	$\mathbf{WC}_{j,2}$ $\mathbf{Perf}_{j,2}$ $\mathbf{Util}_{j,2}$	$\mathrm{P}_{i,2}/\mathrm{P}_{j,2}$
N	$\mathbf{WC}_{j,n}$ $\mathbf{Perf}_{j,n}$ $\mathbf{Util}_{j,n}$	$\mathrm{P}_{i,n}/\mathrm{P}_{j,n}$

construct $O(n)$ models that predict its fitness relative to all other devices. As such, the computational resources for maintaining the models can be made to scale with the number of devices. Also, in large-scale environments, certain collections of devices will be identical and can share models.

4. EVALUATION

The motivation and advantages of relative fitness modeling can be stated as four hypotheses:

- **Hypothesis 1.** Workload characteristics can change across storage devices ($\mathbf{WC}_i \neq \mathbf{WC}_j$) and reduce the accuracy of an absolute model.
- **Hypothesis 2.** A relative model (Equation 2) can reduce the inaccuracies that result from changing characteristics.
- **Hypothesis 3.** Performance and resource utilization can improve prediction accuracy (Equation 3).
- **Hypothesis 4.** Performance ratios (Equation 4) can provide better accuracy than raw performance values (Equation 3).

To test these hypotheses, the accuracy of various CART models can be compared: absolute models (Equation 1), relative models (Equation 2), relative models with performance (Equation 3), and relative fitness models (Equation 4).

4.1. Setup

Experiments are run on an IBM x345 server (dual 2.66GHz Xeon, 1.5GB RAM, GbE, Linux 2.6.12) attached to three iSCSI storage arrays. The arrays have different hardware platforms, software stacks, and are configured with different RAID levels.[a] More specifically,

- **Vendor A** is a 14-disk RAID-50 array with 1GB of cache (400GB 7200 RPM Hitachi Deskstar SATA)
- **Vendor B** is a 6-disk RAID-0 array with 512MB of cache (250GB 7200 RPM Seagate Barracuda SATA)
- **Vendor C** is an 8-disk RAID-10 array with 512MB of cache (250GB 7200 RPM Seagate Barracuda SATA)

The server attaches to each array using an iSCSI device driver[6] that contains counters (below the file system and page cache) for characterizing workloads and measuring their performance. A synthetic workload generator[6] is used to generate numerous workload samples, which we refer to as a *fitness test*. These samples are used to train and test the CART models. Similar results from other workloads (e.g., Postmark, TPC-C), as well as details on the CART algorithm (e.g., tree construction and pruning), can be found in our conference paper.[10]

4.2. Fitness test results

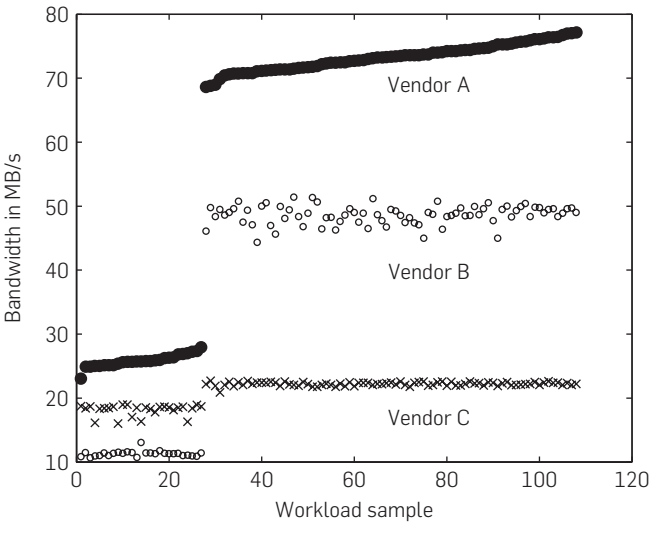
The fitness test compares the performance of the storage arrays across a wide range of workloads (various runs of the workload generator). The measured workload characteristics (**WC**) of each sample include the write percent, the write and read request sizes, the write and read randomness (average seek distance, in blocks, per I/O), and the queue depth (average number of outstanding I/Os). The performance (**Perf**) of each sample run is the average bandwidth (MB/s), throughput (IO/s), and latency (ms). Resource utilization (**Util**) is not used in this evaluation, as this requires modifying storage device software to which we did not have access. A total of 3000 samples are generated.

Over all 3000 samples, Vendor A is the fastest array with an average bandwidth of 25 MB/s, an average throughput of 624 IO/s and an average latency of 37ms. Vendor B is the second fastest (17 MB/s, 349 IO/s, and 45ms). Vendor C is the third (14 MB/s, 341 IO/s, and 84ms). Although Vendor A is the fastest, on average, it is not necessarily the fastest for all sample workloads in the fitness test. There are samples where Vendors B and C do better than A (relative fitness values greater than 1) and cases where they do worse (values lesser than 1). In short, the relative fitness of a device can vary with the workload characteristics.

As an example of how devices can behave similarly, Figure 3 illustrates how the sequential write bandwidth for each array varies for different request sizes and queue depths. From the 3000 samples, we show only the sequential write workloads. There are 120 such samples, sorted by the performance of Vendor A. The graph illustrates the similar performance of the arrays. In particular, the prominent discontinuity in the graph is shared by all arrays (a drop in performance when there are only one or two outstanding requests). Also note how Vendor B is faster than Vendor C to the left of the discontinuity, but slower to the right. Such piecewise functions are ideally suited for CART models.

In support of Hypothesis 1, Table 3 contains averages for the workload characteristics of each sample. Note the variance across devices ($\mathbf{WC}_i \neq \mathbf{WC}_j$), most notably the average spatial randomness of writes, which varies by as much as 38%. In particular, Vendor A experiences the most randomness (an average seek distance of 321MB per write), Vendor B the second most

Figure 3: Device similarity. The performance of each array changes similarly, indicating that the performance of one array is a good predictor of another.



[a] RAID level 0 is striping, 1 is mirroring, 5 is striping with parity, 10 is striping over mirrored pairs (4 in this case), and 50 is striping over RAID-5 parity arrays (2 in this case).

(250MB), and Vendor C the third (233MB). Although masked by the averages in Table 3, the request sizes and queue depths also vary across storage devices for some of the sample workloads.

4.3. Interpreting the models

Of the fitness test samples, 75% are used to build the CART models and 25% are reserved for testing. Figure 4 illustrates four of the bandwidth models, one of each modeling type. For readability, each tree is pruned to a depth of 4, resulting in at most 8 leaves (prediction rules).

The models in Figure 4 predict the performance of Vendor C given observations from Vendor A. CART builds trees top-down, so nodes near the top of the tree have the most information. In particular, note how the relative and relative fitness models learn that the bandwidth of Vendor A is the best predictor of the bandwidth of Vendor C.

As an example of how to use the trees to make a prediction, suppose a workload is running on Vendor A and we want to predict its performance on Vendor C. Also suppose that the workload, as measured by Vendor A, has an average read seek of 2048 blocks, a request size of 64KB, a write percentage <0.5%, a bandwidth of 83 MB/s, and a throughput of 1328 IO/s. The absolute model will predict 75.0 MB/s (see highlighted path in Figure 4a), the relative model (Figure 4b) predicts 75 MB/s, the relative model trained with performance (Figure 4c) predicts 65.0 MB/s, and the relative fitness model (Figure 4d) predicts that Vendor C is 63% of Vendor A or 51 MB/s.

4.4. Modeling accuracy

Recall that 25% of the fitness test samples are reserved for testing, so the performance of each sample is known and can be used to determine the relative error of each prediction.

Table 3: Fitness test workload characteristics.

WC	Vendor A	Vendor B	Vendor C	Maximum Difference (%)
Write percent	40	39	38	5.2
Write size (KB)	61	61	61	0
Read size (KB)	40	41	41	2.5
Write seek (MB)	321	250	233	38
Read seek (MB)	710	711	711	0
Queue depth	23	22	21	9.5

For example, if the performance of Vendor C (for a given test sample) is 45MB/s and the prediction is 51MB/s, the relative error is $\frac{|45-51|}{45} \times 100$, or 13.3%. To quantify the average error of each model (over all test samples), we report the average relative error of the predictions.

As a baseline, Table 4 contains the average relative error of the bandwidth, throughput, and latency predictions for the absolute model. The table is organized pairwise. Workload characteristics ($\mathbf{WC}_j$) are obtained from one array and predictions (P_i) are made for another. For example, the average relative error of the bandwidth predictions when characterizing on Vendor A and predicting for Vendor C is 22% (the top right cell in Table 4).

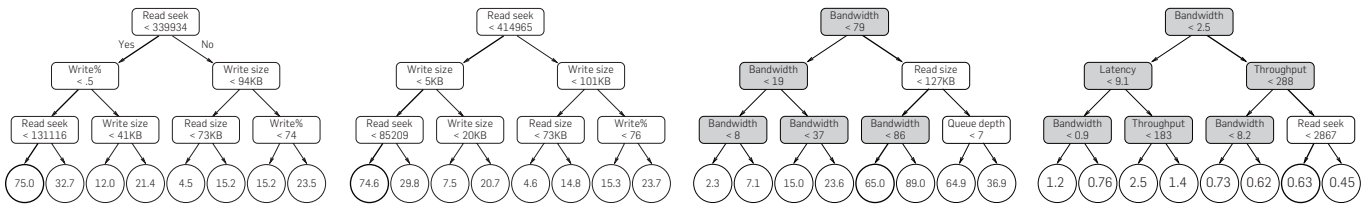
The first observation is that the most accurate predictions occur when the workload is characterized on the same device for which the prediction is being made ($\mathbf{WC}_j = \mathbf{WC}_i$), as indicated by the diagonals in bold. However, if one runs a workload on device *i* to obtain $\mathbf{WC}_i$, there is no need to make a prediction. These predictions are only included to illustrate how changing workload characteristics ($\mathbf{WC}_j \neq \mathbf{WC}_i$) can affect prediction accuracy. For example, the bandwidth prediction error for Vendor A increases from 23% (when characterized on Vendor A) to 29% (when characterized on Vendor B) and 30% (when characterized on Vendor C). Therefore, Table 4 supports Hypothesis 1: changing workload characteristics can affect prediction accuracy.

Figure 5, in contrast, shows the prediction errors for the relative model (Equation 3) and relative fitness model

Table 4: Prediction error for the absolute model. Workload characteristics ($\mathbf{WC}_j$) are obtained from array *j* and predictions (P_i) are made for array *i*.

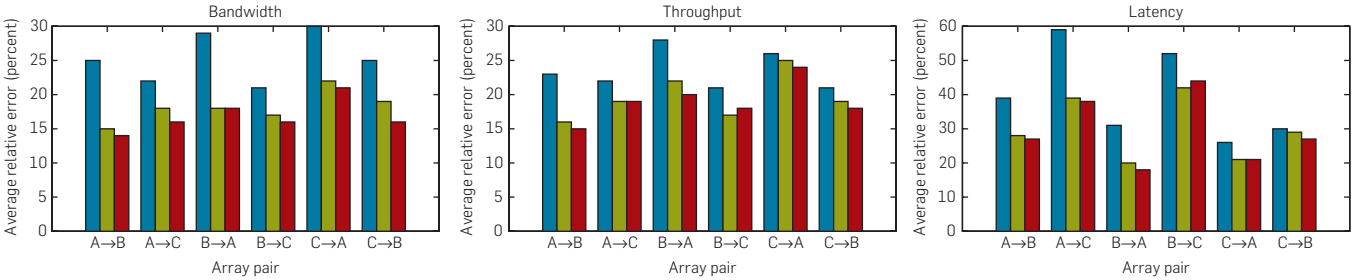
	$\mathbf{Bandwidth}_A$ (%)	$\mathbf{Bandwidth}_B$ (%)	$\mathbf{Bandwidth}_C$ (%)
WC_A	**23**	25	22
WC_B	29	**19**	21
WC_C	30	25	**17**
	$\mathbf{Throughput}_A$ (%)	$\mathbf{Throughput}_B$ (%)	$\mathbf{Throughput}_C$ (%)
WC_A	**20**	23	22
WC_B	28	**15**	21
WC_C	26	21	**14**
	$\mathbf{Latency}_A$ (%)	$\mathbf{Latency}_B$ (%)	$\mathbf{Latency}_C$ (%)
WC_A	**20**	39	59
WC_B	31	**21**	52
WC_C	26	30	**21**

Figure 4: CART models trained to predict the bandwidth of Vendor C. The leaf nodes in the absolute and relative models represent bandwidth predictions; the leaves in the relative fitness model are relative fitness predictions. The absolute model (a) and relative model (b) only use workload characteristics from Vendor A; the relative model (c) with Perf. and relative fitness model (d) also use Vendor A's performance (shaded). All but the absolute model account for changes in the workload characteristics between Vendors A and C.



(a) Absolute (b) Relative (c) Relative with Perf. (d) Relative fitness

Figure 5: Errors of the absolute (first bar in each graph), relative (second), and relative fitness models (third), where *i*→*j* indicates that workload characteristics from array *j* are used to predict the performance of array *i*.



(Equation 4), both of which use performance information to make a prediction; the absolute model prediction errors from Table 4 are shown for comparison. Overall, the relative fitness models reduce the average bandwidth prediction error from 25% to 17%, throughput from 24% to 19%, and latency from 40% to 29%. Moreover, in most cases, the relative fitness model is slightly more accurate than the relative model. These results confirm that models trained with performance can be more accurate (Hypotheses 2 and 3) and that predicting ratios can further improve accuracy (Hypothesis 4).

In summary, workload characteristics can change across devices and impact the accuracy of an absolute model (Hypothesis 1), a relative model can reduce the inaccuracy due to changing workloads (Hypothesis 2), the performance of one device can be used to predict the performance of another (Hypothesis 3), and performance ratios can be better predictors than raw performance values (Hypothesis 4).

5. CONCLUSION

By modeling storage devices relative to one another, relative fitness models can use the observed performance and resource utilization of a workload on one device when making predictions for another. Such modeling addresses many of the challenges associated with workload characterization and, therefore, brings automated storage management a step closer to becoming a practical solution for the data center.

In addition, relative fitness models may find a broader applicability outside of storage management. In the same manner that storage models can exploit performance and resource utilization, so too can models of other data center resources (e.g., application servers).

Acknowledgments

We thank Christos Faloutsos, Arif Merchant, Mic Bowman, and the PDL Consortium (APC, Cisco, Data Domain, EMC, Facebook, Google, Hewlett-Packard, Hitachi, IBM, Intel, LSI, Microsoft, NetApp, Oracle, Panasas, Seagate, Symantec, and VMware). We thank EqualLogic, Intel, IBM, LeftHand Networks, NetApp, Open-E, Seagate, and Sun for equipment donations. This research is sponsored in part by the NSF (CNS-0326453, CCF-0621499, and 0431008) and the Army Research Office (DAAD19-02-1-0389). Matthew Wachs is supported in part by an NDSEG Fellowship.

References

1. Anderson, E. *Simple Table-Based Modeling of Storage Devices*. Technical Report HPL-SSP-2001-4, Hewlett-Packard Laboratories, July 2001.
2. Anderson, E., Hobbs, M., Keeton, K., Spence, S., Uysal, M., Veitch, A. Hippodrome: running circles around storage administration. In *Proceedings of the 1st USENIX Conference on File and Storage Technologies* (*FAST 02*) (Monterey, CA, January 2002), The USENIX Association.
3. Breiman, L., Friedman, J., Olshen, R.A., Stone, C.J. *Classification and Regression Trees*. Chapman and Hall, New York, NY, 1984.
4. Ganger, G.R. Generating representative synthetic workloads: an unsolved problem. In *Proceedings of the 21st International Computer Measurement Group Conference* (*CMG*) (Nashville, TN, December 1996), Computer Measurement Group (CMG).
5. Ganger, G.R., Strunk, J.D., Klosterman, A.J. *Self-* Storage: Brick-Based Storage with Automated Administration*. Technical Report CMU-CS-03-178, Carnegie Mellon University, August 2003.
6. Intel Corporation. Open Storage Toolkit. http://www.sourceforge.net/projects/intel-iscsi.
7. Kelly, T., Cohen, I., Goldszmidt, M., Keeton, K. *Inducing Models of Black-Box Storage Arrays*. Technical Report HPL-2004-108, Hewlett-Packard Laboratories, June 2004.
8. Kurmas, Z., Keeton, K. Using the distiller to direct the development of self-configuration software. In *Proceedings of the 1st International Conference on Autonomic Computing* (*ICAC-04*) (New York, NY, May 2004), IEEE Computer Society.
9. Mesnier, M. *On Modeling the Relative Fitness of Storage*. Ph.D. dissertation. Carnegie Mellon University, Pittsburgh, PA, December 2007.
10. Mesnier, M., Wachs, M., Sambasivan, R.R., Zheng, A., Ganger, G.R. Modeling the relative fitness of storage. In *Proceedings of the International Conference on Measurement and Modeling of Computer Systems* (*SIGMETRICS 2007*) (San Diego, CA, June 2007), ACM.
11. Uysal, M., Alvarez, G.A., Merchant, A. A modular, analytical throughput model for modern disk arrays. In *Proceedings of the 9th International Symposium on Modeling Analysis and Simulation of Computer and Telecommunications Systems* (*MASCOTS-2001*) (Cincinnati, OH, August 2001), IEEE/ACM.
12. Varki, E., Merchant, A., Xu, J., Qiu, X. Issues and challenges in the performance analysis of real disk arrays. *IEEE Transactions on Parallel and Distributed Systems* (*TPDS*) *15*, 6 (June 2004), 559–574.
13. Wang, M., Au, K., Ailamaki, A., Brockwell, A., Faloutsos, C., Ganger, G. R. Storage device performance prediction with CART models. In *Proceedings of the 12th International Symposium on Modeling Analysis and Simulation of Computer and Telecommunications Systems* (*MASCOTS-2004*) (Volendam, The Netherlands, October 2004), IEEE.

Michael P. Mesnier
Intel Corporation
Hillsboro, OR.

Alice X. Zheng
Microsoft Research
Redmond, WA.

Matthew Wachs
Carnegie Mellon University
Pittsburgh, PA.

Gregory R. Ganger
Carnegie Mellon University
Pittsburgh, PA.

Raja R. Sambasivan
Carnegie Mellon University
Pittsburgh, PA.

DOI:10.1145/1498765.1498790

Technical Perspective
Integrating Flash Devices

By Goetz Graefe

FLASH MEMORY NOWADAYS seems to be in every discussion about system architecture—not only in mobile devices from phones to notebook computers, but also in servers, from Web servers to blades and database systems. Sure enough, flash memory boasts multiple qualities and advantages over traditional mass storage, disk drives with rotating platters and moving access arms. These include no noise or vibration, lower power consumption and cooling requirements during both active and idle times, faster access times, and lower cost when calculated with a focus on access performance rather than on capacity. For example, a "flash disk" device providing a standard interface and form factor of traditional SATA disk may cost five times more than a traditional SATA disk, but if it permits 100 times more read operations per second, the cost for access performance is 20 times less. Similarly, flash memory devices offer tremendous advantages over traditional disk drives in terms of power and energy relative to access performance.

On the other hand, flash memory still suffers from two principal weaknesses: reliability and cost relative to capacity. Thus, a system architect is tempted to combine traditional RAM and traditional disk drives with flash memory. The RAM provides write endurance by absorbing the data traffic coming from the CPU caches; omitting RAM in a system architecture and backing up CPU caches with flash memory would lead to very unreliable or expensive computing systems. The traditional disk drives provide cost-efficient storage capacity; many studies have shown that a large fraction of any data collection is rarely accessed after a short initial period of creation and repeated activity. In fact, disk hardware could be divided into capacity-optimized drives, often targeted at consumers, and performance-optimized drives, often targeted at enterprises. Multi-disk strategies could be similarly divided, with RAID-5 and -6 optimized for minimal overhead, minimal power, and maximal capacity and with RAID-1 optimized for performance.

In discussions and designs, flash memory is placed between RAM and disks. Thus, the traditional two-level memory hierarchy (ignoring CPU caches and archival tapes) becomes a three-level hierarchy. For most applications, rewriting the source code to accommodate a deeper memory hierarchy does not make sense. Thus, the flash memory must be integrated in such a way that its presence is hidden except for the performance or cost advantage. One exception may be database management systems, which manage very large data volumes and are thus constantly being updated to take the best advantage of available hardware. For example, researchers and vendors are investigating challenges and opportunities due to deep CPU caches, many-core processors, transactional memory, and flash memory.

Roberts, Kgil, and Mudge represent a perspective that assumes no change in application software, meaning all adaptations for deep memory hierarchy and for flash memory are in the lower levels of system software. Multiple interesting and promising techniques are explored, for example, increasing reliability of multi-level flash memory by using individual pages in single-level mode rather than immediately declaring them bad blocks. This novel technique for graceful degradation of flash memory capacity, if widely adopted in future implementations of the flash translation layer, could greatly increase reliability and cost effectiveness, and speed the adoption of flash memory in all kinds of servers. The race is on among techniques that give flash memory the required reliability and endurance—candidates include over-provisioning, hardware techniques such as those presented here, and software techniques such as a flash translation layer akin to a log-structured file system.

The authors introduce and compare three basic architectures of using flash memory in the memory hierarchy: as "extended system memory," as "storage accelerator," and as "alternative storage device." This characterization resonates with earlier work on the Five-Minute Rule and flash memory (*ACM Queue* 2008). The authors tie these usage models to hardware interfaces, namely memory, PCI, and disk interfaces such as SATA. After reviewing these approaches, the authors synthesize a proposed architecture for a flash-based disk cache. The proposed memory controller partitions the flash memory into separate regions for reading and writing in order to accommodate the need to erase large blocks prior to writing. Future improvements, for example, adaptation of generational garbage collection as proposed for log-structured file systems, will likely build on this foundation. The proposed "flash cache hash table," held in RAM, permits efficient mapping of pages to locations in flash memory; one wonders whether this function could be integrated in the virtual memory management already ubiquitous in operating systems.

The authors present results of a detailed simulation study of their proposed architecture using a full-system simulator. Rather than simply add flash to a system including traditional RAM and traditional disks, they reduce RAM for equal die area before comparing the RAM-and-flash system with the traditional RAM-only system. Equal power consumption could be an alternative metric but is left for future study. Similarly, secondary software effects are omitted, for example, faster access times leading to shorter delays due to virtual memory faults or misses in the buffer pools of file system or database server, such that a lower multi-programming level can mask all those delays, which in turn reduces memory contention and thus the RAM required in the system.

Nonetheless, the results demonstrate that with flash memory in a memory hierarchy, less power and cooling can still result in higher processing bandwidth, and that the proposed programmable controller for flash memory can extend the expected lifetime for a given access rate. With those results, the proposed techniques represent a step toward more efficient storage, servers, and data centers, reducing costs for data center operation and environment emissions.

Goetz Graefe (goetz.graefe@hp.com) is an HP Fellow and member of the Advanced Database Group at Hewlett-Packard Laboratories.

DOI:10.1145/1498765.1498791

Integrating NAND Flash Devices onto Servers

By David Roberts, Taeho Kgil, and Trevor Mudge

Abstract

Flash is a widely used storage device in portable mobile devices such as smart phones, digital cameras, and MP3 players. It provides high density and low power, properties that are appealing for other computing domains. In this paper, we examine its use in the server domain. Wear-out has the potential to limit the use of Flash in this domain. To seriously consider Flash in the server domain, architectural support must exist to address this lack of reliability. This paper first provides a survey of current and potential Flash usage models in a data center. We then advocate using Flash as an extended system memory usage model—OS managed disk cache—and describe the necessary architectural changes. Specifically we propose two key changes. The first improves performance and reliability by splitting Flash-based disk caches into separate read and write regions. The second improves reliability by employing a programmable Flash memory controller. It changes the error code strength (number of correctable bits) and the number of bits that a memory cell can store (cell density) in response to the demands of the application.

1. INTRODUCTION

Data centers are an integral part of today's computing platforms. As cloud computing initiatives provide IT capabilities that incorporate software as a service, it requires internet service providers such as Google and Yahoo to build large-scale data centers hosting millions of servers. Energy efficiency becomes a first-class citizen to address the increasing cost of operating a data center. Data centers based on off-the-shelf general-purpose processors are unnecessarily power hungry, require expensive cooling systems, and occupy a large space. In fact, the cost of power and cooling these data centers contributes to a significant portion of the operating cost. Figure 1 breaks down the annual operating cost for data centers. It clearly shows that the cost of power and cooling servers increasingly contributes to the overall operating costs of a data center.

System memory power (DRAM power) and disk power contribute as much as 50% to the overall power consumption in a data center. Further, current trends suggest that this percentage will continue to increase at a rapid rate as we integrate more memory modules (DRAM) and disk drives to improve throughput.

Fortunately, there are emerging memory devices in the technology pipeline that may address this concern. These devices typically display high density and consume low idle power. Flash, Phase Change RAM (PCRAM) and Magnetic RAM (MRAM) are examples.

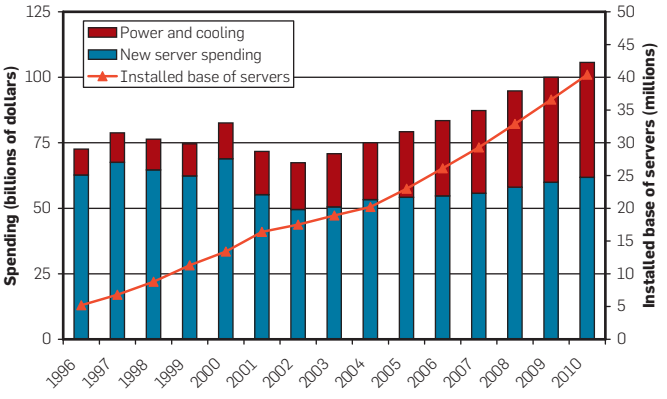
In particular, Flash is an attractive technology that is already deployed heavily in various computing platforms. Today, NAND Flash can be found in handheld devices such as smart phones, digital cameras, and MP3 players. This has been made possible because of its high density and low power properties. These result from the simple structure of Flash cells and its nonvolatility. Its popularity has meant that it is the focus of aggressive process scaling and innovation.

The rapid rate of improvement in density has become the primary driver to consider Flash in other usage models. There are several Flash usage models in the data center that are currently being examined by industry and academia that address rising power and cooling costs, among other things. Two common usage models are disk caches or storage devices. Some efforts have lead to product development,[8,19] while others have influenced storage and memory device standards.[16,18]

This paper provides an overview of the benefits of integrating Flash onto a server. Specifically, in this paper:

1. We provide an analysis of current and potential Flash usage models for servers.
2. We argue that the extended system memory model[10] is the best usage model to reduce data center energy when the contribution of system memory power exceeds the contribution of disk power.
3. We review two architectural modifications to improve NAND-based disk caches.[11] First, we show that by splitting Flash-based disk caches into read and write regions, overall performance and reliability can be improved.

Figure 1: IDC estimates for annual cost spent on powering and cooling servers and purchasing new servers.[17]



A previous version of this paper, entitled "Improving NAND Flash-based Disk Caches" was published in *Proceedings of the International Symposium on Computer Architecture* (ISCA 2008).

Second, we show that a programmable Flash memory controller can improve Flash cell reliability and extend memory lifetime. The first programmable parameter is error correction code (ECC) strength. The second is the Flash cell density—changing from multilevel cells (MLC) to single-level cells (SLC).

2. BACKGROUND

2.1. Properties of a NAND Flash device

Flash memory is a nonvolatile memory device that can be electrically read, written, and erased. Flash memory cells in NAND Flash are connected in series to maximize cell density. Further, to improve Flash density, each Flash memory cell can use multiple threshold voltage levels to store more than one bit per cell. NAND Flash supporting MLC is called MLC NAND Flash. NAND Flash using a single threshold voltage level (technically two levels) is called SLC NAND Flash. Cutting-edge MLC NAND Flash supports 4 bits per cell. There are significant differences in the access time and lifetime of the two types. Although MLC Flash is cheaper and bit density is higher relative to SLC, MLC is slower to read and write and has shorter lifetime by a factor of 10 or more. Typical latencies for read, write, and erase are 25 µs, 250 µs, and 0.5 ms for SLC and 50 µs, 900 µs, and 3.5 ms for 2-bit MLC. The gap between performance and lifetime is getting worse as the number of bits per Flash cell is increased. This may be perfectly acceptable for some applications; for example, a tune in an MP3 player may only be replaced every few days. A disk cache, however, may have all of its locations rewritten several times a day depending on the amount of disk traffic and size of the cache.

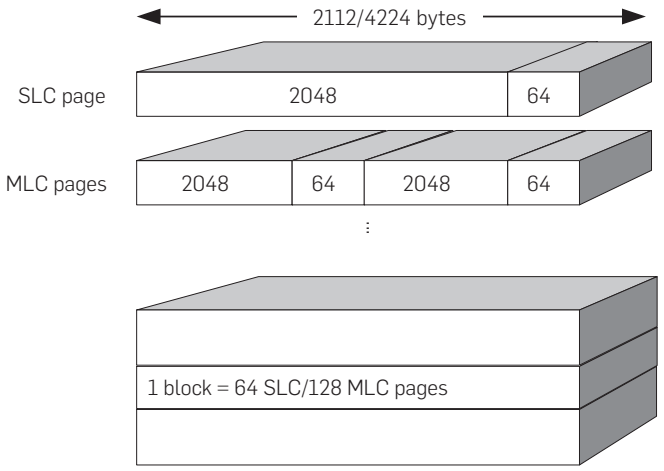
NAND Flash is organized in units of *pages* and *blocks*. A typical Flash *page* is 2KB in size and a Flash *block* is made up of 64 Flash *pages* (128KB). Random Flash reads and writes are performed on a *page* basis and Flash erasures are performed per *block*. A Flash must perform an erase on a *block* before it can write to a *page* belonging to that *block*. Each additional write must be preceded by an erase. Therefore *out-of-place* writes are commonly used to mitigate wear-out. These writes append new data to the end of the log while old data pages are invalidated.

NAND Flash can also be dynamically configured to support multiple Flash memory cell types for each page or block. In fact, such devices are now commercially available, e.g., Samsung's Flex-OneNAND.[6] Figure 2(a) illustrates the organization of an SLC/MLC dual-mode device. Pages in SLC mode consist of 2048 bytes of data area and 64 bytes of "spare" data for ECC bits. When in MLC mode, an SLC page can be split into two 2048 byte MLC pages. Pages are erased together in blocks of 64 SLC pages or 128 MLC pages.

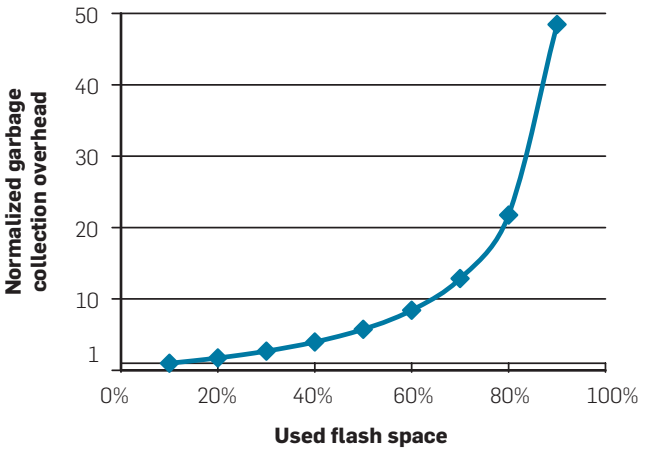
When the number of faulty bits per block exceeds the capabilities of an ECC, blocks are disabled, reducing the capacity of the memory. This is an instance of wear-out affecting system performance over time, especially, for file cache applications.

MLC Flash ages quicker than SLC Flash. An MLC Flash can support fewer reliable write/erase (W/E) cycles due to the smaller threshold voltage margins between bit values. New Flash architectures[6] can circumvent this problem by switching from high-density MLC to lower density or even single-level mode to counter wear-out. No policy currently exists to perform the mode selection, so we propose a mechanism for changing mode, tailored to a disk caching application.

Figure 2: (a) Example dual-mode SLC/MLC Flash bank organization and (b) time spent in garbage collection as a function of the Flash space in use.



(a) Flash block diagram



(b)Garbage collection

Because Flash blocks have a limited number of erases before they develop faulty bits, a *wear-leveling* algorithm attempts to equalize the number of erases performed on each block.[3] This has to be achieved without performing more erases than necessary. The simplest method of wear-leveling is to treat the device as a circular log. New data is written to the next available page and the old page is invalidated. However, wear-leveling causes fragmentation problems. Fragmentation is addressed with garbage collection. The process of garbage collection reads valid pages from erase blocks containing some invalid pages, then writes them to a previously erased block.[4] Garbage collections free up pages that are ready to write new data. This process takes time and increases the amount of wear in the Flash blocks.

The overhead in garbage collection increases as less free space is available on Flash. This becomes a significant

problem, because garbage collection generates extra writes and erases in Flash, reducing performance and endurance as the occupancy of the Flash increases. Figure 2(b) shows how the time spent garbage collecting increases as more Flash space is used. It is normalized to an overhead of 10% and is for a 2GB Flash memory. It can be seen that garbage collection becomes overwhelming well before all of the memory is used.

2.2. NAND Flash usage models in a server

Industry and researchers in academia are making strides to integrate Flash onto the data center. Industry has recently released several Flash-based products and Flash standards targeted for servers, while researchers in academia have recently published several papers proposing techniques for integrating emerging memory technologies including Flash.

NAND Flash usage models pursued by industry and academia can be categorized as follows:

1. Extended system memory usage model: A NAND Flash memory module is connected to the current system memory interface or to a dedicated Flash memory interface.
2. Storage accelerator usage model: A NAND Flash PCI express card is connected to the PCI express interface.
3. Alternative storage device usage model: A Solid State Drive (SSD) replaces or augments the hard disk drive. It is connected to the disk interface. An example would be a SATA SSD.

Each usage model presents a unique set of benefits and challenges. Table 1 qualitatively captures them. The "extended system memory" usage model presents Flash as a part of the system memory. It addresses the rising contribution of power consumed by DRAM in addition to the electrical constraints limiting the integration of more system memory. For example, to increase storage capacity without having to reduce the operating frequency of the memory channel, MetaRAM[14] packs more DRAM onto each DIMM module. Using denser memory such as Flash may serve a similar purpose. However, this usage model requires modification to the operating system kernel. Specifically, the current implementation in the kernel memory manager that supports nonuniform memory architectures needs to be aware of the unique organization and behavior of Flash. Flash reliability management can be performed by the kernel memory manager with the assistance of the Flash controller.

The "storage accelerator" usage model presents Flash as a PCI express device that can be directly managed by the user application. This usage model allows the server application to manage Flash directly as a cache that stores frequently accessed code and data. It reduces the number of accesses to the hard disk drive thereby reducing overall disk power. Further, it may also be used as a way to implement the "extended system memory" usage model but with several drawbacks such as higher latency, lower throughput and added complexity in managing Flash. Flash management is distributed across the user application, device driver stack and the Flash PCI express card firmware. To truly leverage Flash as a "storage accelerator," the user application should be Flash aware. A device driver stack needs to be implemented to support the PCI express device. The device driver stack needs to implement device sharing mechanisms such that other concurrent user applications and kernel components can make use of it simultaneously. In Fusion-io's Solid State Storage[7] they have also shown the "storage accelerator" usage model can expose the Flash PCI express device as an SSD by providing disk emulation features in the device driver stack.

The "alternative storage device" usage model presents Flash as an SSD that replaces a hard disk drive.[15] This usage model improves the latency and throughput to disk and reduces overall disk power consumption in a data center. With appropriate filesystems such as ZFS,[19] it improves storage device scalability in a data center. Industry has heavily adopted this usage model and has recently released several products.[8, 9] These solid state drives are used to implement a storage area network (SAN) or network attached storage (NAS) in a data center. They employ similar reliability features such as RAID, commonly found in a hard disk drive based SAN or NAS. Flash reliability management is performed by the Flash device controller in the SAN or NAS. However, this usage model also requires modification in the kernel, and a complex Flash device controller that is capable of performing intelligent Flash reliability management. A customized filesystem needs to be implemented to fully take advantage of the benefits of Flash.[12, 19] Further, this usage model ties itself to the non-Flash aware features that are found in a hard disk drive interface protocol such as SATA. For example, the device driver can only communicate to disk using SATA commands that are not defined with Flash in mind. On the other hand, the operating system in the "extended main memory" model has full visibility of memory page classification and activity statistics that can be used for more intelligent mapping of data to Flash.

Table 1: Comparison of Flash usage models in a server.

	Primary powersavings	Secondary powersavings	Hardware complexity	OS kernel modification	Application modification	Comments
Extended system memory	DRAM	Disk	Minimal	Medium	None	Extend kernel memory manager to manage Flash devices
I/O accelerator	Disk	DRAM	Medium	Medium	Yes	Need to build I/O accelerator driver stack
Alternative storage device	Disk	DRAM	High	Minimal	None	Need to implement filesystem for Flash

Servers clearly benefit from all three usage models that essentially integrate Flash as a faster hard disk or disk cache. All usage models help (1) reduce unnecessary standby power from hard disk drives and (2) improve overall throughput by reading and writing from disk cache instead of a hard disk drive.

In the remainder of this paper, we examine Flash-based disk cache architectures that improve Flash manageability and reliability in the extended system memory usage model. We believe this usage model is effective in addressing the increasing power consumption in system memory. Our studies on servers have revealed the system memory architecture to be the critical component in delivering high throughput in a data center.[10]

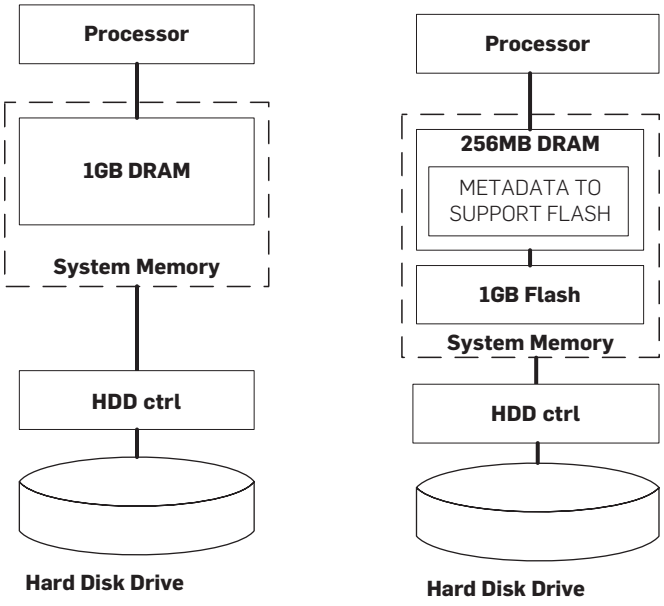
3. PROPOSED ARCHITECTURE

3.1. Architecture of the Flash-based disk cache

The right side of Figure 3 shows the Flash-based disk cache architecture for the extended system memory usage model. Compared to a conventional DRAM-only architecture shown on the left side of Figure 3, our proposed architecture uses a two level disk cache, composed of a relatively small DRAM in front of a dense Flash. The much lower access time of DRAM allows it to act as a cache for the Flash without significantly increasing power consumption. A Flash memory controller is also required for reliability management.

Our design uses a NAND Flash that stores 2 bits per cell (MLC) and is capable of switching from MLC to SLC mode using techniques proposed in Flex-OneNAND[6] and Cho.[5] Finally, our design uses variable-strength ECC to improve reliability while adding the smallest possible delay.

Figure 3: 1GB DRAM is replaced with a smaller 256 MB DRAM and 1GB NAND-based Flash. Additional components are added to control Flash.



(a) Standard system (a) Proposed system

Operating System Support: Our proposed architecture requires additional data structures to manage the Flash blocks and pages. These tables are read from the hard disk drive and stored in DRAM at run-time to reduce access latency and mitigate wear-out. Together, they describe whether pages exist in DRAM or Flash, and specify the various Flash memory configuration options for reliability. For example, the FlashCache Hash Table allows the operating system to quickly look up the location of a file page. The Flash Page Status Table keeps track of the ECC strength, MLC/SLC mode and access frequency for each page. Each Erase block has an entry in the Block Status Table to determine how worn out it is. Finally, the Global Status Table records how quickly the Flash-based disk cache is satisfying requests, and is the number we try to maximize while the system is running.

The storage overhead of the four tables are less than 2% of the Flash size. The FlashCache Hash Table and Flash Page Status Table are the primary contributors because an entry is needed for each Flash page. Our Flash-based disk cache is managed in software (OS code) using the tables described above. We found the performance overhead in executing this code to be minimal.

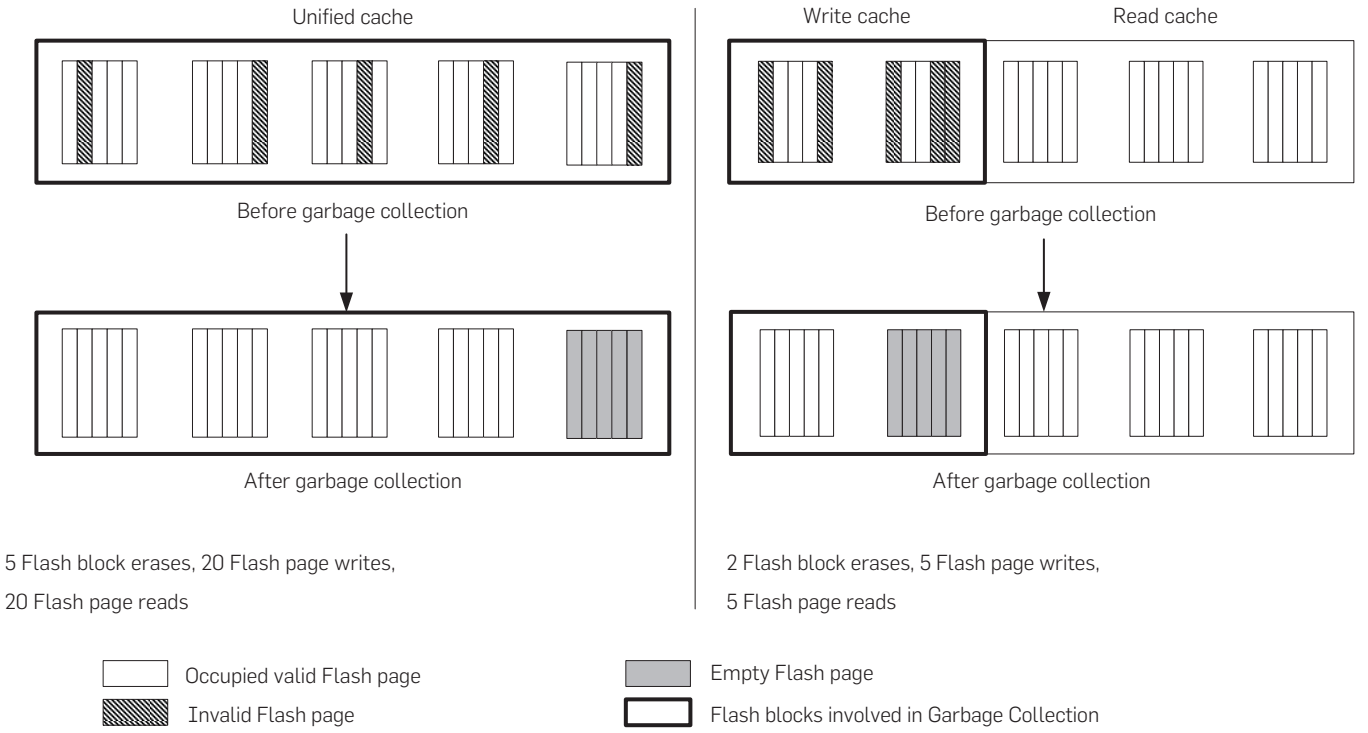
Splitting Flash into Read and Write Regions: We divide the Flash into a read disk cache and a write disk cache. Read caches are less susceptible to *out-of-place* writes, which reduce the read cache capacity and increase the risk of garbage collection. An *out-of-place* write happens when existing data is modified, because Flash has to be erased before it can be written to a second time. It is simple to invalidate the old data page (using the Page Status Table and modifying the Hash Table) then write new data into a previously erased page. However, the invalid pages accumulate as wasted space that will have to be garbage collected later. By splitting Flash into read and write regions, we were able cut down on time consuming garbage collections.

Figure 4 shows an example that highlights the benefits of splitting the Flash-based disk cache into a read and write cache. The left side shows the behavior of a unified Flash-based disk cache and the right side shows the behavior of splitting the Flash-based disk cache into a read and write cache. Figure 4 assumes we have five pages per block and five total blocks in a Flash-based disk cache. Garbage collection proceeds by reading all valid data from blocks containing invalid pages, erasing those blocks and then sequentially re-writing the valid data. In this example, when the Flash-based disk cache is split into a read and write cache, only two blocks are candidates for garbage collection. This dramatically reduces Flash reads, writes, and erases compared to a unified Flash-based disk cache that considers all five Flash blocks. Our studies also show that the overall disk cache miss rate is reduced substantially for online transaction processing (OLTP) applications by splitting the Flash.

3.2. Architecture of the Flash memory controller

Flash needs architectural support to improve reliability and lifetime when used as a cache. Figure 6 shows a high-level block diagram of a programmable Flash memory controller that addresses this need. Requests from the operating system

Figure 4: Example showing the benefits of splitting the Flash-based disk cache into Read and Write caches. In the five erase blocks, pages of data that have been evicted from the cache and invalidated are grayed out. On the left, a unified cache allows pages that are heavily read and written to be placed in any erase block. This results in scattered invalid pages. Our split read/write cache forces read and write-dominated data into two separate sets of erase blocks. As a result, invalid pages are clustered and fewer blocks have to be erased to prepare the invalid pages for another write.



provide the address being accessed and any data to be written. In addition, the OS specifies the strength of ECC and whether the page is in MLC or SLC mode. The controller returns any data that was read along with information concerning the number of errors currently being corrected by the ECC logic.

Our architecture uses a BCH encoder and decoder to perform error correction and a CRC checker to perform error detection. The BCH code guarantees that a number of faulty bits can be corrected. However, as the number of faulty bits increases it takes longer to perform the correction. Doubling the number of correctable bits approximately doubles the time needed to decode the data and extract the correct value. Our system adapts the ECC strength to the appropriate number of faulty bits in each page to achieve graceful Flash wear-out. The relationship between number of correctable bits and erase count is shown in Figure 5. It shows that stronger ECC effectively improves page lifetime. The different lines on the graph show the effects of different levels of variability in the likelihood of bits being faulty. As the standard deviation increases, the number of tolerated erases decreases for any particular error correction strength. It shows that process variability has a negative impact on lifetime and requires more bits to be corrected for the same lifetime. As process technology advances and cells become smaller, the effect of variability will become even more pronounced.

Our programmable Flash memory controller also dynamically controls the density of a Flash page. Density control benefits Flash performance and endurance, because we are able to reduce access latency for frequently accessed pages and possibly improve endurance for aging Flash pages by

Figure 5: Maximum tolerable Flash Write/Erase cycles for varying code strength.

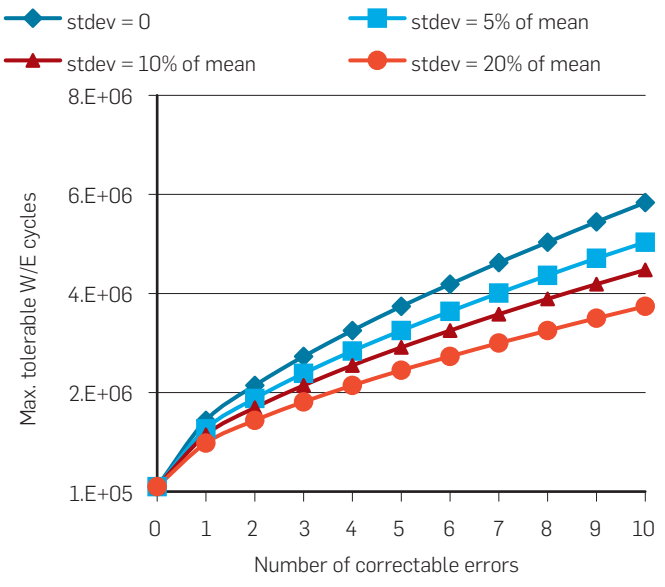
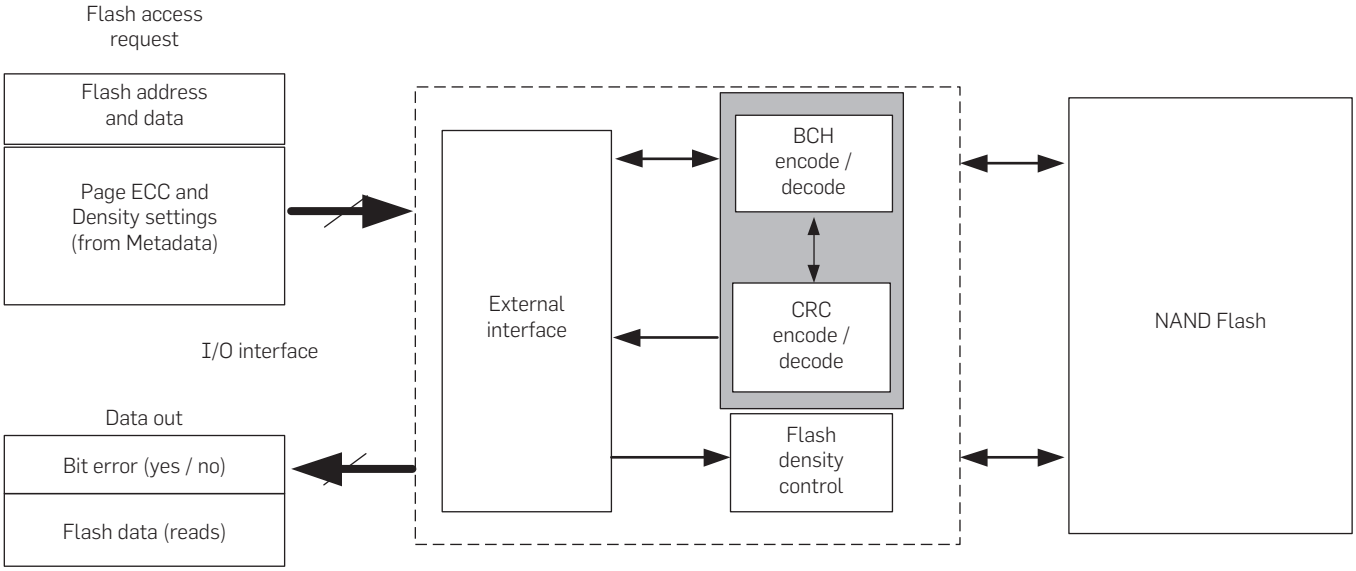


Figure 6: Flash memory controller architecture. The Flash disk cache device driver sends requests to the hardware interface. These requests also specify the ECC strength and density mode of the accessed page. In turn, the controller accesses the Flash chip after performing ECC encoding for a write, or decoding for a read. The device driver software receives any requested data along with an indication of the number of failing Flash bits.



changing MLC pages into SLC pages as needed. To show the potential improvement of Flash performance by controlling density, we present a study using real disk traces.

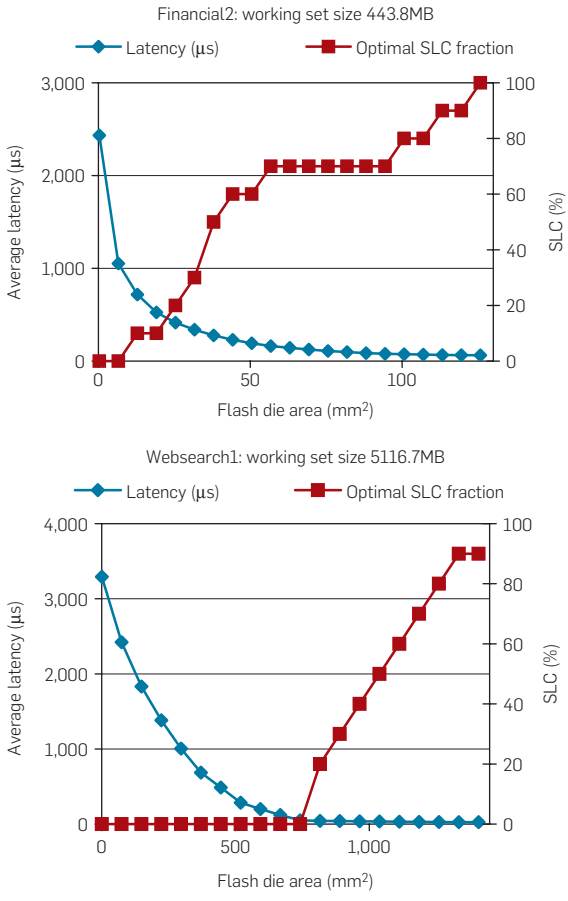
Using disk activity traces from the University of Massachusetts Trace Repository[20] for financial and web search applications, we analyzed the average access latency for different SLC/MLC partitions, for several Flash sizes.

A hybrid allocation of SLC and MLC Flash provides minimum access latency, because it is sometimes more effective to store heavily used data in a faster SLC page and lose one page of storage space. Figure 7 shows the average delay (left *y*-axis) achieved for an optimal partition (right *y*-axis) between SLC and MLC. The *x*-axis shows the Flash memory area and extends far enough to contain the entire working set. As expected, when the size of the cache approaches the entire workload, latency reaches a minimum using only SLC cells. Intermediate Flash sizes provide minimum latency through a combination of MLC and SLC with the most frequently accessed data in the faster SLC cells. The best division between SLC and MLC depends on the access frequencies of the data pages. For example, if there are hot pages that are significantly more active than other pages, the bias will be towards SLC. This type of behavior is exhibited by the Financial2 trace (Figure 7(a)) where the SLC allocation grows rapidly with Flash capacity. If the access frequencies are more uniform (e.g., Websearch1 in Figure 7(b)) it is better to have a bigger (MLC) cache to increase the number of accesses to Flash because going to disk is much slower.

3.3. Operation and dynamic reconfiguration

We use a typical software device driver interface to access the Flash memory controller. The driver specifies which Flash address is to be accessed along with the read or write mode. For a write, the driver also sends the new data to the

Figure 7: Optimal access latency and SLC/MLC partition for various multimode MLC Flash sizes.



controller. Using the configuration tables stored in DRAM, the driver tells the Flash controller what error correction strength and cell density to use for each access. In this section we provide more details on how the cache operates and how its settings are reconfigured on the fly.

Theory of Operation: In this section we summarize how the operating system and controller interact (see Kgil[11] for a full description). The concepts are similar to ordinary disk caches except that it is now a two-level cache. The first level of cache resides in DRAM, and the second level consists of Flash memory. In addition, the Flash portion of the cache has to be reconfigured on the fly to maximize performance and reliability. The DRAM, with fast, uniform read and write latency, no wear-out and no density modes, is easier to handle.

When a file read is performed, the OS searches for the file in the primary disk cache located in DRAM. If the page is found in DRAM, the file content is accessed directly from the primary disk cache—no access to Flash related data structures is required. Otherwise, the OS determines whether the requested file currently resides in the secondary (Flash) disk cache. If the requested file is found, then a Flash read is performed and the Flash content is transferred to DRAM.

If the data is not found in Flash, we first look for an empty Flash page in the read cache. If there is no empty Flash page available, we first select a block for eviction to disk, freeing Flash pages for the newly read data. The data being replaced is usually the "least recently used" (LRU) block so it is unlikely to be needed again. Such an access would have to go all the way to disk, increasing program execution time, so the LRU algorithm reduces the likelihood of this happening. Concurrently, a hard disk drive access is scheduled using the device driver interface. The hard disk drive content is copied to the primary disk cache in DRAM and also the read cache in Flash.

If we write to a file, we typically update/access the page in the primary disk cache and this page is periodically scheduled to be written back to the secondary disk cache and later periodically written back to the disk drive. When writing back to Flash, we first determine whether it already exists on Flash. If it is found in the write region, we update the page by doing an *out-of-place* write to the write cache. If it is found in the read cache, then we move it to the write cache. If it is not found in the Flash, we allocate a page in the write cache.

In the background, garbage collections are triggered when the Flash-based disk cache starts to run out of space. The cached data is also periodically flushed back to disk if it has been modified. Concurrent with the normal cache operation, the reliability management algorithms continuously try to adapt the Flash configuration to provide maximum benefit. We have already seen that the configuration changes with the application software. The next section describes the configuration policies enforced to achieve this.

Reconfiguring the Flash Memory Controller: The Flash Page Status Table (FPST) specifies the reliability control settings for each page of flash. When the OS reads and writes to/from the Flash controller, it also sends configuration bits specifying the various modes for the Flash page. Configuration policies are applied to select those modes, maximizing performance as the application demands change and the Flash eventually develops faulty bits.

There are two main triggers for an ECC strength or density mode change. These are (1) an increase in the number of faulty bits and (2) a change in access (read) frequency. Each trigger is explained below:

When new bit errors are observed and fail consistently due to wear-out, we reconfigure the page. This is achieved by enforcing a stronger ECC or reducing cell density from MLC to SLC mode. We choose the option with the minimum increase in latency using some simple heuristics. They take into account how active that particular page is to determine its impact on the system as a whole. It also considers the current level of wear-out for the page.

Some heavily accessed pages will benefit from being in SLC storage simply because of its lower latency. If a page is in MLC mode and the entry in the FPST field that keeps track of the number of read accesses to a page reaches a limit, we migrate that Flash page to a new empty page in SLC mode. If there is no empty page available, a Flash block is evicted and erased using our wear-level aware replacement policy. Reassigning a frequently accessed page from MLC mode to SLC mode improves performance by improving hit latency. Because many accesses to files in a server platform have a tailed distribution (Zipf) with hot and cold data, improving the hit latency to frequently accessed (hot) Flash pages improves overall performance despite the minor reduction in Flash capacity.

If a Flash page reaches the ECC strength limit and has already been set to SLC mode, the block is removed permanently and never considered when looking for pages to allocate in a disk cache.

4. METHODOLOGY

We evaluated the Flash memory controller and Flash device using a full system simulator called M5.[2] The M5 simulation infrastructure is used to generate access profiles for estimating system memory and disk drive power consumption along with published access energy data. We developed a separate Flash disk cache simulator for reliability and disk cache miss rate experiments where very long traces are necessary, because full system simulators are slow. Given the limitations in our simulation infrastructure, a server workload that uses a large working set of 100–1000's of gigabytes cannot easily be evaluated. We scaled our benchmarks, system memory size, Flash size, and disk drive size accordingly to run on our simulation infrastructure.

We also generated micro-benchmark disk traces to model synthetic disk access behavior. They represent typical access distributions and approximate real disk usage. To properly stress the system, some micro-benchmarks with uniformly random and exponential distributions were also generated.

We used disk traces from University of Massachusetts Trace Repository[20] to model the disk behavior of enterprise level applications like web servers, database servers, and web search. To measure performance and power, we used dbt2 (OLTP) and SPECWeb99 which generated representative disk/disk cache traffic.

5. RESULTS

5.1. System memory and disk energy efficiency

Figure 8 shows a breakdown of power consumption in the system memory and disk drive (left *y*-axis). Figure 8 also shows the measured network bandwidth (right *y*-axis). Throughput measured as network bandwidth is a good indicator of overall system performance as it represents the amount of data that the server can handle in each configuration. We calculated power for a DRAM-only system memory and a heterogenous (DRAM + Flash) system memory that uses a Flash as a secondary disk cache with hard disk drive support. We assume equal die area for a DRAM-only system memory and a DRAM + Flash system memory. Figure 8 shows the reduction in disk drive power and system memory power that results from adopting Flash. Our primary power savings for system memory come from using Flash instead of DRAM for a large amount of the disk cache. The power savings for disk come from reducing the accesses to disk due to a bigger overall disk cache made possible by adopting a Flash. We also see improved throughput with Flash because it displays lower access latency than disk.

5.2. Impact of BCH code strength on system performance

We have already mentioned that BCH latency incurs an additional delay beyond the initial access latency. We simulated the performance of the SPECWeb99 and dbt2 benchmarks to observe the effect of increasing code strength that would occur as Flash wears out. It is assumed that all Flash blocks have the same ECC strength applied. We also measured performance for code strengths (more than 12 bits per page) that are beyond our Flash memory controller's capabilities to fully capture the performance trends.

From Figure 9, we can see that throughput degrades slowly with ECC strength. dbt2 suffers a greater performance loss than SPECWeb99 after 15 bits per page. The disk bound property of dbt2 makes it more sensitive to ECC strength.

5.3. Improved Flash lifetime with reliability support in Flash memory controller

Figure 10 shows a comparison of the normalized number of accesses required to reach the point of total Flash failure where none of the Flash pages can be recovered. We compare our programmable Flash memory controller with a BCH 1-bit error correcting controller. Our studies show that for typical workloads, our programmable Flash memory controller extends lifetime by a factor of 20 on average. For a workload that would previously limit Flash lifetime to 6 months, we show it can now operate for more than 10 years using our programmable Flash memory controller. This was accompanied by a graceful increase in overall access latency as Flash wore out.

6. CONCLUSIONS AND FUTURE WORK

This paper presents the challenges and opportunities in integrating Flash onto a server platform. Flash is an attractive candidate for integration because it reduces power consumption in system memories and disk drives while improving overall throughput. This in turn can reduce the operating cost of a server platform, which is a growing concern in a data center. We presented three key usage models of Flash and examined an architecture for the "extended system memory" usage model. Our proposed architecture carefully manages the Flash and uses it as a secondary disk cache split into a separate read cache and write cache. We observed a dramatic improvement in power consumption and performance. In our simulation studies, a Flash-based disk cache improved the DBT2 database benchmark performance by over 25% while reducing memory and disk power by 44%. For a web server benchmark, performance improvement was around 11% with a power reduction of 73%. This does not account for potentially larger systemwide energy savings obtained from speeding up system response and increasing idle time. Assuming that a server can enter a low-power mode while

Figure 8: Breakdown in system memory and disk power and network bandwidth for architecture with/without a Flash-based disk cache.

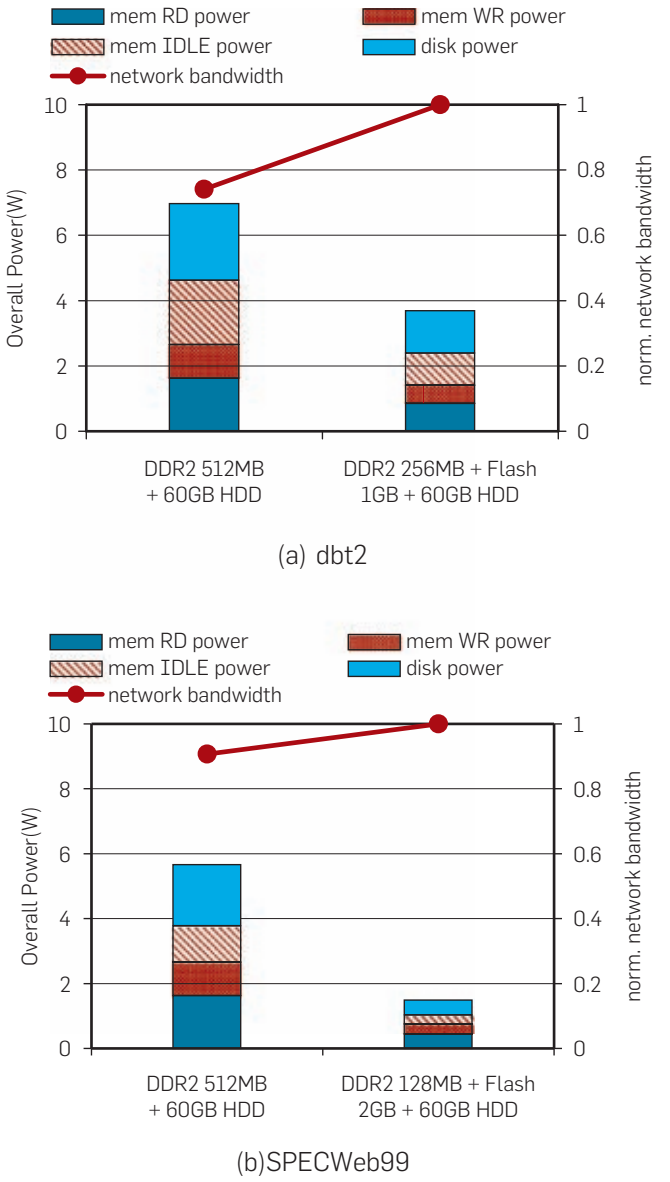


Figure 9: Average throughput as a function of ECC strength. The system used 256MB of DRAM and 1GB of Flash.

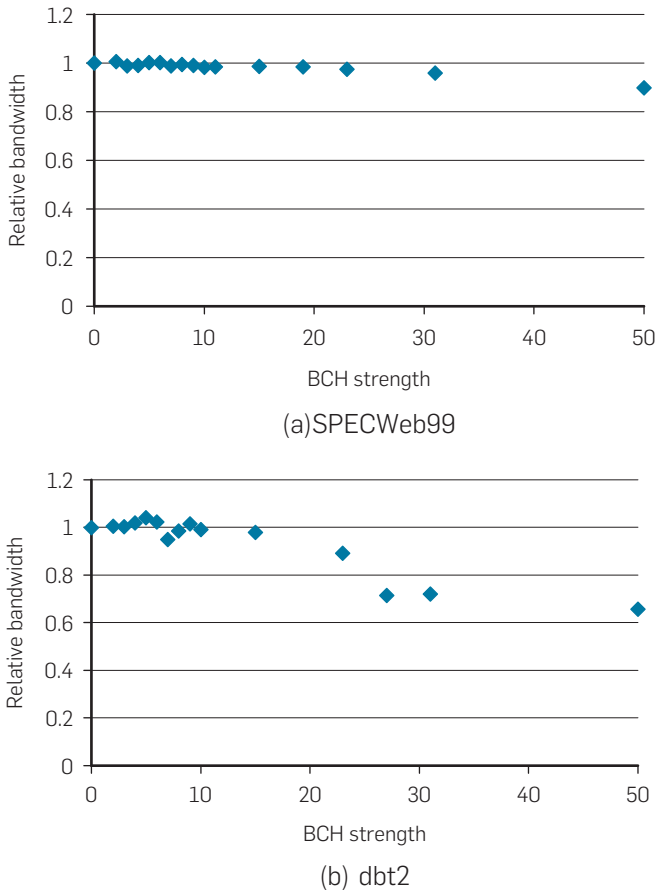
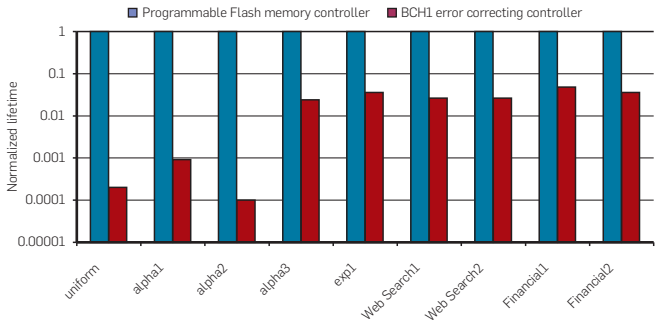


Figure 10: Normalized expected lifetime for a given access rate and the point of total Flash failure.



idle,[13] all components save energy rather than just memory and disk.

We also showed that a Flash memory controller with reliability support greatly improves Flash lifetime. We found that the best configuration of a Flash memory controller is largely dependent upon the access patterns resulting from the application. For example, we found that the typical workload with Zipf access behavior was best served by a Flash configured such that the heavily accessed contents would be located in regions composed of reliable low latency SLC. In general, we found that variable ECC strength gracefully extended Flash lifetime, and that the overhead of ECC is minimized with configurable density. Combining all of our techniques, we saw an average 20× lifetime improvement relative to a system using only a single ECC. We believe our findings are applicable not only to Flash but also to emerging memory technology devices such as PCRAM.[1]

New memory technologies are creating opportunities for increased performance and efficiency in a data center. These disruptive technologies are forcing architects to rethink the current system memory and storage hierarchy in a server. Together, with server virtualization, we believe these memory devices will help realize the objective of building greener data centers. C

References

1. Bedeschi, F. et al. A multi-level-cell bipolar-selected phase-change memory. In *Proceedings of the International Solid-State Circuits Conference* (Feb. 2008), 428–625.
2. Binkert, N., Dreslinski, R., Hsu, L., Lim, K., Saidi, A., Reinhardt, S. The M5 simulator: Modeling networked systems. *IEEE Micro 26*, 4 (Jul./Aug. 2006), 52–60.
3. Chang, L.-P. On efficient wear-leveling for large-scale flash-memory storage systems. In *22nd ACM Symposium on Applied Computing* (*ACM SAC*) (2007).
4. Chang, L.-P., Kuo, T.-W. Real-time garbage collection for flash-memory storage system in embedded systems. *ACM Trans. Embedded Computing Systems 3*, 4 (2004).
5. Cho, T. et al. A dual-mode NAND flash memory: 1-Gb multilevel and high-performance 512-mb single-level modes. *IEEE J. Solid State Circuits 36*, 11 (Nov. 2001).
6. Flex-OneNAND. http://www.samsung.com/global/business/semiconductor/products/fusionmemory/Products_FlexOneNAND.html.
7. Fusion-io's Solid State Storage—A New Standard for Enterprise-Class Reliability. http://www.fusionio.com/PDFs/Whitepaper_Solidstatestorage2.pdf.
8. Hutsell, W, Bowen, J., Ekker, N. Flash Solid-State Disk Reliability. http://www.texmemsys.com/files/f000252.pdf.
9. Intel X18-M/X25-M SATA Solid State Drive. http://download.intel.com/design/flash/nand/mainstream/mainstream-sata-ssd-datasheet.pdf.
10. Kgil, T., Mudge, T. Flashcache: A NAND Flash memory file cache for low power web servers. In *International Conference on Compilers, Architecture and Synthesis for Embedded Systems* (2006).
11. Kgil, T., Roberts, D., Mudge, T. Improving NAND Flash based Disk Caches. In *Proceedings of the International Symposium on Computer Architecture* (*ISCA*) (2008).
12. Leventhal, A. Flash storage today. *ACM Queue* (Aug. 2008).
13. Meisner, D., Gold, B.T., Wenisch, T.F. Powernap: Eliminating server idle power. *ASPLOS* (Mar. 2009).
14. MetaRAMs DDR3 MetaSDRAM Doubles Memory Capacity and Increases Frequency of Future Intel Systems. http://www.metaram.com/pdf/press/MetaRAM_DDR3_08_19_08.pdf.
15. Moshayedi, M., Wilkison, P. Enterprise ssds. *ACM Queue* (Aug. 2008).
16. ONFI: Open NAND Flash Interface. http://www.onfi.org/index.html.
17. Scaramella, J. Enabling Technologies for Power and Cooling. http://h71028.www7.hp.com/enterprise/downloads/Thermal_Logic.pdf.
18. Serial ATA 2.6 Specification. http ://www.sata-io.org.
19. *Solaris ZFS Administration Guide*. 2008.
20. University of Massachusetts Trace Repository. http://traces.cs.umass.edu/index.php/Storage/Storage.

David Roberts (daverobe@umich.edu), Advanced Computer Architecture Lab, Department of CSE, University of Michigan.

Trevor Mudge (tnm@eecs.umich.edu), Advanced Computer Architecture Lab, Department of CSE, University of Michigan.

Taeho Kgil (taeho.kgil@intel.com), Intel Corporation.

Fraunhofer Center – Maryland
Measurement & Knowledge Management Division Researcher

The Fraunhofer Center – Maryland, a non-profit research institute affiliated with the University of Maryland, is looking for a motivated researcher/analyst to join our team. This position offers a unique chance to conduct software engineering research and publish on projects that apply intellectual rigor to real-life problems and have an impact on organizations' practices.

The qualified candidate will work in a collaborative, team-oriented environment. Work activities may include: Collection of quantitative and qualitative data through interviews, surveys, analyses of work products, etc.; designing solutions for storing and analyzing such data; designing and evaluating tools that relate such analyses to customer needs.

Candidates should have: A PhD, Master's, or equivalent degree in a field related to software engineering; good scientific and technical writing skills; familiarity with measuring the effectiveness of software- or system-development processes, practices, or tools; good communication skills and an ability to work effectively with experts from other teams and present research results to customers; good problem-solving skills.

For more information see http://fc-md.umd.edu/jobs

To apply please forward your CV to Forrest Shull (fshull@fc-md.umd.edu)

The Hong Kong Polytechnic University
Department of Computing

The Department invites applications for Professors/Associate Professors/Assistant Professors in Database and Information Systems / Biometrics, Computer Graphics and Multimedia / Software Engineering and Systems / Networking, Parallel and Distributed Systems. Applicants should have a PhD degree in Computing or closely related fields, a strong commitment to excellence in teaching and research as well as a good research publication record. Applicants with extensive experience and a high level of achievement may be considered for the post of Professor/Associate Professor. Please visit the website at http://www.comp.polyu.ed.hk for more information about the Department. Salary offered will be commensurate with qualifications and experience. Initial appointments will be made on a fixed-term gratuity-bearing contract. Re-engagement thereafter is subject to mutual agreement. Remuneration package will be highly competitive. Applicants should state their current and expected salary in the application. Please submit your application via email to hrstaff@polyu.edu.hk. Application forms can be downloaded from http://www.polyu.edu.hk/hro/job.htm. **Recruitment will continue until the positions are filled.** Details of the University's Personal Information Collection Statement for recruitment can be found at http://www.polyu.edu.hk/hro/jobpics.htm.

Internet2
Chief Technology Officer

Internet2 is the foremost U.S. Higher Education advanced networking consortium. Led by the research and education community since 1996, Internet2 promotes the missions of its members by providing both leading-edge network capabilities and unique partnership opportunities that together facilitate the development, deployment and use of revolutionary Internet technologies. Internet2 has an opportunity for a Chief Technology Officer (CTO) for our Research & Development area. The CTO reports directly to the Chief Executive Officer and directs the overall advanced technology program and activities of Internet2. Visit our website at www.internet2.edu/about/staff for complete details of this and other positions.

Send curriculum vitae in electronic mail format to: jobs@internet2.edu

Internet2
Attn: 08-073
1000 Oakbrook, Suite 300
Ann Arbor, MI 48104

Internet2 is a 501(C)3 not-for-profit organization and an equal opportunity employer.

University Corporation for Atmospheric Research
Scientist I

The Computational and Information Systems Laboratory (CISL) at the National Center for Atmospheric Research (NCAR) in Boulder, Colorado, seeks an individual to conduct research and development in the fields of computational science, scientific computing, high-performance computing, and computing systems as it relates to NCAR's mission. We are particularly interested in scientists with experience and expertise in the following areas: technologies and techniques for petascale computing, massively parallel algorithms, optimization for multi/many-core systems, accelerator technologies (e.g., GPGPUs and FGPAs), and software engineering for parallel applications.

Initial consideration will be given to applications received prior to Friday, March 13, 2009. Thereafter, applications will be reviewed on an as-needed basis. Apply online at www.ucar.edu (reference job #9028). *We value diversity.*

AA/EOE

Application materials should include: A statement of research interest; a Curriculum Vitae (CV); and a list of referees from whom we may expect letters of recommendation to be sent. Please ask referees to submit their letters to: SCI09T-DD@ucar.edu

View detailed job description at www.ucar.edu at the Jobs & Opportunities/Careers @ UCAR link.

THE HONG KONG POLYTECHNIC UNIVERSITY
香港理工大學

The Hong Kong Polytechnic University is the largest government-funded tertiary institution in Hong Kong, with a total student headcount of about 28,090, of which 14,260 are full-time students, 10,050 are part-time students, and 3,780 are mixed-mode students. It offers programmes at Doctorate, Master's, Bachelor's degrees and Higher Diploma levels. The University has 27 academic departments and units grouped under six faculties, as well as 2 independent schools and 2 independent research institutes. It has a full-time academic staff strength of around 1,300. The total consolidated expenditure budget of the University is in excess of HK$4 billion per year.

SCHOOL OF DESIGN

Professor / Associate Professor / Assistant Professor in Digital Media

The School of Design, as one of the top design schools in the world, is at the forefront of applying Asian innovation to global opportunities. The School is committed to sustaining excellence in design education, practice, consulting and research; to harnessing the legacy and dynamism of Asian cultures in creating solutions for human needs; and to creating strategic models for products, brands, and systems in local and global markets. The School offers a wide range of programmes at sub-degree, undergraduate and postgraduate levels in areas of Advertising Design, Digital Media, Environment and Interior Design, Industrial and Product Design, Visual Communication Design, Multimedia and Digital Entertainment, Interaction Design, Design Strategies and Practices. Its research and consultancy work are of an applied nature relevant to industrial, commercial and community needs. Please visit the website at http://www.sd.polyu.edu.hk for more information about the School.

The School is now inviting applications for a Professor / Associate Professor / Assistant Professor in Digital Media. The appointee will be in charge of the Multimedia Innovation Centre (MIC) at the School. MIC is an interdisciplinary centre dedicated to research, teaching, training, and outreach activities in the areas of Digital Media, Entertainment Technology, and Video Games. MIC's mission is to advance understanding in the design and development of new products and services in this high-innovation area. Drawing from the Centre's interdisciplinary resources, the appointee will be involved in all aspects of initiating and orchestrating the development of the Centre.

The appointee will be required to (a) oversee the mission, staffing matters and budget of MIC; (b) oversee the Master of Science in Multimedia and Entertainment Technology Programme and develop new programmes as opportunities arise; (c) contribute to teaching at the postgraduate and/or undergraduate levels in the area of Digital Media; (d) network with other institutes and experts to establish important partnerships, share information, and expand research and outreach endeavours; (e) cultivate collaboration with other disciplines, Schools and industry partners to develop new research initiatives; and (f) provide guidance on the application of multimedia technologies and design principles to education, research, and interdisciplinary projects.

Applicants should have (a) a relevant PhD degree plus at least five years' teaching or relevant working experience, OR a relevant master's degree plus at least eight years' teaching or relevant working experience preferably in university administration and leadership experience in the areas of Multimedia, Entertainment Technology, Digital Media Design or related disciplines; (b) a distinguished record of professional, scholarly and/or academic activities and significant background and record in scholarship and publication in Digital Media; (c) qualities of creativity, initiative and leadership; (d) a strong commitment to excellence in teaching, research and professional service.

Applicants with less experience may be considered for appointment at the level of Assistant Professor. The job duty requirements and expectations would be in line with the appointed grade. Applicants should submit a letter of interest and their portfolios including copies of 10 samples of their work in hardcopy, CD or memory stick format with a brief description of the work together with the completed application.

Remuneration and Conditions of Service

Salary offered will be commensurate with qualifications and experience. Initial appointment will be made on a fixed-term gratuity-bearing contract. Re-engagement thereafter is subject to mutual agreement. Remuneration package will be highly competitive. Applicants should state their current and expected salary in the application.

Application

Please submit application form via email to hrstaff@polyu.edu.hk; by fax at (852) 2764 3374; or by mail to **Human Resources Office, 13/F, Li Ka Shing Tower, The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong.** If you would like to provide a separate curriculum vitae, please still complete the application form which will help speed up the recruitment process. Application forms can be obtained via the above channels or downloaded from http://www.polyu.edu.hk/hro/job.htm. **Recruitment will continue until the position is filled.** Details of the University's Personal Information Collection Statement for recruitment can be found at http://www.polyu.edu.hk/hro/jobpics.htm.

NANYANG TECHNOLOGICAL UNIVERSITY

School of Physical and Mathematical Sciences

The Division of Mathematical Sciences (**http://www.spms.ntu.edu.sg/mas**) of the Nanyang Technological University (NTU), Singapore, is looking to add to its tenure-track faculty at all ranks. While we encourage strong candidates from all areas of **Theoretical Computer Science** to apply, we are particularly interested in the following areas:

- Computational Geometry
- Discrete Algorithms
- Computational Number Theory
- Network and Combinatorial Optimization
- Computational Complexity

NTU is a research university, with low teaching loads, excellent facilities, ample research funding and support for conference travel. The Division of Mathematical Sciences consists of active and talented faculty members working in a variety of areas. Its student body includes some of the best in the region. It offers undergraduate programs in mathematical sciences and mathematics & economics, and a graduate program awarding Masters and PhD degrees. Salary and benefits are competitive with the top universities around the world.

We seek people with excellent achievements in both research and teaching. Interested candidates are requested to send the following material to **MASrecruit@ntu.edu.sg**

- Application Letter
- Curriculum Vitae
- Research Statement
- Teaching Statement
- Names of at least three referees

Apart from the above faculty positions, there are also **16** three-to-five year research fellowships (from fresh post-docs to senior research fellows) available. Applicants with a strong background in **one of the areas** of Coding/Cryptography and Computational Mathematics/Image Processing/Computer Vision are encouraged to send their application letters, detailed CVs and the names/contact emails of two references to **ccrg_postdoc@ntu.edu.sg** (for Coding and Cryptography) or **compmath@ntu.edu.sg** (for Computational Mathematics/Image Processing/Computer Vision). In addition, **three to five** postdoc positions in Algorithmic Game Theory/Computational Social Choice are available. Interested applicants should send their detailed CVs, a research statement, and contact details of at least three referees to **Prof. Elkind (eelkind@ntu.edu.sg)**.

www.ntu.edu.sg

Group Term Life Insurance**

10- or 20-Year Group Term Life Insurance*

Group Disability Income Insurance*

Group Accidental Death & Dismemberment Insurance*

Group Catastrophic Major Medical Insurance*

Group Dental Plan*

Long-Term Care Plan

Major Medical Insurance

Short-Term Medical Plan***

Who has time to think about insurance?

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

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

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

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

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

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

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

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

AG5217

MARSH
Affinity Group Services
a service of Seabury & Smith

[CONTINUED FROM P. 112] our first meeting this February, in London. There are already lots of different computer societies in Europe, and they do a lot to energize and support the research base. We're trying to find out what we could do to help and collaborate.

What's your time frame for all of this?
We'd like to see the councils for China, India, and Europe set up and running their own meetings by the end of financial year '09, which technically finishes in June. We're also planning more events like the educational summit in China. Hopefully, we'll have something in India in 2010, and we're looking to have an event in Europe, as well. Then we've got to think about Central and South America, and Africa—it's a big world.

You've also been talking about growing ACM's membership.
ACM has had a steady growth, and we reckon we can get to an even 100,000. But when you think about it, there are hundreds of thousands of people—millions—who work in this area across the world. So we've just started a debate, which we're going to run this year: What would it mean if we tried to double our membership? How would it change ACM?

"What would it mean if we tried to double our membership? How would it change ACM?"

Sounds like there's a lot to be done.
There is a lot of responsibility, but it's also great fun. I've always enjoyed working with and for ACM, and because of its international role you feel that you can really have an impact.

What else is on the agenda?
The other big part of our agenda is improving the image of the field and the health of the discipline. This is all in collaboration with other organizations, like the National Science Foundation and members of the media. We've seen a dramatic drop in the numbers of people interested in careers in computing, and we're working on several projects to help turn this around.

Such as?
Public broadcaster WGBH, in Boston, does a lot to encourage young people to go into science and engineering. So we're working with them to look particularly at Latina and African-American girls. The Educational Policies Committee is also looking at ways to move computing and computer science into the mainstream of policy thinking.

On a more personal note, you were recently honored as Dame Commander. How did that feel?
It is, of course, a huge honor, and thrilling for me and my family. But I think it's also good for the computing community to have one of its own recognized in this way. C

Leah Hoffmann is Brooklyn-based technology writer.

DOI:10.1145/1498765.1498792

Leah Hoffmann

Q&A
Our Dame Commander

Wendy Hall discusses her plans to increase ACM's membership and to create task forces in China, India, and Europe.

A PROFESSOR OF computer science at the University of Southampton and the winner of numerous awards and honors, such as her recent appointment as Dame Commander of the Order of the British Empire, Wendy Hall was elected president of ACM in July 2008.

PHOTOGRAPH BY JON BANFIELD

You're the third female president of ACM, and the first non-North American president. How does that feel?
For me, it's more exciting that I'm the first non-North American president. A lot of times you don't like to do things just because you're a woman—you want to do stuff because you're the best person to do it, in the whole competitive field.

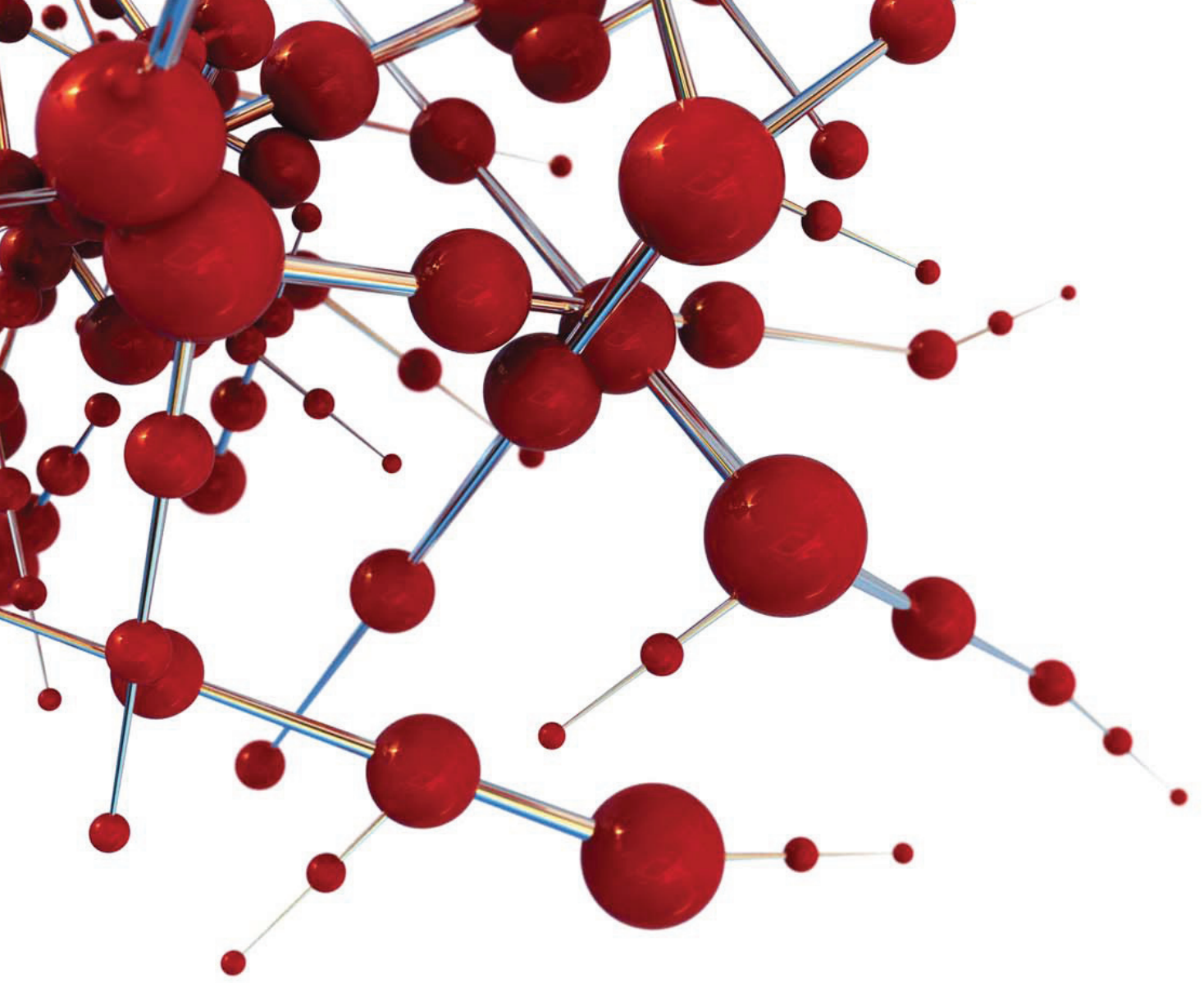
What are your plans for ACM?
ACM is in a good position, with 92,000 members and counting, and we had a fantastic year last year. But we mustn't be complacent. Broadly speaking, I want more people to join ACM, I want more women to join ACM. ACM is a U.S.-based organization, but it reaches out to the whole world through its publications and conferences.

What will you be doing to support those international members?
We're developing a series of task forces to explore what ACM can do in particular areas of the world—China, India, Europe. We need to ask: What can we do to support each of these cultures in their own context?

How will the task forces operate?
The task forces are geographically based, and [ACM CEO] John White is working very hard to get a good quality, diverse membership. The idea is that we start off with a task force, and as it matures, it will become a council in that region. We also want the task force chairs to come to New York and be a part of our discussions. It would be hopeless if it were always done at a distance.

What's been accomplished thus far?
In November, we held a summit largely aimed at Chinese educators. It was very successful, but we want to build on it. There's a huge amount of computing activity in China. At the moment our work is in Beijing—and we'll arrange a meeting of the ACM China Council later in the year—but we must expand it to Shanghai and Hong Kong and other regions. We also have an embryonic task force in India, which held its first meeting in early February.

And in Europe?
We, too, held [CONTINUED ON P. 111]



CHI 2009

DIGITAL LIFE NEW WORLD

Join us in Boston, MA for the 27th Annual CHI Conference, the premier international forum for all aspects of human-computer interaction. Computing is reaching into all parts of modern life. CHI 2009 brings together people working on the design, evaluation, implementation, and study of interactive computing systems for human use. CHI serves as a forum for the exchange of ideas among computer scientists, human factors scientists, psychologists, social scientists, system designers, usability professionals, and end users.

www.chi2009.org

acm Association for Computing Machinery

SIGCHI