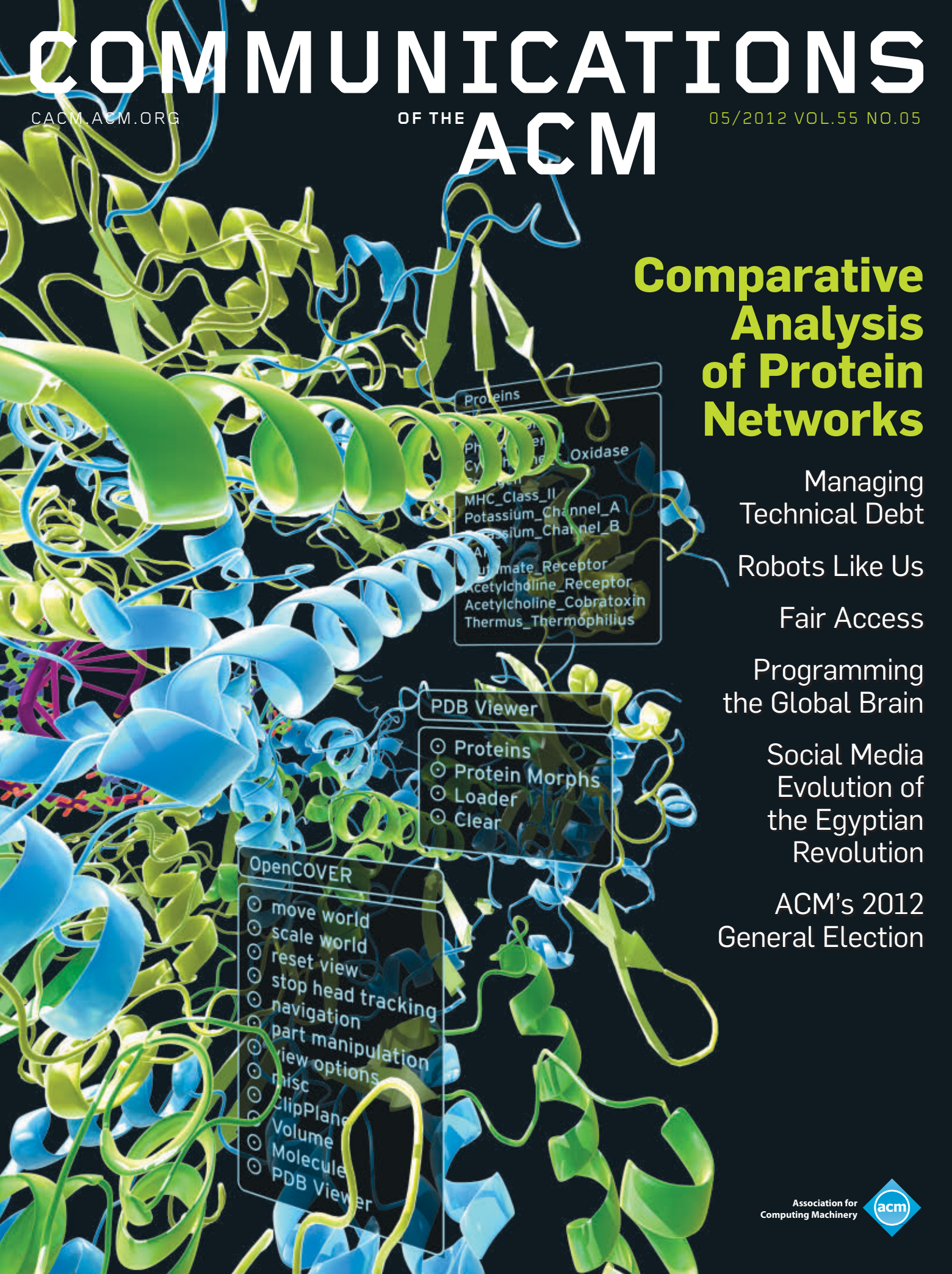
COMMUNICATIONS OF THE ACM

CACM.ACM.ORG

05/2012 VOL.55 NO.05



Comparative Analysis of Protein Networks

Managing Technical Debt

Robots Like Us

Fair Access

Programming the Global Brain

Social Media Evolution of the Egyptian Revolution

ACM's 2012 General Election

Association for Computing Machinery

SIGMOD/PODS 2012
CALL FOR PARTICIPATION

SIGMOD Int. Conference on Management of Data
SIGMOD/SIGACT/SIGART Symp. on Principles of Database Systems

Scottsdale, Arizona, USA - May 20-24, 2012
http://www.sigmod.org/2012/

SIGMOD/PODS Conference Events
- Research papers
- Industrial papers
- Technical demonstrations
- Keynotes
- Tutorials
- Panels on cutting-edge topics
- Undergraduate research papers
- New researcher symposium
- Industrial exhibits
- Recruitment events

Affiliated Workshops
- SIGMOD/PODS Ph.D. Symposium
- Database Mentoring Workshop (DBMe)
- Databases and Social Networks (DBSocial)
- Information Integration on the Web (IIWeb)
- Keyword Search on Structured Data (KEYS)
- Data Engineering for Wireless and Mobile Access (MobiDE)
- Scalable Workflow Enactment Engines and Technologies (SWEET)
- Web and Databases (WebDB)
- Data Management on New Hardware (DaMoN)
- Testing Database Systems (DBTest)
- Semantic Web Information Management (SWIM)

Target Audience
- Researchers
- Educators
- Students
- Practitioners
- System developers
- Application developers
- Recruiters

SIGMOD General Chairs: K. Selçuk Candan, Yi Chen
SIGMOD Program Chair: Luis Gravano
PODS General Chair: Maurizio Lenzerini
PODS Program Chair: Michael Benedikt

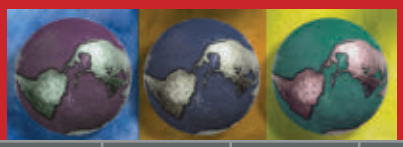
THE ACM A. M. TURING AWARD

by the community◆ from the community◆ for the community

ACM, Intel, and Google congratulate

JUDEA PEARL

for fundamental contributions to artificial intelligence through the development of a calculus for probabilistic and causal reasoning.



"Dr. Pearl's work provided the original paradigm case for how to do statistical AI. By placing structured knowledge representations at the heart of his work, and emphasizing how these representations enabled efficient inference and learning, he showed the field of AI how to build statistical reasoning systems that were actually telling us something about intelligence, not just statistics."

Limor Fix
Director, University Collaborative Research Group
Intel Labs

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

"Judea Pearl is the most prominent advocate for probabilistic models in artificial intelligence. He developed mathematical tools to tackle complex problems that handle uncertainty. Before Pearl, AI systems had more success in black and white domains like chess. But robotics, self-driving cars, and speech recognition deal with uncertainty. Pearl enabled these applications to flourish, and convinced the field to adopt these techniques."

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

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

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

COMMUNICATIONS OF THE ACM

Departments

Last Byte

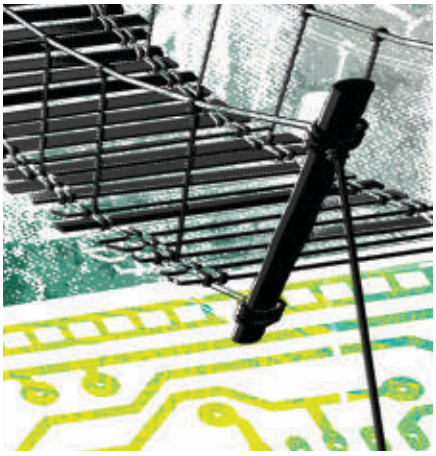
News

Viewpoints

PHOTOGRAPH BY HENRIK SCHARFE, AAU

05/2012
VOL. 55 NO. 05

Practice

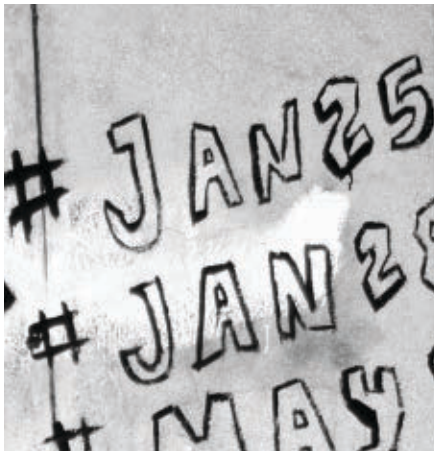


Articles' development led by acmqueue
queue.acm.org

About the Cover: Great strides have been made in the study of proteins—a key to unlocking the workings of a cell. Our cover story (p. 88) examines the problems and practical solutions of comparative analysis of protein networks. This month's cover was inspired by the interactive visualization of proteins from the Protein Data Bank performed by researchers from the University of California, San Diego. Illustration by James Gary.

ILLUSTRATION BY GARY NEILL. PHOTOGRAPH BY JONATHAN RASHAD

Contributed Articles



Review Articles

Research Highlights

Association for Computing Machinery
Advancing Computing as a Science & Profession

COMMUNICATIONS OF THE ACM

Trusted insights for computing's leading professionals.

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

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.

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

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

Single Copies
Single copies of *Communications of the ACM* are available for purchase. Please contact 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:10.1145/2160718.2160719

Moshe Y. Vardi

Fair Access

There has been sound and fury in the Open Access movement over the last few months. On December 16, 2011, The Research Works Act (RWA) was introduced in the U.S. House of Representatives. The bill contained provisions to prohibit open-access mandates for federally funded research, effectively nullifying the National Institutes of Health's policy that requires taxpayer-funded research to be freely accessible online. Many scholarly publishers, including the Association of American Publishers (AAP), expressed support for the bill.

The reaction to the bill and its support by scholarly publishers has been one of sheer outrage, with headlines such as "Academic Publishers Have Become the Enemies of Science." On January 21, 2012, renowned British mathematician Timothy Gowers declared a boycott on Elsevier, a major scholarly publisher, in a detailed blog posting. The boycott movement then took off, with over 8,000 scientists having joined it so far. By the end of February, Elsevier had withdrawn its support of RWA, and its sponsors in U.S. House of Representatives announced that RWA "has exhausted the useful role it can play in the debate."

Now that things have quieted down a bit, we can revisit the arguments for open access, which I discussed first in July 2009, in "Open, Closed, or Clopen Access?" The basic question I would like to address again is what ACM's stance should be with respect to open-access publishing models. (As pointed out in the July 2009 letter, ACM is a Green-Open-Access publisher.)

The arguments in favor of open access have evolved somewhat since the movement emerged about a decade ago. Early arguments talked about broad dissemination of information, under the slogan "information wants to be free." Recent arguments have focused on fairness, as expressed in the headline "Congress Considers Paywalling Science You Already Paid For." But one should not conflate the cost of carrying out research with the cost of publishing the results. Furthermore, universities and academic researchers profit from intellectual property developed under federal funding, and I see no outrage about that.

The real issue seems to me to be the divergence of interests between authors and publishers. Authors of scholarly articles would like to see them disseminated as broadly as possible, while publishers would like to cover the costs of publishing and, possibly, make some profit. This is a true divergence of interests. When it comes to commercial publishers, this divergence, in my opinion, simply cannot be bridged, which explains the outrage directed at Elsevier. As I said in 2009, "Why should for-profit corporations receive products and labor essentially for free?"

In the case of publishing by a professional association, such as ACM, the authors, as ACM members, are essentially also the publishers. Thus, it is up to the association to resolve the tension between the desire for the broadest possible dissemination and the need to cover the cost of publishing. Finding the right balance is one of the major issues the ACM Publication Board deals with on a regular basis. (See "ACM's Copyright Policy" in the October 2011 issue of *Communications*.) The principle guiding the Publication Board is that of *fair access*.

What is fair access? First and foremost, articles published by ACM are broadly available at a very reasonable cost. I recently computed the cost per download at my home institution for articles from the ACM Digital Library, and compared it to the analogous cost for computing-research articles from commercial publishers. The ACM articles were less expensive by at least two orders of magnitude (100x)! Second, pricing should also take into account the readers' ability to pay, which means there should be special pricing for students and institutions in developing countries. Finally, ACM allows self-archiving by authors, and has recently introduced the Author-Izer service, which enables ACM authors to generate and post links on either their personal page or institutional repository for visitors to download the definitive version of their articles from the ACM Digital Library at no charge.

In my opinion, fair access strikes a reasonable compromise between dissemination and covering publishing costs. Other compromises are also possible. For example, ACM could divert funds from other activities, or increase its membership dues and use the additional income to cover publishing costs. ACM could also adopt the author-pays model of most open-access publishers and ask authors to pay publishing fees to cover the cost of publishing. In either case, the decision is up to *us*: ACM members, authors, publishers.

Moshe Y. Vardi, EDITOR-IN-CHIEF

BLOG@CACM

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

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

DOI:10.1145/2160718.2160721 http://cacm.acm.org/blogs/blog-cacm

Likert-type Scales, Statistical Methods, and Effect Sizes

Judy Robertson writes about researchers' use of the wrong statistical techniques to analyze attitude questionnaires.

Judy Robertson
"Stats: We're Doing It Wrong"
http://cacm.acm.org/blogs/blog-cacm/107125
April 4, 2011

It is quite common for HCI or computer science education researchers to use attitude questionnaires to examine people's opinions of new software or teaching interventions. These are often on a Likert-type scale of "strongly agree" to "strongly disagree." And the sad truth is that researchers typically use the wrong statistical techniques to analyze them. Kaptein, Nass, and Markopoulos[3] published a paper in CHI last year that found that in the previous year's CHI proceedings, 45% of the papers reported on Likert-type data, but only 8% used nonparametric stats to do the analysis. Ninety-five percent reported on small sample sizes (under 50 people). This is statistically problematic even if it gets past reviewers! Here's why.

Likert-type scales give ordinal data. That is, the data is ranked "strongly agree" is usually better than "agree." However, it is not interval data. You cannot say the distances between "strongly agree" and "agree" would be the same as "neutral" and "disagree," for example. People tend to think there is a bigger difference between items at the extremes of the scale than in the middle (there is some evidence cited in Kaptein et al.'s paper that this is the case). For ordinal data, one should use nonparametric statistical tests (so 92% of the CHI papers got that wrong!), which do not assume a normal distribution of the data. Furthermore, because of this it makes no sense to report means of Likert-scale data—you should report the mode (entry which occurs most frequently in the dataset).

Which classic nonparametric tests should you use? I strongly recommend the flow chart on p. 274 of *How to Design and Report Experiments* by Field and Hole. This textbook is also pretty good for explaining how to do the tests in SPSS and how to report the results. It also mentions how to calculate effect sizes (see later).

Why is it so common to use parametric tests such as the T-test or ANOVA instead of nonparametric counterparts? Kaptein, Nass, and Markopoulos[3] suggest it is because HCI researchers know that nonparametric tests lack power. This means they are worried the nonparametric tests will fail to find a test where one exists. They also suggest it is because there aren't handy nonparametric tests that let you do analysis of factorial designs. So what's a researcher to do?

Robust Modern Statistical Methods

It turns out that statisticians have been busy in the last 40 years inventing improved tests that are not vulnerable to various problems that classic parametric tests stumble across with real-world data and which are also at least as powerful as classic parametric tests (Erceg-Hurn and Mirosevich[1]). Why this is not mentioned in psychology textbooks is not clear to me. It must be quite annoying for statisticians to have their research ignored! A catch about modern robust statistical methods is that you cannot use SPSS to do them. You have to start messing around with extra packages in R or SAS, which are slightly more frightening than SPSS, which itself is not a model of usability. Erceg-Hurn and Mirosevich[1] and Kaptein, Nass, and Markopoulos[3] both describe the ANOVA-type statistics, which are powerful and usable in factorial designs and works for nonparametric data.

A lot of interval data from behavioral research, such as reaction times, does not have a normal distribution or is heterscedastic (groups have unequal

variance), and so should not be analyzed with classic parametric tests either. To make matters worse, the tests that people typically use to check the normality or heterscedaticity of data are not reliable when both are present. So, basically, you should always run modern robust tests in preference to the classic ones. I have come to the sad conclusion that I am going to have to learn R. However, at least it is free and a package called nparLD does ANOVA-type statistics. Kaptein et al.'s paper gives an example of such analysis, which I am currently practicing with.

Effect Sizes

You might think this is the end of the statistical jiggery pokery required to publish some seemingly simple results correctly. Uh-uh, it gets more complicated. The APA style guidelines require authors to publish effect size as well as significance results. What is the difference? Significance testing checks to see if differences in the means could have occurred by chance alone. Effect size tells you how big the difference was between the groups. Randolph, Julnes, Sutinen, and Lehman[4], in what amounts to a giant complaint about the reporting practices of researchers in computer science education, pointed out that the way stats are reported by computer science education folk does not contain enough information, and missing effect sizes is one problem. Apparently it is not just us: Paul Ellis reports similar results with psychologists in *The Essential Guide to Effect Sizes*.

Ellis also comments that there is a viewpoint that not reporting effect size is tantamount to withholding evidence. Yikes! Robert Cole has a useful article, "It's the Effect Size, Stupid," on what effect size is, why it matters, and which measures one can use. Researchers often use Cohen's *d* or the correlation coefficient *r* as a measure of effect size. For Cohen's *d*, there is even a handy way of saying whether the effect size is small, medium, or big. Unfortunately, if you have nonparametric data, effect size reporting seems to get more tricky, and Cohen's way of interpreting the size of effect no longer makes sense (indeed, some people question whether it makes sense at all). Also, it is difficult for nonexperts to understand.

Common language effect sizes or probability of superiority statistics can solve this problem (Grissom[2]). It is "the probability that a randomly sampled member of a population given one treatment will have a score (y) that is higher on the dependent variable than that of a randomly sampled member of a population given another treatment (y2)" (Grissom[2]). An example from Robert Cole: Consider a common language effect size of 0.92 in a comparison of heights of males and females. In other words "in 92 out of 100 blind dates among young adults, the male will be taller than the female." If you have Likert-type data with an independent design and you want to report an effect size, it is quite easy. SPSS won't do it for you, but you can do it with Excel: PS = U/mn where *U* is the Mann-Whitney U result, *m* is the number of people in condition 1, and *n* is the number of people in condition 2 (Grissom[2]). If you have a repeated measures design, refer to Grissom and Kim's *Effect Sizes for Research* (2006, p.115). PSdep = w/n, where *n* is the number of participants and w refers to "wins" where the score was higher in the second measure compared to the first. Grissom[2] has a handy table for converting between probability of superiority and Cohen's *d*, as well as a way of interpreting the size of the effect.

I am not a stats expert in any way. This is just my current understanding of the topic from recent reading, although I have one or two remaining questions. If you want to read more, you could consult a forthcoming paper by Maurits Kaptein and myself in this year's CHI conference (Kaptein and Robertson[5]). I welcome any corrections from stats geniuses! I hope it is useful but I suspect colleagues will hate me for bringing it up. I hate myself for reading any of this in the first place. It is much easier to do things incorrectly.

Why is it so common to use parametric tests such as the T-test or ANOVA instead of nonparametric counterparts?

References

1. Erceg-Hurn, D. M. and Mirosevich, V. M. (2008). Modern robust statistical methods: an easy way to maximize the accuracy and power of your research. *The American psychologist, 63*(7), 591–601. doi: 10.1037/0003-066X.63.7.591.
2. Grissom, R. J. (1994). Probability of the superior outcome of one treatment over another. *Journal of Applied Psychology, 79*(2), 314–316. doi: 10.1037/0021-9010.79.2.314.
3. Kaptein, M. C., Nass, C., and Markopoulos, P. (2010). Powerful and consistent analysis of likert-type ratingscales. *Proceedings of the 28th international conference on Human factors in computing systems - CHI '10* (p. 2391). New York, NY: ACM Press. doi: 10.1145/1753326.1753686.
4. Randolph, J., Julnes, G., Sutinen, E., and Lehman, S. (2008). A Methodological Review of Computer Science Education Research. *Journal of Information Technology Education*, (7), 135–162. www.jite.org/documents/Vol7/JITEv7p135-162Randolph322.pdf.
5. Kaptein M. and Robertson, J. (In press). Rethinking Statistical Methods for CHI. Accepted for publication in CHI 2012, Austin, TX. http://judyrobertson.typepad.com/files/chi2012_submission_final.pdf

Reader's comment

By replacing ANOVA by nonparametric or robust statistics we risk ending up in another local maximum. Robust statistics are just another way to squeeze your data into a shape appropriate for the infamous "minimizing sum of squares" statistics. Those had their rise in the 20th century because they were computable by pure brainpower (or the ridiculously slow computers in those times).

If HCI researchers and psychologists would just learn their tools and acknowledge the progress achieved in econometrics or biostatistics. For example, linear regression and model selection strategies are there to replace the one-by-one null hypothesis testing with subsequent adjustment of alpha-level. With maximum likelihood estimation, a person no longer needs to worry about Gaussian error terms. Just use Poisson regression for counts and logistic regression for binary outcomes. The latter can also model Likert-type scales appropriately with meaningful parameters and in multifactorial designs.

Once you start discovering this world of modern regression techniques, you start seeing more in your data than just a number of means and their differences. You start seeing its shape and begin reasoning about the underlying processes. This can truly be a source of inspiration.

—Martin Schmettow

Judy Robertson is a lecturer at Heriot-Watt University.

ACM Member News

SARITA ADVE WINS WOMEN OF VISION AWARD

Sarita Adve, professor of computer science at the University of Illinois at Urbana-Champaign, recently won the Anita Borg Institute for Women and Technology's Women of Vision award in the category of Innovation. Her current work involves developing disciplined parallel programming practices and rethinking the multicore memory hierarchy to exploit this discipline.

Adve's interest in what she calls "undisciplined software and software-oblivious hardware" dates back 20 years to her Ph.D. thesis. At that time, the semantics of hardware and the hardware-software interface in parallel computing were ill-defined and difficult to understand. Different vendors used different semantics, which was tolerated because parallel computing was a niche. Those having a hardware-centric point of view used optimizations to make do, but the interface made little sense from a programmability perspective, Adve says. Her contribution was to regard the situation not as a hardware or a software problem, but as a hardware-software problem. "We needed to figure out what programmers want and have hardware agree to provide that in the interface. I proposed a contract; software developers would write programs in this well-structured way, which I formalized as data-race free programs, and the hardware side would provide sequential consistency, which is what programmers like."

Adve likes to choose hard, long-term problems, aiming for a solution that could fundamentally change the field. As for her students, she asks them to tackle underlying phenomena rather than tweaking a system to get a 10% improvement in performance. "Believe in what you do. To become passionate about your work, it has to be really fundamental. They go hand in hand."

—Karen A. Frenkel

PHOTOGRAPH BY KWAME ROSS

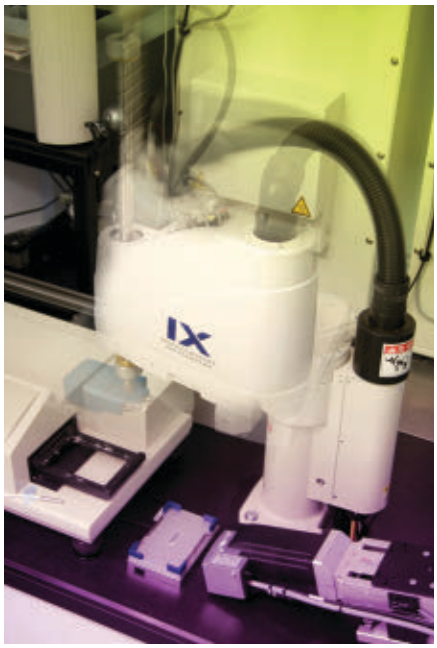
Science | DOI:10.1145/2160718.2160723 Neil Savage

Automating Scientific Discovery

Computer scientists are teaching machines to run experiments, make inferences from the data, and use the results to conduct new experiments.

The power of computers to juggle vast quantities of data has proved invaluable to science. In the early days, machines performed calculations that would have taken humans far too long to perform by hand. More recently, data mining has found relationships that were not known to exist—noticing, for instance, a correlation between use of a painkiller and incidence of heart attacks. Now some computer scientists are working on the next logical step—teaching machines to run experiments, make inferences from the data, and use the results to perform new experiments. In essence, they wish to automate the scientific process.

One team of researchers, from Cornell and Vanderbilt universities and CFD Research Corporation, took a significant step in that direction when they reported last year that a program of theirs had been able to solve a complex biological problem. They focused on glycolysis, the metabolic process by which cells—yeast, in this case—break down sugars to produce energy. The team fed their algorithm with experimental data about yeast metabolism, along with theoretical models in the form of sets of equations that could fit the data.



Adam, a robotic system, operates biological experiments on microtiter places.

PHOTOGRAPH COURTESY OF ABERYSTWYTH UNIVERSITY

The team seeded the program with approximately 1,000 equations, all of which had the correct mathematical syntax but were otherwise random. The computer changed and recombined the equations and ranked the results according to which produced answers that fit the data—an evolutionary technique that has been used since the early 1990s. The key step, explains Hod Lipson, associate professor of computing and information science at Cornell, was to not only rank how the equations fit the data at any given point in the dataset, but also at points where competing models disagreed.

"If you carefully plan an experiment to be the one that causes the most disagreement between two theories, that's the most efficient experiment you can do," says Lipson. By finding the disagreements and measuring how much error the different equations produced at those points, the computer was able to refine the theoretical models to find the one set of equations that best fit the data. In some cases, Lipson says, the technique can even develop a full dynamical model starting from scratch, without any prior knowledge.

Lipson and then-doctoral student Michael Schmidt started developing Eureqa, the algorithm the work was

based on, in 2006. Three years later, they published a paper in *Science*, describing how Eureqa took measurements of the complex motions of a double pendulum and used the evolutionary process to derive Hamiltonian and Lagrangian differential equations that describe the laws of motion.

Lipson's hope is that this approach can expand human capabilities in science. "In classical experimentation, you vary one parameter at a time," he says. A computer can alter all the parameters at once to find the best experiment. "If you have 18 variables, there's no way you can conceptualize that, but the machine can."

The research does not knock humans out of scientific discovery. "The expectation would be that you find the model fast, and you are able to look deeper into your data," Lipson says. "In the end, you do have to have an expert to look at these equations and ponder their meaning and decide whether they're actually useful."

Robots as Principal Investigator

Increasingly, scientists are enlisting the help of robots, such as the ones used in high-throughput drug discovery, to manipulate materials and gather data. "The concept is to try to automate the cycle of scientific research, which involves forming hypotheses and being able to test them," says Ross King, who headed the computational biology group at Aberystwyth University but recently moved to the Manchester Interdisciplinary Biocenter at the University of Manchester where he is a professor of machine intelligence. He used a robotic system he calls Adam to run biological experiments on microtiter plates that hold miniscule amounts of samples, and can run several different experiments within millimeters of each other. Adam compared growth rates of natural yeast cells with a series of genetically altered cells, each of which was missing a different gene, and was able to identify the function of different genes.

Why not enlist a robot to generate data, form a hypothesis, and design experiments to test its hypothesis?

King says robots have already surpassed humans' capabilities in the physical aspects of the experiment, such as the placement of different yeast strains in miniscule holes on the microtiter plates. So why not have a robot conduct the entire experiment—generate data, form a hypothesis, design new experiments to test the hypothesis, and then work with the new data?

The system is not quite that advanced yet, King admits. "I don't think Adam is capable of generating a hypothesis that is cleverer than what a human could do," he says. "I think in maybe 20 years there will be robots able to do almost any experiment in the wet biology lab."

"We're a long way from automating the entire scientific enterprise," says Patrick Langley, head of the Cognitive Systems Laboratory at Stanford University. Langley has argued for the validity of computational discovery since the early 1980s, when he and the late Herbert Simon, a pioneer in artificial intelligence, first wrote about an early scientific discovery program called BACON. The work by Lipson and King, he says, is well within that tradition.

"You can view discovery as a search though a space of laws and models," says Langley. "Historically, people had to search that space themselves." There is no reason, he says, people should not hand off some of that searching to machines, just as they handed off tasks such as focusing astronomers' telescopes. "They'll let scientists be more productive and make more rapid progress."

It is not merely the vast mountains of data the sciences are producing these days that requires computational assistance, Langley explains. There may be plenty to discover in the social sciences, but these are complex systems, with numerous variables often interacting in subtle ways. "They are all really complicated," says Langley, "and we will really need computational tools to go after them."

But automating science also in-

Theory

P vs. *NP* Poll Results

Does *P* = *NP*? Not according to the great majority of computer scientists who responded to the latest opinion poll conducted by William Gasarch, professor of computer science at the University of Maryland, College Park. Gasarch recently polled 150 researchers about the largest unsolved problem in computer science, and found that 81% do not believe *P* = *NP*. In 2002, Gasarch polled 100 researchers on the *P* vs. *NP* problem, and found that 61% did not believe *P* = *NP*, 9% believed *P* = *NP*, and 30% said they did not know or the problem was unsolvable.

Increasingly, researchers do not believe *P* vs. *NP* will be solved anytime soon. In the 2002 poll, 62% of the respondents said an answer would appear before 2100. In the latest poll, the number of respondents who believe so has shrunk to 53%.

"Even though the percentage of people who think it will be solved before 2100 has gone down, I am still surprised it is as high as it is," Gasarch said in an email interview. "People's optimism about the problem surprises me and is significant."

Gasarch believes a proof will take 200–400 years. "My opinion, and frankly anyone's opinion, is really a guess. Having said that, my impression is that we are really at a standstill. There has been no progress and there is no real plan of attack. Geometric complexity theory may be an exception, but that will also take a long time to prove anything."

As for Gasarch, he does not believe *P* = *NP*.

—*Jack Rosenberger*

volves formidable challenges, such as how to scale up algorithms like Eureqa to handle more complex models. While Langley considers Lipson's work to be technically solid, it deals with only a seven-dimensional model. What happens if the computer has to deal with hundreds of equations, all interacting with one another? The computer's runtime could grow exponentially with the complexity of the model. Computer scientists need to develop methods to deal with much more complex models, Langley says.

A Revolution in Science?

It is not difficult to envision how computers could speed up scientific discovery by plowing through vast arrays of data and "spitting out hypotheses" for testing, suggests Bruce Buchanan, professor of computer science, philosophy, and medicine at the University of Pittsburgh. In the 1960s, Buchanan was involved in Dendral, an early scientific discovery computer program that used mass spectrometry to identify unknown organic molecules. Simply sorting out the interesting molecules from vast arrays of information, the way Google identifies the most popular results of a search query, would be useful. "There are so many cases where discoveries could have been made sooner if people were looking at different data, asking better questions," Buchanan says.

The larger question is whether computers could ever go beyond what philosopher Thomas Kuhn called "normal science" and achieve a real scientific revolution. Elihu Abrahams, a condensed matter physicist at Rutgers University, and Princeton University physicist Philip Anderson, expressed their doubts in a letter to *Science* after its publication of separate computational discovery papers by Lipson and King in 2009. "Even if machines did contribute to normal science," they wrote, "we see no mechanism by which they could create a Kuhnian revolution and thereby establish new physical law." Abrahams says he has seen nothing since then to change his mind.

King admits the field has not yet reached that point. Adam "knows far more facts than any human, but it doesn't have a deep understanding of biology," he says. "I think you need a deep understanding to actually manipulate concepts in novel ways."

Computational discovery can certainly make science more efficient and cost effective, and free up scientists to do more deep thinking, he argues. But for machines, King says, "the things which the great scientists do are still a long way off." C

Robots will be able to conduct almost any experiment in a wet biology lab in about 20 years, says Ross King.

Further Reading

Bridewell, W. and Langley, P.
Two Kinds of Knowledge in Scientific Discovery, *Topics in Computer Science 2*, 1, Jan. 2010.

Dzeroski, S. and Todorovski, L. (Eds.)
Computational Discovery of Communicable Scientific Knowledge. Springer, Berlin, Germany, 2007.

King, R., Liakata, M., Lu, C., Oliver, S., and Soldatova, L.
On the formulation and reuse of scientific research, *Journal of the Royal Society Interface 8*, 57, April 2011.

Langley, P.
The computational support of scientific discovery, *International Journal of Human-Computer Studies 53*, 3, Sept. 2000.

Schmidt, M., Vallabhajosyula, R., Jenkins, J., Hood, J., Soni, A., Wikswo, J., and Lipson, H.
Automated refinement and inference of analytical models for metabolic networks, *Physical Biology 8*, 5, Oct. 2011.

Waltz, D. and Buchanan, B.G.
Automating science, *Science 324*, 5923, April 3, 2009.

Neil Savage is a science and technology writer based in Lowell, MA.

Technology

Cloud to Create Jobs Boom

Instead of causing massive unemployment, cloud computing will generate 13.8 million jobs worldwide by the end of 2015, according to a new study from IDC.

Given its potential as an outsourcing catalyst, the cloud has often been linked to future staffing cuts. But the study, which is sponsored by Microsoft, contends that the cloud will fuel growth initiatives by greatly reducing "legacy drag"—the maintenance of legacy systems and routine upgrades—which now accounts for 75% of technology spending.

"Most [CIOs] look at migration to cloud computing as a way to free up existing resources to work on more innovative projects, not to do away with the positions entirely," according to the study, which was written by the IDC research team of John F. Gantz, Stephen Minton, and Anna Toncheva.

IDC came up with its projections by analyzing cloud-spending patterns in more than 40 nations and combining this with data such as available country work force, unemployment rates, and other IT spending developments.

Additional study results:

- **Cloud-related business revenue could increase to $1.1 trillion by 2015.**
- **Small and medium-sized companies will establish the most jobs, with 7.5 million new jobs within this time. Why? Because they are less impacted by legacy drag and are more likely to invest in public cloud services due to capital constraints.**
- **China and India will generate 6.75 million jobs by the end of 2015. In contrast, less than 1.2 million positions will be created in North America.**
- **Public-sector opportunities will grow, but at a much slower pace with just more than 500,000 jobs. Global governments, the authors write, will lag behind because of their "slow pace [in] adopting the new computing style."**

—Dennis McCafferty

Technology | DOI:10.1145/2160718.2160724 Alex Wright

Robots Like Us

Thanks to new research initiatives, autonomous humanoid robots are inching closer to reality.

THE SCIENCE FICTION ROBOTS of our youth always looked reassuringly familiar. From the Iron Giant to C-3PO, they almost invariably sported two arms, a torso, and some type of metallic head.

Real-world robots, alas, have largely failed to live up to those expectations. Roombas, factory assembly arms, and even planetary rovers all seem a far cry from the Asimovian androids we had once imagined.

That may be about to change, however, thanks to a series of engineering advances and manufacturing economies of scale that could eventually bring autonomous humanoid robots into our everyday lives. Before Rosie the Robot starts clearing away the dinner dishes, however, roboticists still need to overcome several major technical and conceptual hurdles.

In June 2011, U.S. President Obama announced the National Robotics Initiative, a $70 million effort to fund the development of robots "that work beside, or cooperatively with people." The government sees a wide range of potential applications for human-friendly robots, including manufacturing, space exploration, and scientific research.

While roboticists have already made major strides in mechanical engineering—designing machines capable of walking on two legs, picking up objects, and navigating unfamiliar terrain—humanoid robots still lack the sophistication to work independently in unfamiliar conditions. With the aim of making robots that can function more autonomously, some researchers are honing their strategies to help robots sense and respond to changing environments.

Honda's well-known ASIMO robot employs a sophisticated array of sensors to detect and respond to external conditions. When ASIMO tries to open a thermos, for example, it uses its sensors to gauge the shape of the object, then chooses a sequence of actions appropriate to that category of object. ASIMO can also sense the contours of the surface under its feet and adjust its gait accordingly.

The newest Geminoid robot is modeled after project collaborator Henrik Scharfe.

Negotiating relationships with the physical world may seem difficult enough, but those challenges pale in comparison to the problem of interacting with some of nature's most unpredictable variables: namely, human beings.

"People and robots don't seem to mix," says Tony Belpaeme, reader in Intelligent Systems at the University of Plymouth, whose team is exploring new models for human-robot interaction.

The problem seems to cut both ways. On the one hand, robots still have trouble assessing and responding to human beings' often unpredictable behavior; on the other hand, human beings often feel uncomfortable around robots.

Roboticists call this latter phenomenon the "uncanny valley"—the sense of deep unease that often sets in when human beings try to negotiate relationships with human-looking machines. The more realistic the machine, it seems, the more uncomfortable we become.

"Robots occupy this strange category where they are clearly machines, but people relate to them differently," says Bill Smart, associate professor of computer science and engineering at Washington University in St. Louis. "They don't have free will or anima, but they seem to."

Perhaps no robot has ever looked so uncannily human as the Geminoids, originally designed by Osaka University professor Hiroshi Ishiguro in collaboration with the Japanese firm Kokoro. To date the team has developed three generations of successively more lifelike Geminoids, each modeled after an actual human being.

The latest Geminoid is a hyper-realistic facsimile of associate professor Henrik Scharfe of Aalborg University, a recent collaborator on the project. With its Madame Tussaud-like attention to facial detail, the machine is a dead ringer for Scharfe. But the Geminoid's movement and behavior, while meticulously calibrated, remains unmistakably, well, robotic.

In an effort to create more fluid human-like interactions, some researchers are starting to look for new perspectives beyond the familiar domains of computer science and mechanical engineering, to incorporate learning from psychology, sociology, and the arts.

Belpaeme's CONCEPT project aspires to create robots capable of making realistic facial expressions by embracing what he calls "embodied cognition." For robots to interact effectively with us, he believes, they must learn to mimic the ways that human intelligence is intimately connected with the shape and movements of our bodies.

To that end, Belpaeme's team is developing machine learning strategies modeled on the formative experiences of children. "Our humanoid robots can be seen as young children that learn, explore, and that are tutored, trained, and taught," he says. The robots learn that language by interacting directly with people, just as a child would.

In order to make those learning exchanges as smooth as possible, the

PHOTOGRAPH BY HENRIK SCHARFE, AAU

team came up with an alternative solution to a mechanical face, instead creating a so-called LightHead that relies on a small projector emitting an image through a wide-angle lens onto a semi-transparent mask. Using a combination of open-source Python and Blender software, the team found it could generate realistic 3D facial images extremely quickly.

"We are very sensitive to the speed with which people respond and with which the face does things," Belpaeme says. "A robot needs to bridge the gap between the internal digital world and the analogue of the external world," he explains. "If a user claps his hands in front of the robot's face, you expect the robot to blink."

Using Psychological Methods

Bill Smart's team at Washington University is taking a different tack to the problem of human-robot interaction, applying psychological methods to explore how human beings react to robots in their midst.

For example, the team has learned that people tend to regard a robot as more intelligent if it seems to pause and look them in the eye. These kinds of subtle gestures can help humans form a reliable mental model of how a robot operates, which in turn can help them grow more comfortable interacting with the machine.

"If we're going to have robots and have them interact, we need a model of how the robot 'thinks,'" says Smart. "We need to design cues to help people predict what will happen, to figure out the internal state of the system."

Smart's team has also pursued a collaboration with the university's drama department, and recently hosted a symposium on human-robot theater. "Theater has a lot to say, but it's hard to tease it out. We don't speak the same language."

Actors provide particularly good role models for robots because they are trained to communicate with their bodies. "If you look at a good actor walking across the stage, you can tell what he's thinking before he ever opens his mouth," says Smart. For actors, these are largely intuitive processes, which they sometimes find difficult to articulate. The challenge for robotics engineers is to translate those expressive impulses into workable algorithms.

Two related problems: Robots have trouble assessing and responding to humans' often unpredictable behavior; humans often feel uncomfortable around robots.

In a similar vein, Heather Knight, a doctoral candidate at Carnegie Mellon University's (CMU's) Robotics Institute, has pursued a collaboration with the CMU drama department to create socially intelligent robot performances. Lately she has entertained stage audiences with Data, a cartoonish humanoid robot with a stand-up routine that she has honed through observation of working comedians in New York City.

"What's new about these robots is that they're embodied," says Knight. "We immediately judge them in the way we do people. Not just in physical terms, but also in terms of how they move and act."

In order to help audiences feel more at ease around her robot, she has found that accentuating the physical differences and making the robot even more "robotic" helps people relate to Data more comfortably.

Another sophisticated cartoonish robot named TokoTokoMaru has recently entertained YouTube audiences with its precise rendition of the famously demanding Japanese Noh dance. Created by robot maker Azusa Amino, the robot relies on aluminum and plastic parts with Kondo servomotors in its joints to create its sinuous dance moves.

Azusa had to make the robot as lightweight as possible while maximizing the number of controlled axes to allow for maximal expression. "Robots have fewer joints than humans, and they can't move as fast," he explains, "so to mimic the motions of human dance, I made a conscious effort to move many joints in parallel and move the upper body dynamically."

Beyond the technical challenges, however, Azusa also had to step into the realm of aesthetics to create a "Japanese-style robot" with a strong sense of character. To accentuate its Japaneseness, the robot wields delicate fans in both hands, while its hair blows in a gentle breeze generated by a ducted fan typically used in radio control airplanes.

While cartoonish appearances may help make a certain vaudevillian class of robots more palatable to theater audiences, that approach might wear thin with a robot designed for everyday human interaction. Take away the wig and the comedy routine, after all, and you are still left with a robot struggling to find its way across the uncanny valley.

"There is something about a thing with two arms and a head that triggers certain affective responses in people," says Smart. "We have to understand how that works to make them less scary." C

Further Reading

Delaunay, F., de Greeff, J., and Belpaeme, T.
A study of a retro-projected robotic face and its effectiveness for gaze reading by humans, *Proceeding of the 5th ACM/IEEE international conference on Human-robot interaction*, Osaka, Japan, March 2–5, 2010.

Knight, H.
Eight lessons learned about non-verbal interaction through investigations in Robot, International Conference on Social Robotics, Amsterdam, The Netherlands, Nov. 24–25, 2011.

Morse, A., De Greeff, T., Belpaeme, T., and Cangelosi, A.
Epigenetic Robotics Architecture (ERA), *IEEE Transactions on Autonomous Mental Development 2*, 4, Dec. 2010.

Park, I., Kim, J., Lee, J., and Oh, J.
Mechanical design of the humanoid robot platform, HUBO, *Advanced Robotics 21*, 11, Nov. 2007.

Smart, W., Pileggi, A., and Takayama, L.
***What do Collaborations with the Arts Have to Say About Human-Robot Interaction?* Washington University in St. Louis, April 7, 2010.**

Alex Wright is a writer and information architect based in Brooklyn, NY.

Society | DOI:10.1145/2160718.2160725 Samuel Greengard

Digitally Possessed

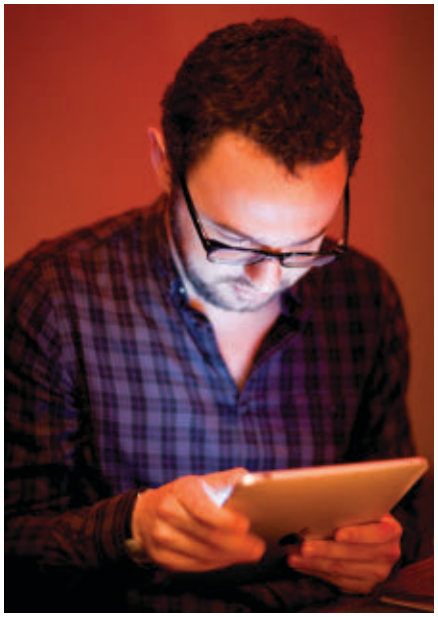
Virtual possessions play an increasingly important role in our daily lives. How we think about them and deal with them is changing the way we think and interact with others.

It is nine o'clock in the evening and a young woman, nestled in a sofa in her living room, is uploading photographs to Facebook and listening to music streaming from her iPad. After a few minutes, she flicks and pinches the screen to navigate to an ebook that she begins reading. An hour later, the woman clicks into the online game FarmVille to check on her cows, chickens, and cornstalks and see how much money she has accumulated since last logging on a couple of days ago.

Today, there is nothing incredibly remarkable about this scenario—unless you consider that every item the woman has viewed, heard, and interacted with resides entirely within a virtual world. She will never actually touch the photos, handle a CD, or feel the currency she has earned in FarmVille. These possessions do not exist in the physical world. They are simply bits and bytes that have been arranged to look and sound like actual objects.

Virtual possessions are changing our world—and our perception of reality. Today, ebooks outsell paperbound books, digital downloads have surpassed CDs in sales, and more than 2.5 billion people around the world use digital cameras to snap a seemingly endless array of photos. Meanwhile, video and audio streaming services such as Netflix, Pandora, and Spotify deliver content on demand, and social media sites like Facebook document our lives in a way that would have been unimaginable only a few years ago.

"Our lives are becoming increasingly digital... more of the things we possess reside inside computers," states Janice Denegri-Knott, a senior lecturer in consumer culture and behavior at Bournemouth Media School. "As possessions become virtual, we think about them and interact with them in entirely different ways. We find new ways to assign meaning to them and incorporate them into our lives."



How we define ourselves is changing as the nature of our possessions changes.

Virtual Objects Take Shape

For nearly half a century, researchers have studied the way people view possessions. What is clear is that possessions help define who we are and what story we present to other people. What is easily overlooked, says Susan Kleine, associate professor of marketing at Bowling Green University, is that virtual possessions are very real—even if they reside only within computers. "There is a psychological appropriation that takes place as an individual connects to an object, item, or idea," she points out.

In the past, bookcases, record racks, and framed photos strategically positioned around a house or office told the story of how we think, what and whom we consider important, and, ultimately, what type of person we consider ourselves. However, to a certain extent, physical objects such as vinyl LPs and photographic paper have always been a way to package content and put it into our hands.

Today, how we define ourselves is changing as the nature of possessions changes. Instead of creating a carefully constructed library in our house, we assemble a digital library on Goodreads.com and share it with others. Instead of creating photo albums, we post pictures on Facebook and YouTube. It is estimated that Facebook users upload 2.7 million photos every 20 minutes. The average person has about 350 pictures at the site.

Further complicating things, entirely new artifacts exist. This includes online gaming avatars; instant message and text message streams; email; virtual currency and social media feeds, badges and awards. Increasingly, these things define our lives and our legacy. "They serve as psychological anchor points for our self-narratives, like souvenirs collected during the course of a journey," observes Vili Lehdonvirta, an economic sociologist who is a visiting fellow at the London School of Economics.

Nowhere is this transformation more apparent than among young people who have grown up in a digital world and, in some cases, cannot relate to the physical objects of the past. "A digital photo or song has value for what it is, but also for what you can do with it," says John Zimmerman, associate professor of human-computer interaction and design at Carnegie Mellon University. "The ability to tag a digital item, comment on it, and share it makes it more meaningful to some people."

In fact, when Zimmerman and other researchers studied a group of 21 teenagers (12 girls and nine boys) in 2011, he found there is a growing belief that digital possessions are more valuable than physical possessions—and that the social interaction surrounding online objects can dramati-

PHOTOGRAPH BY MATT BIDDULPH

cally increase their worth. One girl reported that snapping a large number of photos and posting them online felt like a more honest representation of an event because participants could comment and interact afterward. "There's a shared sense of what happened," she notes.

Teenagers also like the fact they can access content online—photos, music, comments, and badges—at any time and from almost any place, thanks to Facebook and services such as Apple's iCloud, says Zimmerman. Finally, by putting their possessions online, they have a space that is largely free of scrutiny from their parents. This means, among other things, that they can display a photo of a boyfriend or girlfriend online—whereas they might not be allowed to place a picture frame with the same photo in their bedroom. "In some cases, a virtual environment creates a more satisfying and holistic experience," Zimmerman says.

A Changing Picture

It is tempting to think about digital possessions as ephemeral and easily duplicated, Lehdonvirta points out. What is more, because virtual objects do not register with our senses in the way physical objects do, they are not particularly useful in establishing social distinctions. However, the introduction of ubiquitous and always on computing networks—and the widespread adoption of social media—has altered the equation. "This introduces the economics and sociology of ownership into digital goods," he observes.

In *World of Warcraft*, for instance, the publisher creates artificial scarcity by periodically adding 10 new levels to the system to devalue everyone's status. This virtual form of inflation, or artificially imposed scarcity, ensures players will stay engaged and spend more money. Meanwhile, at sites such as Facebook and Twitter, value derives not only from objects and content but also from amassing "friends" and "followers." Lehdonvirta says there have been attempts to trade social media votes for actual money and one company recently sued an ex-employee over the ownership of followers on Twitter. In January 2012, the concept of virtual ownership was affirmed by the Dutch Supreme Court, which upheld a conviction of a boy who stole another boy's possessions in the online game Runescape.

"People not only view a Facebook account as a digital possession, they look at it as a digital storage locker for their lives," says Amber Cushing.

In fact, virtual money and possessions are not an entirely new concept. For years, society has used virtual funds, such as credit and debit cards, to handle transactions and pay for goods. These virtual currencies were not created by government but rather by private organizations.

A bigger question is how virtual possessions influence our thinking and change our behavior. In this new virtual order, a Netflix instant queue becomes a to-do list rather than a personal collection of DVDs. A Kindle book cannot be loaned to a pal or passed on to a child. And so-called friends are people we may have never met or will never interact with in the physical world.

Amber Cushing, a doctoral candidate at The School of Information and Library Science at The University of North Carolina at Chapel Hill, has interviewed dozens of people of varying ages and backgrounds and found there is a common thread in the way people view digital possessions. They define a person's identity; inform others about what is happening in their life; create value or status in some way; and provide a sense of bounded control that may not be possible in the physical world.

The popularity of Facebook, which now claims 845 million active users worldwide, is a perfect example of this thinking. There is a growing desire to document and archive digital experiences, Cushing explains. "People not only view a Facebook account as a digital possession, they look at it as a digi-

Career

Women and IT Jobs

Education advocates have long highlighted the large discrepancy between male and female students studying technical fields such as computer science and mathematics, but a new Anita Borg Institute for Women and Technology (ABI) report, "Solutions to Recruit Technical Women," highlights the challenges the small number of women graduating with technical degrees encounter when they enter the job market.

"There's a pipeline problem," says Denise Gammal, coauthor of the report and director of corporate partnerships at ABI. "We have only about 20% women in the computer science field [as students], and once women get into companies, they encounter a lot of cultural and other barriers. There's a lot of implicit biases that make it difficult for women to get into companies and also make it difficult for women to advance in those companies."

Based on interviews with some of the world's leading technology companies, Gammal and her colleagues were able to identify four key areas where companies are failing women.

First, companies often do not reach enough schools to include many women in their recruitment process and are not fully utilizing social networks to link female prospects to vacant positions. Second, many job descriptions are unwittingly biased toward male prospects. Third, women are underrepresented in most search committees. Finally, companies need to identify ways to retain and promote female talent to deter them from dropping out of the field.

The ABI report focuses heavily on offering best practices for companies. For instance, companies should diversify their hiring committee during the recruitment phase to include different genders and cultures and institute a blind résumé screening process. The report also suggests creating gender-balanced internships to build strong and diverse future work forces.

—Graeme Stemp-Morlock

tal storage locker for their lives. It's a place where they are able to put things they want to retain and where they can reflect on their identity."

Yet, it is also clear that digital possessions create their own set of challenges and frustrations. These range from people deleting photos and tags on a social media site to enduring a hard drive crash and the loss of an entire music collection or photo library, if the data is not backed up. In addition, there is the risk of a device manufacturer or service provider vanishing and taking the content with it, as well as file formats and codecs becoming obsolete and thus rendering old audio recordings or videos unplayable.

Separating co-owned digital items can also prove daunting. In the past, when a couple broke up, dividing a stack of books or CDs was a fairly straightforward proposition. However, divvying up digital content that may reside on multiple computers and have DRM restrictions is vexing. Trying to sort out who controls a jointly owned *World of Warcraft* account or other online content—accessed through shared passwords—can confound and frustrate even the most level-headed individuals.

Not surprisingly, researchers are examining ways to improve how we manage virtual possessions. For example, Zimmerman hopes to encode metadata into virtual things. This might encompass status updates, favorite songs associated with a particular event along with news and weather information. It is also important to view digital spaces differently, he argues. Instead of using a digital picture frame to display hundreds or thousands of random photos, for example, it is possible to dedicate a frame to one person or a particular theme and use multiple frames.

The boundaries and distinctions between digital and physical possessions are blurring as virtual objects become more commonplace.

Kleine says there is also a growing desire to create physical manifestations of virtual things. The popularity of scrapbooking and the yearning to assemble online photo books and print them for posterity is no accident, she says. There is also an evolving desire to capture screen shots of online badges, trophies, and other creations. And 3D printing, which is advancing rapidly, is bridging the digital and physical worlds by giving virtual objects, such as toys, trophies, and characters in games, actual physical form.

In the end, perhaps only one thing is entirely clear: The boundaries and distinctions between digital and physical possessions are blurring as virtual objects become more commonplace. As a result, our thinking and behavior is changing... yet remaining much the same. "Humans have a psychological need to attach to things and hang on to them," says Kleine. "Regardless of whether an object exists in the digital or physical realm, there's a need to feel that it is tangible and real." C

Further Reading

Johnson, M., Lehdonvirta, V., and Wilska, T.
Virtual Consumerism, *Information, Communication & Society 12*, 7, Oct. 2009.

Kleine, S., Kleine, R., and Allen, C.
How is possession "me" or "not me"? Characterizing types and an antecedent of material possession attachment, *Journal of Consumer Research 22*, 3, Dec. 1995.

Lehdonvirta, V.
Consumption, Turku School of Economics, No. A-11, 2009.

Odom, W., Forlizzi, J., and Zimmerman, J.
Virtual possessions, *Proceedings of the 8th ACM Conference on Designing Interactive Systems*, Aarhus, Denmark, August 16–20, 2010.

Odom, W., Forlizzi, J., and Zimmerman, J.
Teenagers and their virtual possessions: Design opportunities and issues, *Proceedings of the 2011 Annual Conference on Human Factors in Computing Systems*, Vancouver, BC, Canada, May 7–12, 2011.

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

Milestones

Computer Science Awards

The National Inventors Hall of Fame and the Computing Research Association recently honored leading computer scientists.

NATIONAL INVENTORS HALL OF FAME

The National Inventors Hall of Fame's 2012 inductees include MIT Institute Professor Barbara Liskov, whose innovations in the design of computer programming languages helped make computer programs more reliable, secure, and easy to use; IBM researchers Lubomyr Romankiw and David Thompson, who invented the first practical magnetic thin-film storage heads; Gary Starkweather, whose laser printer, invented at the Xerox PARC facility, was the first to print images that could be created on a computer; and Apple and Pixar cofounder Steve Jobs, who was posthumously recognized for his technology contributions, which included more than 300 patents and revolutionized entire industries, including personal computing, mobile phones, animated movies, digital publishing, and retailing.

CRA AWARDS

The Computing Research Association awarded the 2012 A. Nico Habermann Award to Lucy Sanders, CEO, National Center for Women & Information Technology (NCWIT); Robert Schnabel, Dean, School of Informatics, Indiana University; and Telle Whitney, CEO and President of the Anita Borg Institute for Women and Technology for their joint efforts to establish and sustain the NCWIT, which is dedicated to encouraging greater participation of women in the development of computing technology. Susan L. Graham, Pehong Chen Distinguished Professor Emerita at the University of California, Berkeley, received 2012 Distinguished Service Award "in recognition of the extraordinary contributions that she has made over more than three decades of dedicated and selfless service and leadership."

—Jack Rosenberger

Education | DOI:10.1145/2160718.2160726 Paul Hyman

A Workshop Revival

The success of Germany's Dagstuhl Seminars has inspired a proliferation of Dagstuhl-like venues, especially in India.

The popularity of selective workshops for computer scientists has reached an all-time high, with various versions of Germany's Dagstuhl Seminars popping up around the globe.

The original Dagstuhl Seminars, launched in 1990, have gradually increased in number from 34 in 2000 to 64 planned for 2012. The program has also broadened over the years to include several annual events designed to support and improve computer science research.

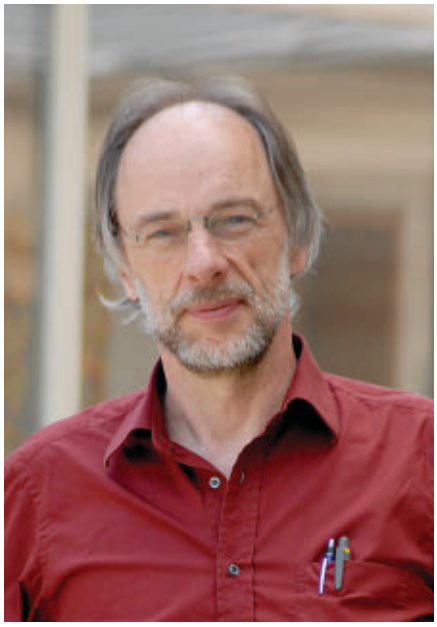
The success of the Dagstuhl concept should be seen in light of the "disturbing trend...in which researchers now attend more and more specialized conferences and workshops without ever meeting colleagues with whom they should communicate," says Reinhard Wilhelm, scientific director of the Schloss Dagstuhl-Leibniz Center for Informatics, which runs the seminars.

"Schloss Dagstuhl partly acts as a repair shop for this lousy trend," Wilhelm explains, "providing a space where insightful scientists can hold workshops involving different communities that should be communicating with each other and that would never meet at a conference."

In April, Wilhelm received ACM's Distinguished Service Award for his work with Dagstuhl Seminars.

Located in a former late-baroque mansion in Saarland, Germany, Schloss Dagstuhl (or Dagstuhl Castle) each week hosts top scientists and promising young researchers from around the world who gather to discuss and debate a particular computer science topic, living and working together for 3–5 days, "resulting in an intense collaboration and exchange of ideas," says Wilhelm.

Workshops are based on proposals submitted by scientists to the Leibniz Center's Scientific Directorate, which meets twice a year. Each proposal is evaluated for its relevance and its potential attractiveness to members of the computer science community.



Reinhard Wilhelm, scientific director of the Schloss Dagstuhl-Leibniz Center for Informatics.

PHOTOGRAPH COURTESY OF REINHARD WILHELM, SCHLOSS DAGSTUHL-LEIBNIZ CENTER FOR INFORMATICS

The Leibniz Center is a member of the Leibniz Association, which consists of 85 research institutes, research libraries, and research museums. Operation of the center is heavily subsidized with 50% of the subsidies coming from the German federal government and 50% from the state governments.

Meanwhile, the success of the seminars has inspired a proliferation of Dagstuhl-like venues, such as Shonan Village in the Tokyo area and the Banff International Research Station for Mathematical Innovation and Discovery in Banff, Canada. There is also discussion of setting up a "Dagstuhl" in southern China.

And, in Mysore, India, the Mysore Park workshops on the Infosys campus are also modeled after the Dagstuhl workshops.

"Schloss Dagstuhl was clearly our inspiration," says Sriram Rajamani, assistant managing director of Microsoft Research India. "Several of us have attended Dagstuhl where it is secluded and the quality of people who get invited and the topics that get discussed are quite high. We wanted to create a similar environment in India. Professor Jay Misra from the University of Texas at Austin and Infosys chairman emeritus N.R. Narayana Murthy supported the idea, and Murthy generously gave space at the Infosys Mysore Campus and funds to run it. We have a scientific board, which peer-reviews proposals to ensure that workshops are of high quality."

Rajamani is chair of the scientific board of Mysore Park.

Just as Dagstuhl is located in a remote area of Germany that is conducive to learning and discussion, Rajamani describes the Infosys Mysore campus as being "wonderful and quiet and a good three-plus hours' drive from Bangalore. Once you enter the campus, you're stuck! You cannot run away for a meeting elsewhere, and you are forced to engage in scientific discussion with your colleagues for 3–4 days. It is a very immersive experience."

Begun just two years ago, there have already been five Mysore Park workshops and several more are planned for this year.

While the growing number of workshop series like Dagstuhl is becoming increasingly important to the computer science community, there is a real need for more, says Rajamani.

"The recipe for a Dagstuhl-like series is quite simple," says Dagstuhl's Wilhelm with a wink. "Copy our concept—and then find an attractive place for it, get strong financial support, and, of course," he quips, "get the right people to run it." C

Paul Hyman is a science and technology writer based in Great Neck, NY.

Association for Computing Machinery

Advancing Computing as a Science & Profession

membership application & *digital library* order form

Priority Code: AD10

You can join ACM in several easy ways:

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

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

Fax
+1-212-944-1318

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

Special rates for residents of developing countries:
http://www.acm.org/membership/L2-3/

Special rates for members of sister societies:
http://www.acm.org/membership/dues.html

Please print clearly

Name

Address

City | State/Province | Postal code/Zip

Country | E-mail address

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

Purposes of ACM

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

I agree with the Purposes of ACM:

Signature

ACM Code of Ethics:
http://www.acm.org/serving/ethics.html

choose one membership option:

PROFESSIONAL MEMBERSHIP:

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

STUDENT MEMBERSHIP:

- o **ACM Student Membership: $19 USD**
- o **ACM Student Membership plus the ACM Digital Library: $42 USD**
- o **ACM Student Membership PLUS Print *CACM* Magazine: $42 USD**
- o **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*. Student membership dues include $15 toward a subscription to *XRDS*. 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.

o Visa/MasterCard o American Express o Check/money order

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

o ACM Digital Library ($99) $ _____

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

Total Amount Due $ _____

Card # Expiration date

Signature

DOI:10.1145/2160718.2160744

Meet the candidates who introduce their plans—and stands—for the Association.

ACM's 2012 General Election

Please take this opportunity to vote.

THE ACM CONSTITUTION provides that our Association hold a general election in the even-numbered years for the positions of President, Vice President, Secretary/Treasurer, and Members-at-Large. Biographical information and statements of the candidates appear on the following pages (candidates' names appear in random order).

ACM members have the option of casting their vote via the Web or by postal mail. The election is being conducted by a third party, ESC, on ACM's behalf and an email message was sent to members in mid-April that contains electronic and mail balloting procedures (please contact acmhelp@electionservicescorp.com or +1-866-720-4357 if you did not receive this message).

In addition to the election of ACM's officers—President, Vice President, Secretary/Treasurer—five Members-at-Large will be elected to serve on ACM Council.

Electronic Balloting Procedures. Please refer to the instructions posted at: https://www.esc-vote.com/acm2012.

You will need your 7-digit ACM Member Number to log into the secure voting site. For security purposes your ACM Member number was not included with your PIN (your PIN was included in the April email message from ESC). Your membership number can be found on your Membership card, or on the label of your copy of *Communications of the ACM*, or by accessing ACM's home page: "Membership," "ACM Web Account," "Forgot Your Password" (https://campus.acm.org/public/accounts/Forgot.cfm)

Paper Ballots. Should you prefer to vote by paper ballot please contact ESC to request a ballot and follow the postal mail ballot procedures (acmhelp@electionservicescorp.com or +1-866-720-4357).

Postal Mail Ballot Procedures. Please return your ballot in the enclosed envelope, which must be signed by you on the outside in the space provided. The signed ballot envelope may be inserted into a separate envelope for mailing if you prefer this method.

All ballots must be received by **no later than 12:00 noon EDT on May 22, 2012**.

The computerized tabulation of the ballots will be validated by the ACM Tellers Committee. Validation by the Tellers Committee will take place at 10:00 a.m. EDT on May 23, 2012.

If you have not done so yet, please take this opportunity to review the candidates and vote via postal mail or the Web for the next slate of ACM officers.

Sincerely,

Gerald Segal, CHAIR, ACM ELECTIONS COMMITTEE

candidates for
PRESIDENT
(7/1/12–6/30/14)

BARBARA G. RYDER
J. Byron Maupin Professor of Engineering
Head, Department of Computer Science
Virginia Tech
Blacksburg, VA, USA

Biography

A.B. in Applied Math, Brown University (1969); M.S. in Computer Science (CS), Stanford University (1971); Ph.D. in CS, Rutgers University (1982). Assoc. Member of Prof. Staff at AT&T Bell Labs, Murray Hill (1971–1976). Asst. Prof. (1982–1988), Assoc. Prof. (1988–1994), Prof. (1994–2001), Prof. II (2001–2008), Rutgers University, Head, Dept. of CS, Virginia Tech (2008–), http://people.cs.vt.edu/~ryder/

Fellow of the ACM (1998) for seminal contributions to interprocedural compile-time analyses. Member, Bd. of Directors, CRA (1998-2001). ACM Presidential Award (2008). SIGPLAN Distinguished Service Award (2001). Rutgers Grad. School Teaching Award (2007). Rutgers Leader in Diversity Award (2006). CRA-W Distinguished Prof. (2004). Prof. of the Year Award from CS Grad Students (2003).

ACM Vice President (2010–2012), Secretary/Treasurer (2008–2010), Council Member-at-Large (2000–2008). Chair, Federated Computing Research Conf (FCRC 2003). SIGPLAN Chair (1995–1997), Vice Chair for Confs (1993–1995), Exec Comm. (1989–1999). General Chair of: SIGSOFT Int'l Symp. on Software Testing and Analysis (ISSTA, 2008), SIGPLAN Conf on History of Programming Languages III (HOPL-III, 2007), SIGPLAN Conf on Programming Language Design and Implementation (PLDI, 1999, 1994). Program Chair of: HOPL-III (2007), PLDI (1991). Recent PCs: PLDI (2012), ISSTA (2010,) ICSE (2009). Member, Outstanding Contribution to ACM Award Comm. and ACM-W Athena Award Comm. ACM Nat'l Lecturer (1985–1988).

Member, Ed Bd: Sci. of Comp. Programming (2009–), ACM Trans on Programming Languages and Systems (TOPLAS, 2001–2007), and IEEE Trans on Software Engineering (2003-2008). Member, Comm. for CRA Snowbird Workshop for New Dept. Chairs (2010). Panelist: CRA Workshops on Academic Careers for Women in CS (1993, 1994, 1996, 1999, 2003), SIGSOFT New SE Faculty Symp. (2003, 2005, 2008). Exec. Champion of CS@VT in NCWIT Pacesetters (2009–). Organizer of NCWIT VA/DC Aspirations in Computing Awards (2011–). Chair, VT College of Engineering HPC Comm. (2009–). Member, VT HPC Infrastructure Investment Comm. (2009–). Member, ADVANCE VT Faculty Advisory Comm. (2008–).

Member: SIGPLAN, SIGSOFT, SIGCSE, ACM, IEEE Computer Society, AWIS, AAUW, EAPLS.

Statement

As a candidate for President, I ask your help in strengthening ACM, the largest independent international computing society in the world. We face major opportunities: in the internationalization of our membership, in computing education (K–12, college, postgraduate), in providing services for computing practitioners and researchers, and in ensuring diversity in the computing profession. My extensive experience as a SIG leader, General Chair of FCRC 2003, 8 years as ACM Council member and 4 years on the ACM Executive Committee as Secretary/Treasurer and Vice President have prepared me well to lead ACM as President.

The ACM Councils in Europe, China and India are established; new members from these regions must be recruited into SIG/ACM leadership. Student chapters in these regions need support. The commitment to hold ACM meetings outside of North America must be continued. We should be actively recruiting contacts from Southeast Asia/Australia and South America for future growth.

We need to continue the efforts of the ACM Education Board and Council on computing curricula and building bridges to K–12 educators (e.g., in the U.S., CS Teachers Association). By collaborating with members of each international ACM Council, we can leverage our experience to support computing education. The specific problems may vary by region, but the need to ensure a quality education in computing is worldwide.

ACM is a membership organization of computing practitioners, managers, and researchers; we need to offer benefits to all of our members. ACM serves as the 'gold standard' of computing research through our sponsored conferences, publications and the Digital Library. We must continue to enhance our support for scholars and students worldwide. We need to find new ways of making our conferences available to a wider audience, synchronously and asynchronously. We should continue to provide new services to researchers (e.g., author pages, Author-Izer). Equally important are new products for practitioners and students such as Tech Packs and Learning Paths. This support must be augmented by new services for life-long learning in the computing profession.

ACM must be a leader in actively supporting diversity in computing. ACM can offer a framework to support continuing and new diversity activities, in collaboration with sister organizations (e.g., CRA, CDC, NCWIT).

The SIGs must remain a strong, integral part of ACM, developing volunteer leaders, providing valuable computing research content for the Digital Library and recruiting students to ACM membership. Without volunteers, ACM cannot thrive. We need to support existing SIGs and continue to look for new areas in computing which can benefit from organization as a SIG.

In closing, I ask for your vote so that my 10 years of SIG leadership, 12 years of ACM leadership, and 37 years of active ACM membership can be put to work on these goals.

candidates for
PRESIDENT
(7/1/12–6/30/14)

VINTON G. CERF
Vice President and Chief Internet Evangelist
Google
Reston, VA, USA

Biography

Vinton G. Cerf has served as VP and chief Internet evangelist for Google since October 2005. Previous positions include SVP of Technology Strategy and SVP of Architecture and Technology for MCI; VP of the Corporation for National Research Initiatives (CNRI); VP of MCI Digital Information Services; principal scientist at DARPA; and assistant professor of computer science and electrical engineering at Stanford University. He is also a distinguished visiting scientist at NASA's Jet Propulsion Laboratory (JPL).

Cerf is the co-designer of the TCP/IP protocols and the architecture of the Internet. Among honors received for this work are the U.S. National Medal of Technology, the ACM A.M. Turing Award, the Presidential Medal of Freedom, the Japan Prize, the Marconi Fellowship, Charles Stark Draper award, the Prince of Asturias award, the IEEE Alexander Graham Bell medal, the NEC Computer and Communications Prize, the Silver Medal of the ITU, the IEEE Koji Kobayashi Award, the ACM Software and Systems Award, the ACM SIGCOMM Award, the CCIA Industry Legend Award, the Kilby Award, the IEEE Third Millennium Medal, and the Library of Congress Bicentennial Living Legend medal. He has been inducted into the National Inventors Hall of Fame and was made an Eminent Member of the IEEE Eta Kappa Nu honor society. He is a Stanford Engineering School "Hero" and received a lifetime achievement award from the Oxford Internet Institute.

Cerf served as chairman of the board of ICANN for seven years. He is honorary chair of the IPv6 Forum, was a member of the U.S. Presidential Information Technology Advisory Committee 1997–2001, and sits on ACM Council and the boards of ARIN, StopBadWare, CosmosID and Gorilla Foundation. He serves on JPL's Advisory Committee and chairs NIST's Visiting Committee on Advanced Technology. He is a Fellow of the ACM, IEEE, AAAS, the American Academy of Arts and Sciences, the Computer History Museum, the Annenberg Center for communications, the Swedish Royal Academy of Engineering, the American Philosophical Society, the Hasso Plattner Institute. He is a member of the US National Academy of Engineering, a Distinguished Fellow of the British Computer Society, and was named one of *People* magazine's "25 Most Intriguing People" in 1994.

Cerf holds a B.S. in Mathematics from Stanford and M.S. and Ph.D. in CS from UCLA. He is the recipient of 20 honorary degrees.

Statement

I have been a member of ACM since 1967 and served as a member of Council in the distant past during which time my primary policy objective was to create the Fellow grade of ACM membership, and currently serve as a Member at Large on Council. I also served for four years as chairman of ACM SIGCOMM.

As President, I consider that my primary function will be to convey to Council and ACM leadership the policy views of the general membership. To this end, I will invite open dialog with any and all members of ACM so as to be informed of their views. I offer to do my best to represent these views and also to exercise my best judgment in the setting of ACM policy and assisting the staff in their operational roles.

It seems clear that ACM can and must take increasing advantage of the online environment created by the Internet, World Wide Web, and other applications supported by this global networking infrastructure. This suggests to me that Council and ACM leadership should be looking for ways to assist Chapters and Special Interest Groups to serve as conduits for two-way flows of information, education, training and expertise. The power and value of ACM membership flows from the contributions and actions of its members.

As a consumer of ACM publications, I am interested in increasing the accessibility and utility of ACM's online offerings, including options for reducing the cost of access to content in this form. By the same token, I am interested in making the face-to-face events sponsored by ACM and its affiliates useful during and after the events have taken place. The possibility of audio, video, and text transcripts of presentations (perhaps starting with keynotes) seems attractive. If these forms of content can be made searchable, their value may well increase and persist over longer periods of time.

I am aware of the time commitment required of this position and can confirm that I am prepared to honor that commitment with the support of my employer, Google.

candidates for
VICE PRESIDENT
(7/1/12–6/30/14)

MATHAI JOSEPH
Advisor, Tata Consultancy Services
Pune, India

Biography

Mathai Joseph is an Advisor to Tata Consultancy Services. He has been a Member-at-Large of the ACM Council since 2008 and Treasurer of the ACM India Council, which he helped to found, since 2009. He received an ACM Presidential Award in 2010.

He was Executive Director, Tata Research Development & Design Centre from 1997–2007. Earlier, he held a Chair in Computer Science at the University of Warwick from 1985–1997 and was Senior Research Scientist at the Tata Institute of Fundamental Research in Mumbai, India. He has been a Visiting Professor at Carnegie Mellon University, Eindhoven University of Technology, University of Warwick, and University of York.

Joseph is a founder of international conference series such as FST&TCS (Foundations of Software Technology and Theoretical Computer Science) in 1981, FTRTFT (Formal Techniques in Real-Time and Fault-Tolerant Systems) in 1988 and SEAFOOD (Software Engineering Approaches to Outsourced and Offshore Development). He set up the workshop series TECS Week in 2003 to promote computer science research in India. He established and managed research collaborations with universities such as University of California Riverside, Columbia University, Georgia Institute of Technology, Indian Institute of Technology Powai, King's College London, Stanford University, University of Wisconsin, and University of York.

His main research interests have been in real-time and fault-tolerant computing, with emphasis on formal specification and timing analysis. He has written and edited a number of books and conference and journal papers in these areas. He has worked on software systems for many years and wrote an early book on the design of a multiprocessor operating system.

Joseph has a Master's degree in Physics from Bombay University and a Ph.D. from the Mathematical Laboratory, University of Cambridge, U.K. He has worked in several countries and now lives in Pune, India. He has been investigating early work in computing in India and is now investigating new courseware for teaching computer science at engineering colleges in India.

Statement

ACM is renowned for representing the best of computer science in its conferences, publications, and awards. Until recently, most ACM activities took place in the Western world but there are now new ACM Regional Councils in China, Europe, and India. Within these regions, participation in ACM activities is growing but outside them there are many countries where computing professionals work in relative isolation.

I was closely involved with the creation of the ACM India Council in 2009 and have been active in its operation since that time. There are now significant ACM events at the national and chapter level and tours by visiting lecturers. A number of new chapters have been formed and membership has doubled in this period. Compared to the number of computing professionals, the size of the Indian software industry and the needs of the academic institutions, the influence of ACM in India is still small but it is growing to play a major role in computer science education and research.

As in India, I believe that in many other regions ACM should be active in growing professional activities and creating links between their education and research communities and the best in computer science. Wider distribution of ACM publications and conferences will not only allow them to be more effective in growing professional activities and promoting high quality research, it will enable them to be more representative of all such work being done across the world.

A great deal of my professional work has been to create forums for the exchange of ideas, new research, and people among industry and educational institutions in India and outside. I would like to work with ACM to grow its activities, within the main regions of ACM activities and beyond them to areas where it is as yet underrepresented. In this, I see a major opportunity for initiatives for the better acceptance of computer science as a core scientific discipline, for improving the quality of education and research and for increasing the understanding of computing as a global manufacturing industry.

candidates for
VICE PRESIDENT
(7/1/12–6/30/14)

ALEXANDER L. WOLF
Professor
Department of Computing
Imperial College London, U.K.

Biography

Alexander Wolf holds a Chair in Computing at Imperial College London, U.K. (2006–present). Prior to that he was a Professor at the Univ. of Lugano, Switzerland (2004-2006), Professor and C.V. Schelke Endowed Chair of Engineering at the Univ. of Colorado at Boulder, U.S. (1992–2006), and Member of the Technical Staff at AT&T Bell Labs in Murray Hill, New Jersey, US (1988–1992).

Wolf earned his MS (1982) and Ph.D. (1985) degrees from the Univ. of Massachusetts at Amherst, from which he was presented the Dept. of Computer Science Outstanding Achievement in Research Alumni Award (2010).

He works in several areas of experimental and theoretical computer science, including software engineering, distributed systems, networking, and databases (see http://www.doc.ic.ac.uk/~alw/ for links to his papers). He is best known for seminal contributions to software architecture, software deployment, automated process discovery (the seed of the business intelligence field), distributed publish/subscribe communication, and content-based networking.

Wolf currently serves as Secretary-Treasurer of the ACM. He is a member of the ACM Council and ACM Executive Committee. He is also a member of the ACM Europe Council. He serves on the editorial board of the Research Highlights section of CACM. Previously, he served as Chair of the ACM SIG Governing Board (2008–2010) and Chair of the ACM Software System Award Committee (2009–2011). He served as Vice Chair (1997–2001) and then Chair (2001–2005) of ACM SIGSOFT, Chair of the ACM TOSEM EiC Search Committee (2006), a member of the SGB Executive Committee (2003–2005), and an SGB representative on the ACM Council (2005–2008). He was a member of the editorial boards of ACM TOSEM and IEEE TSE, and has chaired and been a member of numerous international program committees.

He is a Fellow of the ACM, Fellow of the IEEE, Chartered Fellow of the British Computer Society, holder of a U.K. Royal Society–Wolfson Research Merit Award, winner of two ACM SIGSOFT Research Impact Awards, and is an ACM Distinguished Speaker.

Statement

I have been an active member of ACM for much of my career. Over these 25+ years, computing has become central to advancing society. As a volunteer I have tried to help shape ACM into a similarly central role in advancing the educators, practitioners, researchers, and students at the core of computing. In my recent leadership roles I have contributed to important outreach efforts, including formation of regional councils (so far, Europe, India, and China), formation of SIGs in new areas of computing (so far Bioinformatics, Health Informatics, and High Performance Computing), enrichment of the Digital Library, and a revamp of CACM to be more relevant, informative, and authoritative. There are many ways to measure the impact of these efforts, but a simple one is the growth of ACM into the world's largest computing association.

Essential to these and future initiatives is a clear and convincing long-term strategy that draws together the talents of volunteers and staff, supported by sufficient financial resources. Indeed, the world financial crisis has been a necessary focus of my term as Secretary-Treasurer. Managing financial risk while maintaining the integrity and momentum of the community is a difficult challenge. The crisis has hit different sectors and regions at different times and degrees. For example, the peak effect on academia has lagged that of industry, and Europe and Asia that of North America. This will continue to impact conference planning and regional growth initiatives (e.g., in South America). Nonetheless, our efforts at securing a significant fund balance have contributed to the fundamental stability of the organization, such that we can absorb much of the pain of the crisis and yet still grow ACM's offerings and services.

As an ACM leader, I am regularly asked where the value lies in being a member, a question you may ask yourself each time you renew. Some obvious benefits are CACM, discounted registration fees, and access to professional development materials. But beyond that, most important to me is a less tangible benefit: the opportunity to publicly endorse the activities of an organization whose standards of excellence and commitment to advancing professionalism, scholarship, and innovation are unsurpassed in the computing community. The opportunity to serve as Vice President of such an organization is a privilege and an honor.

candidates for
SECRETARY/TREASURER
(7/1/12–6/30/14)

GEORGE V. NEVILLE-NEIL
Neville-Neil Consulting
New York, NY, USA

Biography

George Neville-Neil is a computer scientist, author, and practicing software engineer who currently builds high-speed, low latency systems for customers in the financial services sector.

Neville-Neil is the co-author of *The Design and Implementation of the FreeBSD Operating System*, the only university quality textbook to cover a broadly deployed, open source operating system, used daily by software engineers around the world.

He has been a member of ACM since his undergraduate days at Northeastern University, where he received his Bachelor of Science degree in computer science.

For the last 10 years Neville-Neil has been an integral part of ACM's *Queue* magazine editorial board, joining it before the first edition of the magazine was published and helping to find, select, and bring to publication the best writing from practitioners in our field.

While working with the *Queue* editorial board, Neville-Neil developed the column, Kode Vicious, which has been a staple of both *Queue* and the recently revamped *Communications of the ACM* for the last eight years.

More recently Neville-Neil has been an active member of the ACM Practitioner Board, which is a key component of ACM's program to broaden its membership.

From 2004 until 2008 Neville-Neil lived and worked in Japan, developing a set of courses dubbed "The Paranoid University" teaching safe and secure programming to thousands of engineers at Yahoo Inc. at all of their offices in Asia, Europe, and North America. While on assignment in Japan, Neville-Neil became fluent in Japanese.

Statement

For the past 25 years I have been associated with the ACM. During that time I have felt that the organization did its best work when it was reaching out beyond its academic focus. It has always been clear that ACM is the preeminent professional organization for computer scientists and researchers, publishing the best papers and organizing the most respected conferences in the field.

When I began working on *Queue* I saw that ACM was serious about the need to bridge the gap between research and the daily practice of programming. The last 10 years have been an exhilarating experience helping people take the best research and turn it into something that software engineers can use every day. It was through my work on *Queue*, and the Practitioner Board, that I saw how ACM could evolve into an even broader professional society.

Being nominated for Secretary/Treasurer showed me that there was another way to help ACM grow and to serve the global community.

Living and working in Asia gave me a unique perspective on ACM's challenges in addressing the coming wave of people in our field.

ACM's ability to help improve computer science education globally is without question, but it is just as important to understand how we can continue our relationship with the people who are building our technological world, throughout their careers.

My experience building systems for the global financial markets has given me insight into how money is managed in large organizations.

Working as the Secretary/Treasurer for ACM carries the same challenges and responsibilities as building systems that exchange millions of dollars every day.

It has always been a pleasure to be an ACM volunteer. I hope that as Secretary/Treasurer I can do even more for the ACM, and help them to do more for others.

candidates for
SECRETARY/TREASURER
(7/1/12–6/30/14)

VICKI L. HANSON
Professor and Chair of Inclusive Technologies
University of Dundee
Dundee, Scotland

Biography

Employment
Professor and Chair of Inclusive Technologies, School of Computing, University of Dundee, UK; IBM Research Staff Member Emeritus. Previously, Research Staff Member and Manager, IBM T. J. Watson Research Center, USA (1986–2008); Research Associate, Haskins Laboratories, New Haven, CT (1980–1986); Postdoctoral Fellow, Salk Institute for Biological Studies, La Jolla, California (1978–1980).

Education
B.A. (University of Colorado, Psychology, 1974); M.A., Ph.D. (University of Oregon, Cognitive Psychology, 1976, 1978).

ACM Activities
SIG Governing Board: Chair (2010–2012), Vice-Chair for Operations (2007–2010), Executive Committee (2005–2007). ACM Council (2010–2012). Founder and Co-Editor-in-Chief of *ACM Transactions on Accessible Computing* (since 2006); Associate Editor, *ACM Transactions on the Web* (since 2005). Outstanding Contribution to ACM Award Committee (2006–2010; Chair, 2009). ACM Student Research Competition: Grand Finals judge (2008–2012). ASSETS conference: General Chair (2002), Program Chair (2010), Steering Committee Chair (2005–2009). CHI conference: Associate Program Chair (2009, 2012). Hypertext conference: Program Committee Co-Chair (2007). CUU conference: General Chair (2003), Associate Program Chair (2000). Multiple program committees. OOPSLA conference: Treasurer (1993–1995, 1999, 2005).

Professional Interests:
Human-computer interaction; accessibility of technology for people with disabilities and the aging population; computer-supported education.

Honors/Awards/Achievements
ACM Fellow (2004). ACM SIGCHI Social Impact Award (2008). Chartered Fellow of the British Computer Society (2009). IBM Corporate Award for Contributions to Accessibility (2009); multiple IBM Outstanding Contribution Awards. More than 120 peer-reviewed and book chapter publications, multiple conference keynotes, awarded 8 U.K. Research Council grants as PI or co-Investigator; 3 NIH grants as PI; 5 other industry and Scottish government grants. Co-inventor on 8 awarded patents. Holder of a UK Royal Society Wolfson Research Merit Award.

Statement

I have been an active ACM volunteer for two decades and am honored by this nomination and the opportunity to continue to serve the organization.

In multiple leadership roles within ACM, I have worked with both volunteers and staff who are committed to increasing the society's reach and impact. ACM is the leading professional society in computing. Its membership has been steadily increasing with significant worldwide growth. Its premier journals, conference proceedings, and magazine publications serve a broad readership of computing professionals, and its educational initiatives are contributing to the development of computing professionals at all stages oftheir career. In the past few years, significant ACM efforts have involved working with computing professionals in China, India, Europe, and South America to understand and address their needs. In a professional environment that is increasingly global, continuing to reach out internationally serves to strengthen not only ACM, but also the computing profession as a whole.

In my current position as Chair of the SIG Governing Board, I have been privileged to work with SIG leaders who contribute so much to ACM's strength as an organization. The SIGs create a large portion of the Digital Library content, draw in new student and professional members, and provide much of ACM's volunteer core. As we move forward, we must continue to support and grow the SIGs to retain this technical vitality.

Financially, ACM is strong even in these economically challenging times. This reflects the very real value members derive from their participation in ACM activities. As Secretary/Treasurer, I will remain committed to growing ACM and supporting its technical, education, and advocacy roles around the world.

candidates for
MEMBERS AT LARGE
(7/1/12 – 6/30/16)

RADIA PERLMAN
Intel Fellow
Redmond, WA, USA

Biography

Radia Perlman, Intel Fellow, is director of Network Technology at Intel. She received her S.B. and S.M in mathematics from MIT, and her Ph.D., in 1988, in computer science from MIT with thesis "Network Layer Protocols with Byzantine Robustness."

While a student at MIT, she created systems for making programming concepts accessible to children as young as 3 years old. Her work is credited for starting the field of "tangible computing," in which computer commands are physical objects that are plugged together to create programs.

At Digital, she designed layer 3 of DECnet. Her innovations include making link state protocols stable, manageable, and scalable. The routing protocol she designed (IS-IS) remains widely deployed today. She also created the Spanning Tree algorithm that has been the heart of Ethernet for decades. After Digital, she worked at Novell and at Sun, designing network and security protocols. Most recently, she designed the concept that became the TRILL standard, which allows Ethernet to use layer 3 techniques such as shortest paths, and multipathing, while remaining compatible with existing endnodes and switches. She has also made significant contributions to network security, including PKI trust models, assured delete, protocols resilient despite malicious insiders, and strong password protocols.

Perlman is the author of the textbook *Interconnections*, and coauthor of *Network Security*. She has served on many program committees, holds over 100 patents, and has taught at MIT, Harvard, and the University of Washington. She was awarded an honorary Ph.D. by KTH (Royal Institute of Sweden), lifetime achievement awards by SIGCOMM (2010) and USENIX (2006), and the Women of Vision award for Technical Innovation by the Anita Borg Institute.

Statement

ACM has played a critical role in shaping the field of computing, and I think there are several areas in which it is well positioned to help the industry in new ways.

Academia and industry would benefit from more cross-fertilization. Industry can bring a fruitful selection of real-world problems, and academia can bring rigor. Most industry people would find a lot of the content at academic conferences incomprehensible and irrelevant. Academics do not have the time or funds to spend enough time at standards meetings to extract conceptual problems to study out of the sea of acronyms and marketing hype.

One idea that might help is to encourage papers at conferences that can organize the field, such as sorting through several overlapping systems to get to the heart of what is really intrinsically different, and the real trade-offs. Some academics would not think such a paper was "novel enough to be publishable," and yet the academic rigor to do this sort of analysis can be very challenging, and influential to the field. It is entertaining to have a panel of people on various sides of controversies debate each other, but having a careful analysis by someone without bias is critical. With content like this, more industry people would attend the conference, and even hallway interaction will spark collaboration.

Another problem in our field is misperceptions that cause some groups to believe they would not be good at it, or that they would not enjoy it. To enrich the industry by bringing in people with diverse thinking styles, perhaps ACM could sponsor contests for young students with interests other than building computers, such as music, art, or social interaction, that will inspire them to realize they can make a difference in our field.

RICARDO BAEZA-YATES
VP of Research for EMEA & LatAm
Yahoo! Research Barcelona
Barcelona, Catalunya, Spain

Biography

Ricardo Baeza-Yates is VP of Yahoo! Research for Europe, Middle East, and Latin America, leading the labs at Barcelona, Spain and Santiago, Chile, since 2006, as well as supervising the lab in Haifa, Israel since 2008. He is also a part-time professor at the Dept. of Information and Communication Technologies of the Universitat Pompeu Fabra in Barcelona, Spain, since 2005. Until 2005 he was a professor and director of the Center for Web Research at the Department of Computer Science of the Engineering School of the University of Chile.

He obtained a Ph.D. from the University of Waterloo, Canada, in 1989. Before he obtained two master's (M.Sc. CS and M.Eng. EE) and the electrical engineering degree from the University of Chile, Santiago.

He is co-author of the best-seller *Modern Information Retrieval* textbook, published in 1999 by Addison-Wesley with a second enlarged edition in 2011, as well as co-author of the 2nd edition of the *Handbook of Algorithms and Data Structures*, Addison-Wesley, 1991; and co-editor of *Information Retrieval: Algorithms and Data Structures*, Prentice-Hall, 1992, among more than 300 other publications.

He has received the Organization of American States award for young researchers in exact sciences (1993) and the CLEI Latin American distinction for contributions to CS in the region (2009). In 2003 he was the first computer scientist to be elected to the Chilean Academy of Sciences. During 2007 he was awarded the Graham Medal for innovation in computing, given by the University of Waterloo to distinguished ex-alumni. In 2009 he was named ACM Fellow and in 2011 IEEE Fellow.

Statement

I became an ACM member back in 1984 when I was a student in Chile. My initial motivation was to be part of the largest CS community in a time when there was no Internet. This feeling of belonging has been constant until today. In 1998 I was invited to be part of the first South American steering committee for the regional ACM Programming Contest, a position that I held for eight years. In 2007 I was invited to become a member of the Publications Board for a three-year term and since 2010, I have been a member of the ACM Europe Council. Thus, I know reasonably well how the ACM works across several continents.

On the technical side, I have been a member of the ACM SIGIR community for many years. I am also a member of SIGACT, SIGKDD, SIGMOD, and SIGWEB. Therefore, I also feel that I know well my research peers. For that reason, although I have received several awards, being named ACM Fellow in 2009 was for me the highest possible recognition given by my peers. So, when I was asked to be a Member at Large candidate, I gladly accepted.

So, based on my ACM experience, as a Member-at-Large my main goal will be to extend the influence of ACM in emerging regions, involving developing countries in Africa, Asia, and Latin America. To achieve this goal we need to promote the ACM Distinguished Speakers Program in places that never had the opportunity of listening to senior members of our community, extend the ACM Programming Contest in countries that do not participate, start agreements with CS associations in the countries where they exist, and, more important, work on making it feasible to have free access to the ACM Digital Library for the ACM student members on these countries. All these actions are easier from Europe due to their cultural and geographical proximity.

candidates for
MEMBERS AT LARGE
(7/1/12 – 6/30/16)

FENG ZHAO
Microsoft Research Asia, Beijing
China

Biography

Feng Zhao is an assistant managing director at Microsoft Research Asia, responsible for the hardware, mobile and sensing, software analytics, systems, and networking research areas. He is also an adjunct professor at Shanghai Jiaotong University, Hong Kong University of Science and Technology, and the University of Washington. Prior to joining MSR-Asia in 2009, he was a Principal Researcher at Microsoft Research Redmond (2004–2009), and founded the Networked Embedded Computing Group. His research has focused on wireless sensor networks, energy-efficient computing, and mobile systems. He has authored or co-authored over 100 technical papers and books, including a book, *Wireless Sensor Networks: An Information Processing Approach*, by Morgan Kaufmann.

An ACM member for over 25 years, Feng was the founding Editor-In-Chief of *ACM Transactions on Sensor Networks* (2003–2010), and founded the ACM/IEEE IPSN conference. Feng served on ACM SIGBED Executive Committee (2004–2010), as TPC Co-Chair for ACM Sensys'05, and on the Steering Committee for CPSWeek (2007–). In 2008, he worked with USENIX and ACM to help start a new workshop series, HotPower, focusing on the emerging topic of sustainable computing.

Feng received his BS from Shanghai Jiaotong University (1984), and MS and Ph.D. in Electrical Engineering and Computer Science from MIT (1988 and 1992, respectively). He taught at Ohio State University as a tenured Associate Professor in Computer Science 1992–1999, and at Stanford University as a Consulting Professor of Computer Science 1999–2006. He was a Principal Scientist at Xerox PARC 1997–2004.

An IEEE Fellow and ACM Distinguished Engineer, Feng received a Sloan Research Fellowship (1994) and NSF and ONR Young Investigator Awards (1994, 1997).

Statement

If elected, I will push to deepen the engagement of ACM with the Asia computing community, building on the momentum initiated by the Council.

In addition to expanding ACM membership, we need to focus on the quality of engagement, such as bringing leading ACM conferences to the region, establishing a regional version when needed, and ensuring an active and *sustained* participation by the local community. The ACM Programming Contest has had a great mindshare among aspiring students at Chinese universities, with a broad participation and success of these schools at the event. Can we learn from these, and expand to other activities?

As computing becomes increasingly central in addressing societal problems in energy, environment, and health that affect the region and the rest of the world, and as these problems increasingly demand more global, coordinated actions, ACM can be the thought leader in advocating a global, collaborative approach to addressing these problems.

Young students do look to role models when deciding what to study and making career choices. Microsoft has been sponsoring the Best Doctoral Dissertation Awards at the Chinese Computer Federation Annual Meeting. ACM can and should play a major role in similar activities, partnering with CCF and other organizations in the region.

Bringing leading lights in computing to Asia for public lectures has proven very successful in increasing public awareness of computing and inspiring young students. An example is the annual 21st Century Computing and TechVista Symposia organized by Microsoft Research together with universities across Asia. But it is equally important for the global community to hear the perspectives of leading researchers from Asia. ACM can help promote these researchers in the global community.

ERIC ALLMAN
Sendmail, Inc.
Chief Science Officer
Emeryville, CA, USA

Biography

Eric Allman graduated from the University of California, Berkeley with a Bachelor's Degree in 1977 and a Master's Degree in 1980. While at Berkeley he was Lead Programmer on the INGRES relational database management system as well as making significant contributions to the Berkeley Software Distribution, notably sendmail and syslog. Sendmail remains among the top MTAs on the Internet, and syslog is the standard logging mechanism used in nearly all open systems and peripherals.

After leaving Berkeley, Allman worked at Britton Lee (later Sharebase) on database interface software and the International Computer Science Institute on the Ring Array Processor project for neural-net-based speech recognition. He returned to U.C. Berkeley at the Mammoth project, building department-wide research infrastructure for the Computer Science Division. He then founded Sendmail, Inc., an email infrastructure company based on his work at Berkeley. He has played many roles at Sendmail and currently holds the title of Chief Scientist.

Allman has been a member of ACM since 1975, is a past board member and officer of USENIX Association, was on the board of Sendmail for several years, and is currently a Trustee of Cal Performances, a U.C. Berkeley-based arts organization. He has written for *TODS, TOCS, USENIX, UNIX Review* magazine, and *ACM Queue* magazine, among others. He also co-authored the C Advisor column for *UNIX Review* for several years.

Allman is a Distinguished Engineer at ACM and is a founding member of the *Queue* editorial review board, on which he continues to serve.

Statement

I have had the opportunity to watch ACM evolve over the last 40 years, starting from my introduction to computers in high school. I was given old copies of *Communications*, which I devoured, joining ACM as an undergraduate. In the 1960s and into the 1970s, CACM was both academic and pragmatic; for example, *Communications* had a Programming Techniques section as well as an Algorithms section that included descriptions of various algorithms (hundreds of them, including code!). As time went on ACM increasingly focused on academics and researchers to the detriment of practitioners such as myself, until 10 years ago when *Queue* magazine was started with practitioners as the express audience. I am proud of having been a founding member of the *Queue* board, and feel that I've been honored to help restore ACM's relevance to people such as myself.

One of the challenges is explaining to practitioners how the ACM is relevant to them. Much of this is just a matter of awareness. ACM already has *Queue*, Tech Packs, the Learning Center, Safari books and videos, to name just a few. While these resources are of immense relevance to practitioners, most of them are relatively unknown in that community.

Another area of importance is international participation. The once English-centric computing world has now expanded into many geographies and languages. ACM needs to broaden its international outreach or risk becoming marginalized. There is a huge amount to be done here, and it will be challenging, but it must proceed.

I have very much enjoyed working on the *Queue* board and with the Practitioner Board and with your vote, look forward to additional participation on the ACM Council.

candidates for
MEMBERS AT LARGE
(7/1/12 – 6/30/16)

MARY LOU SOFFA
Chair, Department of Computer Science
and the Owen R. Cheatham Professor
University of Virginia
Charlottesville, VA, USA

Biography

Mary Lou Soffa is the Chair of the Department of Computer Science and the Owen R. Cheatham Professor of Sciences at the University of Virginia. Prior to Virginia, she was a Professor at the University of Pittsburgh and also served as the Graduate Dean in Arts and Sciences. Her general research interests are in programming languages/compilers and software engineering. Her current focus is on resource management for multicores, program analysis, virtual execution environments, testing, and debugging.

In 1999, Soffa was selected as an ACM Fellow and received the Presidential Award for Excellence in Science, Mathematics and Engineering Mentoring. She received the SIGPLAN Distinguished Service Award in 2003 and SIGSOFT Distinguished Service Award in 2010. She received the Nico Habermann Award from the Computer Research Association in 2006. In 2011, she was selected for the Anita Borg Technical Leadership Award.

Soffa has been active in ACM for many years. She currently serves on the Publications Board and is a member-at-large in ACM Council. She was the ACM SIG Board Council Representative from 2000–2002 and served as SIGPLAN Chair, Vice-Chair and Treasurer. She also served as member-at-large on the SIGSOFT executive committee.

She served on the CRA Board for 10 years and currently serves on CRA-W. She has worked to increase the participation of women and underrepresented minorities for many years. She co-founded the CRA-W Graduate Cohort Program and the CRA-W Cohort for Associate Professors. She has been a member of a number of editorial boards and has served as conference chair, program chair, and program committee member for numerous conferences in both software engineering and programming languages.

Statement

Through the more than 20 years that I have been involved with ACM, it has continually evolved, always striving to provide the services needed by it s members as well as welcoming new members, both in the U.S. and internationally. I have been, and continue to be, committed in helping ACM address both new problems that are emerging as well as ongoing challenges.

Interdisciplinary research and development is a growing national trend in science, and it very much involves computing. With this important direction comes challenges that I believe ACM can help address. How do we properly educate students? How do we evaluate it? What type of journals and conferences are needed? These are some questions that we must start to address.

Another challenge that I would like to help address is the troubling lack of interest in computing shown by students. We need to help ongoing projects that are focused on demonstrating the excitement and opportunities that computing offers to them. Our science and profession depends on a robust pipeline of students engaged in computing and we must work toward that end.

An ongoing challenge is the lack of gender and minority diversity in our field in all levels. Although efforts have been ongoing for a number of years, there still remains much to be done. We need to develop new strategies and culture changes to attract diverse students. I look forward to continuing my work in this area with ACM.

Although there are many challenges to address, we must also maintain and enhance what is working right. Much of success of ACM has come from the SIGs, through their volunteers, high-quality journals and conferences and the digital library. We have to continue to ensure that the SIGs have the support and tools that they need to continue their important work.

P J NARAYANAN
Dean (R&D)
IIIT Hyderabad
Hyderabad, India

Biography

P J Narayanan is a Professor and Dean (R&D) at the International Institute of Information Technology, Hyderabad. His research interests include Computer Vision, Computer Graphics, and Parallel Computing. His Ph.D. thesis dealt with parallel processing for large computer vision problems. At CMU, he developed Virtualized Reality, the first system to capture dynamic action in 3D using 50 cameras. He played a key role in establishing a vibrant research programme at IIIT-H as the first Post-Graduate Coordinator and Dean (R&D). Nvidia recognized him as a CUDA Fellow in 2008 for the pioneering efforts in GPU computing. He has played a key role in making the Indian Conference on Computer Vision, Graphics, and Image Processing (ICVGIP) as a high-quality conference series and in bringing the Asian Conference on Computer Vision to India. He has been an Area Chair or Program Committee member of major conferences in Computer Vision and Computer Graphics as well as IJCAI in 2007. Narayanan has been the Co-Chair of ACM India Council from its inception in 2009. He serves on several committees of DST, MCIT, and CSIR in India, dealing with education and research.

Education:

B.Tech (1984) in CS from the IIT, Kharagpur; MS (1989) and Ph.D. (1992) in CS from the University of Maryland, College Park.

Work experience

- Associate Development Engineer, Lipi group, CMC R&D (1984–1986)
- Teaching and Research Assistant in University of Maryland (1986–1992)
- Research Faculty Member in Robotics Institute of Carnegie Mellon University (1992–1996)
- Scientist D/E and Head of Computer Vision and Virtual Reality group of the Centre for AI & Robotics (CAIR), DRDO, Bangalore (1996–2000)
- Faculty member of IIIT, Hyderabad (2000–)

Statement

Computing has touched every aspect of human life in a short period. Information is abundant today, resulting in fundamental changes in teaching, research, and how we work. Corporations have a truly global work force and are critically dependent on the information infrastructure. Research too has become increasingly global. Academia will also be required to make its teaching and research global.

Internationalization has already taken ACM to small places in India like Patiala and Anakapalle with its chapters. I have been a part of this as ACM India Co-Chair since its inception. We have already set up an award for the best dissertation from India. I am satisfied with the progress so far in India. I am amazed at the dedication and vision of the ACM Council in expanding the role of ACM everywhere. I think much more can be accomplished with communities from South America, the Middle East, and Africa, apart from India, China, and Europe, if ACM makes itself relevant and useful to them.

ACM can leverage its strengths and expand its services to be more attractive to its members. Broadcasting high-profile events like the Turing Award lectures, the Turing 100 celebrations, and events from SIGGRAPH, MultiMedia, and Supercomputing, exclusively to its members will make ACM more attractive. In the era of digital connectivity, ACM can have Distinguished Lectures that are delivered digitally. A periodic 2-hour slot with a Turing Award winner or an accomplished person exclusively for ACM chapters can and will be very popular in countries like India, but also to other places. ACM's DL is already attractive, but can be made more so by expanding its scope to its events and lectures. I would like to explore such ideas as a member of the council. I seek your votes to make this possible.

candidates for
MEMBERS AT LARGE
(7/1/12 – 6/30/16)

EUGENE H. SPAFFORD
Professor and Executive Director
Purdue University CERIAS
West Lafayette, IN, USA

Biography

Eugene H. Spafford (Gene, Spaf) received a B.S. in both math and computer science from SUNY at Brockport ('79). He received his M.S. ('81) and Ph.D. ('86) in CS from Georgia Tech. After a post-doc at Georgia Tech he joined the faculty at Purdue University in 1987 as a professor in CS. He also has courtesy appointments in ECE, Philosophy, and Communication.

His current research interests are prevention, detection, and remediation of information system failures and misuse, with an emphasis on applied information security. He also has been deeply involved in issues of science and national security policy. He founded the COAST Laboratory at Purdue in 1992, which became the Center for Education and Research in Information Security and Assurance (CERIAS) in 1998.

Spaf has received many awards for research, teaching and service, including: National Computing Systems Security Award, ACM SIGSAC Outstanding Contribution, ACM SIGSAC Making a Difference Award, ISSA Hall of Fame, SANS Lifetime Achievement Award, Computer Society Taylor Booth Award, Upsilon Pi Epsilon ABACUS Award, CRA Distinguished Service Award, ACM President's Award, and an Air Force civilian Meritorious Service Medal. He is a Fellow of the ACM, IEEE, AAAS, $(ISC)_2$, and a Distinguished Fellow of the ISSA.

A member of ACM for over 30 years, he has been a member of or chaired nine major ACM committees including the ACM/IEEE Joint Curriculum Taskforce (1989–1991), and the editorial boards of two ACM journals. He was an ACM representative on the CRA Board of Directors (1998–2007) and has served as chair of the ACM's U.S. Public Policy Council since 1998. He is currently a member of several company, government, and institutional advisory boards, including those of the U.S. Air Force University, and the GAO.

Statement

ACM should be more engaged in some of the issues being raised by the increasing use of computing in critical applications. Issues related to computing and networking are now major topics in public policy. ACM can play an important role as a trusted source of expertise and information in these matters, and should be involved in discussing them. Whether those discussions are about the role of early computing education, the issues of privacy in a global community, or the role of social media in political dissent, ACM has members and resources that can make a difference. Over the last few decades, ACM has grown into a major international organization, with much more to offer than the presentation and publication of new research results. More than scholars, we can be leaders at the forefront of changes in the world around us.

I have been actively involved with ACM for over 30 years, as a researcher and as a member involved in issues of security, privacy, ethics, public policy, and education. This has included serving on or chairing several ACM taskforces and committees, representing ACM as a CRA board member, and serving as chair of the ACM's U.S. Public Policy Council for over a decade. These activities have provided me deep insight into both ACM and the role that we can play as informed professionals. I will bring this insight and leadership experience to my position on ACM Council.

As a member of the Council I will support opportunities to help grow ACM as an internationally prominent organization sought out for our technical expertise, not as advocates but as temperate professionals. This will be in addition to strong support of our base functions—conferences, publications, the Digital Library, and SIG activities—and of our increasingly global membership.

DOI:10.1145/2160718.2160727

Peter S. Menell

Law and Technology
Design for Symbiosis

Promoting more harmonious paths for technological innovators and expressive creators in the Internet age.

Throughout history, technologies for instantiating, reproducing, and distributing information have evolved in tandem with the creative industries that cultivate, fund, and distribute literature, music, film, and art. Although the relationship between these technology and content industries is often characterized in contemporary policy debates as deeply polarized and antagonistic, the historical record reflects a more symbiotic and multifaceted story.

Symbiosis in Historical Context

Technologists and technology companies have long played critical roles in the emergence of the content industries and many eventually came to see the protection and promotion of expressive creativity as critical to the growth and development of their businesses. It is no accident that the General Electric Corporation (GE), after acquiring rights to the Marconi wireless patents in the United States, spearheaded the formation of the Radio Corporation of America (RCA), which in turn launched the National Broadcasting Corporation (NBC), one of GE's many subsidiaries and a leading content company. General Electric profited from the manufacture and sale of broadcasting equipment, selling advertising on NBC stations, and selling radio receivers and other consumer electronics products.

The emergence of the American Society of Composers, Authors, and Publishers (ASCAP) during this period posed a significant challenge for the early radio industry. Using litigation and negotiation, ASCAP developed a licensing model that provided the foundation for a highly symbiotic relationship between the radio and later television industries and composers. The radio industry, composers, and American consumers prospered from the innovation, information dissemination, and artistic creativity spawned by this synergistic technology-content ecosystem. The availability, quality, and quantity of content increased the demand for better broadcasting and consumer electronics and vice versa.

In analogous ways, other content industries trace their development to early patentees and the companies commercializing this technology. The Victor Talking Machine Company, so-named according to some sources for having prevailed in a costly phonogram patent battle, would emerge in the marketplace as much through cultivation of the most popular performing artists as the quality of its technology.

While Thomas Edison, the developer of the competing cylinder recording technology, refused to promote "stars" on his medium, the Victor Company paid Enrico Caruso and other leading performers of the day handsomely to work exclusively on the Victor label. As a result, Victor's Victrola players won the phonogram market (and related format war) and its "His Master's Voice" discs dominated the early content sector.

In England, EMI (Electric and Musical Industries) Ltd was formed in 1931 by the merger of the Columbia Graphophone Company and the Gramophone Company (which controlled the "His Master's Voice" record label outside of the United States). This vertically integrated company produced sound recordings as well as recording and playback equipment and became one of the most important record labels of the 20th century.

ILLUSTRATION BY RICHARD MIA

Digital Symbiosis and a Widening New Digital Divide

These patterns continue today with new platforms. Apple (the computer company, not the record label) leveraged the development of the first broadly licensed online music store (iTunes) to great success in the markets for portable music devices (iPods, iPhones, iPads, and a growing family of consumer electronics) as well as music and now television and film programming. Google has used YouTube and Google Books to enhance and expand its advertising platform. YouTube's ContentID and Partner Program, under which many copyright owners pre-clear and derive advertising revenue from use of their content, leverages professional content to expand Google's family of services. Over 3,000 content owners participate in Google's revenue-sharing model. Facebook leverages music streaming services—such as Spotify—to enrich its social networking platform. Microsoft's Xbox, Netflix, and a host of other technology platforms develop and expand the symbiotic pathways that bind technology innovators and content creators. Sony's recent effort to enter the smartphone industry seeks to integrate devices, music, video, and game content seamlessly.

Yet the principal technology and content sectors are deeply divided on how to address the widespread availability of unauthorized copies of copyrighted works on the Internet. The vertical fragmentation of distribution platforms in the Internet age has created a wedge between platform innovators and creative industries that hinders e-symbiosis. Whereas broadcasters historically hosted content and hence bore direct responsibility for infringement, Internet platforms attenuate responsibility. For example, Google profits from advertising impressions and click-through monetization via its search engine and AdSense network that derive from Web sites distributing copyrighted works without authorization. Although Google might be insulated from liability so long as it complies with the Digital Millennium Copyright Act's (DMCA) takedown procedures, its bottom line benefits handsomely from delivering advertisements to people searching for pirated content. The same can be said for payment processors working for Web sites distributing copyrighted works without authority.

The major content industries—music, film, books, as well as some video game manufacturers—have called upon Congress to enact strict new enforcement tools—such as domain name blocking, expanded public enforcement powers, and broader private rights of action—to combat piracy of their works on foreign Web sites. The major Internet companies have strongly opposed these measures on the grounds that they would undermine the Internet's functioning, compromise cybersecurity, and hamper free expression. They also contend that these measures could be easily circumvented. At a minimum, such measures could substantially increase the screening costs of advertising networks and payment processors.

Although such legislation is unlikely to pass in its current form or anytime soon, the pressure to enact such strict legislation will remain as long as the Internet affords easy access to copyrighted works without authorization and successful Internet companies—from search engines to advertising networks, ISPs, and payment processors—profit from illegal distribution. As experience with the DMCA and other efforts to legislate in areas of rapid technological advancement have shown, however, Congress has limited ability to foresee and address technological change. It failed to anticipate peer-to-peer tech-

nology, which emerged within a year of the DMCA's passage, and lacks the sustained attention span or proclivity to adapt legislation in a transparent, broad-based, or systematic manner to deal with new piracy challenges. At the same time, the DMCA's ISP safe harbor is showing its age.

Collaborative Symbiotic Initiatives

Legislation is but one means for achieving collective action to address problems that cut across multiple sectors of the society and the economy. Private concerted action offers another and potentially more effective and sustainable pathway toward surmounting Internet governance challenges. Similarly, technological innovation in copyright enforcement systems can play a critical role in improving the technology-content ecosystem. A key part of the solution to the current challenges over Internet policy relates to the perceived goals of the Internet community. For much of the past two decades, many technology companies have been antagonistic toward or, at best, agnostic about addressing piracy concerns.

This reflects several factors. Cyberlibertarians have advocated the effective abolition of copyright protection on the Internet. The emergence of the Creative Commons and open source software blunt this extreme position. There is little standing in the way of creators pre-committing to sharing their works. And although open source software has made substantial inroads into several key software marketplaces (particularly those exhibiting network effects), shared creative works have had a much more modest impact. Given the continuing popularity of professionally produced and copyrighted works of authorship, the case for abolishing copyright protection on the Internet seems doubtful.

A more serious concern relates to the fear that copyright protection may be chilling technological innovation. Yet the picture is mixed. The DMCA's safe harbors have significantly limited platform innovators' exposure to copyright liability. Furthermore, corporate law's limited liability regime provides substantial insurance against crushing liability. The emergence of new technology platforms (including some that are highly parasitic) over the past decade suggests copyright liability is not significantly dampening innovation. In fact, the argument can be made that the equilibrium has tilted toward too much piracy.

Many technology companies have been antagonistic toward or, at best, agnostic about addressing piracy concerns.

As legitimate Internet content distribution models have emerged, many more technology companies stand to gain from consumers accessing content from legal sources. Authorized vendors—such as iTunes, Amazon, Vevo, Netflix, and Spotify—experience greater traffic and commerce to the extent that illegal alternatives are harder to access. ISPs can better manage their traffic when consumers access content from legitimate sources. As ISPs further integrate cable and content businesses, they will see even greater direct benefits from reduced piracy.

Notwithstanding the recent impasse over rogue Web site legislation, a quieter and more constructive pathway has been in the works. In July 2011, a group of major ISPs (SBC, AT&T, Comcast, Verizon, CSC, and Time Warner Cable) and leading content industry organizations (RIAA and MPAA) entered into a Memorandum of Understanding (MOU) to implement a flexible Copyright Alert System to discourage unauthorized distribution of copyrighted works.[a] The signatories to this MOU committed to implement an escalating system of alerts in response to alleged infringing activities: an Educational Step Copyright Alert; an Acknowledgment Step Copyright Alert; and a Mitigation Measure Copyright Alert Step. The Mitigation Step can include a reduction in upload/download transmission speeds, a step down to a lower-tier service, redirection to a landing page until the matter is resolved, and restrictions on Internet access. The MOU provides for "warning bells" along the alert steps as well as an appeals procedure.

This graduated response system provides a foundation for ISPs and copyright owners to collaborate more constructively in pursuit of a free and less piracy-prone Internet ecosystem. It builds a balanced enforcement system into ISP activities. As this experiment unfolds, the parties will be able to learn more about the ecosystem and how to adapt these techniques to better channel consumers into the legitimate marketplace.

A similar initiative produced the Principles for User Generated Content Services,[b] which encouraged the development and implementation of effective filtering technologies for user-upload Web sites. Although Google did not formally join this initiative, the ContentID system that it implemented for YouTube largely follows the UGC Principles model. In March 2011, Youku.com, China's leading Internet television company, joined the initiative. This is a particularly encouraging development in light of concerns about piracy in China.

Analogous initiatives could potentially address the roles that search engines, advertising networks, and payment processors play in enabling rogue foreign Web sites and piracy-prone cyberlocker businesses. Although it is unlikely that such approaches will entirely bridge the divide between Internet companies and traditional content owners, such collaborations provide the most promising foundation for incubating and testing designs for surmounting the dynamic challenges of Internet governance and forging collaborative relationships that can address new problems as they emerge. C

a See Memorandum of Understanding (Jul. 26, 2011); http://www.copyrightinformation.org/sites/default/files/Momorandum%20of%20Understanding.pdf.

b See Principles for User Generated Content Services; http://www.ugcprinciples.com/.

Peter S. Menell (pmenell@law.berkeley.edu) is the Herman Phleger Visiting Professor of Law (2011–2012) at Stanford Law School and the Robert L. Bridges Professor of Law and Director at the Berkeley Center for Law & Technology, University of California at Berkeley School of Law.

DOI:10.1145/2160718.2160728

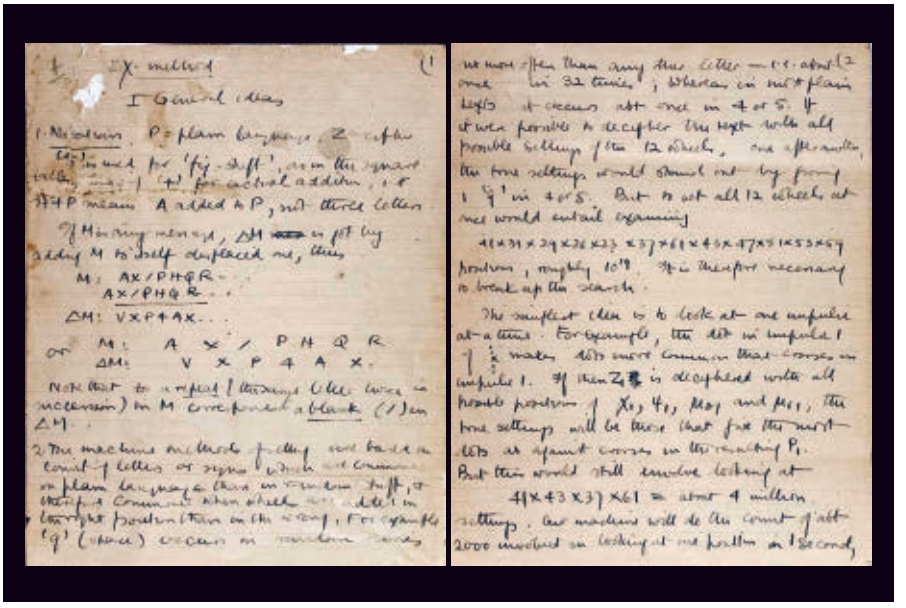
David Anderson

Historical Reflections
The Future of the Past

Reflections on the changing face of the history of computing.

I HAVE ALWAYS thought that Erasmus had his priorities correct when he remarked: "When I get a little money, I buy books; and if any is left I buy food and clothes." For as long as I can recall I have loved secondhand bookshops, libraries, and archives: their smell, their restful atmosphere, the ever-present promise of discovery, and the deep intoxication produced by having the accumulated knowledge of the world literally at my fingertips. So it was perhaps inevitable that, despite having pursued a career in computer science, I would be drawn inexorability toward those aspects of the discipline that touched most directly on the past: the history of computing and preservation.

A great deal of my personal research, particularly on the largely unknown role played by M.H.A. Newman in the post-war development of computers, has obliged me to spend many hours uncovering and studying old letters, notebooks, and other paper-based records. While some of my source material came to me after having been transferred to digital format, none of it was born digital. Future historians of computing will have a very different experience. Doubtless they, like us, will continue to privilege primary sources over secondary, and perhaps written sources will still be preferred to other forms of historical record, but for the first time since the emergence of writing systems some 4,000 years ago, scholars will be increasingly unable to access directly historical material. During the late 20th and early 21st century, letter writing has given way to email, SMS messages, and tweets, diaries have been superseded by blogs (private and public), and where paper once prevailed digital forms are making inroads and the trend is set to continue. Personal archiving is increasingly outsourced—typically taking the form of placing material on some Web-based location in the erroneous belief that merely being online assures preservation. Someone whose work is being done today is likely to leave behind very little that is not digital, and being digital changes everything.



Cracking German codes during WWII. Handwritten notes by M.H.A. Newman.

Digital objects, unlike their physical counterparts, are not capable directly of human creation or subsequent access but require one or more intermediate layers of facilitating technology. In part, this technology comprises further digital objects: software such as a BIOS, an operating system, or a word processing package; and in part it is mechanical, a computer. Even a relatively simple digital object such as a text file (ASCII format) has a surprisingly complex series of relationships with other digital and physical objects from which it is difficult to isolate it completely. This complexity and necessity for technological mediation exists not only at the time when a digital object is created but is present on each occasion when it is edited, viewed, preserved, or interacted with in any way. Furthermore, the situation is far from static as each interaction with a file may bring it into contact with new digital objects (a different editor, for example) or a new physical technology.

The preservation of, and subsequent access to, digital material involves a great deal more than the safe

THE MAX NEWMAN DIGITAL ARCHIVE, THE UNIVERSITY OF PORTSMOUTH FUTURE PROOF COMPUTING GROUP, AND ST. JOHN'S COLLEGE, CAMBRIDGE, U.K.

storage of bits. The need for accompanying metadata, without which the bits make no sense, is understood well in principle and the tools we have developed are reasonably reliable in the short term, at least for simple digital objects, but have not kept pace with the increasingly complex nature of interactive and distributed artifacts. The full impact of the lacunae will not be completely apparent until the hardware platforms on which digital material was originally produced and rendered become obsolete, leaving no direct way back to the content.

Migration

Within the digital preservation community the main approaches usually espoused are migration and emulation. The focus of migration is the digital object itself, and the process of migration involves changing the format of old files so they can be accessed on new hardware (or software) platforms. Thus, armed with a suitable file-conversion program it is relatively trivial (or so the argument goes) to read a WordPerfect document originally produced on a Data General minicomputer some 30 years ago on an iPad 2. The story is, however, a little more complicated in practice. There are something in excess of 6,000 known computer file formats, with more being produced all the time, so the introduction of each new hardware platform creates a potential need to develop afresh thousands of individual file-format converters in order to get access to old digital material. Many of these will not be produced for lack of interest among those with the technical knowledge to develop them, and not all of the tools that are created will work perfectly. It is fiendishly difficult to render with complete fidelity every aspect of a digital object on a new hardware platform. Common errors include variations in color mapping, fonts, and precise pagination. Over a relatively short time, errors accumulate, are compounded, and significantly erode our ability to access old digital material or to form reliable historical judgments based on the material we can access. The cost of storing multiple versions of files (at least in a corporate environment) means we cannot always rely on being able to retrieve a copy of the original bits.

The challenges associated with converting a WordPerfect document are simpler that those of format-shifting a digital object as complex as a modern computer game, or the special effects files produced for a Hollywood blockbuster. This fundamental task is well beyond the technical capability or financial wherewithal of any library or archive. While it is by no means apparent from much of the literature in the field, it is nevertheless true that in an ever-increasing number of cases, migration is no longer a viable preservation approach.

Emulation

Emulation substantially disregards the digital object, and concentrates its attention on the environment. The idea here is to produce a program that when run in one environment, mimics another. There are distinct advantages to this approach: it avoids altogether the problems of file format inflation, and complexity. Thus, if we have at our disposal, for example, a perfectly functioning IBM System/360 emulator, all the files that ran on the original hardware should run without modification on the emulator. Emulate the Sony PlayStation 3, and all of the complex games that run on it should be available without modification—the bits need only be preserved intact, and that is something we know perfectly well how to accomplish.

Unfortunately, producing perfect, or nearly perfect emulators, even for relatively unsophisticated hardware platforms is not trivial. Doing so involves not only implementing the documented characteristics of a platform but also its undocumented features. This requires a level of knowledge well beyond the average and, ideally, ongoing access to at least one instance of a working original against which performance can be measured.

Over and above all of this, it is critically important to document for each digital object being preserved for future access the complete set of hardware and software dependencies it has and which must be present (or emulated) in order for it for it to run (see TOTEM; http://www.keep-totem.co.uk/). Even if all of this can be accomplished, the fact remains that emulators are themselves software objects written to run on particular hardware platforms, and when those platforms are no longer available they must either be migrated or written anew. The EC-funded KEEP project (see http://www.keep-project.eu) has recently investigated the possibility of developing a highly portable virtual machine onto which emulators can be placed and which aims to permit rapid emulator migration when required. It is too soon to say how effective this approach will prove, but KEEP is a project that runs against the general trend of funded research in preservation in concentrating on emulation as a preservation approach and complex digital objects as its domain.

Conclusion

Even in a best-case scenario, future historians, whether of computing or anything else, working on the period in which we now live will require a set of technical skills and tools quite unlike anything they have hitherto possessed. The vast majority of source material available to them will no longer be in a technologically independent form but will be digital. Even if they are fortunate enough to have a substantial number of apparently well-preserved files, it is entirely possible that the material will have suffered significant damage to its intellectual coherence and meaning as the result of having been migrated from one hardware platform to another. Worse still, digital objects might be left completely inaccessible due to either not having a suitable available hardware platform on which to render them, or rich enough accompanying metadata to make it possible to negotiate the complex hardware and software dependencies required.

It is a commonplace to observe ruefully on the quantity of digital information currently being produced. Unless we begin to seriously address the issue of future accessibility of stored digital objects, and take the appropriate steps to safeguard meaningfully our digital heritage, future generations may have a much more significant cause for complaint. C

David Anderson (cdpa@btinternet.com) is the CiTECH Research Centre Director at the School of Creative Technologies, University of Portsmouth, U.K.

DOI:10.1145/2160718.2160730

Joel Waldfogel

Economic and Business Dimensions

Digitization and Copyright: Some Recent Evidence from Music

Examining the effects of stealing on producers and consumers.

Copyright-related industries have been threatened by technological change in the past decade. Events raise questions about whether copyright policy, or its enforcement, requires rethinking. Evidence-based policy-making requires, of course, evidence. That begs the question: what do we really know about piracy and its effects?

As it turns out, there has been an outpouring of research on some copyright-related issues in the past decade. The results may surprise some readers. Reports of shrinking revenue in the copyright-protected industries are a cause for concern and further exploration, but many of the answers needed to inform sensible policy are not yet available. Sensible policy has to ask the right questions, and it has to inform the answers with the right data.

Napster Started It

Much of what is known about copyright concerns the recorded music industry. Yet, recorded music makes up a relatively small part of the copyright-dependent sectors of the economy, which includes motion pictures, software, video games, and publishing, among others.

Why is the music market the favored topic among researchers even though music itself is not intrinsically important? Blame it on Napster. Because file sharing has been widespread since Napster, the music market has experienced a shock with observable effects, namely, a weakening of copyright protection. Most observers agree that technological change has sharply reduced the effective degree of protection that copyright affords since 1999.

From an economist's perspective, copyright is the grant of a monopoly right as a reward—and inducement—for creative innovation. Monopolies

The original logo for a 1980s anti-copyright infringement campaign by the British Phonographic Industry (BPI), a British music industry trade group.

PHOTOGRAPH BY CHRIS ANGLE

are, of course, bad. They raise prices and restrict output. Beyond that, they either transfer resources from buyers to sellers, or worse, they employ high prices that frustrate mutually beneficial transactions. Why do governments grant these monopoly rights? It provides an incentive for creative activity.

Napster illustrates that copyright's effectiveness depends crucially on technology. While the recent technological challenge to copyright could have affected any product that can be digitized—text, audio, or video—in reality the recorded music industry was the first to face the new challenge. And the decade since Napster has seen a dramatic reduction in revenue to the recorded music industry.

The struggles of the recorded music industry have spawned an outpouring of research seeking to document the effect of file sharing on the recording industry's revenue. Organizations representing the recording industry have argued that piracy is the cause of the reduction in revenue they have experienced. They further claim that their experience foreshadows in other creative industries, and that it will have serious consequences for bringing artists' work to market.

What Do We Know?

While the question of whether file sharing displaces legal sales and weakens copyright is interesting—and a vital question for industry—it is clearly not the only question raised by file sharing. Copyright has traditionally allowed the recording industry to generate revenue from recorded music. Weakened copyright may be bad news for sellers of recorded music, but its consequence for consumers depends on what is arguably more important:

Does piracy slow new products from being brought to market?

whether new music continues to be made available. In this column, I consider these issues in more detail.

Although stealing has long been understood to be bad, the effects of stealing on the sellers of products produced with zero marginal costs (the additional cost required to make one more unit available) are somewhat subtle. To see this, first consider the analysis of stealing music that has already been recorded. There are two separate questions: whether stealing harms sellers and whether stealing harms society as a whole.

The effects of stealing on sellers depend on its effects on consumers, who differ in their willingness to pay for the product. Some attach valuations high enough so that, had stealing been impossible, they would instead have purchased the product. When they steal, each unit of theft reduces paid consumption by one entire unit. Thus, their stealing harms sellers one-for-one.

But other consumers attach lower valuations to the product. If stealing had been impossible, they would not have purchased the product. When they steal, it brings about no reduction in revenue to the traditional sellers.

Summarizing, does stealing harm sellers? It depends on whether the instances of unpaid consumption would otherwise have generated sales. That also explains why sellers and society should not view stealing the same way. When low-valution consumers steal, their theft does not harm sellers. But their consumption does generate a gain that otherwise would not have occurred.

This is society's paradox of piracy. If stealing could somehow be confined to these circumstances, then its effects would help consumers without harming producers. Generally, it cannot, and that is why the revenue reduction experienced by sellers can have long-term consequences beyond their direct losses. Indeed, because new products need some threshold level of revenue in order to be made available profitably, transfers from producers to consumers are not benign.

New Products

The need to cover costs motivates the biggest unanswered question in this

debate. Does piracy slow new products from being brought to market? If so, and if consumers care about new products—and there is compelling evidence they do—then in the longer run, stealing can have a devastating effect on both producers and consumers. What does the evidence show?

Quantifying the effect of piracy on sales is an inherently difficult question. First, piracy is an illegal behavior and therefore not readily documented in, say, government statistics. As a result, it is difficult to get data on volumes of unpaid consumption, particularly in ways that can be linked with volumes of paid consumption (more on this later in the column). A second and equally important difficulty is the usual scourge of empirical work in the field, that is, its non-experimental nature.

Broadly, what we know about music piracy is this: the rate of sales displacement is probably far less than 1:1. Supporting this view is corroborating evidence that consumers steal music that, on average, they do not value very highly and would not otherwise purchase.

Still, the volume of unpaid consumption is quite high. So even with a small rate of sales displacement per instance of stealing, it is likely that stealing explains much if not all of the substantial reduction in music sales since Napster. That is, stealing appears to be responsible for much of the harm to the recorded music industry (for an overview, see Leibowitz[3]). Yet, that does not answer the whole question. Another crucial question is whether the reduction in revenue reduces the flow of new products to market.

Perhaps because this is a difficult question to study, there has been almost no research on the topic. Recorded music is the only industry that has experienced a massive reduction in revenue. This can play the role of an "experiment" to see whether the flow of new products declines following a weakening in effective copyright protection. However, it is only one experiment, and researchers do not have the option of rerunning it with slightly different circumstances, so getting definitive answers is challenging.

While some researchers have documented an increase in the number of new recorded music products and independent labels since 2000, this evidence is ambiguous. Most of these products are consumed little if at all (for example, see Handke[1,2] and Oberholzer-Gee and Strumpf[4]).

The stable flow of new products appears to be a puzzle when set against the sharply declining revenue.

It is tempting instead to ask how many products released each year sell more than, say, 5,000 copies. But not so fast! That approach is undermined entirely by the fact that stealing is on the rise, so that the meaning of selling a particular number of copies changes over time.

In recent work I took a new approach (see Waldfogel[5,6]). I document the evolution of quality using critics' lists of the best albums of various times periods (for example, best of the decade). Any album on one of these lists surpasses some threshold of importance. The number of albums on a list released in each year provides an index of the quantity of products passing a threshold over time. I "splice" nearly 100 such lists, and I can see how the ensuing overall index evolves over time, including—importantly—during the "experimental" period since Napster. The index tracks familiar trends in the history of popular music: it rises from 1960 to 1970, falls, then rises again in the early 1990s. It is falling from the mid-1990s until the 1999 introduction of Napster.

What happens next? Rather than falling—as one might expect for an era of sharply reduced revenue to recorded music—the downturn of the late 1990s ends, and the index is flat from 2000 to 2009. While it is of course possible that absent Napster, the index would have risen sharply in a flowering of creative output, it nevertheless seems clear that there has been no precipitous drop in the availability of good new products.

The stable flow of new products appears to be a puzzle when set against the sharply declining revenue. Its resolution may lie in the fact that the costs of bringing new works to market have fallen. Creation, promotion, and distribution have all been revolutionized by digital technology and the Internet.

The Debate Continues?

Beware of simple answers in debates about piracy. Our knowledge remains incomplete. Representatives of copyright-protected industries are understandably concerned about threats to their revenue from piracy. In some cases—for example, recorded music—they can point to compelling evidence that their revenues are down. That is, most evidence available so far shows harm to sellers from consumer theft.

On the longer-run question of continued supply, however, we know far less. Has the supply of quality music declined due to piracy? Probably not. Finding desirable intellectual property institutions—that properly balance the interests of various parties—requires attention to the interests of both producers and consumers. Reports of shrinking revenue in the copyright-protected industries are a cause for concern and further exploration, but they are not alone sufficient to guide policy-making. C

References

1. Handke, C. Digital copying and the supply of sound recordings. Unpublished Manuscript, 2010; http://www.acei2010.com/upload/acei/handke.pdf.
2. Handke, C. Plain destruction or creative destruction? Copyright erosion and the evolution of the record industry. *Review of Economic Research on Copyright Issues 3*, 2 (Feb. 2006), 29–51.
3. Liebowitz, S.J. File sharing: Creative destruction or just plain destruction? *Journal of Law and Economics 49*, 1 (Apr. 2006), 1–28.
4. Oberholzer-Gee, F. and Strumpf, K. File-sharing and copyright. *NBER's Innovation Policy and the Economy series 10*. J. Lerner and S. Stern, Eds. MIT Press, 2009.
5. Waldfogel, J. Bye, bye, miss American pie: The supply of new recorded music since Napster. National Bureau of Economic Research Working Paper 16882, March 2011b.
6. Waldfogel, J. Music piracy and its effects on demand, supply, and welfare. *Innovation Policy and the Economy 12* (June 2011), J. Lerner and S. Stern, Eds.

Joel Waldfogel (jwaldfog@umn.edu) is a professor and the Frederick R. Kappel Chair in Applied Economics in the Carlson School of Management at the University of Minnesota.

DOI:10.1145/2160718.2160729

Alexander Repenning

Education

Programming Goes Back to School

Broadening participation by integrating game design into middle school curricula.

EDUCATIONAL PROGRAMMING ENVIRONMENTS have tried to broaden the participation of women and minorities in computer science education by making programming more exciting and accessible. Starting in 1991, our research at the University of Colorado with AgentSheets explored the idea of supporting students with game design and simulation building through drag-and-drop interfaces. Early on, AgentSheets enabled students to build and share their creations through the Web[4] and more recently AgentCubes[3] is exploring the idea of 3D fluency through gentle slope 3D. Others followed with slightly different aims. For instance, Alice explored the idea of storytelling and Scratch the idea of animation. However, the point of this column is not to compare programming environments but to explore why programming is still not used in public schools, and particularly middle schools, in spite of these tools.

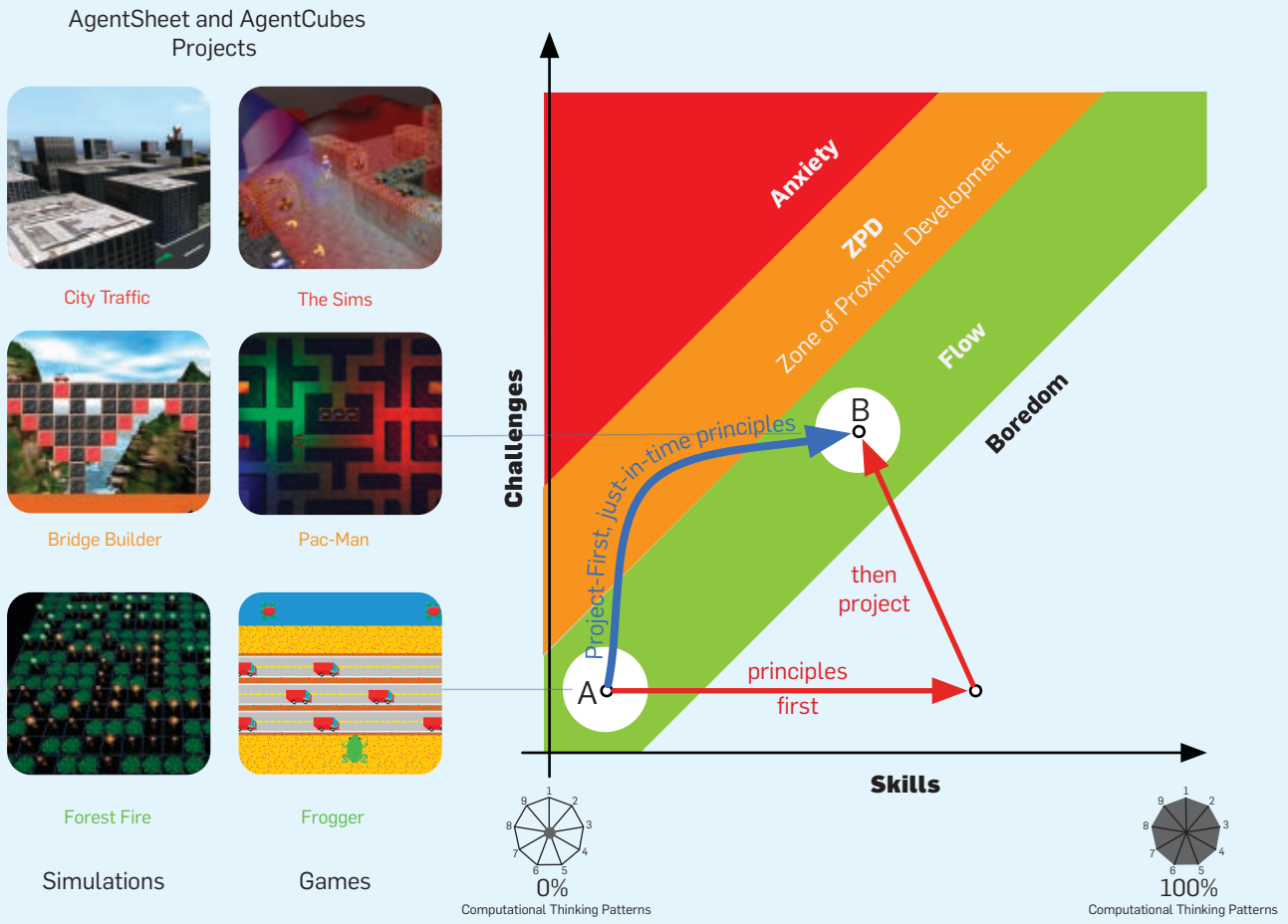
While the broadening participation situation at the high school level still looks bleak[1] it is much worse at the middle school level.[5] To the degree that programming is found at middle schools at all, it is usually offered as after-school programs. Middle school is an essential period of life during which students, especially girls and minority students, make decisive but often-unfortunate career decisions such as "science is not for me." How can we shift middle school computer science education from isolated after-school efforts to a systemic model in which computer science is integrated into the school curriculum and taught in required classes at districtwide levels? Our Scalable Game Design Initiative, with over 8,000 participants, is showing great initial promise and is formulating a new approach in various settings that include inner city, remote rural, and Native American communities. But before I describe our strategies and results, I will begin with a story.

The ability to create a playable game is essential if students are to reach a profound, personally changing "Wow, I can do this" realization.

Six years ago one of our teachers decided to use AgentSheets to introduce game design as a programming activity in his regular computer education class. Previously, his class was limited to topics such as keyboarding and PowerPoint. When I visited his class for the first time, I was truly surprised. I immediately noticed a completely different participant composition. Instead of the typical single girl found in the computer club, about 50% of the students were female—there was also a large percentage of minority students. I asked students if they liked the game design activity. They did. However, many also indicated they would never have gone to a Friday afternoon computer club to do programming. One student summarized her perception of computer science as "hard and boring." The basically universal excitement about game design in a required class suggested a strategy to simultaneously increase the exposure of student interest in computer science and broaden participation.

Since then, the NSF Innovative Technology Experiences for Students and Teachers (ITEST) program enabled us to expand this experience into a complete strategy for middle school computer science education. We were able to study the systemic integration of a game design-based computational thinking (CT) curriculum into middle schools. The resulting Scalable Game Design[6] framework (see the accompanying figure) includes a curriculum of increasingly advanced game design activities that range from basic classic 1980s arcade games such as Frogger (bottom-left in the figure) to more contemporary games such as The Sims. The framework includes sophisticated artificial intelligence concepts (top-left in the figure). The right-hand side shows our Zones of Proximal Flow dia-

Zones of Proximal Flow: The Scalable Game Design framework includes a CT curriculum balancing CT challenges with CT skills. A Project-First learning path advances students from basic skills to more advanced ones effectively.



gram (a combination of Csíkszentmihályi's Flow diagram with Vygotsky's Zone of Proximal Development conceptualization). The essence of Scalable Game Design is that programming challenges and skills should be balanced and there are different paths, some better suited than others for broadening participation, along which students can advance their skills and tackle more advanced challenges.

The four main goals and approaches of the Scalable Game Design framework are described here.

Exposure: Develop a highly adoptable middle school CT curriculum integrated into existing computer education and STEM courses so that a potentially very large and diverse group of children is exposed to CT concepts. The small number of middle schools that offer programming-related after-school programs attract only a small number of students. A successful computer club at a middle school may attract about 20 students consisting almost exclusively of boys already interested in programming. In the Boulder Valley School District in Colorado, a technologically progressive and affluent district, only one of 12 middle schools offers a computer club. With Scalable Game Design introduced to the curriculum, 10 of the 12 middle schools now offer CT classes. Additionally, most of the students participate, resulting in an extremely high participation of girls and minority students. Out of the over 8,000 study participants, 45% are girls and 48% are minority students. In some of the larger schools we have 400 students per year per school participating. It is clear that a curriculum-integrated approach has a much higher potential for systemic impact compared to an after-school program. However, to reach this kind of exposure school districts must see direct value in CT education and find ways to integrate it into existing courses.

The left side of the figure shows some AgentSheets/AgentCubes STEM simulations and games employed in existing courses to address STEM and educational technology standards. STEM simulations require computational thinking tools such as AgentSheets that can run advanced simulations with thousands of objects, include scientific visualization, and provide means to export data to Excel and other tools. We have found that most schools have the capacity to incorporate these activities into existing courses. With over 16 million middle school students in the U.S., the potential exposure of students to CT through this strategy is enormous.

Motivation: Create a scalable set of game design activities ranging from low-threshold to high-ceiling activities

so that students with no programming background can produce complete and exciting games in a short amount of time while still moving on a gradual trajectory to the creation of highly sophisticated games. Scalable Game Design has a uniquely low threshold for teachers and students making game design accessible across gender and ethnicity. We have developed a professional development program based on approximately 35 contact hours in which we train teachers to have students build their first playable game from scratch in about a week (for example, 5 lessons x 45 minutes). The ability to create a playable game is essential if students are to reach a profound, personally changing "Wow, I can do this" realization.

On the right side of the figure, zones of motivation are delineated based on challenge versus skills. These zones are Anxiety, Zone of Proximal Development, Flow, and Boredom. In middle schools we have found the path called Project-First is significantly more effective than paths relying on learning many principles first without the presence of a concrete and interesting challenge. The Project-First path maneuvers students into the Zone of Proximal Development where, with proper support, they quickly learn relevant CT concepts. Motivational levels, as measured by the expressed interest to continue with similar classes, are extremely high: 74% for boys and 64% for girls; and 71% for white and 69% for minority students. In most schools these are not self-selected students. While there are no well-established baselines, our teachers considered a desire by one-third of their students to continue a success.

Education: Build computational instruments that analyze student-produced projects for CT skills so that learning outcomes can be objectively measured. These outcomes include learning trajectories and transfer of CT concepts from game design to simulation building. What do students really learn? While CT definitions are still somewhat forthcoming, one school director created a succinct statement of expectation—"I would want to walk up to a student participating in game design and ask: Now that you can make space invaders, can you also make a science simulation?" This question of transfer should be at the core of computational thinking education. If students learn to build a game but have no notion of how to transfer their skills into science simulations, then game design has no educational justification for being in a curriculum. We devised the Computational Thinking Pattern Analysis[2] as an analytical means of extracting evidence of CT skill directly from games and simulations built by students. Most excitingly, this lets us go beyond motivational research and study the educational value of game design, including the exploration of learning trajectories and transfer from game design to STEM simulation building. To enable this transfer, the professional development of teachers stresses these computational thinking patterns as a means of promoting CT.

Pedagogy: Systematically investigate the interaction of pedagogical approaches and motivational levels so that teachers can broaden participation. With school sites in Alaska, California, Georgia, Ohio, South Dakota, Texas, and Wyoming and over 10,000 student-created games and simulations, we were able to explore a uniquely rich set of motivational and educational data.[6] We found the main common factor supporting motivational levels and skills across different school contexts, gender, and ethnicity was scaffolding. Of all the factors we considered, scaffolding was the only significant one. Scaffolding, a pedagogical aspect indicating the degree and kind of support provided by a teacher, was assessed through classroom observation. Direct instruction, which provides a very high degree of scaffolding, highly polarized motivational levels between boys and girls. With direct instruction a teacher provides step-by-step instructions at a detailed level (for example, "click this button," "paint the frog green"). Direct instruction is particularly unappealing to girls. With less scaffolding, such as with guided discovery, a teacher employs a more inquiry-based approach that includes classroom discussion (such as "what should we do?" and "how can we do this?"). In guided discovery, the motivational levels of girls not only approached the motivational levels of boys but often exceeded it. In a number of classes, using guided discovery raised the motivational levels of girls to 100%. This is exciting because it suggests that broadening participation is not a question of difficult-to-change factors such as school affluence. Preconceived notions such as lower interest of girls in programming turned out to be unsubstantiated. Even in cases where most literature would suggest a gender effect, for instance if there were significantly fewer girls in a class, we found that the right level of scaffolding could raise the level of motivation in girls to be even higher than that of boys.

Conclusion

We believe we have found a systemic strategy for integrating CT education in middle schools in a way that exposes a large number of students and is appealing to girls as well as to underrepresented students. The Scalable Game Design project will continue as an NSF Computing Education for the 21st Century (CE21) project to advance a research based framework for broadening participation. We invite interested schools to participate. C

This work is supported by the National Science Foundation under grant numbers 0848962, 0833612, and 1138526. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.

References

1. Computer Science Teachers Association (CSTA) Board of Directors. *Achieving Change: The CSTA Strategic Plan*, 2008.
2. Ioannidou, A., Bennett, V., Repenning, A., Koh, K., and Basawapatna, A. Computational thinking patterns. Paper presented at the 2011 Annual Meeting of the American Educational Research Association (AERA) in the symposium "Merging Human Creativity and the Power of Technology: Computational Thinking in the K–12 Classroom" (Apr. 8–12, 2011, New Orleans, LA).
3. Ioannidou, A., Repenning, A. and Webb, D. AgentCubes: Incremental 3D end-user development. *Journal of Visual Language and Computing* (2009).
4. Repenning, A., and Ambach, J. Tactile programming: A unified manipulation paradigm supporting program comprehension, composition, and sharing. In *Proceedings of the 1996 IEEE Symposium of Visual Languages*, (Boulder, CO, Computer Society), 1996, 102–109.
5. Webb, H. Injecting computational thinking into career explorations for middle school girls. In *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC '11*, (Pittsburgh, PA, Sept. 2011), IEEE Computer Society, Los Alamitos, CA.
6. Webb, D.C., Repenning, A., and Koh, K. Toward an emergent theory of broadening participation in computer science education. ACM Special Interest Group on Computer Science Education Conference, (SIGCSE 2012), (Raleigh, North Carolina, Feb. 29–Mar. 3, 2012).

Alexander Repenning (ralex@cs.colorado.edu) is a computer science professor at the University of Colorado, a member of the Center for Lifelong Learning and Design at the University of Colorado, and the founder of AgentSheets Inc.

DOI:10.1145/2160718.2160731

Abraham Bernstein, Mark Klein, and Thomas W. Malone

Viewpoint
Programming the Global Brain

Considering how we can improve our understanding and utilization of the emerging human-computer network constituting the global brain.

New ways of combining networked humans and computers—whether they are called collective intelligence, social computing, or various other terms—are already extremely important and likely to become truly transformative in domains from education and industry to government and the arts. These systems are now routinely able to solve problems that would have been unthinkably difficult only a few short years ago, combining the communication and number-crunching capabilities of computer systems with the creativity and high-level cognitive capabilities of people. And all this is supercharged by the emergent generativity and robust evaluation that can come from the many eyes of large crowds of people. As the scale, scope, and connectivity of these human-computer networks increase, we believe it will become increasingly useful to view all the people and computers on our planet as constituting a kind of "global brain."

We still only poorly understand, however, how to "program" this global brain. We have some stunning success stories (such as Wikipedia, Google), but most applications still fail, or require a long series of trial-and-error refinements, for reasons we only poorly understand. Because people are involved, programming the global brain is deeply different from programming traditional computers. In this Viewpoint, we will consider this challenge, exploring the nature of these differences as well as issuing a call to arms describing open research challenges that need to be met in order to more fully exploit the enormous potential of the global brain.

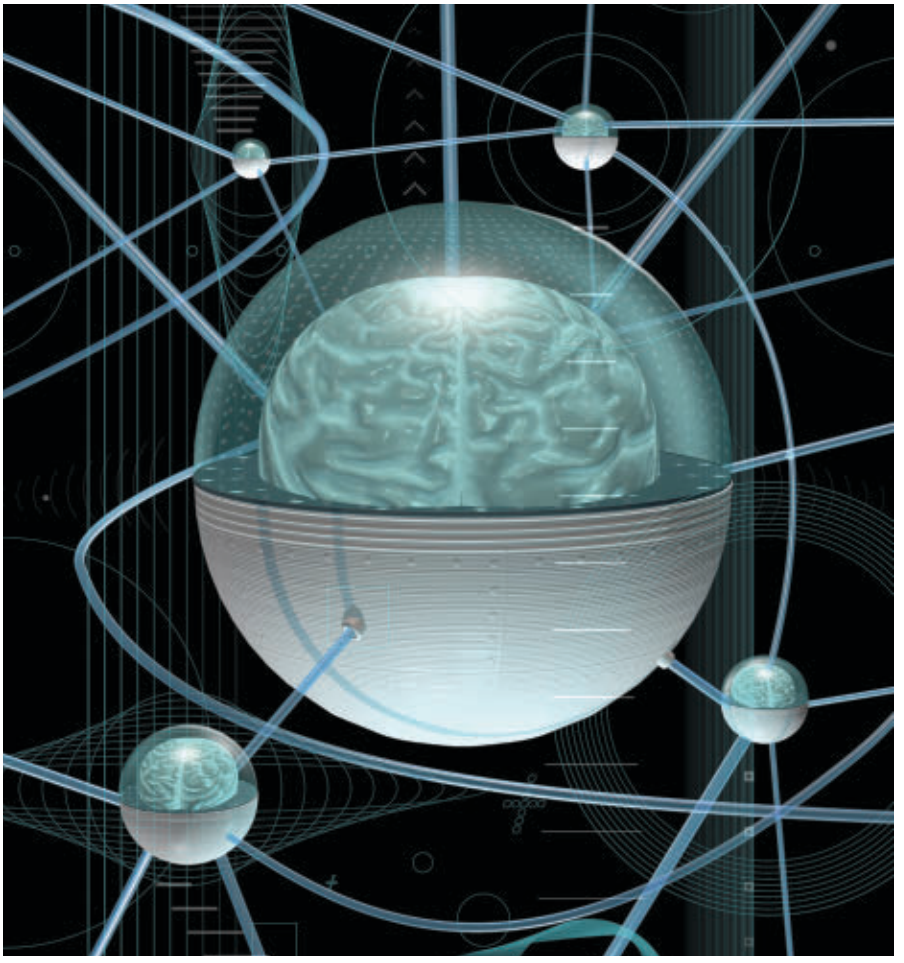


ILLUSTRATION BY JOHN HERSEY

What Makes the Global Brain Different?

There are already literally hundreds of compelling examples of the global brain at work, collectively representing the contributions of many millions of people and computers.[3,10] These range

from systems where individuals perform simple micro-tasks (http://mturk.com) to where they compete to solve complex engineering problems (http://innocentive.com). Some systems harness the "wisdom of crowds" in contexts that range from "citizen science"[4] (http://fold.it, http://galaxyzoo.org) to predicting box office performance (http://hsx.com). Open idea ecologies (where crowds of people share, recombine, and refine each other's creative outputs) have produced remarkable results for everything from videos (http://youtube.com) and encyclopedias (http://wikipedia.org) to software (Linux). Systems have been developed that look for individual task-focused "geniuses" (http://marketocracy.com) or, conversely, datamine the activity traces of millions of Internet users (Google's search engine). While these systems cover an enormous range of approaches, it has become clear from these experiences that programming the global brain is different from programming traditional computers in some fundamental ways. Some of the most important differences include:

▸ *Motivational diversity:* People, unlike current computational systems, are self-interested and therefore require appropriate incentives—anything from money, fame, and fun to altruism and community—to perform tasks. These incentives have to be carefully designed, moreover, to avoid people gaming the system or causing outright damage. In some cases, one may even use their motivation to do one task to accomplish another, as in reCAPTCHA, where people OCR documents as a side effect of passing a human versus bot test.[12]

▸ *Cognitive diversity:* In most computer systems we deal with a limited range of diversity—in terms of memory, speed, and device access. People, by contrast, vary across many dimensions in the kinds of tasks they can do well, and their individual strengths are only incompletely understood at best. This implies qualitative differences in how (and how well) we can expect to match tasks and resources in a global brain context.

▸ *Error diversity:* With traditional computers, we worry much more about outright failure than other kinds of errors. And the other errors are usually highly deterministic and limited in diversity because a relatively small range of software is typically replicated across millions of computers. People, by contrast, are prone to a bewildering and inconsistent variety of idiosyncratic deviations from rational and accurate performance. The global brain, therefore, calls for a radically more capable quality assurance oriented toward the particular kinds of errors that occur with human participants. Fortunately, the global brain also provides access, at least currently, to a huge human "cognitive surplus,"[11] so that, for instance, quality mechanisms based on previously unthinkable levels of redundancy have become practical.[10]

Because people are involved, programming the global brain is deeply different from programming traditional computers.

These attributes lead, in turn, to the possibility of new, and potentially troubling, forms of *emergence*. Crowds of people, when engaged in solving interdependent problems, can evince emergent behaviors that range from groupthink (where decision-makers converge prematurely on a small subset of the solution space) to balkanization (where decision-makers divide into intransigent competing cliques) to chaotic dynamics (for example, stock market bubbles and crashes). While emergence is, of course, not unique to the global brain, it is probably made much more challenging by the unprecedented combination of microsecond computer and communications speeds, globe-scale interdependencies, and human diversity.

The Need for New Programming Metaphors

How, then, can we effectively program a global brain, characterized as it is by unique challenges (and opportunities)? We believe a fundamental requirement is developing powerful new programming metaphors that more accurately reflect the ways people and computers can work together in the global brain. For instance, today's innovative collective intelligence systems embody a set of common design patterns,[8] including *collections* (where people create independent items, such as YouTube videos), *collaborations* (where people create interdependent items, such as Linux modules), and various kinds of *group and individual decisions* (such as *voting*, *averaging*, *social networks*, and *markets*). These design patterns, in turn, can be embodied in various programming metaphors, such as:

▸ *An idea ecology:* The global brain can host a constant ferment of idea generation, mutation, recombination, and selection, analogous to biological evolution. In this context, programming consists of soliciting collections of items and specifying a fitness function for choosing among them. Interesting examples of this include the MATLAB open programming contests (for software) and YouTube (for videos).

▸ *A web of dependencies:* Many important problems (such as product, process, and policy definition) can be viewed as collaborations, where multiple diverse agents try to solve interdependent pieces of a larger problem. The global brain can detect when conflicts appear between sub-solutions, as well as guide agents toward a globally consistent result. In this context, programming includes defining the task decomposition and specifying ways of managing the interdependencies among sub-problems. Early examples of this include virtual mockups such as the digital pre-assembly system Boeing used in the design of the 777 aircraft.

▸ *An intellectual supply chain:* For some problems, we can view the global brain as a supply chain, where a sequence of tasks and information flows among people and machines can be specified in advance. In this context, programming can be defined in terms already familiar to computer scientists as processes and dataflows. Interesting examples of this idea include the Turkit[6] and Crowdforge[5] systems, which have been applied to such tasks as writing and editing articles.

▸ *A collaborative deliberation:* The global brain can also be used to enact decision processes where people and software systems select issues to consider, enumerate and critique solution alternatives, and then choose some subset of these solutions. In this context, programming can be viewed as defining the rules for identifying issues and enumerating, critiquing, and selecting solutions.

▸ *A radically fluid virtual organization:* Sometimes it is useful to view the global brain as a collection of evanescent virtual organizations, which rapidly coalesce, perform, and disband in light-speed open markets. In this context, programming includes identifying the task requirements and selecting among the organizations that are offering to perform the task. Interesting examples of this idea include odesk.com and elance.com.

▸ *A multi-user game:* Many tasks can be presented as a multi-user game, where useful outcomes are achieved as a result, sometimes unintentional, of playing the game. In this context, programming consists of specifying the rules and incentives for game play. Interesting examples of this include fold.it and the Google Image Labeller.

Making such global brain programming metaphors a reality will, in turn, require progress along several fronts:

Creating "social operating systems." An operating system, in the context of a single computer, manages the allocation of hardware and software resources such as memory, CPU time, disk space, and input/output devices. A social operating system, in addition to doing all these things, will also have to manage the mustering and allocation of human resources to tasks. This will require fast, robust infrastructures for contracts, payments, or other motivational elements, as well as scalable task-to-resource matchmaking such as markets. These will be challenging problems because people (unlike hardware resources) are diverse in all the ways we have described. But providing easy-to-use solutions for the problems of finding and motivating human participants—rather than requiring each system developer to solve this problem individually—will greatly facilitate programming the global brain.

Making global brain metaphors a reality will require progress along several fronts.

Defining new programming languages. Conventional programming languages are, out of necessity, fully prescriptive, describing the algorithms to be executed in exhaustive detail. Such languages are often not a good match, however, for specifying tasks with human participants. The programming languages for the global brain will, therefore, need to support a "specificity frontier" of varying degrees of detail in task definition.[2] One end of this frontier involves defining programs that allocate highly specific micro-tasks to people and link them into larger workflows. In the middle ground, we may use constraint-based programs, which specify (for example, in a game setting) the goals as well as the *limits* on how they can be achieved, but not *how* they should be achieved. At the far end of the specificity frontier, programming may be limited to simply stating incomplete goal specifications. Additionally, we need to expand the range of abstractions such programming languages offer. While traditional programming languages incorporate constructs such as loops and recursion, a global brain programming language may also need to incorporate abstractions such as group decision processes, contests, and collaborative steps.[1,9]

Promulgating new software engineering skills. Programmers will need to develop new mind-sets about, for example, such basic concepts as what a "program" is and what "testing" and "debugging" mean. They will need to become not just software architects and algorithm implementers, but also *organizational* or even *societal* architects able to think systematically about things like motivations, coalitions, emergence, and so on. Perhaps most fundamentally, they will need to transition from a purely "command-and-control" perspective for organizing people to one oriented around cultivating and coordinating[7] societies made up of many diverse independent players.

A Call to Arms

We have attempted to identify some of the key challenges, opportunities, and strategies involved in programming the emerging global brain. Learning to do this well is, perhaps, even more urgent than many people realize. Our world is faced with both existential threats of unprecedented seriousness (such as the environment) and huge opportunities (such as for scientific and social progress). We believe that our ability to face the threats and opportunities of the coming century will be profoundly affected by how well, and soon, we can master the art of programming our planet's emerging global brain. C

References

1. Ahmad, S., Battle, A., and Kamvar, S. The Jabberwocky programming environment for structured social computing. UIST '11 (Santa Barbara, CA, Oct. 16–19, 2011).
2. Bernstein, A. How can cooperative work tools support dynamic group processes? Bridging the specificity frontier. In *Proceedings of the International Conference on Computer Supported Cooperative Work* (CSCW 2000) 2000, ACM.
3. Doan, A., Ramakrishnan, R., and Halevy, A. Crowdsourcing systems on the World-Wide Web. *Commun. ACM 54*, 4 (Apr. 2011), 86–96.
4. Hand, E. Citizen science: People power. *Nature 466* (2010), 685–687.
5. Kittur, A., Smus, B., and Kraut, R. Crowdforge: Crowdsourcing complex work. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '11)*. ACM, New York (2011).
6. Little, G. et al. TurKit: Human computation algorithms on mechanical turk. In *Proceedings of the 23rd Annual ACM Symposium on User Interface Software and Technology (UIST '10)*. ACM, New York, 2010, 57–66.
7. Malone, T.W. *The Future of Work*. Harvard Business School Press, Boston, MA, 2004.
8. Malone, T.W., Laubacher, R., and Dellarocas, C. The collective intelligence genome. *Sloan Management Review 51*, 3 (Spring 2010), 21–31.
9. Minder, P. and Bernstein, A. CrowdLang—First steps towards programmable human computers for general computation. In *Proceedings of the 3rd Human Computation Workshop (HCOMP 2011)*, AAAI-Press.
10. Quinn, A.J. and Bederson, B.B. Human computation: A survey and taxonomy of a growing field. In *Proceedings of CHI 2011*, (Vancouver, BC, Canada, May 7–12, 2011).
11. Shirky, C. *Cognitive Surplus: Creativity and Generosity in a Connected Age*. Penguin Press, New York, 2010.
12. von Ahn, L. et al. reCAPTCHA: Human-based character recognition via Web security measures. *Science 321* (2008), 1465–1468.

Abraham Bernstein (bernstein@ifi.uzh.ch) is a professor at the University of Zurich's Department of Informatics and heads the Dynamic and Distributed Information Systems Group.

Mark Klein (m_klein@mit.edu) is a principal research scientist at the MIT Center for Collective Intelligence in Cambridge, MA.

Thomas W. Malone (malone@mit.edu) is the director of the MIT Center for Collective Intelligence and the Patrick J. McGovern Professor of Management at the MIT Sloan School of Management in Cambridge, MA.

DOI:10.1145/2160718.2160732

Armando Fox and David Patterson

Viewpoint
Crossing the Software Education Chasm

An Agile approach that exploits cloud computing.

Via a remarkable alignment of technologies, the future of software has been revolutionized in a way that also makes it *easier* to teach. Cloud computing and the shift in the software industry[a] toward Software as a Service[b] (SaaS) using Agile development has led to tools and techniques that are a much better match to the classroom than earlier software development methods. In addition, modern programming frameworks for Agile development like Ruby on Rails demonstrate that big ideas in programming languages can deliver productivity through extensive software reuse. We can leverage such productivity to allow students to experience the whole software life cycle repeatedly within a single college course, which addresses many criticisms from industry about software education. By using free-trial online services, students can develop and deploy their SaaS apps in the cloud without using (overloaded) campus resources. Our enthusiasm for Agile, SaaS, and cloud computing is not based just on improving students' employability; rather, we are excited that they can learn and practice software engineering fundamentals, and that they use them even after graduation. Our 50,000 online students are a testimony to the popularity of the material and the scalability of our SaaS "courseware."

The Complaints and the Defense

While new college graduates are good at coding and debugging,[1] employers complain about other missing skills that are equally important. A standard faculty reaction to such criticisms is that we are trying to teach principles that prepare students for their whole careers; we are not trade schools that teach the latest superficial fads.

To understand the complaints in more depth, we spoke to representatives from a half-dozen leading software companies. We were struck by the unanimity of the number-one request from each company: that students learn how to enhance sparsely documented legacy code. In priority order, other requests were making testing a first-class citizen, working with nontechnical customers, performing design reviews, and working in teams. We agree that the social skills needed to work effectively with nontechnical customers, work well in teams, and perform effective design reviews are helpful for the students' whole careers—the question is how to fit them

Turning software concepts into Rails development tools.

SWEBOK Concept	Agile Version	Rails Tool
Software Requirements	User Stories, Behavior-Driven Design (BDD)*	Cucumber (http://cukes.info)
Project Management	Velocity, Version Control	Pivotal Tracker (http://www.pivotaltracker.com/), GitHub (https://github.com/)
Software Verification and Testing	Test-driven development (TDD)	RSpec (unit/functional testing; http://rspec.info/), SimpleCov (coverage measurement; http://rubygems.org/gems/simplecov)
Software Maintenance	Refactoring to control complexity	metric-fu (captures metrics of code complexity e.g., cyclomatic and ABC; http://metric-fu.rubyforge.org/), pingdom (monitors 99%ile response times to deployed app from around the world; http://pingdom.com/). IDEs such as Aptana (http://aptana.com/) and RubyMine (http://www.jetbrains.com/ruby/) include refactoring support, method name autocompletion, visualizing dependencies among classes, and so on.
Software Lifecycle	Iterations	See Figure 3

*BDD is a variation of TDD that suggests writing higher-level acceptance or integration tests first before writing code.

a Virtually every shrink-wrap program is offered as a service, including PC standard-bearers like Office (see Office 365; http://www.microsoft.com/en-ca/office365/online-software.aspx) and TurboTax (see TurboTax Online; http://turbotax.intuit.com/personal-taxes/online/compare.jsp).

b Instead of binaries that must be installed on a local computer, SaaS delivers software and associated data as a service over the Internet, often via a thin program on client devices such as a browser.

into a course. Similarly, no one questions the value of emphasizing testing—the question is how to get students to take it seriously.

Instructors respond that even if we agreed with the recommendations, time constrains how much one class can cover. A typical undergraduate workload of four courses per term and a 50-hour workweek gives students about 12 hours per week per course, including lectures, labs, exams, and so forth. This works out to approximately 120 hours per quarter to 180 hours per semester, or just three to four weeks for a full-time developer!

A Classroom Opportunity: Agile Development

The Agile Manifesto signaled a paradigm shift for many software applications. This approach embraces change as a fact of life; small teams of developers continuously refine a working but incomplete prototype until the customer is happy with the result, with the customer offering feedback with every iteration. Agile emphasizes *Test-Driven Development*[c] (TDD) to reduce mistakes, which addresses industry's request to make testing a first-class citizen; *user stories*[d] to reach agreement and validate customer requirements, which addresses working with nontechnical customers; and *velocity*[e] to measure progress. The Agile software philosophy is to make new versions available every two weeks. The assumption is basically continuous code refactoring over its lifetime, which develops skills that can also work with legacy code. Clearly, small teams and multiple iterations of incomplete prototypes could match the classroom.

Note that we do not tell students that Agile is the only way to develop software; indeed, we explain Agile is inappropriate for safety-critical apps, for example. We believe that new programming methodologies develop and become popular in response to new opportunities, so we tell students to expect to learn new methodologies and frameworks in the future. Our experience is that once students learn the classic steps of software development and have a positive experience in using them via Agile, they will use these key software engineering principles in other projects no matter which methodology is used (see Figure 5).

The only hope for addressing all the concerns from industry within the course time constraints is to use a highly productive programming framework.

A Classroom Target for the Post-PC Era: "I Do and I Understand"

To motivate students, it is helpful to use a platform that allows them to create compelling apps. In this emerging post-PC era, mobile applications for smartphones and tablets and Software as a Service (SaaS) for cloud computing are both compelling. (50,000 students are evidence that SaaS is indeed compelling.) As you can teach the principles with either target, given the time constraints mentioned earlier, why not pick the platform that has the most productive tools? Our view is that the only hope for addressing all the concerns from industry within one course is to use a highly productive programming framework.

Our experience is that the Rails ecosystem has by far the best tools to support test-driven development, behavior-driven design, and Agile processes, many of which are made possible by intellectually deep Ruby language features such as closures, higher-order functions, functional idioms, and

c In TDD you first write a failing test case that defines a new feature, and then write code to pass that test.

d A user story is a few nontechnical sentences that captures a feature the customer wants to include in the app.

e Velocity is calculated by estimating units of work per user story and then counting how many units are completed.

Calendar of Events

May 15–17
Computing Frontiers Conference,
Caligari, Italy,
Sponsored: SIGMICRO,
Contact: John Feo,
Email: john.feo@pnl.gov,
Phone: 509-375-3768

May 15–16
Workshop on Software and Compilers for Embedded Systems,
Sankt Goar, Germany,
Contact: Henk Corporaal,
Email: h.corporaal@tue.nl

May 16–21
The 9th Annual Conference on Theory and Applications on Models and Computation,
Beijing, China,
Contact: Li Angsheng,
Email: angsheng@ios.ac.cn

May 21–24
6th International Conference on Pervasive Computing Technologies for Healthcare,
San Diego, CA,
Contact: Dr. Rosa I. Arriaga,
Email: arriagea@cc.gatech.edu

May 21–25
Shape Modeling International,
College Station, TX,
Contact: Ergun Akleman,
Email: ergun.akleman@gmail.com,
Phone: 979-845-6599

May 21–25
The 2012 International Conference on Collaboration Technologies and Systems,
Denver, CO,
Contact: Geoffrey Fox,
Email: gcfexchange@gmail.com

May 22–25
International Working Conference on Advanced Visual Interfaces,
Capri Islands (Naples), Italy,
Contact: Tortora Genoveffa,
Email: tortora@unisa.it

May 23–25
Euro-American Conference on Telematics and Information Systems,
Valencia, Spain,
Contact: Samper J. Javier,
Email: jjsamper@uv.es

metaprogramming. The accompanying table shows critical topics from the SWEBOK (software engineering body of knowledge),[5] the Agile technique for that topic, and the Rails tool that implements that technique. Because these tools are lightweight, seamlessly integrated with Rails, and require virtually no installation or configuration—some are delivered as SaaS—students quickly begin learning important techniques directly by doing them. We agree with Confucius: "I hear and I forget. I see and I remember. I do and I understand." Students see and use tools that we can explain and check, rather than just hear lectures about a methodology and then forget to use them in their projects.

Figure 1. An example feature for a cash register application and one "happy path" user story for that feature.

```
Feature: Division
  In order to avoid silly mistakes
  Cashiers must be able to calculate a fraction

  Scenario: Regular numbers
    Given I have entered 3 into the calculator
    And I have entered 2 into the calculator
    When I press divide
    Then the result should be 1.5 on the screen
```

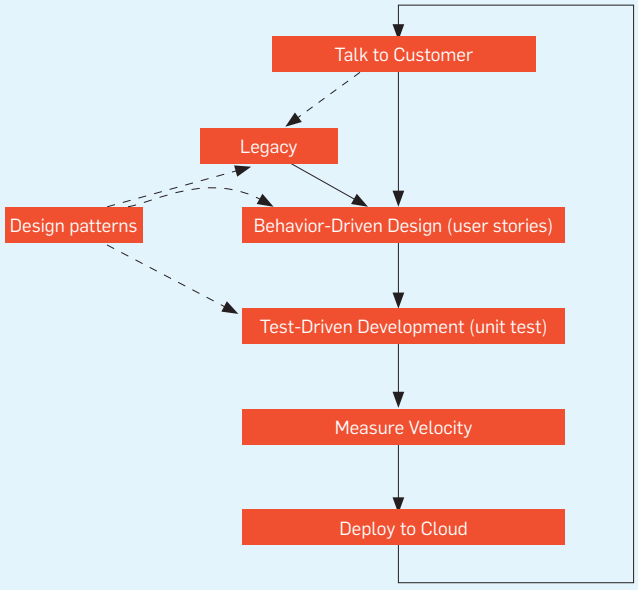
Figure 2. The key section of the Cucumber and Ruby code that automates the acceptance test for the user story in Figure 1 by matching regular expressions.

```
Given /I have entered (\d+) into the calculator/ do |n|
  @calc.push n.to_i
end

When /I press (\w+)/ do |op|
  @result = @calc.send op
end

Then /the result should be (.*) on the screen/ do |result|
  @result.should == result.to_f
end
```

Figure 3. One Agile iteration in our course.



For example, the Cucumber tool automates turning customer-understandable user stories into acceptance tests. Figure 1 is an example feature for a cash register application and one "happy path" user story (called a scenario in Cucumber) for that feature (see http://en.wikipedia.org/wiki/Cucumber_%28software%29). Note that this format is easy for the nontechnical customer to understand and help develop, which is a founding principle of Agile and addresses a key criticism from industry. Cucumber uses regular expressions to match user stories to the testing harness. Figure 2 is the key section of the Cucumber and Ruby code that automates the acceptance test by matching regular expressions.

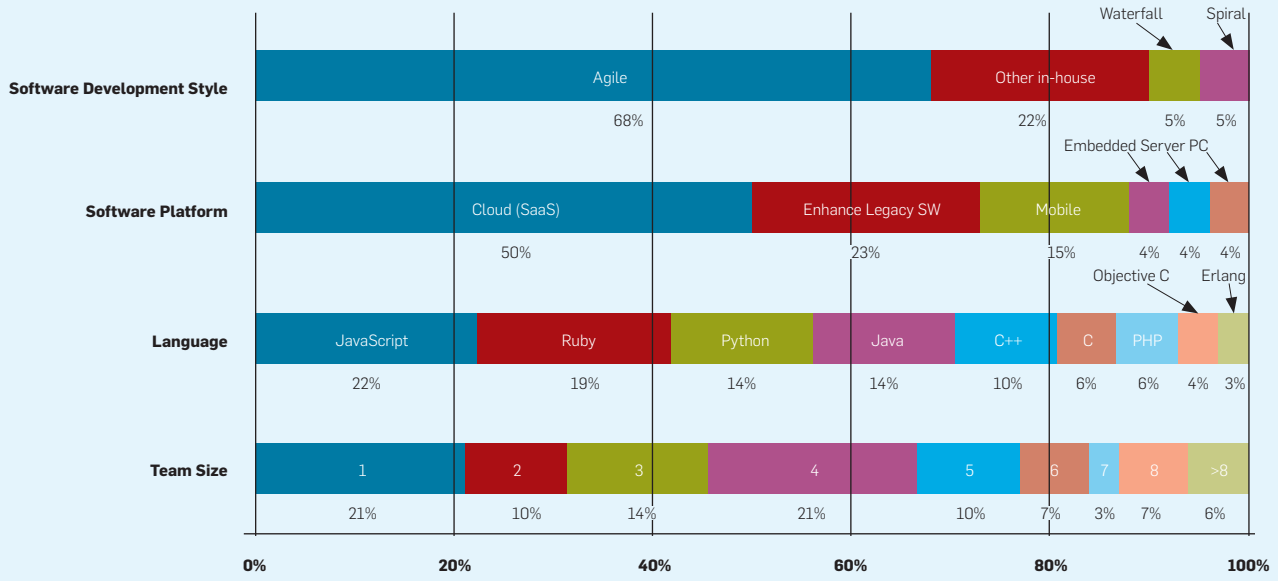
Such tools not only make it easy for students to do what they hear in lecture, but also simplify grading of student effort from a time-intensive subjective evaluation by reading code to a low-effort objective evaluation by measuring it. SimpleCov measures test coverage, *saikuro* measures cyclomatic complexity of the code, *flog* measures code assignment-branch-condition complexity, *reek* comments on code quality by highlighting "code smells,"[f] Cucumber shows the number of user stories completed, and Pivotal Tracker records weekly progress and can point out problems in balance of effort by members of teams. Indeed, these tools make it plausible for the online course to have automatically gradable assignments with some teeth in them.

Compared to Java and its frameworks, Rails programmers have found factors of three to five reductions in number of lines of code, which is one indication of productivity.[6,8] Rails also helps with a criticism of Agile in that TDD and rapid iteration can lead to poor software architecture. We do teach design patterns (see Figure 3). Indeed, the Rails framework follows the Model View Controller (MVC)[g] design pattern

f Code smells highlight to sections of code that may be hard to read, maintain, or evolve.

g MVC is a design pattern that separates input logic, business logic, and user interface logic yet provides a loose coupling between them.

Figure 4. Survey results of software experience for former Berkeley students now in industry. (The waterfall software development process is characterized by much of the design being done in advance of coding, completing each phase before going on to the next one. The spiral model combines features of a prototyping model with the waterfall model and is intended for large projects.)



to simplify development of the classic three-tiered applications of cloud computing. In addition, because of cloud computing, deploying their projects in the same horizontally scalable environment used by professional developers is instant, free for small projects, and requires neither software installation nor joining a developer program. In particular, it frees the course from instructional computers, which are often antiquated, overloaded, or both.

One criticism of the choice of Ruby is its inefficiency compared to languages like Java or C++. Since hardware has improved approximately 1,000X in cost-performance since Java was announced in 1995 and 1,000,000X since C++ was unveiled in 1979,[7] the efficiency of low-level code matters in fewer places today than it used to. We think using the improved cost-performance to increase programmer productivity makes sense in general, but especially so in the classroom. Note that for cloud computing, horizontal scalability can trump single-node performance; deploying as SaaS on the cloud in this course lets us teach (and test) what makes an app scalable across many servers, which is not covered elsewhere in our curriculum. Once again, without using the cloud to teach the class, we could not offer students the chance to experiment with scalability.

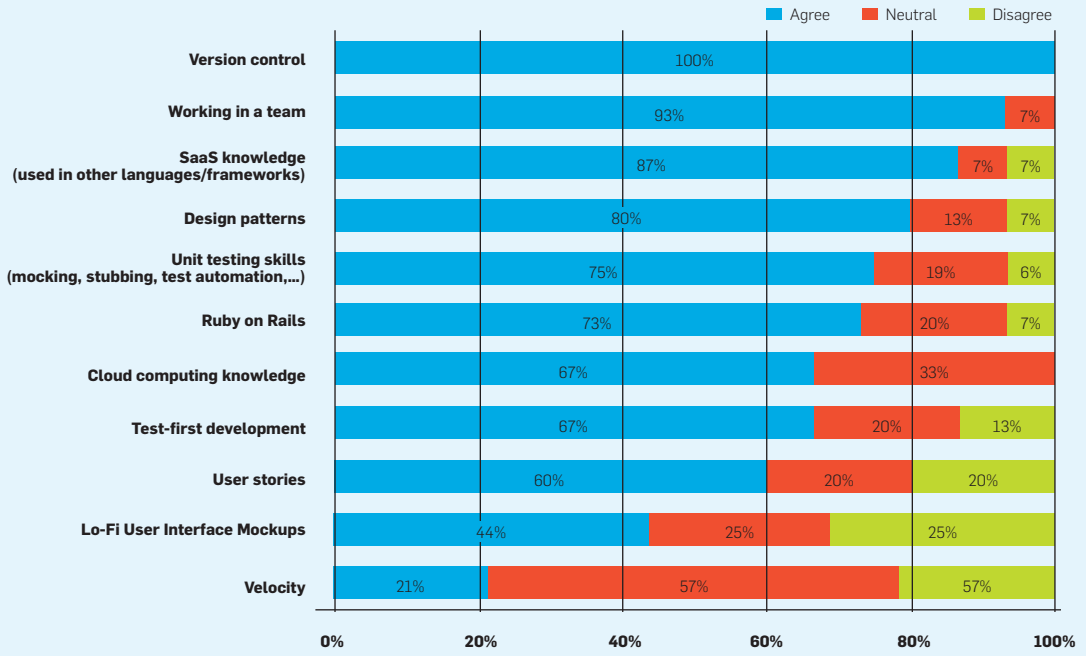
An Example Course

We use this approach to teach a software course, which currently has more than 100 juniors and seniors, at UC Berkeley. We also are offering the first part as a massive open online course (MOOC) to 50,000 online students, most of whom work in the IT industry and graduated from college five to 10 years ago.[3] (The MOOC tests the scalability of the tools and automatic grading of programming assignments.) Students follow steps shown in Figure 3, which turns the concepts and tools listed in the table into a logical process. Additional tools facilitate tying together the steps of this process; for example, Autotest monitors the source code tree in the background, automatically rerunning tests and user stories via RSpec/Cucumber in response to even minor code changes, giving immediate feedback if something breaks. Following the Agile philosophy, they deploy on Amazon Web Services (AWS) (via Heroku) multiple times during the course. The teaching staff evaluates iterations by interacting with the deployed app on AWS and by using Pivotal Tracker to check velocity and stories remaining to be implemented.

A typical faculty reaction to the request for students to deal with poorly documented legacy code is that it should not exist if students are taught good practices. However, despite decades of well-meaning instructors expounding on the importance of properly factored and highly documented code, our industry colleagues assure us the legacy code problem will continue to persist. Thus, on this point we quote Shimon Peres: *"If a problem has no solution, it may not be a problem, but a fact—not to be solved, but to be coped with over time."* Hence, if we can find principles that teach how to cope with legacy code, they *would* be appropriate for the classroom since it is a long-lasting challenge.

To learn to deal with legacy apps, our students learn and enhance a large, popular, well-written (but poorly documented) open source app written in Rails. We use Typo, a blogging framework containing 30,000 lines of Ruby and 15,000 lines of JavaScript, which suggests that it has the functionality of a Java program of 100,000 to 150,000 lines of code. We choose a Rails app rather than a Java app to maintain the programmer productivity we need to fit within our time budget. Feathers[2] argues that the first step is to write tests for legacy code to ensure understanding before modification, so *enhancing*

Figure 5. Ranked results from survey of former Berkeley students on whether course topics listed in the table and Figure 3 were useful in their industrial projects. These earlier versions of the course did not offer enhancing legacy code, design reviews, and working with nontechnical customers.



legacy code fits surprisingly well with the test-driven development of Agile.

To learn to communicate with nontechnical customers, for the Berkeley course we asked for projects from nonprofit organizations. The projects are typically less than 3,000 lines of code, with typically two to three times more code for testing than for the app. Teams of four or five students meet with customers on each Agile iteration for feedback on the current prototype and to prioritize the next features to add. (Moreover, we encourage students to connect these applications to Facebook or Twitter, which gives students the chance to deal with users as well as nontechnical customers within a single course.) Teams members do design reviews for each other as part of a biweekly class laboratory section, as Agile favors frequent code check-ins and design reviews. (Online students do not do projects.)

Using the Course Content Afterward

Figure 4 shows the survey results of Berkeley students from two earlier course offerings. Just 22 of the 47 respondents had graduated, and just 19 had done significant software projects. The percentages indicate the results of their 26 software projects. We were surprised that Agile software development was so popular (68%) and that the cloud was such a popular platform (50%). Given that no language was used in more than 22% of the projects, our alumni must be using Agile in projects that use languages other than Ruby or Python. All the class teams had four or five students, which directly matches the average team size from the survey.

Once again, Agile development and Rails were not selected because we expected them to dominate students' professional careers upon graduation; we use them to take advantage of their productivity so we can fit several critical ideas into a single college course in the hope they will use them later no matter what methodology, programming language, and framework.

Figure 5 shows the students' ranking of the topics the table and Figure 3 in terms of usefulness in their industrial projects. Most students agreed that the top nine topics in the course were useful in their jobs. Once again, we were pleased to see that these ideas were still being used, even in industrial projects that did not rely on Agile or on Rails. The two lower ranked topics were Lo-Fi User Interface Mockups, which makes sense since few developers work on the UI of a user-facing project, and Velocity, as progress can be measured in other ways in industry.

Although a small sample and not a conclusive user study, our survey offers at least anecdotal evidence that students of this course *do* continue to use successful software development techniques in later software projects of all kinds.

Conclusion

Using Agile to develop SaaS apps via highly productive tools like Rails and deploying them using cloud computing cultivates good software practices and pleases many stakeholders:

- Students like it because they get the pride of accomplishment in shipping code that works and is used by people other than their instructors, plus they get experience that can help land internships or jobs.
- Faculty like it because students actually use what they hear in lecture, even after graduation, and they experience how big CS ideas genuinely improve productivity. Virtual machines reduces hassles for faculty plus the cloud allows for more interesting program-

ming assignments—which the testing and code evaluation tools of Rails can help grade—thereby allowing us to offer a MOOC with 50,000 students.

▸ Colleagues in industry like it because it addresses several of their concerns. An example is this quote from a Googler: *"I think what you're doing in your class is amazing. I'd be far more likely to prefer graduates of this program than any other I've seen. As you know, we're not a Ruby shop, but I feel this is a good choice for the class to be able to get real features done. Java or C++ would take forever."*[4]

We received similar comments from colleagues at Amazon, eBay, and Microsoft, none of which are "Ruby shops." As we expected, leading software companies prefer students learn important ideas rather than steer us to teach specific languages and tools used inside those companies.

We believe Agile+Cloud+Rails can turn a perceived weakness of the CS curriculum into a potential strength. If you are a potentially interested instructor, we would be happy to help you cross the long-standing chasm between what industry recommends and what academia offers. C

References

1. Begel, A. and Simon, B. Novice software developers, all over again. In *ICER '08: Proceedings of the 4th International Workshop on Computing Education Research* (Sept. 2008).
2. Feathers, M. *Working Effectively with Legacy Code*, Prentice Hall, 2004.
3. Fox, A. and Patterson, D. Software engineering for Software as a Service; http://www.saas-class.org, March 2012 and May 2012.
4. Green, B. Private Communication, 2011.
5. IEEE. Guide to the Software Engineering Body of Knowledge (SWEBOK), 2004.
6. Ji, F. and Sedano, T. Comparing extreme programming and waterfall project results. *Conference on Software Engineering Education and Training 2011* (2011).
7. Patterson, D.A. and Hennessy, J.L. *Computer Organization and Design: The Hardware/Software Interface*. Revised 4th Edition, Morgan Kaufmann Publishers, 2012.
8. Stella, L.F.F., Jarzabek, S. and Wadhwa, B. A comparative study of maintainability of Web applications on J2EE, .NET, and Ruby on Rails. *WSE 2008. 10th International Symposium on Web Site Evolution* (Oct. 3–4, 2008), 93–99.

Armando Fox (fox@cs.berkeley.edu) is an adjunct associate professor at UC Berkeley and a co-founder of the Berkeley RAD Lab.

David Patterson (pattrsn@cs.berkeley.edu) is the E.H. and M.E. Pardee Chair of Computer Science at UC Berkeley and is a past president of ACM.

We thank our colleagues at Amazon Web Services, eBay, Facebook, GitHub, Google, Heroku, Microsoft, and Pivotal Labs for their feedback and suggestions on this Viewpoint, for their support and feedback during the development of the course, and for their donations of services to support the online course.

DOI:10.1145/2160718.2160733

Article development led by acmqueue
queue.acm.org

Shortcuts that save money and time today can cost you down the road.

BY ERIC ALLMAN

Managing Technical Debt

IN 1992, WARD CUNNINGHAM published a report at OOPSLA[2] in which he proposed the concept of technical debt. He defines it in terms of immature code: "Shipping first-time code is like going into debt." Technical debt is not limited to first-time code, however. There are many ways and reasons (not all bad) to take on technical debt.

Technical debt often results from the tension between engineering "best practices" and other factors (ship date, cost of tools, and the skills of engineers that are available, among others). Roughly speaking, technical debt is acquired when engineers take shortcuts that fall short of best practices. This includes sneaking around an abstraction because it is too hard (or impossible) to figure how to "do it right," skipping or scrimping on documentation (both in the code and external documentation), using an obscure or incomplete error message because it is just too hard to create something more informative, implementing code using a simple but slow algorithm even though they know that a better algorithm will be needed in production, using `void*` when you really should have created an appropriate `union*`, using build tools that do not quite work for the system at hand, skimping on good security practices, not writing unit tests, and so forth. Admit it—you have all done one or more (or maybe all) of these things at some point or another in your career. (Technical debt may also be taken on intentionally as a strategy to save time or money; more about that later.)

Not all debt (whether technical or financial) is bad. Few of us can afford to pay cash for a house, and going into debt to buy one is not financially irresponsible, provided that we know how to pay it back. In contrast, charging up luxury items on a credit card, knowing very well that your paycheck will not cover them, is usually a recipe for disaster. Using a simple but slow algorithm in a prototype can be exactly the correct path, as long as you have a plan for how you are going to update the code before it ships. That means allowing time in the schedule, making sure the issue is tracked so it does not get lost in the shuffle, knowing when you implement the code that a good algorithm actually does exist that will work in this instance, and trusting that management will support you.

Understanding, communicating, and managing technical debt can make a huge difference in both the short- and long-term success of a system. (Note that although this article focuses on technical debt in software engineering, many of these principles can be applied to other technical disciplines.)

Comparison with Financial Debt

Going into financial debt usually has three important properties. First, the person making the loan wants it to be repaid eventually. Second, you usually have to pay it back with interest—that is, you pay back more money than you got in the first place. Third, if it turns out you cannot pay it back, there is a

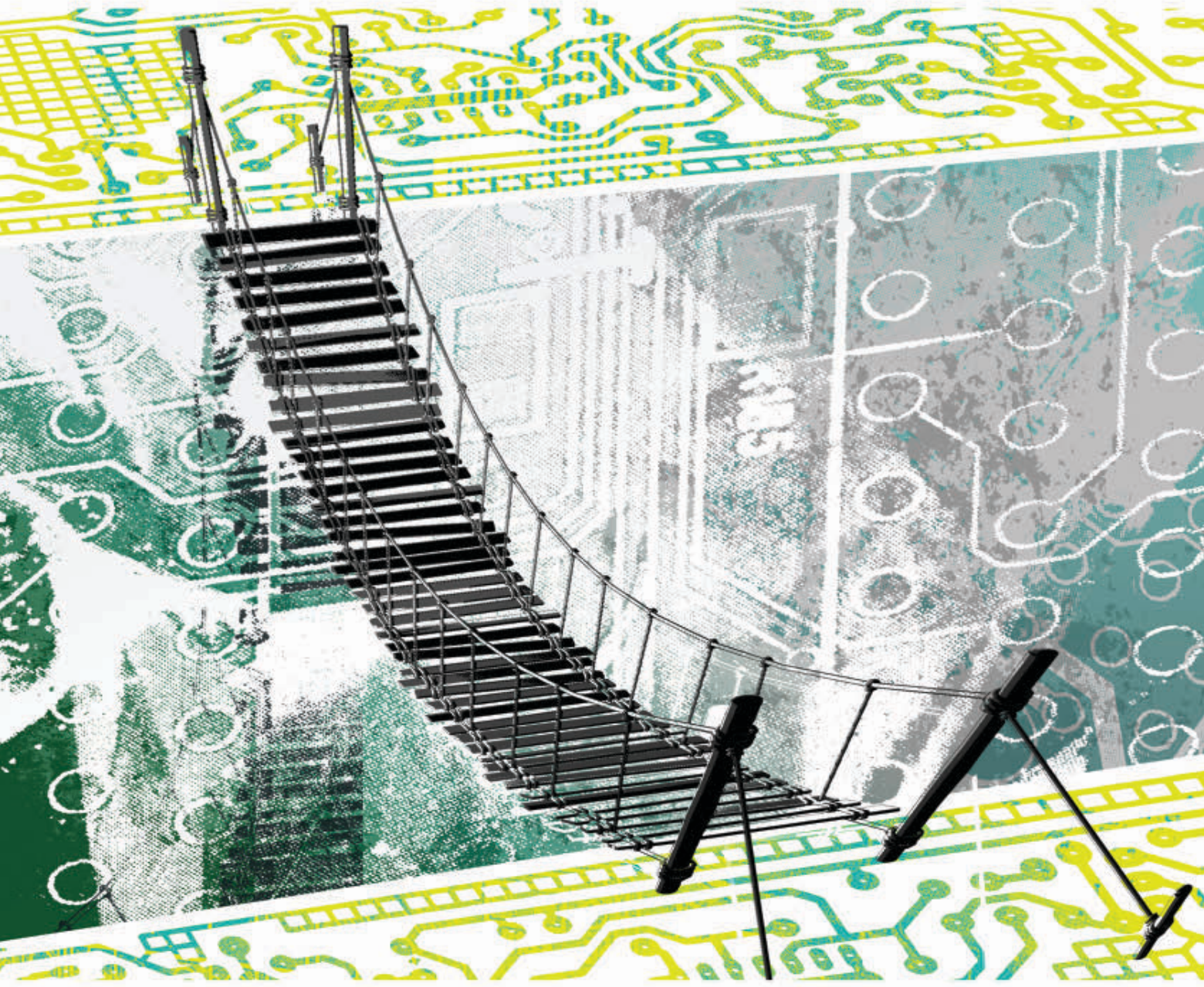


ILLUSTRATION BY GARY NEILL

very high cost, be it declaring bankruptcy, losing your house, or (if you borrowed from the wrong person) a long walk off a short pier wearing cement shoes.

Technical debt is similar in some ways, but different in others. Although you don't have to pay back the debt on any fixed schedule (and some debts may never need to be paid back), you generally do have to pay back (that is, rewrite the code or otherwise fix the problem) the parts that affect you or your customers in a significant way. The "interest" is accrued every time you or anyone else (support-desk workers, future programmers, customers, and so on) working with your system is delayed because of bugs, performance problems, inexplicable misfeatures, time spent researching what has gone wrong when the system could have given a more explicit error message, and so on. Failure to fix problems can result in the utter collapse of a system—the customer gives up and goes elsewhere, the system becomes so slow and brittle that it has to be rewritten from scratch, or in extreme cases the company is forced to close its doors.

There are some significant differences as well. Perhaps the most pernicious one is that the person who takes on technical debt is not necessarily the one who has to pay it off—in fact, most of the time the one who takes on the debt can shuffle the costs on to other people, which encourages taking on debt. Far too many developers do not maintain their own code. Many companies have a policy that software moves from a development mode that is staffed by their best programmers to a maintenance mode staffed by second-tier engineers (who are paid less but often have far more difficult jobs than the premier team). Sometimes it isn't even anyone in your organization who is paying the interest: it's the users who have to pay. Developers are rewarded more on implementation speed than long-term maintainability and may have moved on to a different project or company before the real cost is paid. This gives the initial developer little incentive to do the job right the first time.

Unlike financial debt, technical debt almost never has to be paid off in its entirety. Most (probably all) production systems have warts that do

not have significant impact on the usability or long-term maintainability of the final system. Very few systems have no `TODO` or `FIXME` or `XXX` comments somewhere in the source code. Note that the cost of paying back technical debt comes in the form of the engineering time it takes to rewrite or refactor the code or otherwise fix the problem. If the interest you ultimately accrue is less than the cost of paying back the debt, there is no point in paying it back in the first place. The problem is that it can be difficult to know in advance which debts will ultimately have the highest cost.

For example, when U.C. Berkeley's CalMail system went down in November 2011, the problem was traced to deferred maintenance—in particular, the decision to postpone updating the system even though it was known to be near capacity.[5] One disk in a RAID died, shortly followed by a second, and the cost of rebuilding the array reduced capacity sufficiently to create a crisis. Murphy's law needs to be taken into consideration when deciding how much technical debt to accept. In the CalMail case, individual hardware failures were expected in the base design, but multiple failures, happening during a historically high usage spike, created a condition that was not quickly resolvable. According to Berkeley's associate vice chancellor for information technology and chief information officer, Shelton Waggener, "I made the decision not to spend the million dollars to upgrade CalMail software for only 12 months of use given our plan to migrate to new technology. We were trying to be prudent given the budget situation, (but) in retrospect it would have been good to have invested in the storage upgrade so we might have avoided this crisis." This is a case where technical debt was taken on intentionally but turned out to be a bad gamble. Had the system survived that 12-month window, the school likely would have saved $1 million during a budget crunch.

There is a saying to the effect that there are three variables in engineering: time, functionality, and resources—pick two. In fact, there is a fourth variable: debt. Of these four variables, you can set any three of them, but you can never set all four; something just has to give, and very commonly debt is the free variable in the equation. Debt can seem "free" at first, but technical debt tends to build on itself. If the acquisition of debt involves interest in the form of increased effort to maintain and extend the system, then as you take on debt it gets harder and takes longer to do maintenance and extension. This is one form of collapse under debt: if all of your "income" (in the form of effort) is spent paying off interest and nothing is left over to move the system forward, then that system is stuck. This is especially obvious if productivity is measured in lines of code produced per day, a measure that should be relegated to the fires of hell. You do not have many choices left: add effort (hire more engineers), abandon the system and move on, or go bankrupt. In this sense, the interest on technical debt is actually compound interest, or put another way: if you don't stay on top of the debt, then the payments go up over time.

Consider these other interesting comparisons. Steve McConnell, CEO and chief software engineer at Construx Software, distinguishes between unintentional and intentional debt, which in turn is broken up as short-term (tactical) versus long-term (strategic) debt.[6] He also notes that when a system is nearing end of life, incurring debt becomes more attractive, since all debt is retired when a system is decommissioned. He also makes some interesting observations on how to communicate the concept of technical debt to nontechnical people, in part by maintaining the "debt backlog" in a tracking system and exposing it in terms of dollars rather than something more tech oriented.

In a slightly different analysis, software designer Martin Fowler breaks down technical debt on two axes: reckless/prudent and deliberate/inadvertent.[3] He describes reckless-deliberate debt as "we don't have time for design," reckless-inadvertent as "what's layering?" and prudent-deliberate as "we must ship now and deal with consequences." This exposes a fourth class of technical debt that doesn't map easily to the financial model: prudent-inadvertent, which he describes as "now we know how we should have done it."

Another analysis of technical debt that resonates with some people is that managing technical debt is a way of managing risk in the technical realm. Software consultant Steve Freeman discusses this by comparing technical debt with an unhedged (or "naked") call option.[4] Either case (risk or unhedged calls) allows for the possibility that debt may never need to be paid back; indeed, big money can be made by taking appropriate risks. Naked calls, however, can also lose all of their value—essentially, the risk is unlimited. This doesn't often happen (most of the time when you lose, you lose only some of your money, not all of it), but it can happen, much as the wrong choice of technical debt can result in disaster.

So far we have spoken of technical debt as though it were unique to coders. This is far from true. For example, the operations department incurs its own kind of debt. Avoiding a disk-array upgrade is a trade-off between technical debt and financial costs. Failure to consider power and cooling requirements when adding new, hotter equipment to a machine room is a debt. Failure to automate a simple-but-tedious manual process is a debt. Systems administrators who have neither (for lack of desire, inspiration, or time) documented the systems they support nor trained co-workers before going on vacation are another example. The comparison of technical debt to risk management is often starker in these noncode-related situations: you are betting that you will not run out of disk space or bandwidth, that you will not have a super-hot day, that your system will not become so successful that the manual process becomes a bottleneck, or that nothing will go wrong while you are in Machu Picchu.

Certain staffing issues can lead to another form of technical debt: having parts of systems that are understood by only one person. Sometimes this happens because the staff is spread too thin, but it can also be caused by insecure individuals who think that if they keep everyone else in the dark, then they will be indispensable. The problem, of course, is that everyone moves on eventually.

Managing Your Debt

Technical debt is inevitable. The issue is not eliminating debt, but rather managing it. When a project starts, the

team almost never has a full grasp on the totality of the problem. This is at the root of the failure of the waterfall model of software development, which posits that all requirements can be finalized before design begins, which in turn can be completed before the system is implemented, and so forth. The argument seems good: the cost to make a change goes up exponentially as the system is developed, so the best path is to get the early stages done right before moving on. The reality is that requirements always change ("requirements churn"). It is often better to have a working prototype (even though it is not complete or perfect) so that you and the customers can start gaining experience with the system. This is the philosophy behind Agile programming, which accepts some technical debt as inevitable but also mandates a remediation process ("plan for change").

As necessary as technical debt may be, however, it is important that the strategic parts of it be repaid promptly. As time goes on, programmers move to other companies, and the people who agreed to various compromises have moved on to other projects, replaced by others who do not see it the same way. Failure to write the documentation (both internal and external) for the initial prototype may be a good trade-off, but the longer it goes the more difficult it is to write—if only because human memory is transient, and if you show most people code they wrote a year ago they will have to study it to remember why they did it that way. Code that is intended to have a limited life span may be immune to these concerns, but many short-term "prototypes" end up getting shipped to customers. Unfortunately, Fred Brooks' statement in *The Mythical Man-Month*, "Plan to throw one away; you will, anyhow,"[1] seems all too often to be corrupted to, "Make your prototype shippable; it will, anyhow." These two statements are not contradictory.

Equally as important is that some forms of technical debt are so expensive that they should be avoided entirely whenever possible. Security is an area where taking shortcuts can lead to disaster. You never want to say, "We're using passwords in the clear today, but we will come back someday and change it to challenge-response," in anything other than very early prototypes that no one but you will ever see. This is a recipe for disaster if it ever gets accidentally deployed. You also want to avoid enshrining "bet your company" shortcuts in code. If for some reason you have no choice (for example, because during development other engineers have to write code that will interface with yours and you can't afford to keep them waiting), keep a journal of "debts that must be repaid before release." It's amazing how easy it can be to forget these things if they aren't written down.

Understanding, communicating, and managing technical debt can make a huge difference in both the short- and long-term success of a system.

Release cycles can make a considerable difference in the rate of acquisition and disposal of technical debt. The modern trend to "release early and often," especially in the context of Web-based services, has made it much easier to take on technical debt but has also made it easier to resolve that debt. When well-managed, this can be a blessing—taking on debt earlier allows you to release more functionality earlier, allowing immediate feedback from customers, resulting in a product that is more responsive to user needs. If that debt is not paid off promptly, however, it also compounds more quickly, and the system can bog down at a truly frightening rate. Another side of Web-based services in particular is that a correct but inefficient solution can actually cost your company more money—for example, in the form of server-farm rental fees. Fortunately, this makes the debt easy to translate into dollar terms, which nontechnical stakeholders usually find easier to understand than assertions about maintainability.

Not all technical debt is the result of programmer laziness. Some is imposed by management or other departments, especially when they do not understand how pernicious this debt can be. Customers usually buy features, not long-term maintainability, so marketing departments often encourage engineering to move on to the next great thing rather than spending the time necessary to consolidate, clean up, and document the existing system. To them, taking these steps is an unnecessary cost—after all, the system works today, so why does engineering need to spend time gilding the lily?

There is another aspect to technical debt to consider: it occurs in many

ways and is ongoing. It can come from the design or implementation phases, of course, but can also occur in the operational phase. For example, a computer system may have had a UPS (uninterruptible power supply) designed and installed, but deferred maintenance—in the form of failing to test those units and replace batteries—can render them useless. Disk arrays may be adequate when specified, but as the system grows they must be upgraded. This can be especially hard when attempting to extract dollars from a cash-strapped management to upgrade something that, from their perspective, works fine.

Management all too often aids and abets this problem. The current business mantra of "shareholder value" would be fine if shareholders were patient enough to reward long-term value creation. Instead the tendency is to think quarter to quarter rather than decade to decade, which puts immense pressure on everyone in the organization to produce as much as possible as quickly as possible, regardless of the longer-term costs (as indicated by the old lament, "there's never time to do it right, but there's always time to do it over"). Pushing costs into the future is considered a good strategy. This strongly encourages assumption of technical debt. An indicator of this is when engineering is perpetually in "crunch mode" rather than using crunches sparingly. How many companies advertise being "family friendly" on their Web sites and in their corporate values statement while encouraging their employees to work 60-hour weeks, penalizing "slackers" who work 40-hour weeks and then go home to their families? In these environments, the assumption of inappropriate technical debt is nearly impossible to avoid.

This is not to say that management is always wrong. There are appropriate times to accrue debt. If my child needed a critical medical treatment I wouldn't refuse just because it meant taking on debt, even if it would be expensive to pay back. Likewise, management has a responsibility to customers, employees, and (yes) investors that can sometimes impose uncomfortable requirements. Debt taken on with eyes open and in a responsible way is not a bad thing. U.C. Berkeley's CIO made a bet that turned out wrong, but it could have been a winning bet. He knew he was making it, and he took responsibility for the problem when the roof did cave in. The difficulty is when management doesn't understand the debt they are taking on or takes it on too easily and too often, without a plan for paying it back. In a past job I argued that we needed more time to build a system, only to be blamed by management when we had a high defect rate that was directly attributable to the artificially short schedule that was imposed against my better judgment. In this case, management didn't understand the debt, ignored warnings to the contrary, and then didn't take responsibility when the problems manifested.

Technical debt is inevitable. The issue is not eliminating debt, but rather managing it. When a project starts, the team almost never has a full grasp on the totality of the problem.

Cost of Debt from Various Perspectives

Technical debt affects everyone, but in different ways. This is part of the problem of managing the debt—even if you understand it from your perspective, there are other legitimate ways to view it.

Customers. It may seem that the customers are the ultimate villains (and victims) in this affair. After all, if they were more patient, if they demanded less from the products and gave the company more time to do the job right the first time, none of this would happen (or maybe not). True, customers can sometimes focus more on features (and sadly, sometimes on marketing fluff) than long-term maintainability, security, and reliability, yet they are the ones who are most badly injured. When the mobile network goes out, when they cannot get their work submitted on time, when their company loses business because they are fighting the software, they pay. Ultimately, it's all about doing what the customers need, and customers need software that works, that they can understand, that can be maintained and extended, that can be supported, and (ultimately) that they like using. This cannot happen without managing the technical debt at every level through the process, but customers seldom have any control over how that debt is managed. It is worth noting that customers who are paying for bespoke solutions general-

ly have more control than customers who buy software "off the rack," who for the most part have to use what they are given. At the same time, when you are building software for particular customers, you may be able to negotiate "debt repayment" releases (probably not using that term).

Help Desk. Those who work on the help desk deserve a special place in heaven—or occasionally in hell. Customers seldom call to say how happy they are; they generally have a rather different agenda. Help-desk personnel suffer from almost every aspect of technical debt: poorly designed interfaces, bad or nonexistent documentation, slow algorithms, etc. In addition, things that may not seem to affect them directly (such as obscurity in the code itself) will have an indirect effect: customers get more ornery the longer it takes to fix their problem. Though the help desk is the customers' primary input to the internal process, the desk often has no direct access to the people who can solve the problem.

Operations. In a service-oriented environment the operations people (those who carry the beepers 24/7 and who have to keep everything working) are far too often the cannon fodder on the frontlines; they can spend much of their time paying for decisions that other people made without consulting them. Sometimes they get to look at the code, sometimes not. In any case they get to look at the documentation—if it exists. The (minimal) good news is that they may be able to pass the problem off to a maintainer as long as they can come up with an acceptable work-around. The rise of the DevOps movement—the concept that operations folks need to work with developers early in the cycle to make sure that the product is reliable, maintainable, and understood—is a positive development. This is a great way of reducing long-term technical debt and should be strongly encouraged.

Engineers. Engineers fall into two roles: the developers who write the code and the people who have to repair, extend, or otherwise maintain that code (these may be the same engineers, but in many places they are not). At first glance, the initial developers seem to be the major creators of technical debt, and they do have strong incentive to take on debt, but as we have seen, it can come from a number of sources. In its early days technical debt is almost invisible, because the interest payments haven't started coming due yet. Doing a quick, highly functional initial implementation makes the programmer look good at the cost of hampering engineers who join the party later. In some cases, those programmers may not even realize they are taking on the debt if they have limited experience maintaining mature code. For this reason, an average-speed, steady, experienced programmer who produces maintainable code may be a better long-term producer and ultimately higher-quality engineer than a "super-stud programmer" who can leap tall prototypes in a single bound but has never had to maintain mature code.

Marketing. These customer-facing people often have to take the brunt of customer displeasure. They can often be the people pushing hardest for short product development times because they are pressured by sales and the customers to provide new functionality as quickly as possible. When that new functionality does not work properly in the field, however, they are also the ones on the wrong side of the firing line. In addition, pressure for quick delivery of new features often means that later features will take even longer to produce. Great marketing people understand this, but all too often this is not a concept that fits the marketing world model.

Management. There is good management and bad management. Good management understands risk management and balances out the demands of all departments in a company. Bad management often favors a single department to the detriment of others. If the favored department is marketing or sales, management will be inclined to take on technical debt without understanding the costs. Management also pays a price, however. It is not true that "there is no such thing as bad publicity," especially when your company appears to be circling the drain. Management should have no difficulty embracing the concept of managing financial debt. It is also to their advantage to manage technical debt.

Summary

Technical debt can be described as all the shortcuts that save money or speed up progress today at the risk of potentially costing money or slowing down progress in the (usually unclear) future. It is inevitable, and can even be a good thing as long as it is managed properly, but this can be tricky: it comes from a multitude of causes, often has difficult-to-predict effects, and usually involves a gamble about what will happen in the future. Much of managing technical debt is the same as risk management, and similar techniques can be applied. If technical debt is not managed, then it will tend to build up over time, possibly until a crisis results.

Technical debt can be viewed in many ways and can be caused by all levels of an organization. It can be managed properly only with assistance and understanding at all levels. Of particular importance is helping nontechnical parties understand the costs that can arise from mismanaging that debt.

Related articles on queue.acm.org

Coding Smart: People vs. Tools
Donn M. Seeley
http://queue.acm.org/detail.cfm?id=945135

IM, Not IP (Information Pollution)
Jakob Nielsen
http://queue.acm.org/detail.cfm?id=966731

Outsourcing: Devising a Game Plan
Adam Kolawa
http://queue.acm.org/detail.cfm?id=1036501

References

1. Brooks, F. *The Mythical Man-Month*, Anniversary Edition. Chapter 11. Addison-Wesley, Reading, PA, 1995.
2. Cunningham, W. The WyCash portfolio management system. OOPSLA 1992, Experience Report; http://c2.com/doc/oopsla92.html.
3. Fowler, M. Technical debt quadrant, 2009; http://martinfowler.com/bliki/TechnicalDebtQuadrant.html.
4. Freeman, S. Bad code isn't technical debt, it's an unhedged call option. Higher-order Logic; http://www.higherorderlogic.com/2010/07/bad-code-isnt-technical-debt-its-an-unhedged-call-option/.
5. Grossman, S. Calmail crashes last multiple days. *The Daily Californian* (Dec. 1, 2011); http://www.dailycal.org/2011/12/01/calmail-crashes-last-multiple-days/.
6. McConnell, S. Technical Debt. Construx Conversations: Software Best Practices; http://blogs.construx.com/blogs/stevemcc/archive/2007/11/01/technical-debt-2.aspx.

Eric Allman has been a programmer, a maintainer, a help-desk survivor, both frontline and executive management, a consultant, a technology writer, and occasionally even a customer. He appreciates the difficulty and sometimes idiocy of all of those roles.

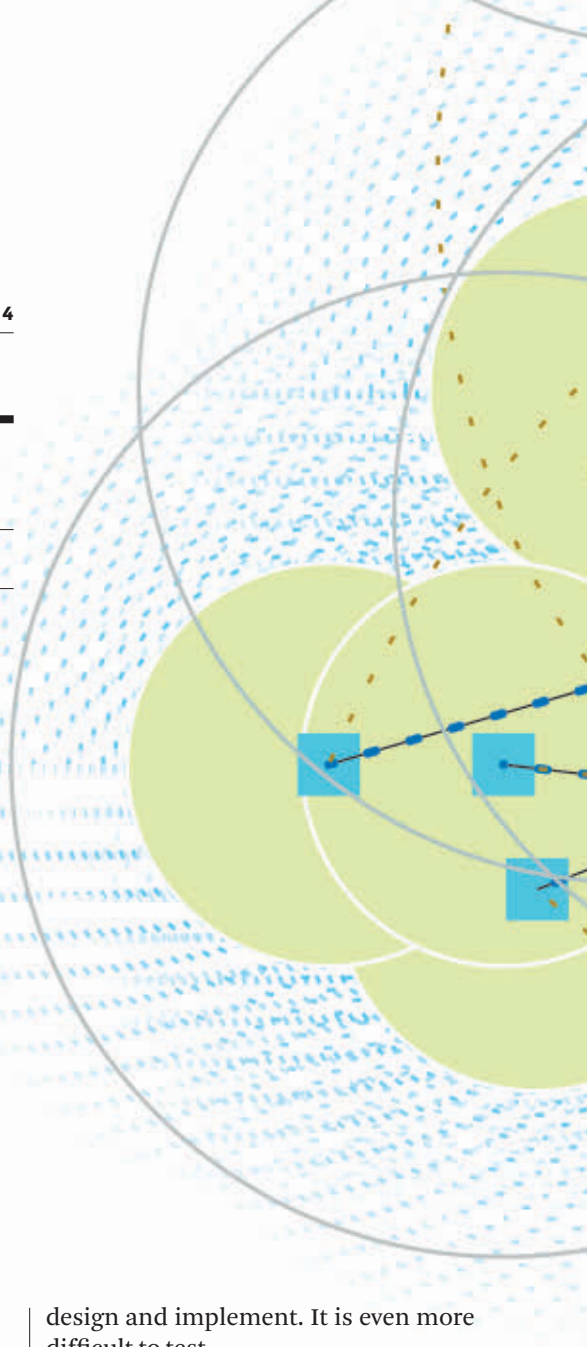
DOI:10.1145/2160718.2160734

Article development led by acmqueue
queue.acm.org

Messages may be retried. Idempotence means that's OK.

BY PAT HELLAND

Idempotence Is Not a Medical Condition



THE DEFINITION OF *distributed computing* can be confusing. Sometimes, it refers to a tightly coupled cluster of computers working together to look like one larger computer. More often, however, it refers to a bunch of loosely related applications chattering together without a lot of system-level support.

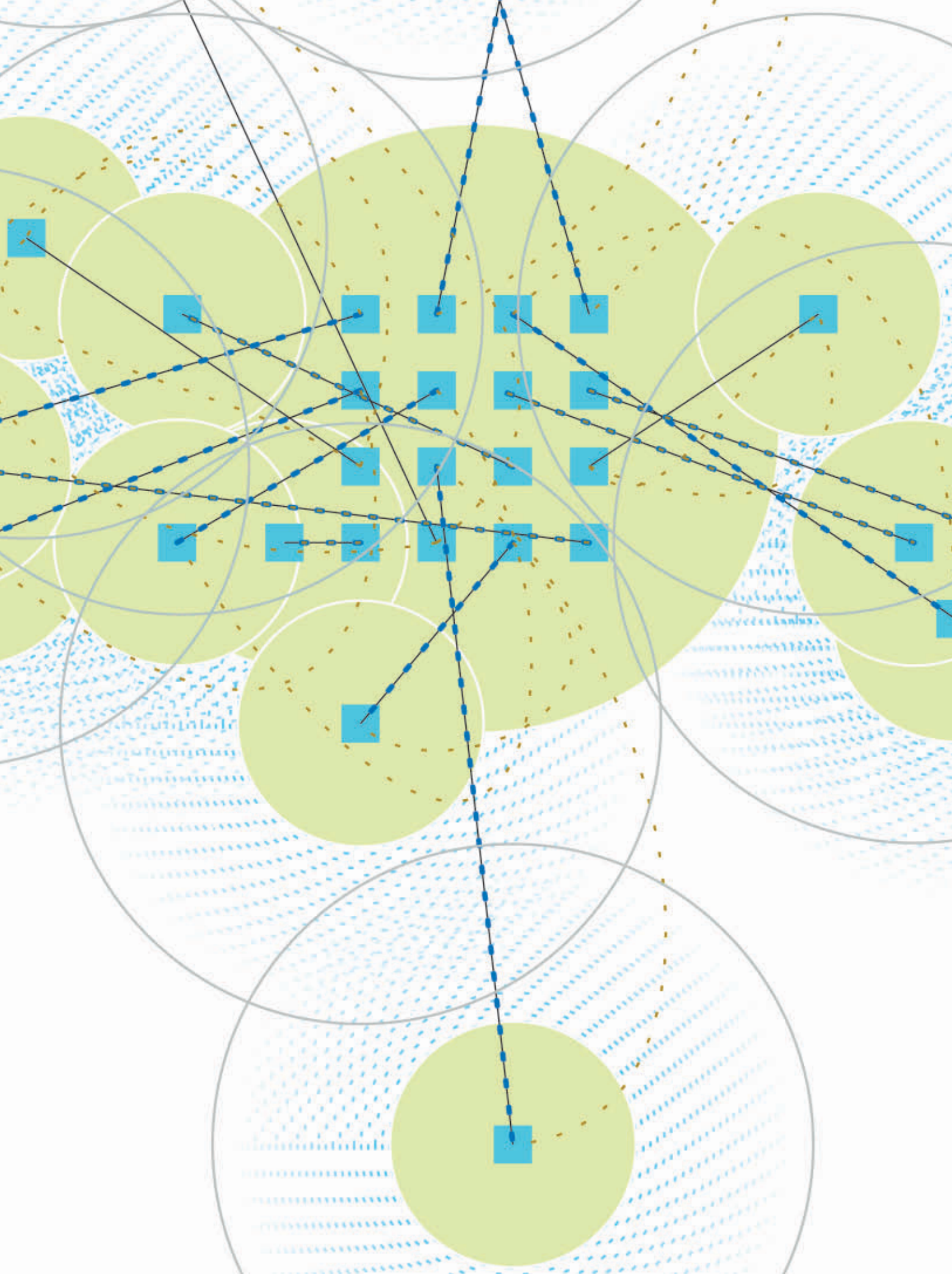
This lack of support in distributed computing environments makes it difficult to write applications that work together. Messages sent between systems do not have crisp guarantees for delivery. They can get lost, and so, after a timeout, they are retried. The application on the other side of the communication may see multiple messages arrive where one was intended. These messages may be reordered and interleaved with different messages. Ensuring the application behaves as intended can be difficult to design and implement. It is even more difficult to test.

In a world full of retried messages, *idempotence* is an essential property for reliable systems. *Idempotence* is a mathematical term meaning that performing an operation multiple times will have the same effect as performing it exactly one time. The challenges occur when messages are related to each other and may have ordering constraints. How are messages associated? What can go wrong? How can an application developer build a correctly functioning app without losing his or her mojo?

Messaging in a Service-Oriented World

This article considers messaging in a service-oriented environment consisting of a collection of servers that share the work. The hope is that you can

ILLUSTRATION BY STUDIO TONNE



scale by adding more servers when the demand increases. This opens up the question of how later messages can locate a service (running on a server) that remembers what happened before.

Here, the problem is framed by describing the kind of application under consideration by looking at services and how long-running work can run across them, and by considering the need for messaging across applications that evolve independently in a world where they share relatively simple standards.

Messages and Long-Running Work

Sometimes related messages arrive much later, perhaps days or weeks later. These messages are part of the same piece of work performed by an application attempting to work across multiple machines, departments, or enterprises. Somehow the communicating applications need to have the information necessary for correlating the related messages.

Predictable and consistent behavior is essential for message processing, even when some of the participants have crashed and restarted. Either the earlier messages did not matter or the participant needs to remember them. This implies some form of durability capturing the essence of what was important about the earlier messages so the long-running work can continue. Some systems do need the information from the earlier message to process the later ones but do not make provisions for remembering the stuff across system crashes.

Of course, there is a technical term for unusual behavior when a system crash intervenes; it is called a *bug*.

The shape and form of applications continue to evolve as years go by, transitioning from mainframes to minicomputers to PCs to departmental networks. Now scalable cloud-computing networks offer new ways of implementing applications in support of an ever-increasing number of messaging partners.

As the mechanisms for implementing applications change, the subtleties of messaging change. Most applications work to include messaging with multiple partners using semi-stable Internet standards. These standards support the evolving environment but cannot eliminate some of the possible anomalies.

Figure 1. Applications talk to their local plumbing.

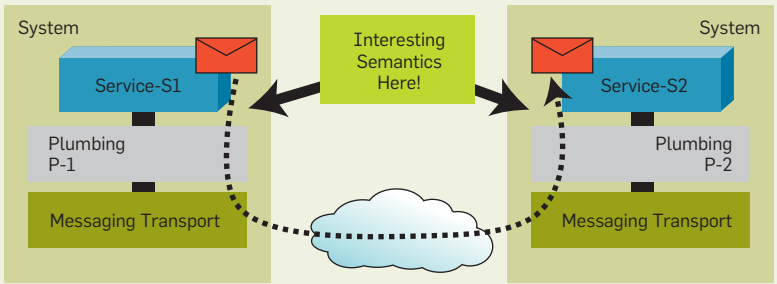
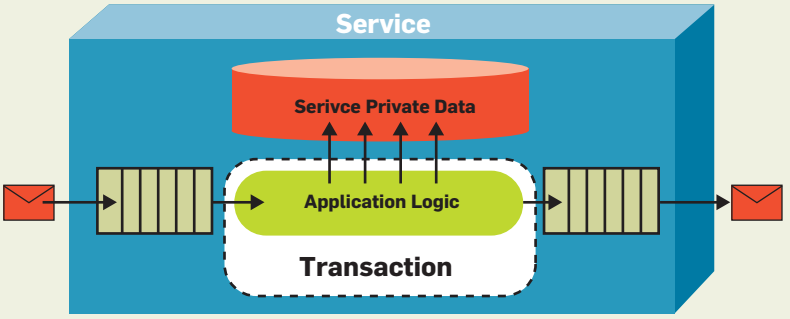


Figure 2. Sometimes, the plumbing will transactionally tie the consumption of the incoming message to changes in the application's database and to outgoing messages.



Imagining a Dialog Between Two Services

Here, we consider the challenges of communicating between two parties. (Other messaging patterns, such as pub-sub, are not addressed here.) It imagines an arbitrary communication sequence of messages between these two parties that are related to each other and assumes that somehow the two programs have a notion of the work accomplished by this dialog. Even with the best "messaging plumbing" support, a two-party messaging dialog between two services has complications to be addressed.

Applications often run on top of some form of "plumbing." Among other services, the application's plumbing offers assistance in the delivery of messages. It likely offers help in naming, delivery, and retries and may provide an abstraction that relates messages together (for example, request-response).

As an application designer, you can only understand what *you* see as you interact with the plumbing on your client, server, or mobile device. You can anticipate or infer what happens farther down the line, but all of it is intermediated by your local plumbing (see Figure 1).

When a local application (or service) decides to engage in a dialog with another service, it must use a name for the intended partner. The identity of a partner service for the first message specifies the desire to chat with a service doing the application's work but not yet engaged in some joint project. That is really different from connecting to the same service instance that has processed previous messages.

A relationship is established when messages are sent in a dialog. Processing subsequent messages implies that the partner remembers earlier messages. This relationship implies an identity for the partner in the midst of the dialog that is different from a fresh partner at the beginning of a dialog.

The representation of the mid-dialog identity, and how the later messages are routed to the right place, are all part of the plumbing. Sometimes

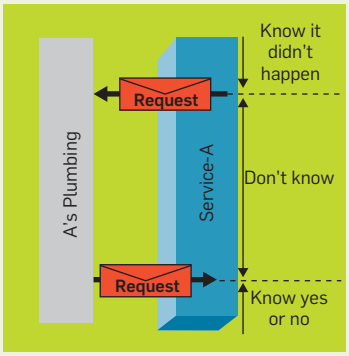
the plumbing is so weak that the application developer must provide these mechanisms, or the plumbing may not be universally available on all the communicating applications, so the applications must delve into solving these issues. For now, let's assume that the plumbing is reasonably smart, and look at the best-case behavior an application can expect.

Messages, Data, and Transactions

What happens when the application has some data it is manipulating (perhaps in a database with transactional updates)? Consider the following options:

- Message consumption is recorded *before* processing. This is rare because building a predictable application is very difficult. Sometimes the message will be recorded as consumed, the application sets out to do the work, and the transaction fails. When this happens, the message is effectively not delivered at all.
- The message is consumed as part of the database transaction. This is the easiest option for the application, but it is not commonly available. All too often, the messaging and database systems are separate. Some systems, such as SQL Service Broker[4] provide this option (see Figure 2). Sometimes, the plumbing will transactionally tie the consumption of the incoming message to changes in the application's database and to outgoing messages. This makes it easier for the application but requires tight coordination between messaging and database.
- The message is consumed *after* processing. This is the most common case. The application must be designed so that each and every message is idempotent in its processing. There is a failure window in which the work is successfully applied to the database, but a failure prevents the messaging system from knowing the message was consumed. The messaging system will redrive the delivery of the message.

Figure 3. The interesting semantic occurs as an application talks to the local plumbing on its box.



Nothing beats a plumber who can alleviate your worries and make everything "just work." It is especially nice if that plumber shares the same understanding of you and your needs.

Outgoing messages are typically queued as part of committing the processing work. They, too, may be either tightly coupled to the database changes or allowed to be separate. When the database and messaging changes are tied together with a common transaction, the consumption of the message, the changes to the database, and the enqueuing of outgoing messages are all tied together in one transaction (see Figure 2). Only *after* enqueuing an outgoing message will the message depart the sending system. Allowing it to depart before the transaction commits may open up the possibility of the message being sent but the transaction aborting.

In-Order Delivery: History or Currency?

In a communication between two partners, there is a clear notion of sending order in each direction. You may be sending to me while I am sending to you, meaning there is fuzziness in the ordering for messages going past each other, but only one message at a time is sent in a specific direction.

A listener can easily specify that it does not want out-of-order messages. If you sent me *n* different messages in a specific order, do I really want to see message *n–1* when I've already seen message *n*? If the plumbing allows the application to see this reordering, then the application very likely has to add some extra protocol and processing code to cope with the craziness.

Suppose the messages *always* come in order. There are two reasonable application behaviors:

- *History (no gaps).* Not only will the messages be delivered in order, but also the plumbing will respect the sequence and not allow gaps. This means that the plumbing will not deliver message *n* in the dialog sequence to the app if message *n–1* is missing. It may work to get message *n–1* under the covers, but

that is transparent to the application. This is very useful for business process workflows and many other cases. Losing a message would make the application's design very, very difficult, so the app wants no gaps.

▸ *Currency (deliver the latest—gaps or no gaps).* Sometimes the delay to fill in the gap is more of a problem than the gap. When watching the price for a specific stock or the temperature gauge of a chemical process in a factory, you may be happy to skip a few intermediate results to get a more timely reading. Still, order is desired to avoid moving back in time for the stock price or temperature.

Knowing What You Don't Know When Sending Messages

When a communicating application wants to request some work to be done by its partner, there are a few stages to consider:

▸ *Before you send the request.* At this point, you are very confident the work hasn't been done.

▸ *After you send but before you receive an answer.* This is the point of confusion. You have absolutely no idea if the other guy has done anything. The work may be done soon, may already be done, or may never get done. Sending the request increases your confusion.

▸ *After you receive the answer.* Now, you know. Either it worked or it did not work, but you are less confused.

Messaging across loosely coupled partners is inherently an exercise in confusion and uncertainty. It is important for the application programmer to understand the ambiguities involved in messaging (see Figure 3). The interesting semantic occurs as an application talks to the local plumbing on its box. This is all it can see.

Every application is allowed to get bored and abandon its participation in the work. It is useful to have the messaging plumbing track the time since the last message was received. Frequently, the application will want to specify that it is willing to wait only so long until it gives up. If the plumbing helps with this, that's great. If not, the application will need to track on its own any timeouts it needs.

Some application developers may push for no timeout and argue it is OK to wait indefinitely. I typically propose they set the timeout to 30 years. That, in turn, generates a response that I need to be reasonable and not silly. *Why is 30 years silly but infinity is reasonable?* I have yet to see a messaging application that really wants to wait for an unbounded period of time.

When considering the behavior of the underlying message transport, it is best to remember what is promised. *Each message is guaranteed to be delivered zero or more times!* That is a guarantee you can count on.

When Your Plumber Does Not Understand You

Nothing beats a plumber who can alleviate your worries and make everything "just work." It is especially nice if that plumber shares the same understanding of you and your needs.

Many applications just want to have a multimessage dialog between two services in which each accomplishes part of the work. I have just described what can be expected in the best case when you and your plumber share clear notions of:

▸ How you should talk to your plumbing.

▸ Who you are talking to on the other side.

▸ How transactions work with your data (and incoming and outgoing messages).

▸ Whether messages are delivered or you skip messages to get the latest.

▸ When the sender knows a message has been delivered and when it is ambiguous.

▸ When a service may abandon the communication and how the partner learns about it.

▸ How you test the timeout.

These challenges assume you have met the plumber of your dreams who has implemented great support for your messaging environment. It is rarely that clean and simple. On the contrary, the application developer needs to watch out for a slew of issues.

Some messaging systems offer guaranteed delivery. These systems (for example, MQ-Series)[1] will typically record the message in a disk-based queue as a part of accepting the send of the message. The consumption of the message (and its removal from the queue) happens either as part of the transaction stimulated by the message or only after the transactional work has been completed. In the latter case, the work may be processed twice if there is a glitch.

One challenge in the classic guaranteed-delivery queue system occurs when the application gets a message

Figure 4. In most loosely coupled systems, messages may arrive multiple times and out of order.

that it cannot process. Guaranteed delivery means the messaging system delivered it, but there is no guarantee the message was well formed or, even if it was, that the tempestuous application did something reasonable with the message. Before my wife started doing our household bills and it was my responsibility, the reliable delivery of the electric bill to our house was only loosely correlated to the electric company receiving its money.

When considering the behavior of the underlying message transport, it is best to remember what is promised. *Each message is guaranteed to be delivered zero or more times!* That is a guarantee you can count on. There is a lovely probability spike showing that most messages are delivered one time.

If you do not assume the underlying transport may drop or repeat messages, then you will have latent bugs in your application. More interesting is the question of how much help the plumbing layered on top of the transport can give you. If the communicating applications run on top of plumbing that shares common abstractions for messaging, some help may exist. In most environments, the app must cope with this issue by itself.

Why Isn't TCP Enough?

TCP has had a major impact on unifying the ways in which we perform data communication.[5] It offers exactly-once and in-order byte delivery *between two communicating processes*. It offers no guarantees once the connection is terminated or one of the processes completes or fails. This means it covers only a small portion of the landscape visible to developers building reliable applications in a loosely coupled distributed system. Realistically, the application layers on top of TCP and must solve many of the same problems all over again.

Requests get lost, so just about every messaging system retries transmitting. The messaging system often uses TCP, which has its own mechanism to ensure the reliable delivery of bytes from process to process. TCP's guarantees are real but apply only to a single process chatting with exactly one other process. Challenges arise when longer-lived participants are involved.

Consider, for example, HTTP Web requests. HTTP typically shuts down the TCP connection between requests. When a persistent HTTP connection is used, the TCP connection is typically left alive, but there is no guarantee. This means any use of HTTP on top of TCP may result in multiple sends of the HTTP request. For this reason, most HTTP requests are idempotent.[3]

In scalable Web-service worlds, we are constantly reimplementing the same sliding window protocol[2] that is so ubiquitous in TCP, where the endpoints are running processes. The failure of either of the processes means the failure of the TCP connection. In a long-running messaging environment implemented by a collection of servers, the semantics of the endpoint are more complex. As the representation of an endpoint and its state evolves, so do the messaging anomalies that are (hopefully) managed by the plumbing. More likely, they are incrementally solved by the application as patches to surprising bugs. Either way, even application developers will need a copy of Andrew Tanenbaum's classic book, *Computer Networks*, at their fingertips.[2]

Defining Idempotence

To review, idempotence means that multiple invocations of some work are identical to exactly one invocation.

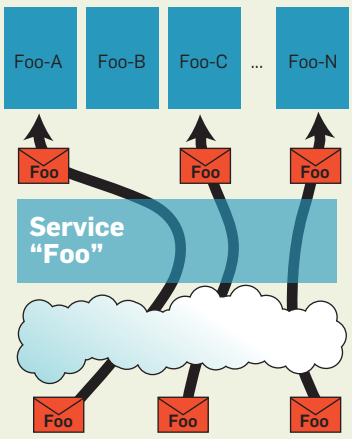
- Sweeping the floor is idempotent. If you sweep it multiple times, you still get a clean floor.
- Withdrawing $1,000,000,000 is not idempotent. Stutering and retrying might be annoying.
- Processing withdrawal 'XYZ' for $1,000,000,000 if not already processed is idempotent.
- Baking a cake is not idempotent.
- Baking a cake starting from a shopping list (if you do not care about money) is idempotent.
- Reading record *X* is idempotent. Even if the value changes, any legitimate value for *X* during the window between the issuance of the read and the return of the answer is correct.

The definition of idempotent in computer usage is: "Acting as if used only once, even if used multiple times." While this is true, there are frequently side effects of the multiple attempts. Let's consider a few side effects that are not typically considered semantically relevant:

- *Heaps.* Imagine a system that uses a heap during the processing of a request. You would naturally expect the heap might become more fragmented with multiple requests.
- *Logging and monitoring.* Most server systems maintain logs that allow analysis and monitoring of the system. Repeated requests will influence the contents of the logs and the monitoring statistics.
- *Performance.* The repeated requests may consume computation, network, and/or storage resources. This may be a tax on the throughput of the system.

These side effects are not relevant to the semantics of the application behavior, so the processing of an idempotent

Figure 5. When communicating with Service Foo, multiple machines may implement the service.



request is still considered idempotent even if side effects exist.

Challenges of Multimessage Interactions

Any message may arrive multiple times, even after a *long* while. Think of a messaging system as containing a bunch of Machiavellian gnomes who are watching your messages float by so they can interject a copy of a message at precisely the worst time for your application (see Figure 4). In most loosely coupled systems, messages may arrive multiple times. Furthermore, related messages may be delivered out of order.

A typical approach to this problem is to use request-response and then ensure the messages processed are idempotent. This has the benefit of the application seeing the delivery of the message via a positive response. If the application gets a response from its intended partner, then it is confident that the message has actually arrived. Because of retries, the receiving application may receive the message many times.

This challenge is further compounded when multiple messages are involved in making a piece of longer-running work happen. The duration of the work and the allowable failures for the work must be considered, as in the following cases:

- *Lightweight process state.* Sometimes the failure of a process at either end of the shared computation can cause the entire workflow to be abandoned. If that is the case, then the intermediate state can be kept in the running process.
- *Long-running durable state.* In this case, the long-running work must continue even if one of the partner systems crashes and restarts. This is common for business processes that may last days or weeks but still have a meaningful messaging interaction.
- *Stateless application servers.* Many architectures have the computation separated from the state. The state may be kept in a durable database on a single server, replicated across a bunch of durable data stores, or any number of other novel ways of storing the state.

In all of these cases, the system must behave correctly in bringing together the state (which is the result of memories from earlier messages) with the new message. Messages may be merged into the state in different orders and those orders may be affected by failures of one or more of the systems.

Many systems implement a target application with a pool of load-balanced servers. This works well when the incoming message makes no assumptions about previous communications and previous state. The first message is routed to one of the servers and gets processed (see Figure 5). When communicating with Service-Foo, multiple machines may be implementing the service. This puts interesting burdens on the implementation of Service-Foo and can show up in anomalous behavior visible to the communicating partner.

Challenges may arise when the second (or later) message arrives and expects the partner not to have amnesia. When chatting with multiple messages, you assume your partner remembers the earlier messages. The whole notion of a multimessage dialog is based on this.

Figure 6. Service behavior is in the eye of the beholder.

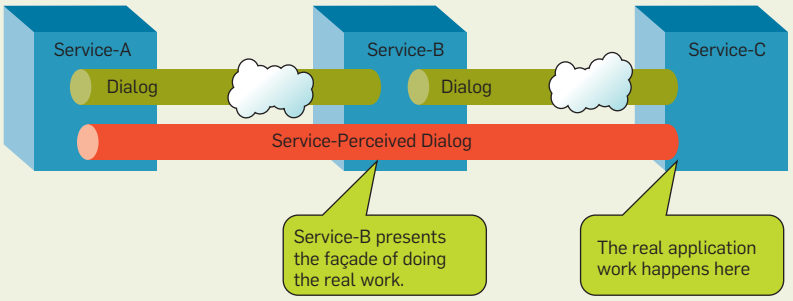
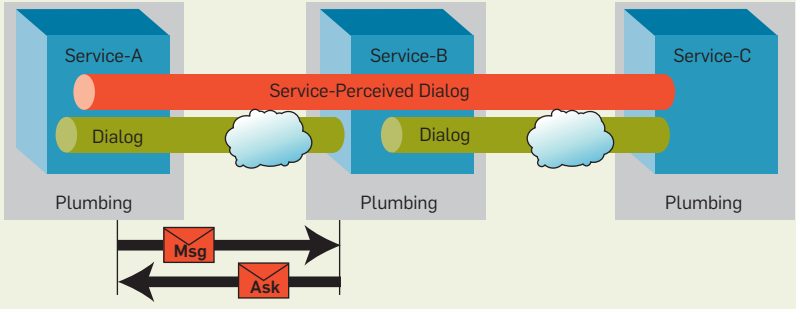


Figure 7. Transport and plumbing acknowledgments should not be visible to applications.



What Do You Mean the Work Is Not Done on the Premises?

When Service A talks to Service B, you do not know where the work is really done. Service A believes it is doing work with Service B, while Service B may actually subcontract all the work to Service C (see Figure 6). This is not in itself a problem, but it can magnify the failure possibilities seen by Service A.

You cannot assume the work is actually done by the system you are chatting with. All you know is that you have a messaging protocol, and, if things are going well, appropriate messages come back from the named partner according to the protocol's definition. You simply do not know what goes on behind the curtain.

You know a message has been delivered only when the answer comes back from the partner doing the work. Tracking the intermediate waypoints does not help to know that the work will get done. Knowing a FedEx package has reached Memphis will not tell you that your grandmother will receive her box of chocolates.

When a messaging transport sends an acknowledgment (ACK) to a sender, it means the message has been

received at the next machine. It says nothing about the actual delivery of the message to the destination and even less about any processing the application may do with the message. This is even more complicated if there is an intermediate application that subcontracts the work to another application service (see Figure 7). Transport and plumbing acknowledgments cannot be visible to applications or there may be bugs introduced when the destination service is reconfigured. The ACK tells Service A's plumbing that Service B's plumbing has the message, but does not tell Service A anything about Service C's receipt of the message. Service A must not act on the ACK.

ACK means sending the message again won't help. If the sending application is made aware of the ACK and acts on that knowledge, then it may cause bugs if and when the real work fails to materialize.

Plumbing Can Mask Ambiguities of Partnership (but rarely does)

It is possible for the message-delivery plumbing to have a formalized notion of a long-running dialog. The plumbing must define and implement the following:

- *State.* How is the state information from a partially completed dialog represented by the application? Remember, either the state must survive failures of a component or the dialog must cleanly fail as a consequence of the failure—just forgetting the early messages is bad.
- *Routing.* How do later messages find a service (and server) that can locate the state and correctly process the new message without amnesia about the earlier messages?
- *Dialog semantics.* How can the dialog provide correct semantics even when the real work of the dialog is subcontracted to somebody way the heck over yonder? Part of this is properly masking transport issues (such as ACKs) that will only cause problems if seen by the application.

Thus, it seems that messaging semantics are intimately tied to naming, routing, and state management. This inevitably means the application designer is faced with challenges when the database comes from a plumbing supply house that is different from that of the messaging system. One system that does provide crisp dialog semantics is SQL Service Broker.[4] It does so by holding the messaging and dialog state in the SQL database.

Figure 8. The first message sent to a service must be idempotent.

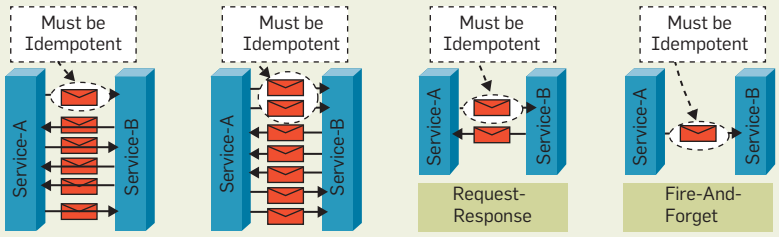
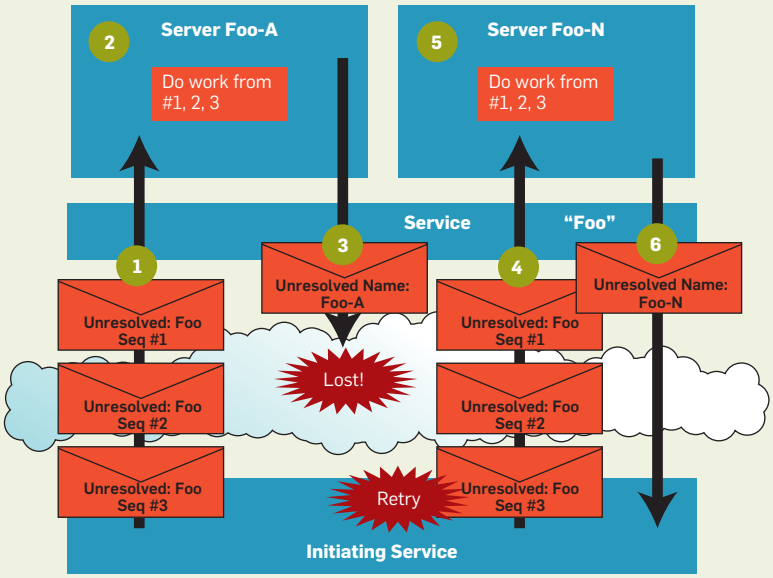


Figure 9. The first messages to a scalable service must be idempotent



Talking to a Service

Services are designed to be black box. You know the service address, start working with it, chatter back and forth, and then finish. It turns out that, even with the support of some plumbing to provide you with dialogs, there are issues with repeated messages and lost messages. These challenges are compounded when the target service is implemented in a scalable fashion. Of course, that means you may start out interacting with a service that is not yet scalable, and, as it grows, certain new obstacles arise.

A dialog goes through three stages in its lifetime:

- *Initiation.* Messages are sent from the dialog initiator to the target of the dialog. The initiator does not know if the target is implemented as a single server or a load-balanced pool.
- *Established.* Messages may flow in both directions in full-duplex fashion. The messaging system has now ensured that either multiple messages will land at the same server in a load-balanced pool or the server processing the message will accurately access the state remembered from the previous messages (and behave as if it were the same server).
- *Closing.* The last message (or messages flowing together in the same direction) gets sent.

Each of these stages of communication offers challenges, especially the initiation and closing stages.

When the first message in a dialog is sent, it may or may not need retrying.

In a load-balanced server environment, the two retries may land at different back-end servers and be processed independently.

From the sender's perspective, the application tosses a message into the plumbing with some name that hopefully will get it to the desired partner. Until you hear a response, you cannot tell if the message was received, the message was lost, the response was lost, or the work is still in progress. The request must be idempotent (see Figure 8).

The first message sent to a service must be idempotent because it may be retried in order to cope with transmission failures. Subsequent messages can count on some plumbing to help (provided that the application runs on top of the plumbing). After the first message, the plumbing can know enough about the destination of the message (in a scalable system) to perform automatic duplicate elimination. During the processing of the first message, the retries may land in different portions of the scalable service, and then automatic duplicate elimination is not possible.

Sometimes a server receives the first message (or messages) from a dialog, then a retry of the message (or sequence of messages) is rerouted to another server in the load-balanced pool. This can go awry in two ways:

- The messaging protocol is designed to keep the state in the target server and responds to the initiator with a more detailed address of the server within the pool for subsequent messages.
- The session state is kept separate from the load-balanced server, and subsequent messages in the protocol include session-identity information that allows the stateless load balancer to fetch the session state and keep going where the protocol left off.

These two approaches are equally challenged by a retry. The first attempt will not be correlated with the second attempt that happened as a result of the retry. This means the initiation messages in a dialog protocol *must be idempotent* (see Figure 9).

When an initiating service sends a sequence of messages to a load-balanced service, the first message must be idempotent. Consider the following events:

1. A sequence of messages (1, 2, and 3) is sent to service Foo, and the load-balancer selects Foo A.
2. The work for the messages is performed at Foo A.
3. The answer is sent back indicating future messages should target Foo A as a resolved name. Unfortunately, the reply is lost by the flaky network transport.
4. A retry to service Foo happens, and the messages are sent to server Foo N.
5. The work for messages 1, 2, and 3 is performed at server Foo N.
6. The response is sent back to the initiator who now knows to keep chatting with Foo N.

Somehow, we must ensure the redundant work performed at Foo A is not a problem.

Figure 10. An application knows the initiation-stage has completed when it hears from the partner.

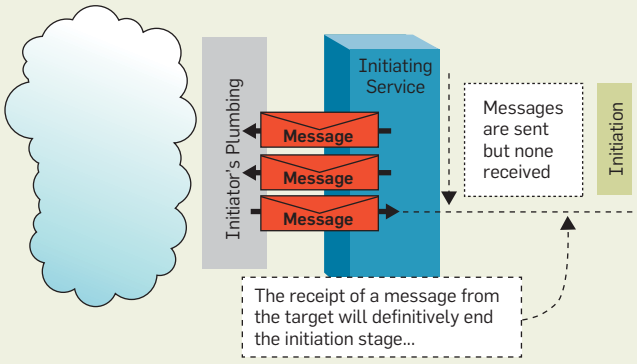
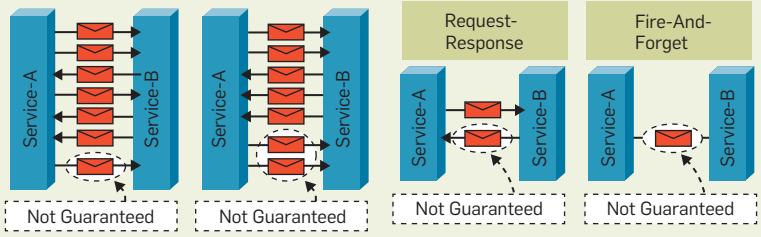


Figure 11. The last messages sent in the same direction cannot be guaranteed.



Accurately Ending the Initiation Stage

An application knows it has reached a specific partner when a message is returned from its local plumbing. If the other service has responded on the dialog, then there is a specific server or a bound session state for the dialog. While the application may not directly see the binding to the specific resources in the partner, they must have been connected by the time an application response is seen.

Prior to the application's receipt of a partner application's message from its local plumbing, any message that is sent will possibly be rerouted to a new (and forgetful) implementation of the partner.

Only after you have heard from the application on the other side may you exit the initiation stage of the dialog (see Figure 10). An application can see only what its plumbing shows it. When an app starts sending messages to its partner, it has the initiation-stage ambiguity and must ensure that all messages have a semantic for idempotent processing. When the local plumbing returns a message from the partner, the local application can be assured that the plumbing has resolved the initiation-stage ambiguity and now messages can be sent without having to ensure that they are idempotent.

Guaranteeing Idempotence of the First Messages

Everything you say as an application in

a dialog before the first answer may be retried. That can cause a world of trouble if the early messages cause some serious (and non-idempotent) work to happen. What's an application developer to do?

There are three broad ways to make sure you do not have bugs with the initialization stage:

▸ *Trivial work.* You can simply send a message back and forth, which does nothing but establish the dialog to the specific partner in the scalable server. This is what TCP does with its SYN messages.

▸ *Read-only work.* The initiating application can read some stuff from the scalable server. This won't leave a mess if there are retries.

▸ *Pending work.* The initiating application can send a bunch of stuff, which the partner will accumulate. Only when subsequent round-trips have occurred (and you know which specific partner and the actual state for the dialog has been connected) will the accumulated state be permanently applied. If the server accumulates state and times out, then the accumulated state can be discarded without introducing bugs.

Using one of these three approaches, the application developer (and not the plumbing) will ensure that no bugs are lying in wait for a retry to a different back-end partner. To eliminate this risk completely, the plumbing can use TCP's trick and send a trivial round-trip set of messages (the SYN messages in TCP), which hooks up the partner without bothering the application layered on top. On the other hand, allowing the application to do useful work with the round-trip (for example, read some data) is cool.

The Closing-Stage Ambiguity

In any interaction, *the last message from one application service to another cannot be guaranteed*. The only way to know it was received is to send a message saying it was. That means it is no longer the last message.

Somehow, some way, the application must deal with the fact that the last message or messages sent in the same direction may simply go poof in the network. This may be in a simple request-response, or it may be in a complex full-duplex chatter. When the last messages are sent, it is just a matter of luck if they actually get delivered (see Figure 11). In each interaction between two partners, the last messages sent in the same direction cannot be guaranteed. They may simply disappear.

The penultimate message can be guaranteed (by receiving the notification in the ultimate message). The ultimate message must be best effort. This complexity is typically part of designing an application protocol. The application must not really care if that last message is received because you cannot really tell if it is.

As we have seen, most loosely coupled systems depend on the application designer to consider repeated processing of a request. The complications are made even worse in a scalable implementation of a service. Application designers must learn to live with the reality of idempotence in their daily life.

Conclusion

Distributed systems can pose challenges to applications sending messages. The messaging transport can be downright mischievous. The target for the message may be an illusion of a partner implemented by a set of worker bees. These, in turn, may have challenges in their coordination of the state of your work. Also, the system you *think* you are talking to may be, in fact, subcontracting the work to other systems. This, too, can add to the confusion.

Sometimes an application has plumbing that captures its model of communicating partners, lifetime of the partners, scalability, failure management, and all the issues needed to have a great two-party dialog between communicating application components. Even in the presence of great supporting plumbing, there are still semantic challenges intrinsic to messaging.

This article has sketched a few principles used by grizzled old-timers to provide resilience even when "stuff happens." In most cases, these programming techniques are used as patches to applications when the rare anomalies occur in production. As a whole, they are not spoken about too often and rarely crop up during testing. They typically happen when the application is under its greatest stress (which may be the most costly time to realize you have a problem).

Some basic principles are:

▸ Every message may be retried and, hence, must be idempotent.

▸ Messages may be reordered.

▸ Your partner may experience amnesia as a result of failures, poorly managed durable state, or load-balancing switchover to its evil twin.

▸ Guaranteed delivery of the last message is impossible.

Keeping these principles in mind can lead to a more robust application.

While it is possible for plumbing or platforms to remove some of these concerns from the application, this can occur only when the communicating apps share common plumbing. The emergence of such common environments is not imminent (and may never happen). In the meantime, developers need to be thoughtful of these potential dysfunctions in erecting applications.

Acknowledgments

Thank you to Erik Meijer and Jim Maurer for their commentary and editorial improvements.

Related articles on queue.acm.org

BASE: An Acid Alternative
Dan Pritchett
http://queue.acm.org/detail.cfm?id=1394128

A Co-Relational Model of Data for Large Shared Data Banks
Erik Meijer and Gavin Bierman
http://queue.acm.org/detail.cfm?id=1961297

Testable System Administration
Mark Burgess
http://queue.acm.org/detail.cfm?id=1937179

References

1. IBM. WebSphere MQ; http://www-01.ibm.com/software/integration/wmq/.
2. Tanenbaum, A.S. *Computer Networks*, 4th ed. Prentice Hall, 2002.
3. World Wide Web Consortium, Network Working Group. 1999. Hypertext Transfer Protocol—HTTP1.1; http://www.w3.org/Protocols/rfc2616/rfc2616.html.
4. Wolter, R. An introduction to SQL Server Service Broker (2005); http://msdn.microsoft.com/en-us/library/ms345108(v=sql.90).aspx.
5. Transmission Control Protocol; http://www.ietf.org/rfc/rfc793.txt

Pat Helland has worked in distributed systems, transaction processing, databases, and similar areas since 1978. For most of the 1980s, he was the chief architect of Tandem Computers' TMF (Transaction Monitoring Facility), which provided distributed transactions for the NonStop System. With the exception of a two-year stint at Amazon, Helland worked at Microsoft Corporation from 1994–2011 where he was the architect for Microsoft Transaction Server and SQL Service Broker. He also contributed to Cosmos, a distributed computation and storage system that provide back-end support for Bing.

DOI:10.1145/2160718.2160735

Article development led by acmqueue
queue.acm.org

Web and mobile applications are increasingly composed of asynchronous and real-time streaming services and push notifications.

BY ERIK MEIJER

Your Mouse Is a Database

AMONG THE HOTTEST buzzwords in the IT industry these days is "big data," but the "big" is something of a misnomer: big data is not just about volume, but also about velocity and variety:[4]

- The *volume* of data ranges from a small number of items stored in the closed world of a conventional RDBMS (relational database management system) to a large number of items spread out over a large cluster of machines or across the entire World Wide Web.
- The *velocity* of data ranges from the consumer synchronously pulling data from the source to the source asynchronously pushing data to its clients, at different speeds, ranging from millisecond-latency push-based streams of stock quotes to reference data pulled by an application from a central repository once a month.
- The variety of data ranges from SQL-style relational tuples with foreign-/primary-key relationships to coSQL[6]-style objects or graphs with key-value pointers, or even binary data such as videos and music.

If we draw a picture of the design space for big data along these three dimensions of volume, velocity, and variety, then we get the big-data cube shown in Figure 1. Each of the eight corners of the cube corresponds to a (well-known) database technology. For example, the traditional RDBMS is at the top-back corner with coordinates (small, pull, fk/pk), meaning that the data sets are small; it assumes a closed world that is under full control by the database, clients synchronously pull rows out of the database after they have issued a query, and the data model is based on Codd's relational model. Hadoop-based systems such as HBase are on the front-left corner with coordinates (big, pull, fk/pk). The data model is still fundamentally rectangular with rows, columns, and primary keys, and results are pulled by the client out of the store, but the data is stored on a cluster of machines using some partitioning scheme.

When moving from the top plane to the bottom plane, the data model changes from rows with primary and foreign keys to objects and pointers. On the bottom-left corner at coordinates (small, pull, k/v) are traditional O/R (object/relational) mapping solutions such as LINQ to SQL, Entity Framework, and Hibernate, which put an OO (object-oriented) veneer on top of relational databases. In the front of the cube is LINQ to Objects with coordinates (big, pull, k/v). It virtualizes the actual data source using the `IEnumerable<T>` interface, which allows for an infinite collection of items to be generated on the fly. To the right, the cube changes from batch processing to streaming data where the data source asynchronously pushes a stream of items to its clients. Streaming database systems with a rows-and-columns data model such as Percolator, StreamBase, and StreamInsight occupy the top-right axis.

Finally, on the bottom right at coordinates (big, push, k/v), is Rx (Reactive Extensions), or as it is sometimes

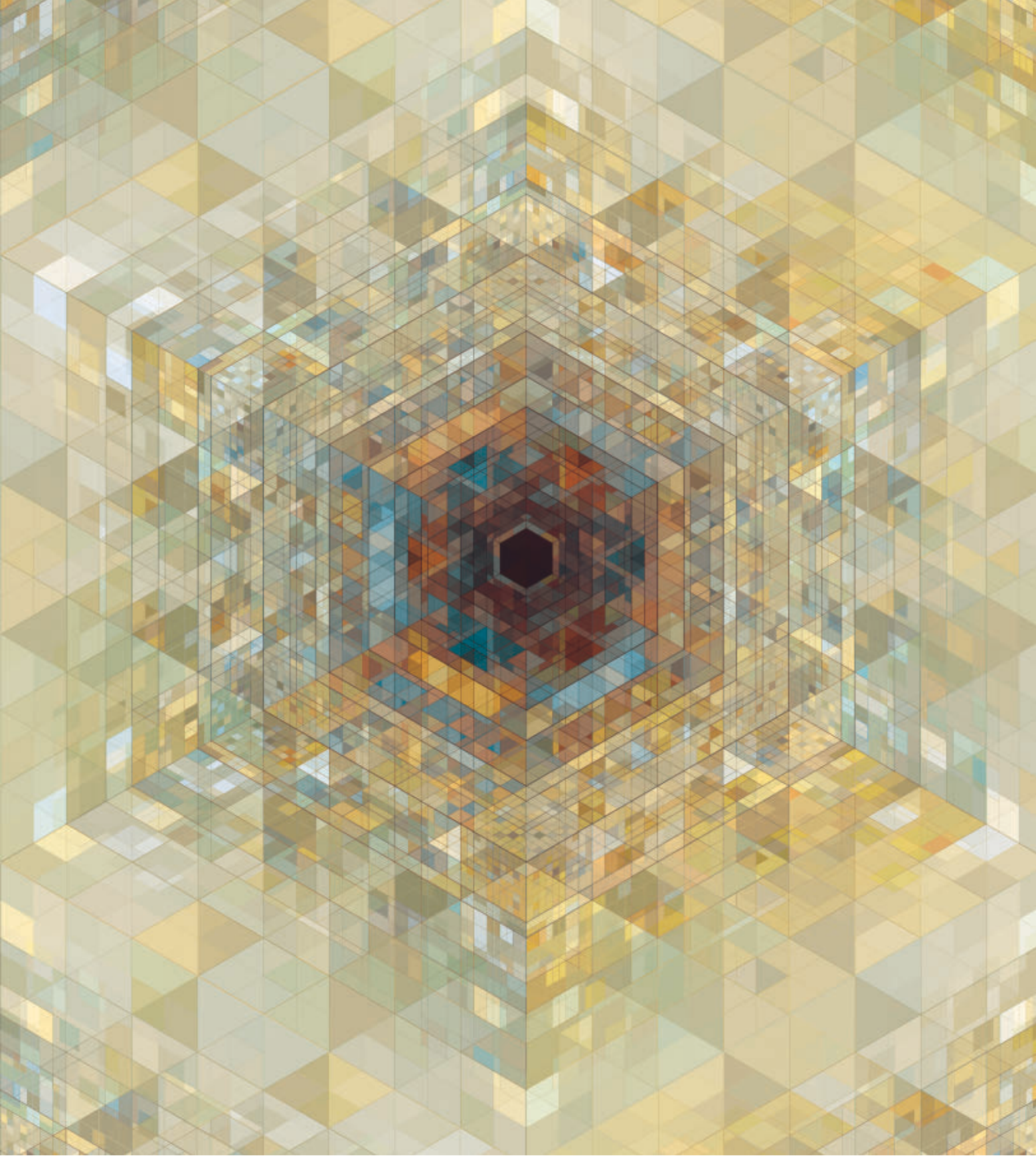


ILLUSTRATION BY ANDY GILMORE

called, LINQ to Events, which is the topic of this article.

The goal of Rx is to coordinate and orchestrate event-based and asynchronous computations such as low-latency sensor streams, Twitter and social media status updates, SMS messages, GPS coordinates, mouse moves and other UI events, Web sockets, and high-latency calls to Web services using standard object-oriented programming languages such as Java, C#, or Visual Basic.

There are many ways to derive Rx, some involving category theory and appealing to mathematical duality, but this article shows how every developer could have invented Rx by crossing the standard JDK (Java Development Kit) `Future<T>` interface with the GWT (Google Web Toolkit) `AsyncCallBack<T>` inter-

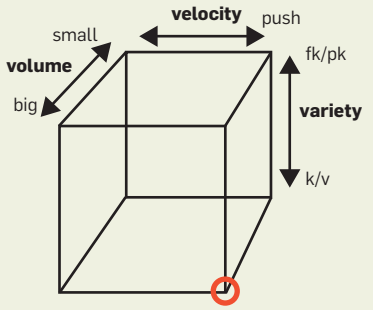
face to create the pair of interfaces `IObservable<T>` and `IObserver<T>` that model asynchronous data streams with values of type `T`. This corresponds to the well-known Subject/Observer design pattern. The article then shows how to write a simple Ajax-style application by exposing UI events and Web services as asynchronous data streams and composing them using a fluent API.

Crossing Futures And Callbacks

The GWT developer documentation contains a slightly apologetic section called "Getting Used to Asynchronous Calls,"[3] which explains that while asynchronous calls at first sight look cruel and unnatural to the developer, they are a necessary evil to prevent the UI from locking up and allow a client to have multiple concurrent outstanding server requests.

In GWT the asynchronous counterpart of a synchronous method, say `Person[] getPeople(…)`, that makes a synchronous cross-network call and blocks until it returns an array of Person, would return void and take an additional callback argument `void getPeople(…, AsyncCallback<Person[]> callback)`. The callback interface has two methods: `void onFailure(Throwable error)`, which is called when the asynchronous call throws an exception; and `void onSuccess(T result)`, which is called when the asynchronous call successfully returns a result value. Given an asynchronous function such as `getPeople`, an invocation typically passes an anonymous interface implementation that handles the success and failure callbacks, respectively, as illustrated in Figure 2.

Figure 1. 1 Big-data cube.



While the commitment to asynchrony in GWT is laudable, it misses a huge opportunity by not further refining and unifying the asynchronous programming model across the entire framework. For example, the `RequestBuilder` class for making direct HTTP calls uses the `RequestCallback` interface that has two methods `onError` and `onResponseReceived` that are virtually isomorphic to the methods of the `AsyncCallback` interface previously discussed.

Both the `AsyncCallback` and `RequestCallback` interfaces assume that asynchronous calls deliver their results in one shot. In the example, however, returning the elements of the Person array incrementally in a streaming fashion makes perfect sense, especially when the result set is large or even infinite. You can asynchronously stream back results by allowing the `onSuccess` method to be called multiple times, once for each additional chunk of the result array, and by adding a method `void onCompleted()`, which is called when all chunks have been delivered successfully. Let's call this derived interface `Observer<T>` to indicate that it can observe multiple `T` values before completing and to reflect the standard Java nongeneric `observer` interface.

With `Observer<T>` instead of `AsyncCallback<T>`, the possible sequences of interaction between an asynchronous computation and its client are: successful termination after $i \geq 0$ values; unsuccessful termination after j values; or an infinite stream of values that never completes, as shown in Figure 3.

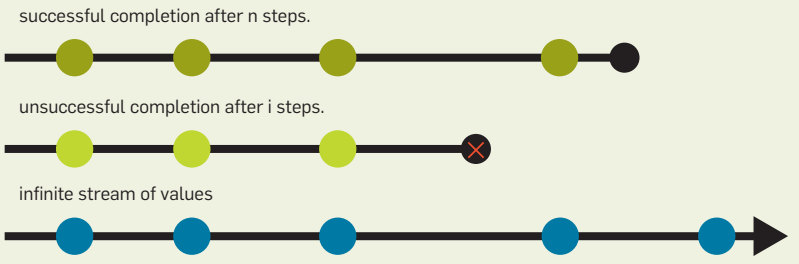
Another downside of passing callbacks directly as parameters to asynchronous methods is that revoking a callback from being invoked is tricky once the call has been made, leaving you with just a void in your hands. Suppose, for example, that the function `getPeople` streams back the names of the people who have signed up for a marketing promotion every minute, but that you are not interested in receiving more than the first thousand names. How do you achieve this later if you did not anticipate this pattern when you made the call and received back `void`. Even if the asynchronous call delivers at most one value, you may choose later to ignore or cancel the call by timing out after not receiving a result within a certain time interval. Again, this is possible if you anticipated this when passing the callback into `getPeople`, but you cannot change your mind later.

These hitches are symptoms of the fact that asynchronous computations and streams are not treated as first-class values that can be returned from methods, stored in variables, and so on. The next section shows how to make asynchronous data streams into proper values by introducing an

Figure 2. Asynchronous fetching data from a service.

```
service.getPeople(startRow, maxRows, new AsyncCallback<Person[]>() {
   void onFailure(Throwable error) { …code to handle failure… }
   void onSuccess(Person[] result) { …code to handle success… }
});
```

Figure 3. Possible sequences of interaction when using `Observer<T>`.



additional container interface that represents the asynchronous computation itself and on which you register the callbacks to be notified about the results of the computation. Now asynchronous methods can return a value that represents the pending asynchronous computation or stream instead of just void. In particular, this allows you to change your mind after making the call and filter, manipulate, or transform the computation at will.

The Java SDK already provides (single-shot) asynchronous computations as first-class values in the form of the Future<T> interface, whose principal method T get() retrieves the result of the computation and blocks when the underlying computation has not yet terminated:

```
interface Future<T>
{
  boolean cancel(boolean mayInterrup-
  tIfRunning);
  T get();
  T get(long timeout, TimeUnit unit);
  boolean isCancelled();
  boolean isDone();
}
```

Note that in principle, Future<T> could be used to produce multiple values. In this case each call to get() would block and return the next value once it is available, as long as isDone() is not true. This is similar to the iterable interface. In the rest of this article, asynchronous computations are assumed to return streams of multiple results.

While futures do provide a first-class representation of asynchronous computations, the get method is blocking. Fortunately, you can make the JDK interface Future<T> nonblocking by supplying the T get() method with a callback of type Observer<T> (the interface introduced to extend the AsyncCallback<T> interface of GWT). Note that the blocking isCancelled and isDone methods are no longer needed because that information is transmitted via the callback as well. For simplicity ignore the second overload of get since you can easily reconstruct that later. Applying these changes, the nonblocking version of the Future<T> interface looks like this:

```
interface Future<T>
{
  boolean cancel(boolean mayInterrup-
  tIfRunning);
  void get(Observer<T> callback);
}
```

You are not yet done refactoring. Instead of canceling the future as a whole via the cancel method, it makes more sense to cancel just the particular outstanding call to get per observer. This can be achieved by letting get return an interface that represents a cancelable resource. Moreover, since you have already called get, there is no need to specify the mayInterruptIfRunning parameter, because the computation is already running at that point and you can encode the Boolean by deciding whether or not to call cancel. Lastly, you can make the cancel method nonblocking by returning void instead of boolean. You could try to make cancel return a Future<boolean> instead, but then you would fall into an endless recursive rabbit hole of asynchrony. As it turns out the java.io.Closable interface precisely fits the bill, resulting in the following mutation of Future<T>:

```
interface Future<T> { Closable
get(Observer<T> callback); }
```

Note that calling the close() method of the Closable interface returned by a subscription may or may not actually cancel the underlying computation because a single observable may have multiple observers (disposing, say, of the subscription to mouse moves, which should not stop your mouse from working). Since that particular observer is not notified of any further values, however, from its perspective the computation has terminated. If needed, the class that implements IObservable<T> could cancel the computation in some other way.

Instead of Future<T> and Observer<T>, .NET has the standard IObservable<T> and IObserver<T> interfaces; and instead of Closable, it has IDisposable. Values of type IObservable<T> (or Observer<T> depending on your preferred programming language) represent asynchronous data streams, or event streams, with values of type T.

```
interface IObservable<T>
{
   IDisposable Subscribe(IObserver<T>
   observer);
}
interface IObserver<T>
{
   void OnNext(T value);
   void OnError(Exception error);
   void OnCompleted();
}
interface IDisposable
{
   void Dispose();
}
```

A closer inspection of the resulting interface trinity reveals a generic variation of the classic Subject/Observer interface[2] for the publish/subscribe pattern, a staple in the tool chest of object-oriented programmers for decades for dealing with event-based systems. JDK 1.0 already supports this pattern via the (nongeneric) Observable class and the Observer interface. In .NET, the Rx library supports the pattern.

The Rx library makes some additional behavioral assumptions about the IObserver<T> and IObservable<T> interfaces that are not expressed by their (syntactic) type signatures:

- The sequence of calls to an instance of the IObserver<T> interface should follow the regular expression OnNext(t)* (OnCompleted() | OnError(e))?. In other words, after zero or more OnNext calls, either one of OnCompleted or OnError will optionally be called.
- Implementations of IObserver<T> can assume to be synchronized; conceptually they run under a lock, similar to regular .NET event handlers, or the reactor pattern.[10]
- All resources associated with an observer should be cleaned up at the moment OnError or OnCompleted is called. In particular, the subscription returned by the Subscribe call of the observer will be disposed of by the observable as soon as the stream completes. In practice this is implemented by closing over the IDisposable returned by Subscribe in the implementation of the OnError and OnCompleted methods.
- When a subscription is disposed externally, the IObservable<T> stream

should make a *best-effort* attempt to stop all outstanding work for that subscription. Any work already in progress might still complete as it is not always safe to abort work in progress but should not be signaled to unsubscribed observers. This contract ensures it is easy to reason about and prove the correctness of operators and user code.

Asynchronous Data Streams

To create an instance of an `Observable<T>` in Java, you would use anonymous inner classes and define an abstract base class `ObservableBase<T>` that takes care of enforcing the Rx contract. It is specialized by providing an implementation of the subscribe method:

```
Observable<T> observable = new
ObservableBase<T>()
{
   Closable subscribe(Observer<T>
   observer) { … }
};
```

Since .NET lacks anonymous interfaces, it instead uses a factory method `Observable.Create` that creates a new observable instance from an anonymous delegate of type `Func<IObservable<T>, IDisposable>` that implements the `Subscribe` function:

```
IObservable<T> observable =
Observable.Create<T>(
         IObserver<T> observer =>
         { … }
      );
```

Just as in the Java solution, the concrete type returned by the `Create` method enforces the required Rx behavior.

Once you have a single interface to represent asynchronous data streams, you can expose existing event- and callback-based abstractions such as GUI controls as sources of asynchronous data streams. For example, you can wrap the text-changed event of a `TextField` control in Java as an asynchronous data stream using the delicious token salad illustrated in Figure 4.

You can think of a UI control, the mouse, a text field, or a button as a streaming database that generates an infinite collection of values for each time the underlying control fires an event. Conversely, objects with settable properties such as lists and labels can be used as observers for such asynchronous data streams.

Asynchronous data streams represented by the `IObservable<T>` interface (or `Observable<T>` in Java) behave as regular collections of values of type `T`, except that they are push-based or streaming instead of the usual pull-based collections such as arrays and lists that implement the `IEnumerable<T>` interface (or in `Java iterable<T>`). This means you can wire asynchronous data streams together using a fluent API of standard query operators to create complex event-processing systems in a highly composable and declarative way.

For example, the `Where` operator takes a predicate of type `Func<S,bool>` and filters out all values for which the predicate does not hold

Figure 4. Exposing GUI elements as sources and sinks for asynchronous data streams.

```
Observable<string> TextChanges(JTextField tf){
   return new ObservableBase<string>(){
      Closable subscribe(Observer<string> o){
            DocumentListener l = new DocumentListener(){
             void changedUpdate(DocumentEvent e {
                    o.OnNext(tf.getText());};
                tf.addDocumentListener (l);
                return new Closable() {
                   void close(){tf.removeDocumentListener(l);}}}}}
```

Every time the `changedUpdate` event fires, the corresponding asynchronous data stream of type `Observable<string>` pushes a new string to its subscribers, representing the current content of the text field.

Likewise, you can expose objects with setters as sinks of asynchronous data streams by wrapping them as observers. For example, expose a `javax.swing.JList<T>` list into an `Observer<T[]>` by setting the `listData` property to the given array whenever `onNext` is called:

```
Observer<T[]> ObserveChanges(javax.swing.JList<T> list){
            return new ObserverBase<T[]>() {
               void onNext(T[] values){ list.setListData(values); }}}
```

Figure 5. The Where operator.

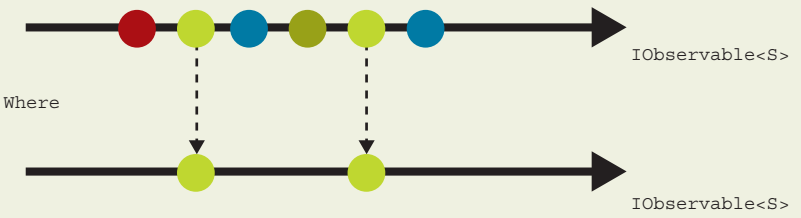
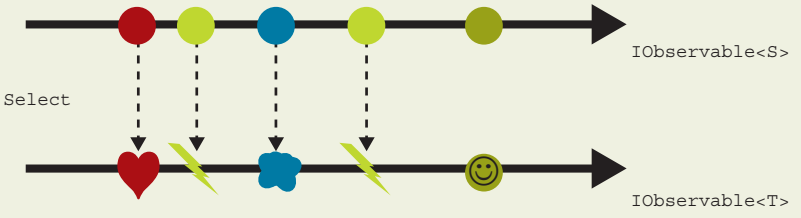


Figure 6. The Select operator.



an input-observable collection of type IObservable<S>, precisely the same as its cousin that works on pull-based IEnumerable<T> collections. Figure 5 illustrates this.

Using this operator, you can cleanse a text field input exposed as IObservable<string> stream and remove all empty and null strings using the query expression input. Where(s=>!string.IsNullOrEmpty(s)).

In Java 8 with lambda expressions and defender methods, the code would look very similar to the C# code shown here, just using -> instead of => for lambdas and different casing of variable names. Even without those upcoming Java language features, however, you can approximate a fluent interface in Java—as in FlumeJava[1] or Reactive4Java[9]—for manipulating event streams using standard query operators. For example, by having the operators as methods on ObservableBase<T>, you can write the filter example as:

```
input.Where<T>(new Func<string,T>{
 Invoke (string s){
   return !(s == null || s.length() == 0); }}
```

To save us all from too much typing, however, the next couple of examples are provided only in C#, even though nothing is C# or .NET specific.

The Select operator takes a transformation function Func<S,T> to transform each value in the input data stream of type IObservable<S>. This produces a new asynchronous result stream of type IObservable<T>, again exactly like the IEnumerable<T>-based version, as seen in Figure 6.

The SelectMany operator is often used to wire together two data streams, pull-based or push-based. SelectMany takes a source stream of type IObservable<S> and an inflator function of type Func<S, IObservable<T>>, and from each element in the original source stream generates a new nested stream of zero, one, or more elements. It then merges all intermediate asynchronous data streams into a single output stream of type IObservable<T>, as shown in Figure 7.

The SelectMany operator clearly shows the difference between the asynchronous nature of IObservable<T> and the synchronous nature of IEnumerable<T>. As Figure 7 shows, values on the source stream appear asynchronously, even as you are still producing values from a previous inflator function. In the IEnumerable<T> case, the next value is pulled from the source stream only after all values from the inflator function have been produced (that is, the output stream is the concatenation of all subsequent inflator-produced streams, not the nondeterministic interleaving), shown in Figure 8.

Sometimes it is convenient to generate the output stream of an asynchronous stream using a more sequential pattern. As shown in Figure 9, the Switch operator takes a nested asynchronous data stream IObservable<IObservable<T>> and produces the elements of the most recent inner asynchronous data stream that has been received up to that point. This produces a new nonnested asynchronous data stream of type IObservable<T>. It allows later streams to override earlier streams, always yielding the "latest possible results," rather like a scrolling news feed.

At this point, the determined reader may attempt to create his or her own implementation of Switch, which, as it turns out, is surprisingly tricky, especially dealing with all edge conditions while also satisfying the Rx contract.

Using the fluent API previously introduced, you can program the prototypical Ajax "dictionary suggest" program in a few lines. Assume you have

Figure 7. The SelectMany operator.

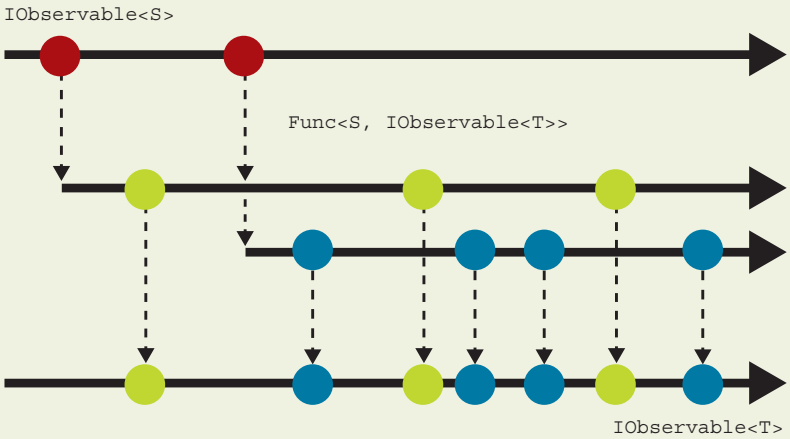
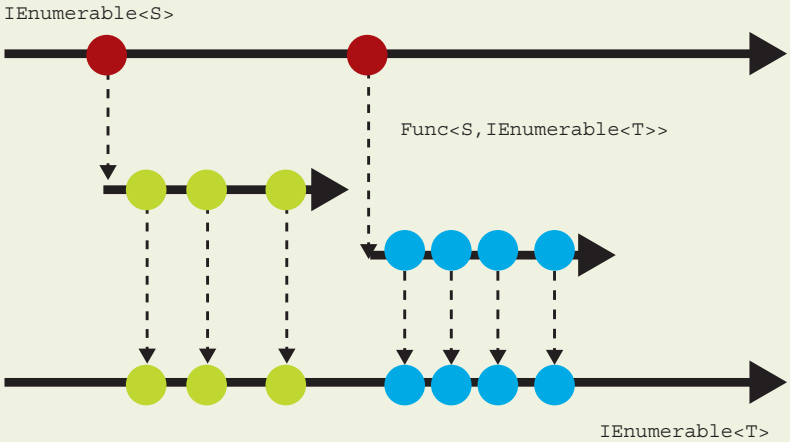


Figure 8. Concatenation of all subsequent inflator-produced streams.



a dictionary-lookup Web service that, given a word, asynchronously returns an array of completions for that word via the following method:

```
IObservable<string[]> Completions(string word){ … }
```

Assume also that using these helpers, you have exposed a GUI text field input as an `IObservable<string>` to produce the value of the text field every time the input changes, and you have wrapped a label output as an `IObserver<string[]>` to display an asynchronous data stream of string arrays. Then you can wire up a pipeline that asynchronously calls the `Completions` service for every partial word typed into the text field but displays only the most recent result on the label:

```
TextChanges(input)
.Select(word=>Completions(word))
.Switch()
.Subscribe(ObserveChanges(output));
```

Figure 9. The Switch operator.

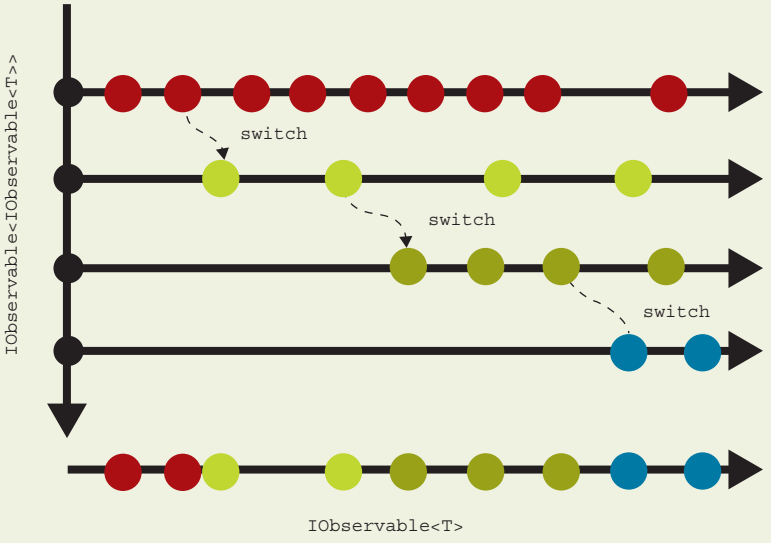


Figure 10. Asynchronous call made to Completions in response to a change in the input.

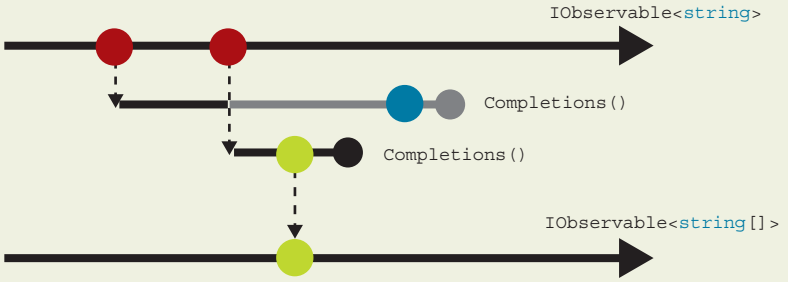
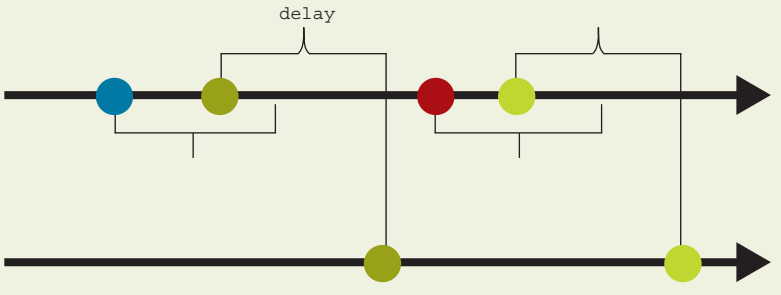


Figure 11. The Throttle operator.



The effect of the `Switch` operator is that every time another asynchronous call is made to `Completions` in response to a change in the input, the result is switched to receive the output of this latest call, as shown in Figure 10, and the results from all previous calls that are still outstanding are ignored.

This is not just a performance optimization. Without using `Switch`, there would be multiple outstanding requests to the `Completion` service, and since the stream is asynchronous, results could come back in arbitrary order, possibly updating the UI with results of older requests.

This basic dictionary example can be improved by inserting a few more operators into the query. The first operator is `IObservable<T> DistinctUntilChanged(IObservable<T> source)`, which ensures that an asynchronous data stream contains only distinct contiguous elements—in other words, it removes adjunct elements that are equivalent. For the example, this ensures that `Completions` is called only when the input has actually changed.

Second, if the user types faster than you can make calls to the Web service, a lot of work will go to waste since you are firing off many requests, only to cancel them immediately when the input changes again before the previous result has come back. Instead, you want to wait at least `N` milliseconds since the last change using the operator `IObservable<T> Throttle(IObservable<T> source, TimeSpan delay)`. The throttle operator samples an asynchronous data stream by ignoring values that are followed by another value before the specified delay, as shown in Figure 11.

The throttle operator drops events that come in at too high a rate; however, one can easily define other opera-

tors that aggregate events in (tumbling) windows or sample the input stream at certain intervals.

The final Ajax program presented here is a dataflow pipeline that invokes the dictionary service only if there has not been a new distinct value fired by input in the last 10 milliseconds; and it ignores the result of the service if a new input value appears before the completion of the previous value has returned:

```
TextChanges(input)
.DistinctUntilChanged()
.Throttle(TimeSpan.FromMilliSeconds(10))
.Select(word=>Completions(word))
.Switch()
.Subscribe(ObserveChanges(output));
```

Of course, the `IObservable<T>` interface is not restricted to UI events and asynchronous computations but applies equally well to any push-based data stream such as tweets, stock stickers, and GPS position, and of course the kinds of asynchronous service invocations for GWT we started off with. For example, you can model a Twitter stream that filters on a given hashtag as a method:

```
IObservable<Tweet>
TweetsByHashTag(string hashtag, …)
```

This will push an (infinite) stream of tweets to the client that called the function. Further, looking inward (rather than outward), observers are natural expressions of "completion ports" for asynchronous I/O and co-processor operations such as those from DSPs (digital signal processors) and GPUs (graphics processing units).

Show Me The (Co)Monads!

So far we have been able to avoid the "M" word (and the "L" word as well), but there's no more hiding it. If we ignore operational concerns such as exceptions, termination, and canceling subscriptions and boil things down to their essence, the `IObservable<T>` and `IObserver<T>` interfaces represent functions of type `(T->())->()`, which is the continuation monad, the mother of all monads, and a co-monad.

Historically, we did not discover the Rx interfaces by the refactorings performed in this article. Instead we applied the definition of *categorical duality* from Wikipedia literally to the `IEnumerable<T>` and `IEnumerator<T>` interfaces for pull-based collections, and thus derived the `IObservable<T>` and `IObserver<T>` interfaces completely mechanically by swapping the arguments and results of all method signatures, not guided by any operational intuition in the process.

Note that our model of asynchronous data streams makes no special assumptions about time. This makes the approach different from the typical reactive programming approaches in functional programming such as Fran or FlapJax that emphasize (continuous) time-varying values, called behaviors, and SQL-based complex event-processing systems such as StreamBase and StreamInsight that also emphasize time in their semantic model. Instead clocks and timers are treated just as regular asynchronous data streams of type `IObservable<DateTimeOffset>`. We parameterize over concurrency and logical clocks by another interface `IScheduler` (slightly simplified here), which represents an execution context that has a local notion of time on which work can be scheduled in the future:

```
interface IScheduler
{
  DateTimeOffset Now { get; }
  IDisposable Schedule(Action work,
TimeSpan dueTime)
}
```

Java programmers will immediately see the correspondence with the executor interface that in the Java SDK plays the same role of abstracting over the precise introduction of concurrency.

Conclusion

Web and mobile applications are increasingly composed of asynchronous and streaming services, emphasizing the velocity aspect of the three V's of big data. This article has shown how to expose asynchronous data streams as push-based collections of type `IObservable<T>` (in contrast to pull-based collections of type `IEnumerable<T>`) and how to query these asynchronous data streams using the fluent API operators provided by the Rx library. This popular library is available for .NET and JavaScript (including bindings for prevalent frameworks such as JQuery and Node) and also ships in the ROM of Windows Phone. F#'s first-class events are based on Rx, and alternative implementations for other languages, such as Dart[8] or Haskell,[7] are created by the community.

To learn more about LINQ in general and Rx in particular, read the short textbook *Programming Reactive Extensions and LINQ*.[5]

Related articles on queue.acm.org

Nine IM Accounts and Counting
Joe Hildebrand
http://queue.acm.org/detail.cfm?id=966720

Debugging in an Asynchronous World
Michael Donat
http://queue.acm.org/detail.cfm?id=945134

Scripting Web Services Prototypes
Christopher Vincent
http://queue.acm.org/detail.cfm?id=640158

References

1. Chambers, C., Raniwala, A., Perry, F., Adams, S., Henry, R. R., Bradshaw, R., Weizenbaum, N. FlumeJava: Easy, efficient, data- parallel pipelines. *Proceedings of the ACM SIGPLAN Conference on Programming Design and Implementation* (2010); https://dl.acm.org/citation.cfm?id=1806638.
2. Eugster, P. Th., Felber, P. A., Guerraiou, R., Kermarrec, A-M. The many faces of publish/ subscribe. *ACM Computing Surveys 35*, 2 (2003), 114–131; https://dl.acm.org/citation.cfm?id=857076.857078.
3. Google Web Toolkit. Getting used to asynchronous calls (2007); http://www.gwtapps.com/doc/html/com.google.gwt.doc.DeveloperGuide.RemoteProcedureCalls.GettingUsedToAsyncCalls.html.
4. Laney, D. 3D data management: Controlling data volume, velocity, and variety. Application Delivery Strategies (2001); http://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data-Management-Controlling-Data-Volume-Velocity-and-Variety.pdf.
5. Liberty, J., Betts, P. *Programming Reactive Extensions and LINQ*. Apress, New York, 2011; http://www.apress.com/9781430237471.
6. Meijer, E. and .Bierman, G. A co-relational model of data for large shared data banks. *Commun. ACM 54*, 4 (Apr. 2011), 49–58.
7. Reactive-bacon; https://github.com/raimohanska/reactive-bacon.
8. Reactive-Dart; https://github.com/prujohn/Reactive-Dart.
9. Reactive4java; https://code.google.com/p/reactive4java/.
10. Wikipedia. Reactor pattern; https://en.wikipedia.org/wiki/Reactor_pattern.

Erik Meijer (emeijer@microsoft.com) has been working on "Democratizing the Cloud" for the past 15 years. He is perhaps best known for his work on the Haskell language and his contributions to LINQ and the Rx Framework.

DOI:10.1145/2160718.2160736

Twitter sentiment was revealed, along with popularity of Egypt-related subjects and tweeter influence on the 2011 revolution.

BY ALOK CHOUDHARY, WILLIAM HENDRIX, KATHY LEE, DIANA PALSETIA, AND WEI-KENG LIAO

Social Media Evolution of the Egyptian Revolution

THE 2011 EGYPTIAN revolution, which drew inspiration from protests in Tunisia and led to widespread anti-government uprisings throughout North Africa and the Middle East, began with protests on January 25, 2011, culminating February 11, 2011 with the resignation of President Hosni Mubarak.[1] The protests and the revolution and their reflection in social media garnered enormous worldwide media attention. While some analysts said social media like Twitter and Facebook were not critical to the revolution,[4] others, including a number of activists on the scene, credit social media with helping the movement achieve critical mass.[2,3] Whatever role Twitter and other social media played, there is no question that the protesters, as well as media and journalists, made extensive use of social networking sites, tweeting and texting on the protests, with the messages read and commented on by people around the world. In this sense, the stream of tweets represents an enormous, unfiltered history of events from a multitude of perspectives, as well as an opportunity to characterize the revolution's events and emotions.

» key insights

- **Scalable analytics helps make sense of the influence on and outcome of large-scale social and political events, as they unfold.**
- **Advanced algorithms for sentiment and response analysis helps differentiate behavior patterns of groups within different communities.**
- **Twitter discussion on the Egyptian revolution was much different from other Twitter discussions in terms of who was tweeting and what was said.**

PHOTOGRAPH BY DYLAN MARTINEZ/REUTERS

We set out to weigh more than 800,000 tweets on topics related to the revolution so we could present our analysis here from three different perspectives:[a] First, how sentiment evolved in response to unfolding events; how the most influential tweeters and popular tweets shed light on the most influential Twitter users and what types of tweets reverberated most strongly; and how user sentiment and follower relationships relate in terms of dynamic social network characteristics and sentiment. Using these metrics, we compared Egypt-related topics during the revolution to other popular early-2011 trending topics (such as politics, sports, and entertainment) (see the sidebar "Pulse of a Revolution").

a http://www.pulseofthetweeters.com

Evolving Emotion

To get a sense of how Twitter activity reflected the events of the revolution, we selected six Egypt-related topics that trended at least three days during the period January 25–February 11, 2011, where a trending topic was a subject identified by Twitter that notably increased how much it was discussed.[5] We also selected topics that trended several times to see how conversations developed in response to the events of the revolution.

We collected thousands to hundreds of thousands of tweets for each day Egypt-related topics were trending and assessed the sentiment of daily tweets. We classified the sentiment for each tweet as positive, negative, or neutral based on a sentiment-analysis technique described by Zhang et al.[6] As Twitter does not require users provide their geographical location, we were unable to isolate the tweets coming from within Egypt itself.

Figure 1a lists the volume of tweets per day, Figure 1b lists cumulative tweets, and Table 1 outlines some of the revolution's major events, along with the number of tweets and the average sentiment for four Egypt-related Twitter topics. Note that we collected this data in real time in our lab for all trending topics available from Twitter.

The first day the topics trended coincided with the January 25 "Day of Rage" protest in Cairo's Tahrir Square, with the Cairo topic showing mostly negative sentiment at the time. On January 27, the Egyptian government began limiting access to social media and the Internet within the country's national network. Later, many of the tweets were likely tweeted from out-

side Egypt. Correspondingly, "egypt" alone trends on January 27, while "egypt," "egyptians," and "cairo" accounted for most of the traffic during the government-imposed Internet blackout. On January 29, Mubarak fired his cabinet and installed Omar Suleiman as vice president, coinciding with a peak in the number of tweets on the topics of "egypt" and "cairo" and a peak in sentiment on "egyptians." Immediately afterward, the sentiment value for most topics decreased, possibly due to decreased interest outside Egypt or the realization that Mubarak would not willingly step aside as president. The Egyptian government ended its blockade of Internet traffic February 2, coinciding with a significant spike in number of tweets and negative sentiment on the topic of Tahrir. While most Egypt-related topics stopped trending February 4, termed by the protesters the "Day of Departure," the protests continued until Mubarak's eventual resignation February 11.[1] Meanwhile,

tweets on "hosni_mubarak" became progressively more negative, so eventually, by February 6, 85% of the tweets on the topic were negative.

Influencers and Tweets

For a clearer view of Twitter users and tweets on Egypt, we looked at influential users and tweets on several Egypt-related topics, ranking their influence based on their follower relationships and on how tweets propagated through the follower network.

For each day the topics in Figure 1 were trending, we identified the top 10 most influential users worldwide. Table 2 lists the people and organizations that influenced these topics most, along with a breakdown of influential users in Figure 2a. Many users at the top of these lists belonged to major news organizations, including Al Jazeera and CNN, though New York-based Egyptian journalist Mona Eltahawy[b] established herself as a trusted

b http://www.monaeltahawy.com

PHOTOGRAPH BY JONATHAN RASHAD

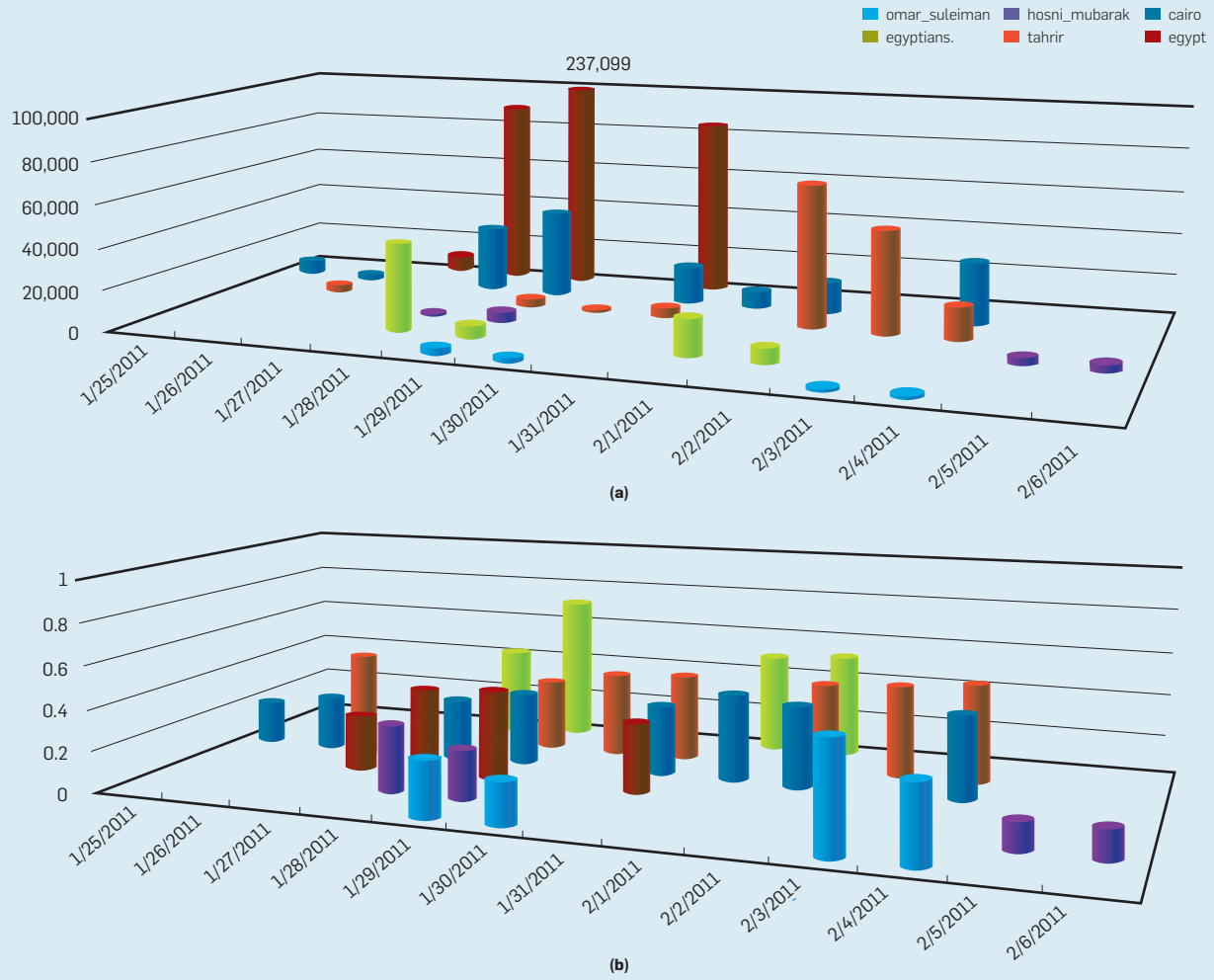
Table 1. Events during the revolution reflecting tweet volume compared with sentiment (per day and cumulatively); positive tweet fraction is the daily or cumulative number of tweets with positive sentiment divided by number of tweets with positive or negative sentiment. Only dates corresponding to major events are shown; for tweets and sentiment for each day the topics were trending, see Figure 1; N/T indicates the topic did not trend that day (daily) or did not yet trend (cumulative).

			Number of tweets		Positive tweet fraction	
Date	Major Events	Topic	Daily	Cum.	Daily	Cum.
Jan. 25	"Day of Rage" protests in Cairo signal start of major changes in Egypt	cairo	6,892	6,892	0.207	0.207
		egypt	N/T	N/T	N/T	N/T
		hosni_mubarak	N/T	N/T	N/T	N/T
		tahrir	N/T	N/T	N/T	N/T
Jan. 27	Egyptian government begins limiting Internet access in Egypt	cairo	N/T	39,752	N/T	0.219
		egypt	8,395	8,395	0.267	0.267
		hosni_mubarak	N/T	N/T	N/T	N/T
		tahrir	N/T	3,053	N/T	0.413
Jan. 29	Mubarak dismisses his cabinet and appoints Omar Suleiman as Vice President of Egypt	cairo	41,049	80,801	0.354	0.315
		egypt	237,099	334,612	0.448	0.435
		hosni_mubarak	4,670	5,759	0.236	0.251
		tahrir	3,776	6,829	0.209	0.376
Feb. 2	Egyptian government ends blocking of Internet access	cairo	15,341	122,575	0.406	0.338
		egypt	N/T	418,855	N/T	0.418
		hosni_mubarak	N/T	5,759	N/T	0.251
		tahrir	68,389	80,616	0.428	0.424
Feb. 6	Egypt-related topics stop trending on Twitter; Mubarak resigns Feb. 11	cairo	N/T	152,463	N/T	0.355
		egypt	N/T	418,855	N/T	0.418
		hosni_mubarak	3,656	12,719	0.145	0.191
		tahrir	N/T	146,439	N/T	0.44

Table 2. Top five most influential individuals and organizations tweeting on the Egyptian revolution, along with number of times each appeared in the daily list of top influencers.

Name	Count	Description	User names (individual count)
Al Jazeera	32	Arabic news channel based in Doha, Qatar; name translates as "the island"	AJEnglish (11), Dima_AlJazeera (6), AJGaza (5), AlanFisher (3), FatimaNaib (3), mohamed (2), SherineT (2)
CNN	17	American news network founded by Ted Turner	bencnn (4), cnnbrk (4), CNNLive (3), natlsecuritycnn (3), vhernandezcnn (3)
Mona Eltahawy	14	Egyptian journalist and public figure	monaeltahawy (14)
Reuters	14	News agency with headquarters in London	Reuters (10), JimPathokoukis (4)
The Nation	10	Left-leaning weekly magazine with headquarters in New York	jeremyscahill (6), thenation (4)

Figure 1. Tweet volume (a) and sentiment (b) per day for six topics related to the Egyptian revolution; tweet sentiment was measured in terms of "positive sentiment fraction," or the fraction of tweets with positive sentiment.

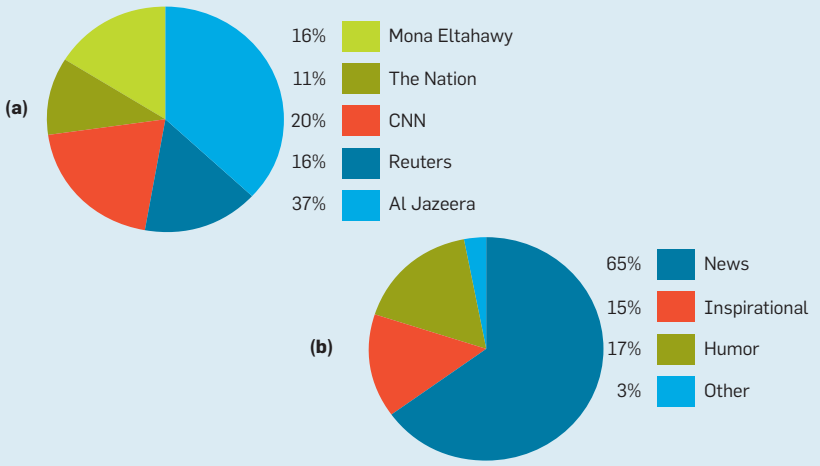


source, with influence across a number of topics, and the most influential single user worldwide on Egyptian topics during the revolution.

We also analyzed the most influential tweets to characterize the simultaneous conversations that were taking place. We define influential tweets as those that were frequently "retweeted" or repeated by users to spread a tweet's message.

We collected the most frequent retweets for each day the topics in Figure 1 were trending. In order to develop an understanding of what these influential tweets were saying, we sampled the 10 most influential and categorized them manually. We found most fit into three main categories: news (updates on major events), inspirational (human-interest stories), and humor. Table 3 lists popular tweets from each category,

Figure 2. Breakdown of the most influential users and most frequently retweeted tweets late January to early February 2011 worldwide; influential users in (a) represent 43% of the top 10 influencers for the days the Egyptian topics were trending.



and Figure 2 outlines the distribution of these "most popular" tweets.

Almost two-thirds of the most popular tweets on Egypt-related topics were news, communicating important updates on unfolding events, while inspirational and humor tweets each represented 15%–20% of the popular tweets. The topic "egyptians" was the source of more than half of the inspirational tweets.

Not Another Super Bowl

To see how Egypt compared to other trending Twitter topics, we chose several related and unrelated topics that trended at the same time across several categories in Figure 3. We chose them by scanning the topics that trended during the same period and selecting those covering four categories—politics, entertainment, sports, and tech/business—to include other popular topics during the same period. For each one, we collected data on the network of follower relationships between users tweeting on a particular topic and the sentiment expressed in the tweets on the topic. We analyzed aspects of the data for both Egyptian and non-Egyptian topics to determine what characteristics of Twitter users and their tweets separated the Egyptian revolution from other Twitter topics (see Figure 4).

We found that while users tweeting on Egypt-related topics had more followers tweeting on the same topic, the clustering coefficient of the follower networks was lower. The clustering coefficient, or "fraction of transitive triples," is defined in graph theory as the fraction of follower relationships with users in common forming a triangle; for example, if user A follows user B and user B follows (or is followed by) user C, how often does A follow C or vice versa? Follower networks for the Egyptian topics were essentially less "clique-ish," forming fewer tight groups despite having a similar number of follower relationships. This fact suggests the users tweeting on Egypt-related topics came from several different social groups both inside and outside Egypt.

We also found users tweeted more on average on Egypt-related topics, though their tweets were much more likely to be retweets, or echoes of tweets by other, presumably more authoritative, users, and more likely to express negative sentiment. With so many tweets on Egypt-related topics expressed in negative terms, the average user sentiment for Egypt-related networks (number of positive tweets minus negative tweets) was negative, notably unlike the other categories we considered.

Pulse of a Revolution

Our Web site (http://www.pulseofthetweeters.com) has been collecting information on all trending topics on Twitter since Feb. 2010. Pulse of the Tweeters (see the figure here) allows users to search trending topics or just scan for topics that are similar to reveal how tweeters feel about particular topics and provide a breakdown of positive vs. negative sentiment concerning them, along with total number of tweets. Users also browse tweeters who have posted on a topic, sorting by influence, number of followers, and number of tweets. They can also find tweeters' home pages and most recent tweets on the topic by clicking on user names.

We gleaned the information on the site through Twitter's representational state transfer API to collect daily trending topics, as well as information on all tweets related to the topic. We continually analyze tweet and user data for sentiment and influence using techniques we have developed at the Center for Ultra-scale Computing and Information Science at Northwestern (Alok Choudhary, director).

Pulse of the Tweeters.

Conclusion

We took an in-depth look at the ways the Egyptian revolution, late Janu-

Table 3. Example popular tweets.

Category	Example tweets
News	**RT @BreakingNews**: 3 private jets leave Cairo airport under heavy security; #Egypt parliament speaker to make major announcement - NBC #Jan25
	RT @cnnbrk: Clarification: Key members resign posts in Egypt's ruling party. Hosni Mubarak remains head of party & president.
Inspirational	**RT @paulocoelho**: To people in Tahrir Square: We are all Egyptians today/Hoje ns somos todos egpcios
	RT @NickKristof: Here's Nawal el-Saddawi, famed women's leader. She's 80-but told me she will sleep in #Tahrir. http://twitpic.com/3w5bsc
Humor	**RT @TheOnion**: Hosni Mubarak Reaches Out To Twitter Followers For Ideas On How To Keep Regime Intact #Egypt #jan25
	RT @pourmecoffee: Egyptians camped out in Cairo. For Americans, think of it as like the release of new Apple product if Apple made freedom.

ary to early February 2011, was discussed and debated as they unfolded through Twitter. The discussion was marked by strong negative sentiment less cohesive than for other types of Twitter topics. Moreover, a significant portion of the discussion reflected broadcast news of ongoing events, with most influential users and tweets delivering news and a large proportion of messages as a way to repost this news for others within, as well as outside, Egypt.

For more on other popular Twitter topics, including sentiment analysis and influential users, see our Web site http://www.pulseofthetweeters.com.

Figure 3. Twitter topics by category.

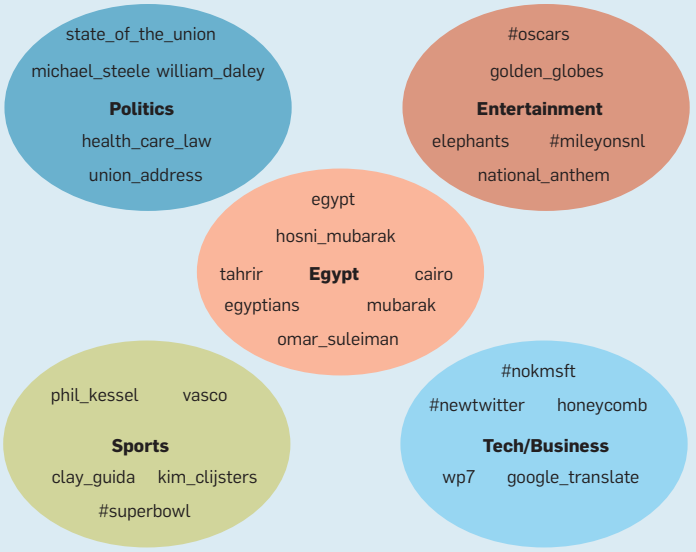
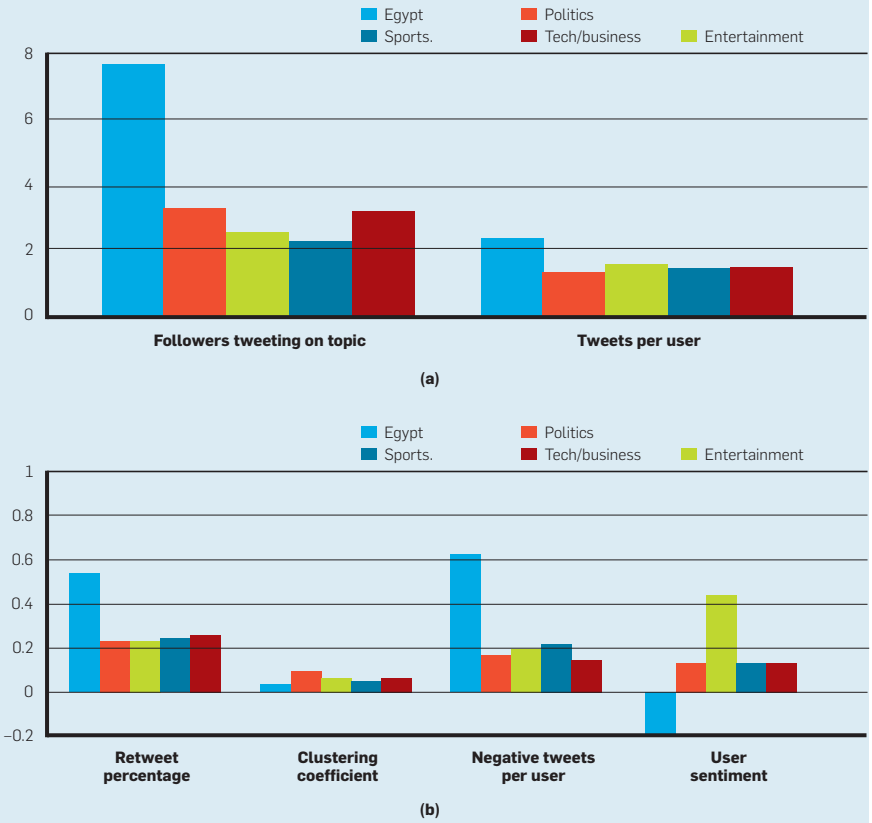


Figure 4. Twitter networks covering Egypt compared with other Twitter topics.



Acknowledgments

This work is supported in part by National Science Foundation awards CCF-0621443, OCI-0724599, CCF-0833131, CNS-0830927, IIS-0905205, OCI-0956311, CCF-0938000, CCF-1043085, CCF-1029166, and OCI-1144061 and in part by U.S. Department of Energy grants DE-FC02-07ER25808, DE-FG02-08ER25848, DE-SC0001283, DE-SC0005309, DE-SC0005340, and DE-SC0007456.

References

1. Al Jazeera. Timeline: Egypt's revolution (Feb. 14, 2011); http://english.aljazeera.net/news/middleeast/2011/01/201112515334871490.html
2. Beaumont, P. The truth about Twitter, Facebook, and the uprisings in the Arab world. *The Guardian* (Feb. 25, 2011); http://www.guardian.co.uk/world/2011/feb/25/twitter-facebook-uprisings-arab-libya
3. Ghonim, W. Interviewed by Harry Smith. Wael Ghonim and Egypt's new age revolution. *60 Minutes* (Feb. 13, 2011); http://www.cbsnews.com/stories/2011/02/13/60minutes/main20031701.shtml?tag=contentMain;contentBody
4. Gladwell, M. Does Egypt need Twitter? *The New Yorker* (Feb. 2, 2011); http://www.newyorker.com/online/blogs/newsdesk/2011/02/does-egypt-need-twitter.html
5. Twitter Blog. To trend or not to trend... (Dec. 8, 2010); http://blog.twitter.com/2010/12/to-trend-or-not-to-trend.html
6. Zhang, K., Cheng, Y. Liao, W.-K., and Choudhary, A. Mining millions of reviews: A technique to rank products based on importance of reviews. In *Proceedings of the 10th International Conference on Entertainment Computing* (Liverpool, U.K., Aug. 3–5, 2011).

Alok Choudhary (choudhar@eecs.northwestern.edu) is the John G. Searle Professor of Electrical Engineering and Computer Science at Northwestern University, Evanston, IL, and a fellow of the AAAS, ACM, and IEEE.

William Hendrix (whendrix@northwestern.edu) is a postdoctoral research associate in the Department of Electrical Engineering and Computer Science at Northwestern University, Evanston, IL.

Kathy Lee (kml649@eecs.northwestern.edu) is a graduate student in the Department of Electrical Engineering and Computer Science at Northwestern University, Evanston, IL.

Diana Palsetia (dianapalsetia2015@u.northwestern.edu) is a graduate student in the Department of Electrical Engineering and Computer Science at Northwestern University, Evanston, IL.

Wei-keng Liao (wkliao@eecs.northwestern.edu) is a research associate professor in the Department of Electrical Engineering and Computer Science at Northwestern University, Evanston, IL.

DOI:10.1145/2160718.2160737

Applied to almost 3,500 articles it reveals computing's (and *Communications*') culture, identity, and evolution.

BY DANIEL S. SOPER AND OFIR TUREL

An *n*-Gram Analysis of *Communications* 2000–2010

GAINING A DEEP, tacit understanding of an institution's identity is a challenge, especially when the institution has a long history, large geographic footprint, or regular turnover among its employees or members. Though the cultural themes and historical concepts that have shaped an institution may be embedded in its archived documents, the volume of this material may be too much for institutional decision makers to grasp. But without a solid, detailed understanding of the institution's identity, how can they expect to make fully informed decisions on behalf of the institution?

Many scientific disciplines suffer from this phenomenon, with the problem especially pronounced in computing.[1] A constant influx of new technologies, buzzwords, and trends produces an environment marked by rapid change, with the resulting instability making it difficult to establish a stable identity.[2,3] As archived institutional artifacts, articles in journals and other media reflect and chronicle the field's evolving identity. Unfortunately, humans are simply unable to digest it all. However, by leveraging a computational method known as *n*-gram analysis, it may be possible for computer scientists and scholars alike to unlock the secrets within these vast collections and gain insight that might otherwise be lost. If the history and identity of computing are encoded on the pages of journals, systematically analyzing them is likely to yield a better understanding of where the field has been and where it might

» key insights

- ***N*-gram analysis is a simple but extremely useful method of extracting knowledge about an institution's culture and identity from its archived historical documents.**
- ***N*-gram analysis can reveal surprising and long-hidden trends that show how an institution has evolved.**
- **Knowledge gained from *n*-gram analyses can substantially improve managerial decision making.**

be headed. After all, "We are what we write, we are what we read, and we are what we make of what we read."[4]

Here, we address the identity problem by prescribing the same medicine for ourselves—technology and algorithms—we often prescribe for others. We present a culturomic analysis[5] of *Communications* showing how natural language processing can be used to quantitatively explore the identity and culture of an institution over time, inspired by the *n*-gram project released in 2010 by Google labs (http://ngrams.googlelabs.com). In natural language processing, an *n*-gram can be viewed as a sequence of words of length *n* extracted from a larger sequence of words.[12] Google's project allows for quantitative study of cultural trends based on combinations of words and phrases (*n*-grams) appearing in a corpus of more than five million books published as early as the 15th century. The central theory behind the project is that when taken together, the words appearing in a collection of books reveal something about human culture at the time the books were written. Analyzing these words computationally makes it possible to study cultural evolution over time.

Applying a similar analytical approach to the articles in *Communications* would allow us to better understand the culture, identity, and evolution of computing. With a view toward portraying its value for institutional-identity data mining, we present several findings that emerged from our *n*-gram analysis of *Communications*. Though our effort here focuses on a single scientific journal, we hope it engenders future studies that evaluate and compare the cultures and identities of a basket of institutions, providing a deeper understanding of their history and evolution over time.

Figure 1. Structural changes in *Communications* from 2000 to 2010.

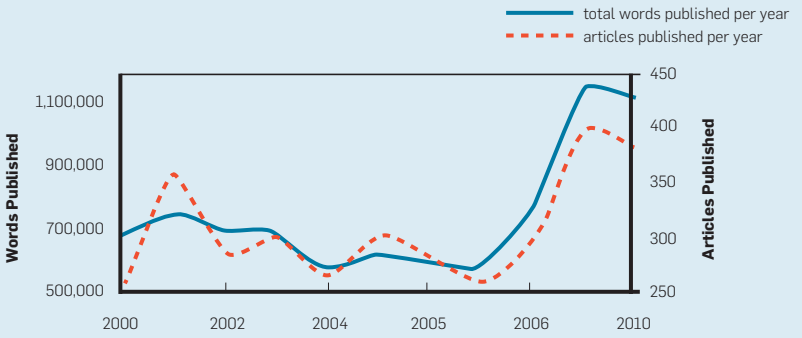


Table 1. Major trends (growth and decline) in terms published in *Communications* from 2000 to 2010.

Growing in Popularity		Declining in Popularity	
Term	**% Change**	**Term**	**% Change**
Google	+18,151%	perceptual	-9,459%
queue	+11,160%	wrapper	-9,459%
cloud	+10,833%	biometrics	-9,295%
VM	+6,439%	CORBA	-9,295%
IT professionals	+5,703%	telemedicine	-8,969%
parity	+5,151%	disintermediation	-7,991%
workload	+5,151%	multimedia	-7,665%
venue	+4,844%	transcription	-7,502%
polynomial time	+4,599%	personalization	-6,425%
DRAM	+4,292%	user profile	-6,197%
test cases	+4,231%	e-commerce	-5,382%
theorem	+4,016%	e-business	-4,281%
OOP	+3,741%	satellites	-4,240%
science and engineering	+3,557%	AOL	-4,158%
emulator	+3,434%	OCR	-4,077%

Method

To appreciate how the identity of *Communications* has evolved, we first constructed a corpus of the complete text of every article it published from 2000 to 2010.[a] We also collected metadata for all these articles, including title, author(s), year published, volume, and issue. In total, our corpus contained 3,367 articles comprising more than 8.1 million words. To put this in perspective, consider that if you were to spend 40 hours per week reading *Communications*, you would need more than four months to read every article published from 2000 to 2010.

With our corpus complete, we next constructed a software system to tokenize, or split the text of each article into a series of *n*-grams. For example, René Descartes' famous phrase "cogito ergo sum"[10] can be subdivided into three *1*-grams (cogito, ergo, and sum), two *2*-grams (cogito ergo, and ergo sum), and one *3*-gram (cogito ergo sum). As this illustrates, the number of *n*-grams that could potentially be extracted from a large corpus of text greatly exceeds the number of words in the corpus itself. This situation has serious scaling and performance implications for a corpus with millions of words, so to avoid them we limited our analysis to include *n*-grams with a maximum length of $n = 4$.

To address the challenges of punctuation, we adopted the same method used by the developers of Google's *n*-gram project for digitized books,[13] treating most punctuation marks as separate words during the *n*-gram construction process. The phrase "Elementary, my dear Watson" would be tokenized as, say, five words:

Elementary , my dear Watson

a Only the text of each article was included in the database; excluded was trailing matter (such as acknowledgements and references).

PHOTOGRAPH BY ALICIA KUBISTA

Notable exceptions to this rule include currency symbols, decimal components of numbers, and apostrophes indicating possessive case. A term like "$5.95" would be treated as a *1*-gram, while "Euler's constant" would be treated as a *2*-gram. For a more general rule for tokenization, developers might consider splitting tokens that contain a special character only if the character is adjacent to whitespace or a linefeed.

We ignored case in the construction of our *n*-gram corpus. Had we retained case sensitivity, a term (such as "computer science") would have been treated as distinct from the term "Computer Science." While ignoring case vastly reduced the potential number of *n*-grams the system might encounter, it also involved a few negative implications for search specificity. Without case sensitivity, the term "IT" (for information technology) would be considered identical to, say, the word "it." Despite this drawback, we concluded that the overall benefit of ignoring case outweighed its cost.

Broadly speaking, our analysis of how *Communications* evolved from 2000 to 2010 was predicated on the idea that the level of importance or relevance of a particular concept is reflected in how often the concept is mentioned over time. We therefore had to compute the frequency with which every *n*-gram in the corpus appeared in *Communications* during each year of the analysis. For example, if the *n*-gram "e-commerce" was mentioned 273 times in 2000 but only 23 times in 2010,[b] we might infer the concept of e-commerce had become less important in *Communications* over time. However, direct frequency comparisons can be deceiving because they do not account for potential growth or decline in the number of words *Communications* published over time. It was therefore necessary for us to calculate relative frequencies for each *n*-gram. We thus divided *n*-gram frequencies for each year by the total number of words appearing in the corpus during that year in order to produce a standardized measure of frequency that would allow valid comparisons between *n*-grams from year to year.[13] The standardized frequency values resulting from this process indicated how often a particular *n*-gram appeared in *Communications* during a particular year relative to the total quantity of text published in it that year. Standardized frequencies are not, of course, the only means *n*-grams can be compared over time. Indeed, other, more sophisticated information-theoretic measures (such as entropy and cross-entropy) can also be used for this purpose.

The result was a vast database containing more than 160 million *n*-grams and their associated years and standardized frequencies. From it we then selected the one million unique *n*-grams exhibiting the most absolute change over time, reasoning that the frequencies of less-interesting

b These were the actual *n*-gram frequencies for "e-commerce" during 2000 and 2010.

n-grams (such as "how" and "the") would remain relatively stable from year to year. Alternatively, a linguistically informed approach to reducing the size of the search space could be done through part-of-speech (POS) tagging, such that only those *n*-grams identified as noun phrases would be added to the dataset. However, state-of-the-art POS taggers are only about 95% accurate, implying that many interesting *n*-grams could have been overlooked had we taken this approach. Nevertheless, POS-based *n*-gram identification remains an option, especially when the corpus to be analyzed is extremely large.

Finally, we constructed a Web-based system to enable us to query, graph, and explore our *Communications* *n*-gram database, plot and analyze multiple *n*-grams simultaneously, and combine related search terms into a single result. For example, the search phrase "cellphone+cellphones, smartphone+smartphones" would produce a graph containing two lines, one representing the combined frequencies of the terms "cellphone" and "cellphones" over time, the other representing the combined frequencies of the terms "smartphone" and "smartphones" over time. To try out our *Communications* *n*-gram tool, see http://www.invivo.co/ngrams/cacm.aspx.

If, in the aggregate, *Communications* reflects what is happening in computing, then perhaps existing industry standards should be refined to more closely approximate real-world practice.

Findings

Though we cannot expect to identify all ways the computing field has evolved in a single article, we do aim to provide a point of embarkation for future research. Beginning with big-picture considerations, we are confident saying the structure and content of *Communications* evolved significantly from 2000 to 2010. An analysis of our metadata revealed several striking, large-scale structural changes from 2000 to 2010. Over that time, *Communications* published an average of 306 articles per year, each containing an average of about 2,400 words. However, these averages obscured underlying trends showing that both the number of articles published per year and the average length of each article grew significantly, especially in more recent years. These trends (see Figure 1) imply *Communications* was providing more value to its readers than in it had previously, since more recent issues contained more articles and words than earlier issues.

Changing Focus

Continuing our investigation, we next extracted the 15 terms that experienced the most growth or decline in popularity in *Communications* from 2000 to 2010 (see Table 1). We hope you find at least a few trends in the table that are unexpected or interesting; indeed finding them is a primary goal of large-scale data mining. For us, we noticed that several of the terms showing the most growth were related to science and technology, while several of the declining terms were related to business and management. But is this observation anecdotal or a broader pattern in *Communications*? To answer, and to show how *n*-gram analyses can be integrated with more traditional analytic techniques, we conducted an interaction analysis comparing the *n*-gram frequencies for terms related to business and management with those related to science and technology. We identified related terms using Thinkmap's Visual Thesaurus software (http://www.visualthesaurus.com), which is specifically designed for this purpose. We then extracted *n*-gram frequencies for the resulting lists of related terms, using these values to conduct our interaction analysis (see Figure 2). As shown in the figure, the average frequency of business- and management-related terms declined steadily from 2000 to 2010, while science- and technology-related terms became more common. Our interaction analysis indicated that the observed disparity was highly significant ($t_{[5052]} = 2.834, p < 0.01$), providing statistical evidence of *Communications*' evolving identity.

Changes in Style

The style of the writing in *Communications* also evolved from 2000 to 2010. Authors seemed to be transitioning from the traditional academic style of writing, adopting instead a less-formal, more personal voice. Evidence of this change can be seen in the increasing use of words that refer directly to an article's author(s) (such as "I" +143% and "we" +137%) and in the increased frequency authors spoke

directly to their readers through such words as "you" (+222%) and "your" (+205%). Interesting to note is while the content of *Communications* became more scientific and technical, it was presented in a way that was less scientific and technical. A possible effect of this change is that the content of *Communications* became more accessible to a wider, more diverse audience. We, too, found ourselves adopting this more personal style when writing this article.

Our *n*-gram analysis also revealed changes in *Communications*' use of gender-related terms from 2000 to 2010. On average, masculine pronouns (such as "he," "his," and "him") appeared 277% more often than feminine pronouns (such as "she," "hers," and "her"). Moreover, the gap widened from 190% in 2000 to more than 290% in 2010. One possible explanation is the gender gap between male and female computing professionals also grew and was wider in 2010 than it was at any time in the previous 25 years.[14]

World Events

When major events occur somewhere in the world, how and to what extent does *Communications* respond? Do such events influence the computing profession? To answer, we conducted an *n*-gram analysis that included three types of world events: a natural disaster (Hurricane Katrina), a terrorist attack (9/11), and a health crisis (2003 SARS outbreak); Figure 3 shows a common pattern with respect to how *Communications* reacted to such events. Specifically, a major world event would first appear in *Communications* shortly after it occured, with discussion of the event—measured by how often it was mentioned—growing quickly for a short time thereafter. This finding indicates *Communications* is not insulated from major world events but rather embraces them and actively contributes to their discussion in the global forum. After a few years, however, *Communications*' interest in a major event would decline sharply. Nevertheless, even after experiencing this precipitous drop, major world events still tend to be mentioned occasionally in *Communications* over the following years.

Systems Development Life Cycle

The systems development life cycle (SDLC) is one of the most ubiquitous and enduring components of the computing profession. With respect to the four principal phases of SDLC—planning, analysis, design, and implementation—industry standards generally recommend that approximately 15% of time and resources budgeted for a systems development project go to planning, 20% to analysis, 35% to design, and 30% to implementation.[9] To what extent, then, does discussion of them in *Communications* mirror the level of interest recommended by industry standards? To answer, we again turned to *n*-gram analysis to compute the average frequency with which each SDLC phase was mentioned in *Communications* from 2000 to 2010. Dividing the value for each phase by the overall sum of the average frequencies yielded the relative average frequencies for *Communications* in Table 2.

If we accept industry standards as canonical, then the values in the table suggest *Communications* underemphasized the SDLC planning and implementation phases while overemphasizing analysis and design. However, *Communications* would seem to agree

Figure 2. *Communications*' changing focus.

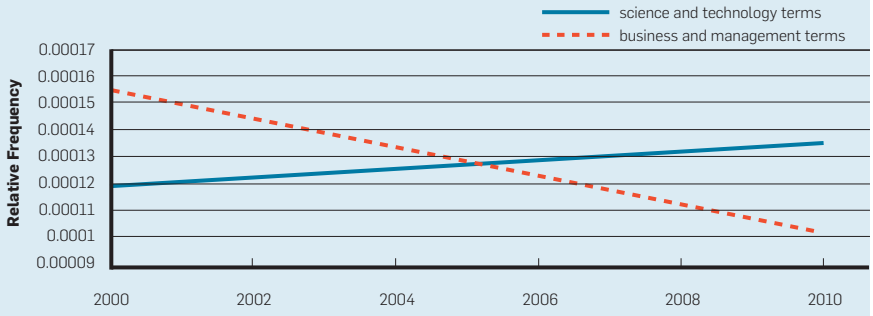


Figure 3. *Communications*' response to major world events.

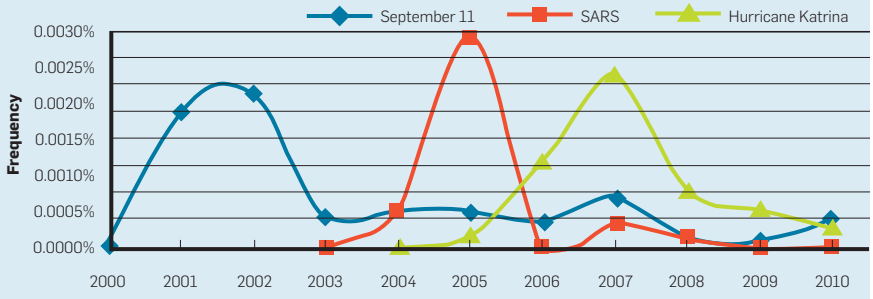


Table 2. Interest in the phases of SDLC in *Communications* compared to industry standards.

	Level of Interest	
SDLC Phase	**Industry Standard**	**Communications**
Planning	15%	8%
Analysis	20%	29%
Design	35%	46%
Implementation	30%	17%

in principle with industry standards that design deserves the most attention and planning the least. The overall discrepancies between these two sources also raise another interesting point: If, in the aggregate, *Communications* reflects what is happening in computing, then perhaps existing industry standards should be refined to more closely approximate real-world practice.

Special Sections

From 2000 to 2010, *Communications* featured special sections that included a number of articles on a specific topic. But did these sections engender long-term interest in the topic being addressed or was the effect more fleeting? To answer, we selected three topics that were the focus of special sections: spyware,[6] digital rights management,[7] and democracy.[8] Our only criterion in selecting them was that they were published closer to 2000 than to 2010, making it easier to identify long-term effects (see Figure 4).

The figure shows special sections generated a spike of interest in the topic during the calendar year the section was published. This interest was sustained for a short time before declining rapidly, eventually returning to a near-zero steady state. Although they can be expected to increase the visibility of a topic in the short-term, special sections did not seem to engender lasting interest in the topics they addressed, at least in *Communications*. Whether this observation holds for other journals is an interesting empirical question but cannot be answered within the scope of this article.

Technology Preferences

If the articles published in *Communications* truly reflect the state of the art in computing, then a *Communications* *n*-gram analysis focused on specific technologies should help reveal differences between the technological preferences of computing professionals and those of the general public. To this end, we compared *Communications* *n*-gram frequencies for different Web browsers, operating systems, and search engines in 2010 against the market shares of the same products among the general public during the same year.[15] The results, which speak to the comparative popularity of different technologies among computing professionals and the general public, are outlined in Figure 5.

The figure indicates that the preferences of computing professionals with respect to Web browsers and operating systems differed markedly from those of the general public. Specifically, there appeared to be more diversity among the preferences of computing professionals regarding the technologies, while usage patterns among the general public were much more homogeneous. One explanation might be that computing professionals simply have a more nuanced understanding of the advantages and disadvantages of the various technology options available to them and are more likely to orient their preferences toward the technologies that serve their needs best. If this is indeed the case, then the search engine results in the figure represent a powerful testament to the perceived superiority of Google's search technology.

Technology Life Cycles

Finally, we wondered whether *Communications*' interest in a particular technology proxies the location of the technology along its product life cycle. To answer, we conducted an *n*-gram analysis of three mass-market commercial Apple products—iPod, iPhone, and iPad—that were arguably at different points in their respective life cycles (see Figure 6).

As shown in the figure, the frequency a particular product appeared in *Communications* spoke to the location of the technology along its own unique trajectory. For example, interest in the iPod in *Communications* grew sharply from 2004 to 2005, corresponding to a 500% increase in sales during that pe-

Figure 4. Effect of special sections on long-term topic interest.

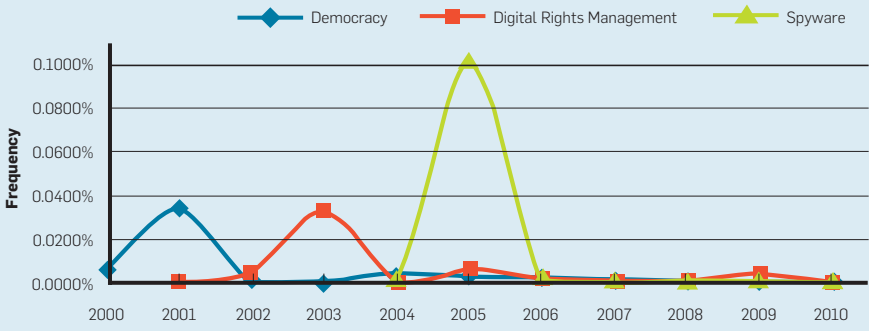


Figure 5. Technology preferences compared.

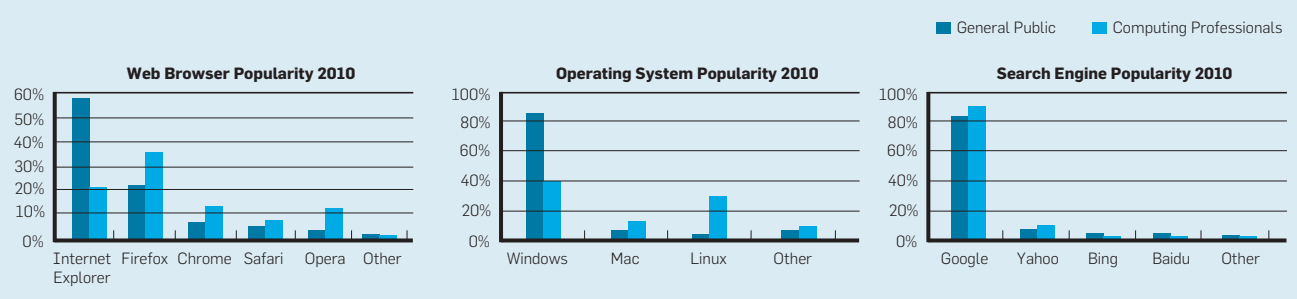
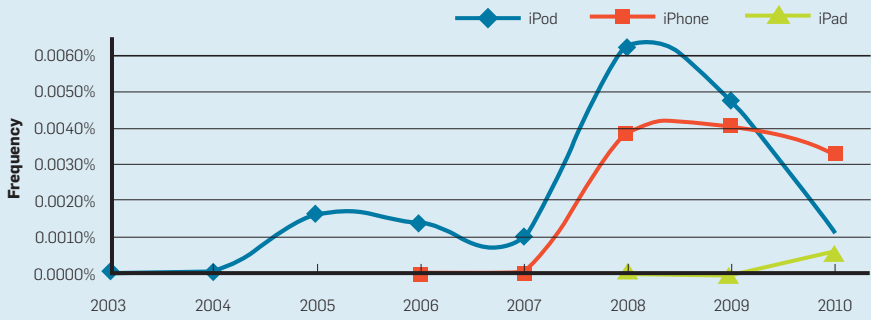


Figure 6. Interest in mass-market commercial Apple products.



riod.[11] However, after the sudden enormous popularity of the iPhone in 2007 and 2008, interest in the iPod began to wane, as the ongoing incorporation of digital music capabilities into smartphones (such as the iPhone) made the iPod an increasingly redundant technology. Beginning around 2009 interest in the iPhone also began to decline among computing professionals, while at the same time interest in Apple's most current innovation—iPad—continued to increase. Together, these results demonstrate the viability of using *n*-gram analyses to help study technological evolution over time.

Conclusion

Findings of a 2010 U.S. Department of Labor longitudinal study indicate college-educated workers hold an average of 11 different jobs by the time they are 44.[16] This surprising statistic has important ramifications for all organizations, since it implies skilled workers (such as managers) will, on average, hold their positions of leadership and productivity for only a few years before moving on. Reflecting such short tenure, managers inevitably struggle to develop the sort of deep, tacit understanding of their organizations that is critical to strategic decision making. If such knowledge can be found within the archived document artifacts produced by an institution over time, then *n*-gram analyses like those we have presented here may prove invaluable for sifting through the content of these vast collections of archival documents, as well as contribute to improvements in managerial decision making.

With respect to our conclusions, we found more recent issues of *Communications* contained substantially more content than earlier issues. The nature of this content is also changing, as articles published in *Communications* are trending away from managerial and business subjects to focus instead on more technical and computational subjects. Despite this trend, the writing style in *Communications* became less formal and more personal over time, helping it reach a wider audience.

In addition to these structural changes, we also pursued an assortment of analyses that, together, demonstrate the potential of the *n*-gram method for institutional data mining. We invite you to conduct your own *Communications* analyses using our *n*-gram tool at http://www.invivo.co/ngrams/cacm.aspx. Broadly speaking, *n*-grams are a powerful tool for gaining insight into textual data that might otherwise go unnoticed. Though the analyses we discussed here target a single publication, the *n*-gram method itself is suitable for a range of analytic situations, including virtually any large corpus of text data. For example, historical documents could be analyzed to identify long-hidden trends, fads, and modes of thought. Blogs or tweets could be analyzed to produce a near-real-time snapshot of the global consciousness. And software source code could be analyzed for style and/or efficiency. Combined with optical character recognition, *n*-grams could be leveraged to sift through mountains of archived paper documents. Indeed, the possibilities for *n*-gram analyses are almost limitless.

Finally, we hope this article serves as evidence for why scholarly journals should not be viewed as immutable objects but as living entities evolving and responding to their environments. To this end, analyses like ours should be conducted periodically to codify and chronicle a publication's history, helping refine its identity over time. This is especially important with long-lived journals (such as *Communications*) that serve as standard-bearers for their respective disciplines. When applied to a broad portfolio of publications, such analyses would help readers, authors, advertisers, and editors alike better understand journals more objectively and perhaps even glimpse what the future holds for disciplines like computer science. C

References

1. Agarwal, R. and Lucas, J.H.C. The information systems identity crisis: Focusing on high-visibility and high-impact research. *MIS Quarterly 29*, 3 (Sept. 2005) 381–398.
2. Benbasat, I. and Zmud, R.W. The identity crisis within the IS discipline: Defining and communicating the discipline's core properties. *MIS Quarterly 27*, 2 (June 2003) 183–194.
3. Bhattacherjee, A. and Gill, G. Whom are we informing? Issues and recommendations for MIS research from an informing sciences perspective. *MIS Quarterly 33*, 2 (June 2009), 217–235.
4. Bloomer, M., Hodkinson, P., and Billett, S. The significance of ontogeny and habitus in constructing theories of learning. *Studies in Continuing Education 26*, 1 (Mar. 2004), 19–43.
5. Bohannon, J. Google Books, Wikipedia, and the future of culturomics. *Science 331*, 6014 (Jan. 14, 2011), 135.
6. Crawford, D. Editorial pointers. *Commun. ACM 48*, 8 (Aug. 2005), 5.
7. Crawford, D. Editorial pointers. *Commun. ACM 46*, 4 (Apr. 2003), 5.
8. Crawford, D. Editorial pointers. *Commun. ACM 44*, 1 (Jan. 2001), 5.
9. Dennis, A., Wixom, B.H., and Roth, R.M. *Systems Analysis and Design, Fourth Edition*. John Wiley & Sons, Inc., Hoboken, NJ, 2009.
10. Descartes, R. *Principia Philosophiae*. Kluwer Academic, Norwell, MA, 1984, originally published 1644.
11. Dowling, S. Apple Reports First Quarter Results: Quarterly Revenue & Net Income Highest in Apple's History. Apple Computer, Cupertino, CA, 2005; http://blog.ttnet.net/resources/2005/01/13/apple-reports-first-quarter-results-quarterly-revenue-net-income-highest-in-apples-history-20050113
12. Manning, C.D. and Schütze, H. Foundations of Statistical Natural Language Processing. MIT Press, Cambridge, MA, 1999.
13. Michel, J.-B., Shen, Y.K., Aiden, A.P., Veres, A., Gray, M.K., Team, T.G.B., Pickett, J.P., Hoiberg, D., Clancy, D., Norvig, P., Orwant, J., Pinker, S., Nowak, M.A., and Aiden, E.L. Quantitative analysis of culture using millions of digitized books. *Science 331*, 6014 (Jan. 14, 2011), 176.
14. Misa, T.J. *Gender Codes: Why Women Are Leaving Computing*. John Wiley & Sons, Inc., Hoboken, NJ, 2010.
15. NetMarketShare. Usage Share Statistics for Internet Technologies. NetApplications.com, Aliso Viejo, CA, 2010; http://netmarketshare.com/
16. U.S. Bureau of Labor Statistics. *Number of Jobs Held, Labor Market Activity, and Earnings Growth Among the Youngest Baby Boomers: Results From a Longitudinal Survey*. U.S. Department of Labor, Washington, D.C., 2010; http://www.bls.gov/news.release/nlsoy.nr0.htm

Daniel S. Soper (dsoper@fullerton.edu) is an assistant professor in the Information Systems and Decision Sciences Department of the Mihaylo College of Business and Economics at California State University, Fullerton.

Ofir Turel (oturel@fullerton.edu) is a professor in the Information Systems and Decision Sciences Department of the Mihaylo College of Business and Economics at California State University, Fullerton.

DOI:10.1145/2160718.2160738

Examining tools that provide valuable insight about molecular components within a cell.

BY NIR ATIAS AND RODED SHARAN

Comparative Analysis of Protein Networks: Hard Problems, Practical Solutions

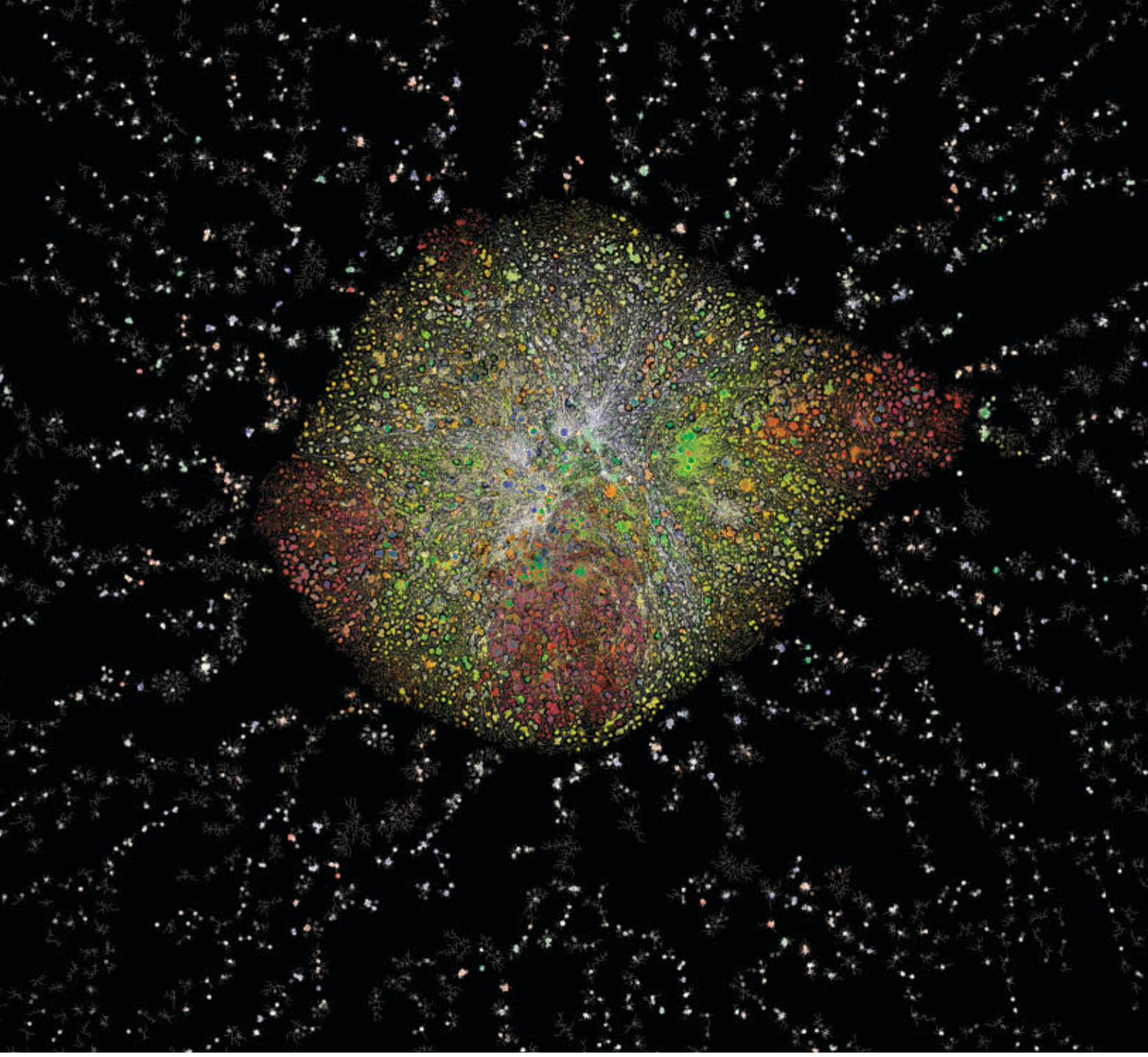
A HOLY GRAIL of biological research is deciphering the workings of a cell—the elementary unit of life. The main building blocks of the cell are macromolecules called proteins; they are key factors in driving cellular processes and determining the structure and function of cells. Proteins do not work in isolation but rather physically interact to form cellular machineries or transmit molecular signals. A modification of a single protein may have dramatic effects on the cell; indeed, many diseases (for example, Huntington's disease[26]) are the result of small changes to a single protein and, consequently, to its set of interacting partners and functionality. The mapping of proteins and their interactions and the interpretation of this data are thus a fundamental challenge in modern biology with important applications in disease diagnosis and therapy.[15]

The last two decades have witnessed a great shift in biological research. While classical research focused on a single gene or subsystem of a specific organism, the emergence of high-throughput technologies for measuring different molecular aspects of the cell has led to a different, systems-level approach. By this approach, genome-wide data is used to build computational models of certain aspects of the cell, thereby generating new biological hypotheses that can be experimentally tested and used to further improve the models in an iterative manner.

A prime example for this technological revolution is the development of techniques for measuring protein–protein interactions (PPIs). Historically, such interactions were measured at small scale—one or few interactions at a time. The development of automated, large-scale measurement technologies such as the yeast two-hybrid system[10] and the co-immunoprecipitation assay[1] has enabled the mapping of

» key insights

- **The explosion of biological network data necessitates methods to filter, interpret, and organize this data into modules of cellular machinery.**
- **The comparative analysis of networks from multiple species has proven to be a powerful tool in detecting significant biological patterns that are conserved across species and in enabling their interpretation.**
- **Comparative network analysis presents hard computational challenges such as graph and subgraph isomorphism and detecting heavy subgraphs; these can be tackled to near-optimality by a combination of heuristic, parameterized, and integer programming-based approaches.**



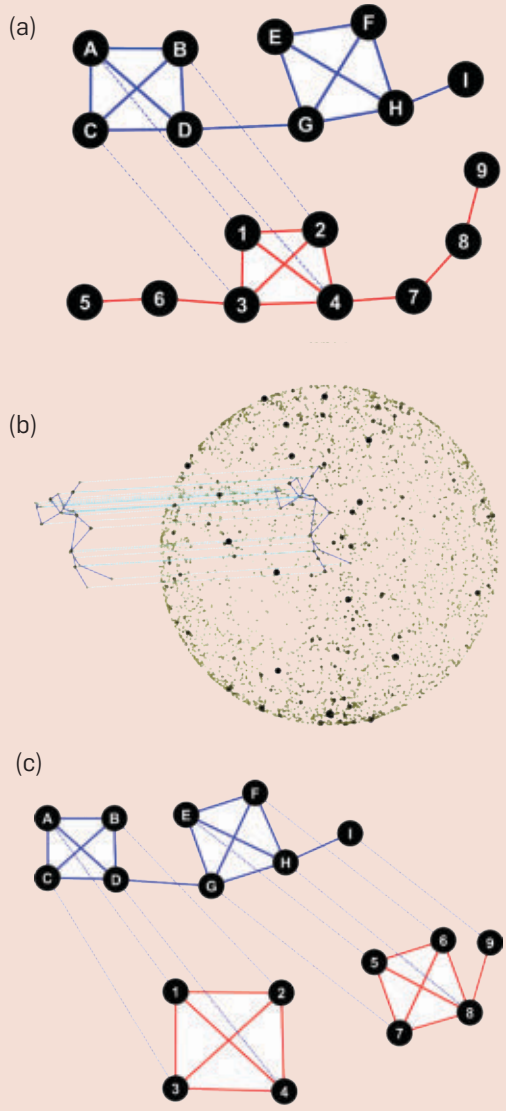
PROTEIN NETWORK VISUALIZATION BY EDWARD MARCOTTE AND ALEX ADAI

the entire interactome of a species in a single experiment.

Since the first publication of PPI data in yeast,[37] dozens of large-scale assays have been employed to measure PPIs in a variety of organisms including bacteria,[25] yeast, worm,[20] fly,[12] and human.[36,27] Protein interaction data is being accumulated and assessed in numerous databases including DIP,[28] BioGRID,[35] and more. Nevertheless, PPI data remains noisy and incomplete. The reliability of different experimental sources for protein-protein interactions has been estimated to be in the range of 25%–60%.[8] A recent experimental assessment of PPIs in yeast[39] estimated that even in this well-mapped organism, the set of reproducible and highly confident interactions covers only 20% of the yeast's interaction repertoire.

The low quality of the data has driven the use of cross-species conservation criteria to focus on the more reliable parts of the network and infer likely functional components. The basic paradigm was borrowed from the genomic sequence world, where sequence conservation (across species) often implies that the conserved region is likely to retain a similar biological function.[3,24] This evolutionary principle has motivated a series of works that aim at comparing multiple networks to extract conserved functional components at two different levels: the protein level and the subnetwork level. On the protein level, proteins whose network context is conserved across multiple species are likely to share similar functions.[34] On the subnetwork level, conserved subnetworks are likely to correspond to true functional components, such as protein complexes, and to have similar function.[32] In both cases, biological knowledge in any one of the species can be transferred to the others, allowing the annotation of networks in an efficient and accurate manner.[30]

Figure 1. Computational problems in comparative network analysis.



(a) In *local network alignment*, we wish to identify local regions that are similar across multiple networks. The similarity combines topological similarity and node similarity. When looking for matching dense subgraphs, the solution may align nodes {1, 2, 3, 4} to either {*A, B, C, D*} or {*E, F, G, H*}.

(b) In *network querying*, instances of a small network are searched for in another, usually much larger, network. For illustration purposes, assume we define a network based on the sky map where nodes represent the stars. Close stars are connected by an edge and the similarity between stars is determined according to their luminosity (graphically represented by both size and color). A known constellation may serve as a query to look for similar patterns.

(c) In *global network alignment*, the goal is to align all the nodes from one network with those of the other network, while optimizing node and edge similarity. In the given example, assuming all nodes are equally similar, a clique such as the one given by the nodes {1, 2, 3, 4} could be locally aligned to either {*A, B, C, D*} or {*E, F, G, H*}. However, when globally aligning the entire networks, the additional information given by the topology of the nodes {5, 6, 7, 8, 9} disambiguates the choice.

In this review, we survey the field of comparative network analysis with an emphasis on the arising computational problems and the different methods that have been used to tackle them, starting from heuristic approaches, going through parameterized algorithms that perform well on practical instances, and ending with optimal integer linear programming (ILP)-based solutions that rely on powerful, yet available, industrial solvers. We demonstrate the applications of these methods to predict protein function and interaction, infer the organization of protein-protein interaction networks into their underlying functional modules, and link biological processes within and across species.

A Roadmap to Network Comparison Techniques

We view a PPI network of a given species as a graph $G=(V, E)$, where V is the set of proteins of the given species and E is the set of pairwise interactions among them. In a network comparison problem, one is given two or more networks along with sequence information for their member proteins. The goal is to identify similarities between the compared networks, which could be either local or global in nature (Figure 1). The underlying assumption is that the networks have evolved from a common ancestral network, and hence, evolutionarily related proteins should display similar sequence and interaction patterns. For ease of presentation, we focus in the description below on pairwise comparisons, but the problems and their solutions generalize to multiple networks.

Most algorithms for network comparison score the similarity of two subnetworks by first computing a many-to-many mapping between their vertices (with possibly some unmatched vertices in either network) and then scoring the similarity of proteins and interactions under this mapping. Proteins are commonly compared by their associated amino-acid sequences, using a sequence comparison tool such as BLAST.[3] The similarity score of any two sequences is given as a *p*-value, denoting the chance of observing such sequence similarity at random. Significant *p*-values imply closer evolutionary distance and, hence, higher chances of sharing similar functions. Interactions are compared in a variety of ways; the simplest and most common of which is to count the number of *conserved interactions*. Formally, given a mapping Φ of proteins between two networks (associating proteins of one network with sets of proteins in the other network), an interaction (u, v) in one species is said to be *conserved* in the other species if there exist $u' \in \Phi(u)$ and $v' \in \Phi(v)$ such that u' and v' interact.

Historically, the first considered problem variant was *local network alignment* (Figure 1a), where the goal is to identify local regions that are similar across the networks being compared. To this end, one defines a scoring function that measures the similarity of a pair of subnetworks, one from each species, in terms of their topology and member proteins. To guide the search for high scoring, or significant matches, the scoring function is often designed to favor a certain class of subnetworks, such as dense subnetworks that serve as a model for protein complexes,[13,16,32] or paths that serve as a model for protein pathways.[17,18] In the related *network querying* problem (illustrated in Figure 1b in an astronomical context), a match is sought between a query subnetwork, representing a known functional component of a well-studied species, and a relatively unexplored network of some other organism. The match could be exact (that is, an isomorphic

subgraph under some mapping of the proteins between the two species) or inexact, allowing unmatched nodes on either subnetwork. This problem was first studied by Kelley et al.[17] in the context of local network alignment; its later development accompanied the growth in the number of mapped organisms.[5,7,9,33] The third problem that has been considered is *global network alignment* (Figure 1c), where one wishes to align whole networks, one against the other.[4,34] In its simplest form, the problem calls for identifying a 1-1 mapping between the proteins of two species so as to optimize some conservation criterion, such as the number of conserved interactions between the two networks.

All these problems are NP-hard as they generalize graph and subgraph isomorphism problems. However, heuristic, parameterized, and ILP approaches for solving them have worked remarkably well in practice. Here, we review these approaches and demonstrate their good performance in practice both in terms of solution quality and running time.

Heuristic Approaches

As in other applied fields, many problems in network biology are amenable to heuristic approaches that perform well in practice. Here, we highlight two such methods: a local search heuristic for local network alignment and an eigenvector-based heuristic for global network alignment.

NetworkBLAST[32] is an algorithm for local network alignment that aims to identify significant subnetwork matches across two or more networks. It searches for conserved paths and conserved dense clusters of interactions; we focus on the latter in our description. To facilitate the detection of conserved subnetworks, NetworkBLAST first forms a network alignment graph,[17,23] in which nodes correspond to pairs of sequence-similar proteins, one from each species, and edges correspond to conserved interactions (see Figure 2). The definition of the latter is flexible and allows, for instance, a direct interaction between the proteins of one species versus an indirect interaction (via a common network neighbor) in the other species. Any subnetwork of the alignment graph naturally corre-

Figure 2. The NetworkBLAST local network alignment algorithm. Given two input networks, a network alignment graph is constructed. Nodes in this graph correspond to pairs of sequence-similar proteins, one from each species, and edges correspond to conserved interactions. A search algorithm identifies highly similar subnetworks that follow a prespecified interaction pattern. Adapted from Sharan and Ideker.[30]

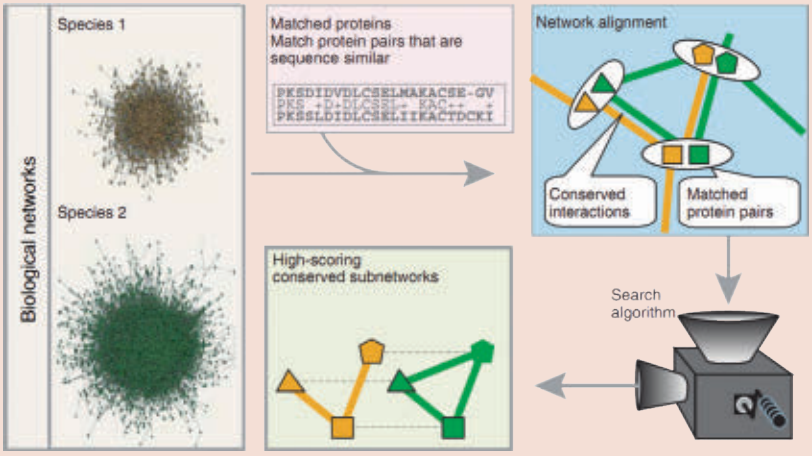
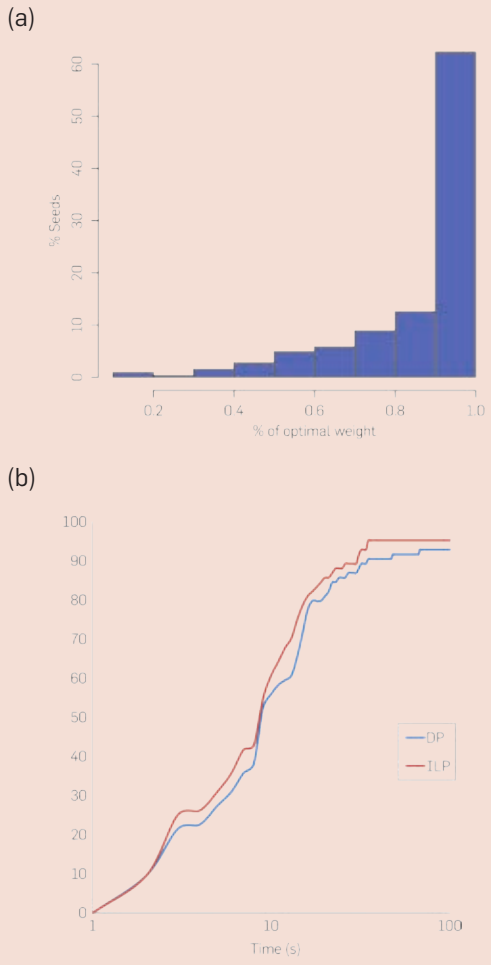


Figure 3. Performance comparison of computational approaches.



(a) An evaluation of the quality of NetworkBLAST's output clusters. NetworkBLAST was applied to a yeast network from Yu et al.[39] For every protein that served as a seed for an output cluster, the weight of this cluster was compared to the optimal weight of a cluster containing this protein, as computed using an ILP approach. The plot shows the % of protein seeds (*y*-axis) as a function of the deviation of the resulting clusters from the optimal attainable weight (*x*-axis).

(b) A comparison of the running times of the dynamic programming (DP) and ILP approaches employed by Torque.[7] The % of protein complexes (queries, *y*-axis) that were completed in a given time (*x*-axis) is plotted for the two algorithms. The shift to the left of the ILP curve (red) compared with that of the dynamic programming curve (blue) indicates the ILP formulation tends to be faster than the dynamic programming implementation.

Finding a Hairpin in a Haystack

Cells react to stimulation by propagating signals from sensory proteins to a set of target proteins. Given a signaling pathway of a well-studied species, it is interesting to query it within networks of less well-studied species. QPath is an exact path query method that is based on color coding. It extends the basic color coding formulation by allowing one to query a weighted network for inexact matches. That is, each edge of the network has a confidence associated with it, and the goal is to find high-scoring subnetworks that are similar to the query while allowing some flexibility in the matches (see Figure 5b). Specifically, there could be up to N_{ins} insertions of vertices to the match that are not aligned against the query's proteins, and up to N_{del} deletions of vertices from the query that are not aligned against vertices in the matching subnetwork. The algorithm aims at finding an inexact match that optimizes a scoring function that combines: (i) sequence similarity—every pair of matched proteins (q, v) contributes a sequence similarity term $\sigma(q, v)$; (ii) interaction confidence—every edge (u, v) on the matched path contributes a weight term $w(u, v)$; (iii) insertion penalty—c_{ins} per insertion; and (iv) deletion penalty—c_{del} per deletion. The relative importance of each of these terms is learned automatically by QPath; for clarity, we assume that all terms are equally important in the description below.

For a given coloring of the network by $k + N_{ins}$ colors, QPath employs a dynamic programming formulation to find the highest scoring match with up to N_{ins} insertions and N_{del} deletions. Denote the color of a vertex v by $c(v)$. Denote by $W(i, v, S, \theta_{del})$ the score of an optimal alignment of the first i nodes of the query that ends at a vertex v in the network, induces θ_{del} deletions, and visits nodes of each color in S. Then,

$$W(i, v, S, \theta_{del}) = \max_{u \in N(v)} \begin{cases} W(i-1, u, S \setminus \{c(v)\}, \theta_{del}) + w(u, v) + \sigma(q_i, v) & (u, v) \in E \\ W(i, u, S \setminus \{c(v)\}, \theta_{del}) + w(u, v) - c_{ins} & (u, v) \in E \\ W(i-1, v, S, \theta_{del} - 1) - c_{del} & 0 < \theta_{del} \le N_{del} \end{cases}$$

The best scoring path is obtained using a standard dynamic programming backtracking starting at

$$\operatorname*{argmax}_{v \in V,\, S \subseteq \{1, \ldots, k+N_{ins}\},\, \theta_{del} \le N_{del}} W(k, v, S, \theta_{del}).$$

sponds to a pair of potentially matching subnetworks. NetworkBLAST scores such a subnetwork by the density of its corresponding intra-species subnetworks versus the chance that they arise at random, assuming a random model that preserves the node degrees.

After constructing the alignment graph, the algorithm proceeds to identify high-scoring subnetworks. This is done by starting with a seed of at most four nodes, and applying a local search to expand it. Each node serves as the center of a seed, along with at most three of its neighbors. The search iteratively adds or removes a node that contributes most to the score, as long as the score increases (and up to an upper bound of 15 nodes). The effectiveness of this search strategy can be quantified by comparing to an exhaustive search when such is possible. Figure 3(a) presents such a comparison when analyzing a single (yeast) network from Yu et al.,[39] searching the best cluster containing each of the network's proteins. It can be seen that the greedy heuristic produces near-optimal clusters (up to 20% deviation in score) in about 75% of the cases, with an average of merely 13% deviation from the optimal score. Notably, NetworkBLAST requires only a few minutes to run, while the exhaustive (ILP-based) approach took several hours while limiting the solver to five minutes per seed. For seven out of a total of 326 seeds, the solver could not find an optimal solution within the allotted time.

While NetworkBLAST can be used to align multiple networks, the size of the alignment graph grows exponentially with the number k of networks and becomes prohibitive for $k = 4$. Interestingly, the NetworkBLAST alignment strategy can be mimicked without having to explicitly construct the alignment graph. Instead, Kalaev et al.[16] show that one can build a linear-size *layered alignment graph* where each layer contains the PPI network of a single species and inter-layer edges connect similar proteins. The main observation is that a set of proteins that are sequence similar, one from each species, translates to a size-k subgraph that contains a protein (vertex) from each species and is connected through the sequence similarity edges. Such a subgraph must have a spanning tree, which can be looked for using dynamic programming.

To exemplify the algorithm, consider the implementation of NetworkBLAST's local search strategy and let us focus on the addition of new k-protein "nodes" (that is, these would have been nodes of the alignment graph) to the growing seed. The latter requires identification of k inter-species proteins that induce a connected graph on the interlayer edges and contribute most to the seed's weight. As the contribution of each protein to the score can be easily computed and the total contribution is the sum of individual protein contributions, the optimal "node" to be added can be identified in a recursive fashion. That is, the corresponding spanning tree is obtained by merging two neighboring subtrees that span distinct species subsets whose union is the entire species set. This computation takes $O(3^k l)$ time in total, where l is the number of inter-layer edges.

For global network alignment, both heuristic and exact (ILP) approaches exist. Here, we highlight one such approach by Singh et al.[34] that is based on Google's PageRank algorithm. The idea is to score pairs of proteins, one from each species, based on their sequence similarity as well as the similarity of their neighbors. A maximum matching algorithm is then used to find a high-scoring 1-1 alignment between the compared networks.

Singh et al. formulate the computation of the pairwise scores as an eigenvalue problem. Denote by R the score vector to be computed (over all pairs of interspecies proteins). Let $N(v)$ denote the set of nodes adjacent to v, and let A be a stochastic matrix over pairs of inter-species proteins, where $A_{(u,v),(u',v')} = 1/|N(u')||N(v')|$ if and only if $\{u, u', v, v'\}$ induce a conserved interaction (that is, (u, u') and (v, v') interact). Finally, denote by B a normalized pairwise sequence similarity vector. The goal is to find a score vector R in which the similarity score of a pair of proteins combines the prior sequence

information with a weighted average of the similarity scores of their neighbors. Thus,

$$R = \alpha AR + (1-\alpha)B$$

where α is a parameter balancing the network-based and sequence-based terms. R can be found efficiently using the power method algorithm, which iteratively updates R according to the above equation and converges to the analytical solution $R = (I - \alpha A)^{-1}(1 - \alpha)B$.

As mentioned earlier, some manifestations of the global alignment problem can be solved to optimality using ILP. For instance, in Klau,[19] a 1-1 mapping that maximizes the number of conserved edges between the two networks is sought. Interestingly, while the eigenvector solution is heuristic in nature, it performs as well as an exact solution in terms of the number of conserved edges it reveals and its correspondence to a biological ground truth. Indeed, in a recent paper,[22] the performance of the heuristic approach of Singh et al. was compared to that of the exact ILP formulation. The algorithms were used to pairwise align the PPI networks of yeast, worm, and fly. Notably, for all three species pairs, the number of conserved edges in the alignment proposed by the heuristic method was equal to that of the ILP approach. As further shown in Mongioví and Sharan,[22] both approaches gave comparable results when their alignments were assessed against a gold standard database of cross-species protein matches (precisely, the HomoloGene database of clusters of orthologous genes[29]).

Exact Approaches

In contrast to the heuristic methods highlighted here, which do not provide any guarantee of the quality of the obtained solution, exact approaches guarantee optimality at the cost of speed. Two general methodologies for efficient, yet exact, solutions have been common in network analysis: fixed parameter and ILP formulations. Fixed parameter tractable problems can be solved in time that is, typically, exponential in some carefully chosen parameter of the problem and polynomial in the size of the input networks. As we will describe, many variants of the network querying problem are amenable to fixed parameter approaches, as the query subnetworks can be often assumed to have a small, bounded size. The other methodology we demonstrate is based on reformulating the problem at hand as an integer linear program and applying an industrial solver such as CPLEX[14] to optimize it. While arriving at a solution in a timely fashion is not guaranteed (as integer programming is NP-hard[11]), in practice, on current networks, many of these formulations are solved in reasonable time.

We start by describing a parameterized approach—color coding—that has been extensively used in network

Querying via an Integer Linear Program

In the biological domain, where one wishes to query known protein machineries in the network of another species, oftentimes the topology of the query is not known (only the identity of the member proteins is known). One possible way to tackle this scenario, applied by Torque,[7] is to assume that the query and, hence, the sought matches are connected. Torque is based on an ILP, which expresses the connectivity requirement by simulating a flow network, where an arbitrary node serves as a sink draining the flow generated by all the other nodes that participate in the solution. In detail, the ILP formulation uses the following variables: (i) a binary variable c_v for each note v, denoting whether it participates in the solution; (ii) a binary variable e_{uv} for each edge (u, v), denoting whether it participates in the solution subnetwork; (iii) a pair of rational variables f_{uv}, f_{vu} for each edge (u, v), representing both the magnitude and direction of the flow going through it; (iv) a binary variable r_v that marks the sink node; and (v) a binary variable g_{vq} for every pair of sequence-similar network and query nodes (v, q), denoting whether q is matched with v.

The following set of constraints is immediately derived from the model and expresses the requirements that (i) the solution should span k nodes; (ii) only one node may serve as a sink; and (iii) an edge is part of the solution only when its two endpoints are:

$$\sum_{v \in V} c_v = k$$

$$\sum_{v \in V} r_v = 1$$

$$e_{vu} \le \frac{1}{2} c_v + \frac{1}{2} c_u \quad \forall (v, u) \in E$$

Let Q denote the set of query proteins, $|Q| = k$, and let $\Phi(v) \subseteq Q$ denote the (possibly empty) subset of query proteins that are sequence similar to v. To obtain an adequate match between the solution and query proteins, the following constraints are added to ensure that (i) a network protein may match at most one query protein; (ii) all query proteins are matched with exactly one network protein (assuming, for simplicity, that there are no insertions or deletions); and (iii) only nodes that are part of the solution may be matched.

$$\sum_{q \in \Phi(v)} g_{vq} \le 1 \quad \forall v \in V$$

$$\sum_{v \in V} g_{vq} = 1 \quad \forall q \in Q$$

$$g_{vq} \le c_v \quad \forall v \in V, q \in \Phi(v)$$

To maintain a legal flow, one also needs to ensure that (i) the pair of flow variables associated with a given edge agrees on its direction and magnitude; (ii) the flow may only pass through edges that participate in the solution; and (iii) source nodes generate flow that is drained by the sink. These conditions are formulated by the following constraints:

$$f_{vu} = -f_{uv} \quad \forall (v, u) \in E$$

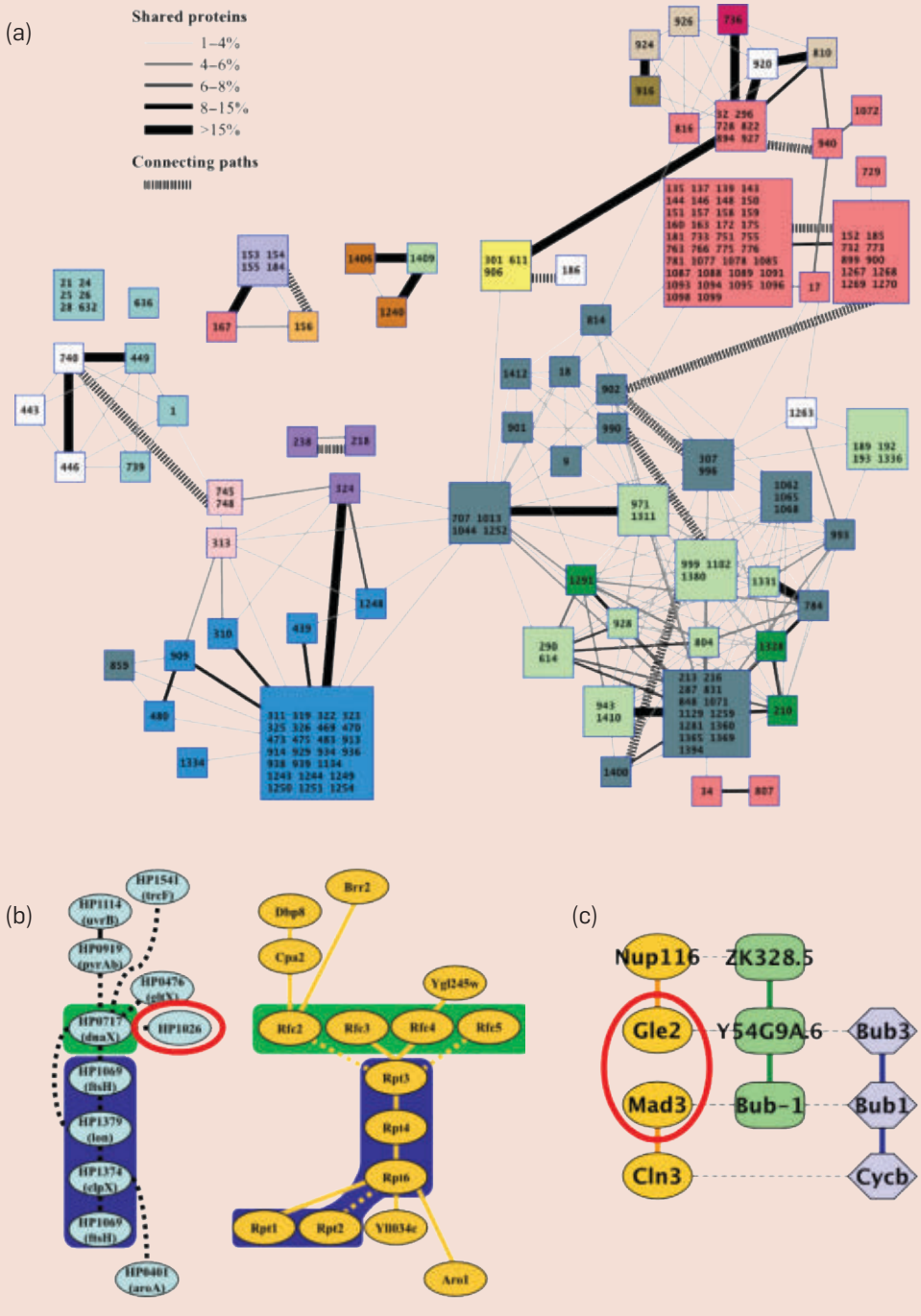
$$f_{vu}, f_{uv} \le (k-1) e_{vu} \quad \forall (v, u) \in E$$

$$\sum_{u \in N(v)} f_{vu} = c_v - k \cdot r_v \quad \forall v \in V$$

Together, the above constraints restrict the solutions to take the form of a connected subnetwork spanning exactly k nodes that are sequence similar to their respective matches in the query. Finally, denoting the weight of edge (u, v) by $w(u, v)$, the objective is to maximize the weight of the solution subnetwork:

$$\max \sum_{(u,v) \in E} w(u, v) e_{uv}$$

Figure 4. Insights derived from a multiple network alignment.



(a) A global map of conserved protein machineries (dense clusters and paths, denoted by arbitrary IDs) in the alignment of the PPI networks of yeast, worm, and fly. Clusters with high overlap (>15%) that share an enriched function are grouped together into regions (squares colored by the enriched function; color legend is omitted for simplicity). Solid links indicate overlapping regions; their thickness is proportional to the % of shared proteins. Hashed links indicate conserved paths connecting different clusters.

(b) Aligning yeast and bacterial networks suggests that DNA replication (yeast proteins Rfc2/3/4/5 and bacterial protein dnaX; shaded in green) and protein degradation (yeast proteins Rpt1/2/3/4/5 and bacterial proteins ftsH, lon, and clpX; shaded in purple) are functionally linked. This conserved region also sheds light on the function of the bacterial protein HP1026 (circled in red). Its inclusion in this region, interaction with the bacterial replication machinery (dnaX), and placement opposite of the yeast replication proteins all point to its involvement in DNA replication.

(c) Interaction prediction. The yeast proteins Gle2 and Mad3 (circled in red) are not known to interact. However, the interaction of their sequence-similar proteins (Y54G9A.6 and Bub-1 in worm and Bub3 and Bub1 in fly) and their inclusion in a conserved subnetwork suggest that the interaction is conserved in yeast as well. Adapted from Sharan et al.[32] and Kelley et al.[17]

querying applications. Color coding was originally developed by Alon et al.[2] for searching for structured size-k subgraphs, such as simple paths and trees, within a graph. Its complexity is $2^{O(k)}m$, where m is the size of the searched graph. Color coding is based on the idea that by randomly assigning k distinct colors to the vertices of the graph, the task of finding a simple subgraph translates to that of finding a *colorful* subgraph, namely, one spanning k distinct colors. For certain classes of tree-like subgraphs, the subsequent search can be efficiently implemented using dynamic programming. Since a particular subgraph need not be colorful in a specific color assignment, multiple color assignments should be considered to retrieve a desired subgraph with high probability. Precisely, the probability that a graph of size k is colorful is $k!/k^k > e^{-k}$; hence, in expectation, e^k iterations of the algorithm suffice to detect the desired subgraph.

In the context of comparative network analysis, color coding was mainly used to tackle network querying problems, where typically the query subnetwork is small (5–15 proteins), motivating the use of this parameterized approach. One specific example is the QPath[33] method for querying

paths in a network. QPath extends the basic color coding formulation by querying weighted networks for inexact matches (see the accompanying sidebar "Finding a Hairpin in a Haystack"). The algorithm takes minutes to run with length-7 queries and up to three insertions (unaligned match vertices) and three deletions (unaligned query vertices). Efficient heuristics to color the network can be used to reduce its time even further.[9,21] In a follow-up work,[9] the QPath approach was extended to handle queries of bounded treewidth and a heuristic solution was offered for general queries.

Our last highlighted method uses an ILP formulation to optimally solve a different variant of the network querying problem, where the topology of the query is not known. This scenario is very common when querying for protein complexes, where the underlying interaction structure is rarely available[39] but the member proteins are assumed to be connected. Hence, instead of searching for a particular interaction pattern, the goal is to find a matching subgraph that is *connected*. In Bruckner et al.,[7] an ILP solution to this problem is given. The main challenge in formulating the problem as an ILP is to express the connectivity of the solution sought. The Torque algorithm[7] solves this problem by modeling the solution subgraph as a flow network, where one of its nodes is arbitrarily designated as a sink, capable of draining $k - 1$ units of flow, and all the other nodes are set as sources, each generating one unit of flow. A set of constraints requires that the total flow in the system is preserved. The detailed program is given in the accompanying sidebar "Querying via an Integer Linear Program."

Notably, there is also a parameterized approach to this querying problem. The approach is based on the observation that a connected subgraph can be represented by its spanning tree, so the querying problem translates to that of finding a tree of k distinct vertices. The latter problem can be solved using the color coding technique.[6,7] Interestingly, for most instances, the dynamic programming approach is empirically slower than running the ILP formulation through a solver, as demonstrated in Figure 3(b).

The Power of Comparative Network Analysis

The successful application of comparative network analysis approaches depends not only on their computational attributes but also on their biological relevance. Here, we give examples for the applications of several of the reviewed approaches and the biological insights they have enabled. We demonstrate the power of comparative network analysis approaches by comparing their performance with that of methods that are either sequence-based and, thus, cannot exploit the network information, or single-species-based and as a result are more prone to noise in the network data.

The most intuitive use of comparative network analysis is to gather support for computational predictions from multiple species. A prime example for this use is the inference of protein complexes or pathways from PPI data. For instance, in Sharan et al.,[31] a cross-species analysis is used to identify yeast-bacterial conserved complexes. By comparing the inferred complexes to known complexes in yeast, it is shown that the comparative analysis increases the specificity of the predictions (compared to a yeast-only analysis) by a significant margin, albeit at the price of reduced sensitivity.

In Sharan et al.,[32] the local alignment of three networks (yeast, worm, and fly) was used to identify their conserved protein machineries (Figure 4a); those were used in a systematic manner to infer protein function (Figure 4b) and interaction (Figure 4c), showing superior performance compared to that of a sequence-based analysis. In brief, NetworkBLAST was used to identify high-scoring triplets of matching subnetworks across the three networks. Whenever the proteins in a conserved subnetwork were enriched for a certain function and at least half the proteins in the subnetwork were annotated with that function, the rest of the subnetwork's proteins were predicted to have that function as well. This prediction strategy yielded 58%–63% accuracy across the three species (in a cross-validation test), versus 37%–53% accuracy for a sequence-based method that predicts the function of a protein based on its most sequence-similar protein in the other species. The conserved subnetworks were further used to predict novel PPIs in the following manner: a pair of proteins were predicted to interact if two sequence-similar proteins were known to interact in another species (directly or via a common network neighbor) and, additionally, if the four proteins co-occurred in one of the conserved subnetworks. Remarkably, this strategy yielded >99% specificity in cross-validation. Experimental validation of 65 of these predictions gave a success rate in the range of 40%–52%. In comparison, sequence-based predictions that do not use the conserved subnetwork information yielded success rates in the range of 16%–31%.[18,32]

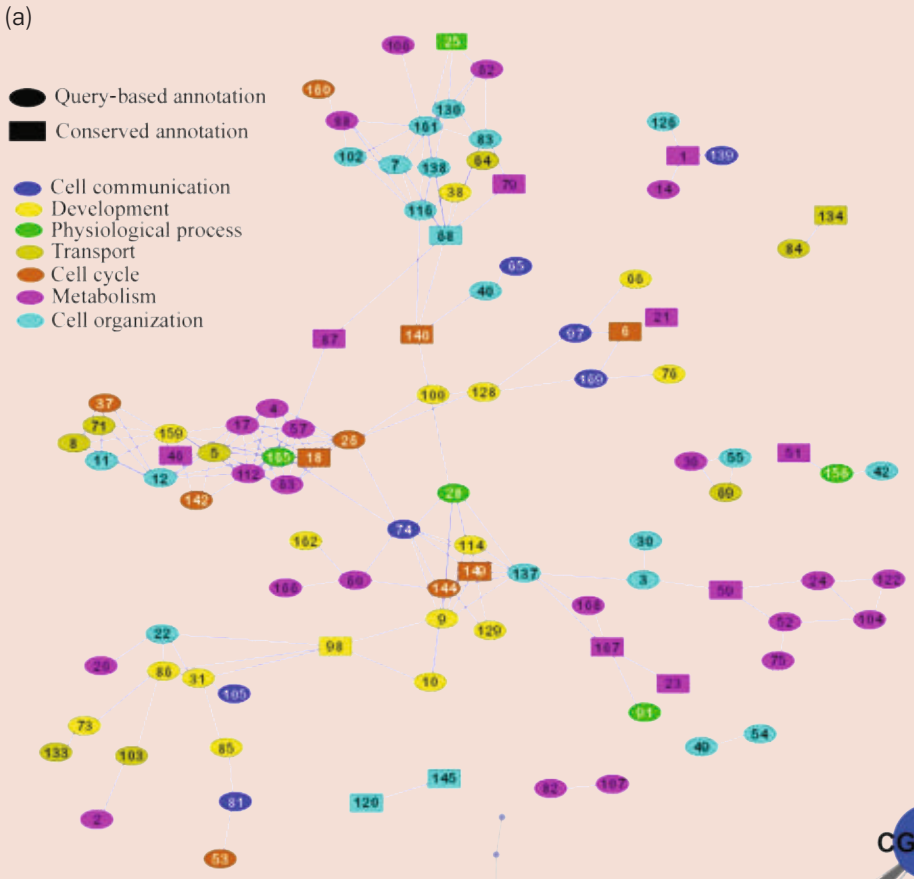
The transfer of annotations across species can go beyond single proteins to whole subnetworks. For instance, in Shlomi et al.,[33] paths in the yeast network served as queries for the fly network. The resulting matches were annotated with the function of the query (whenever the query was significantly enriched with some functional annotation; see Figure 5a), and the predictions were tested versus the known functional annotations in the fly. Overall, the annotation was accurate in 64% of the cases, compared to 40% for sequence-based annotation.

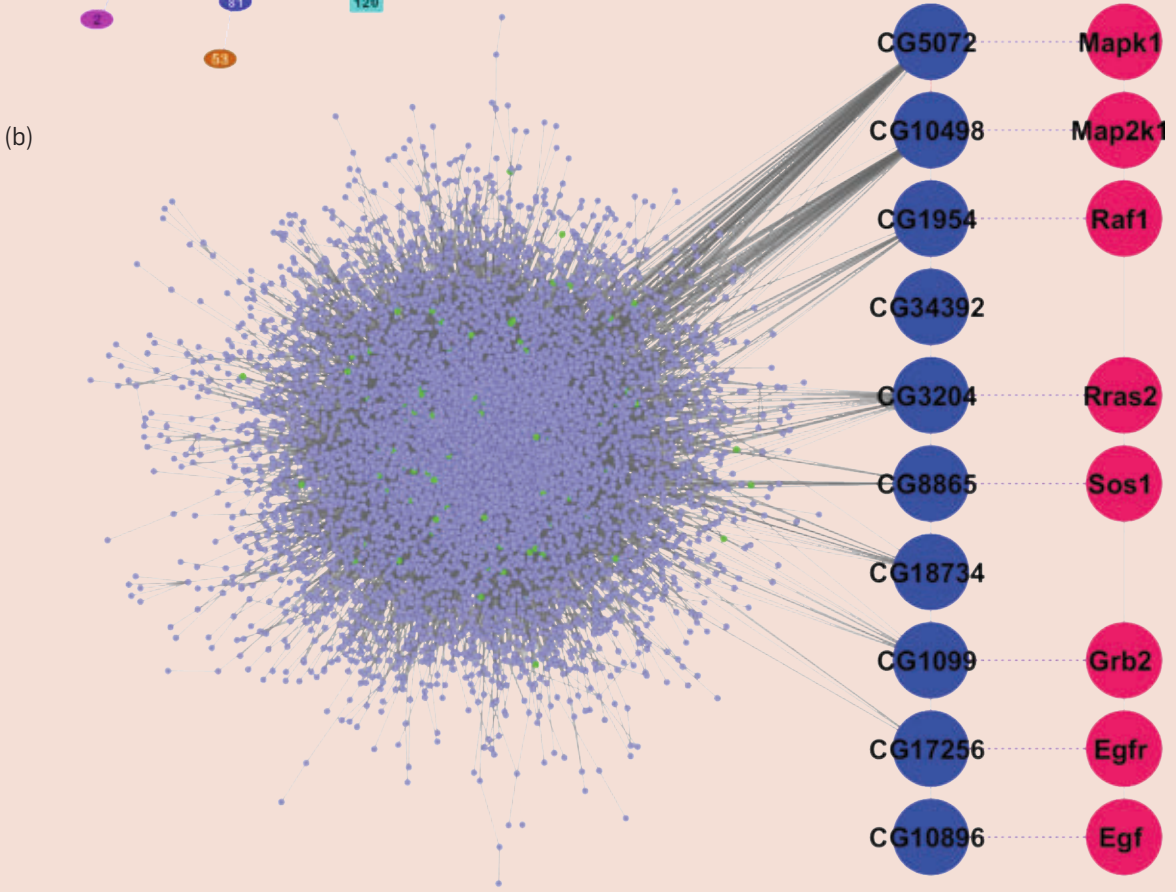
Network comparison schemes can be used to gain additional insights on protein function, interaction, and evolution. Both local and global network alignments suggest aligned pairs (or, more generally, sets) of proteins as functionally similar. Accordingly, they have been used to identify proteins that have evolved from a common ancestor and retained their function (so called *functional orthologs*).[4,38] In Bandyopadhyay et al.,[4] it is shown that in a majority of the cases, the aligned pairs were not the highest sequence similar ones. In addition, the conserved subnetworks often connect cellular processes that may work together in a coordinated manner (Figure 4b). The evidence is particularly compelling when the link is supported by multiple species.

Conclusion

The explosion of molecular data in the last two decades is revolutionizing biological and, consequently, computational research. The arising computational problems require

Figure 5. Path queries.





(a) A yeast-fly conserved pathway map. Pathways from yeast were used to query the fly network. Nodes represent best-match pathways and are connected if they share more than two proteins. Nodes are colored according to their predicted function based on the yeast query proteins. Best-match pathways in which significantly many proteins are annotated with the predicted function appear as boxes.

(b) A querying example. QPath was applied to query the human MAPK pathway (red nodes) within the fly network (blue nodes). The best scoring pathway (dark blue nodes) contains two insertions relative to the query. Fly proteins that are similar to the query are shown in light green. Adapted from Shlomi et al.[33]

practical solutions that can cope with an evergrowing scale. In addition to heuristic approaches, it is often of interest to compute exact solutions that can potentially lead to new insights about the biological problem at hand. The combination of parameterized approaches and powerful linear programming solvers has enabled the development of efficient, yet exact, methods to solve some of the key problems in comparative network analysis.

The application of comparative analysis tools to available network data has provided valuable insights on the function and interplay among molecular components in the cell. While much progress has already been made, new computational techniques will need to be developed to cope with the flood of genomic data that is expected to arrive in the coming years. These will span thousands of organisms and diverse molecular aspects. The arising challenges will involve the organization of this data into high-quality networks, data imputation through the integration of multiple information sources, multiple network alignment, and, ultimately, the propagation of curated or experimentally derived annotations through the aligned networks. Hybrid solution approaches that try to combine different techniques depending on the problem instance (see, for example, Bruckner et al.[7]) may be key to meeting those challenges.

Acknowledgments

We thank Sharon Bruckner for contributing Figure 2b. Atias was partially funded by the Edmond J. Safra Bioinformatics Program. This work was supported by a research grant from the Israel Science Foundation (Grant no. 241/11). C

References

1. Aebersold, R., Mann, M. Mass spectrometry-based proteomics. *Nature 422* (2003), 198–207.
2. Alon, N., Yuster, R., Zwick, U. Color-coding. *J. ACM 42* (July 1995), 844–856.
3. Altschul, S.F., Gish, W., Miller, W., Myers, E.W., Lipman, D.J. Basic local alignment search tool. *J. Mol. Biol. 215*, 3 (Oct. 1990), 403–410.
4. Bandyopadhyay, S., Sharan, R., Ideker, T. Systematic identification of functional orthologs based on protein network comparison. *Genome Res. 16*, 3 (Mar. 2006), 428–435.
5. Banks, E., Nabieva, E., Peterson, R., Singh, M. Netgrep: fast network schema searches in interactomes. *Genome Biol. 9* (2008), R138.
6. Betzler, N., Fellows, M.R., Komusiewicz, C., Niedermeier, R. Parameterized algorithms and hardness results for some graph motif problems. In *Proceedings of the 19th annual symposium on Combinatorial Pattern Matching* (Berlin, Heidelberg, 2008), CPM '08, Springer-Verlag, 31–43.
7. Bruckner, S., Hffner, F., Karp, R.M., Shamir, R., Sharan, R. Topology-free querying of protein interaction networks. *J. Comput. Biol. 17*, 3 (Mar. 2010), 237–252.
8. Deng, M., Sun, F., Chen, T. Assessment of the reliability of protein-protein interactions and protein function prediction. In *Proceedings of the 8th Pacific Symposium on Biocomputing* (2003), 140–151.
9. Dost, B., Shlomi, T., Gupta, N., Ruppin, E., Bafna, V., Sharan, R. QNet: a tool for querying protein interaction networks. *J. Comput. Biol. 15*, 7 (Sep. 2008), 913–925.
10. Fields, S. High-throughput two-hybrid analysis: the promise and the peril. *FEBS J. 272* (2005), 5391–5399.
11. Garey, M., Johnson, D. Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman and Co., San Francisco, 1979.
12. Giot, L., Bader, J., Brouwer, C., Chaudhuri, A., Kuang, B., Li, Y., Hao, Y., Ooi, C., Godwin, B., Vitols, E., Vijayadamodar, G., Pochart, P., Machineni, H., Welsh, M., Kong, Y., Zerhusen, B., Malcolm, R., Varrone, Z., Collis, A., Minto, M., Burgess, S., McDaniel, L., Stimpson, E., Spriggs, F., Williams, J., Neurath, K., Ioime, N., Agee, M., Voss, E., Furtak, K., Renzulli, R., Aanensen, N., Carrolla, S., Bickelhaupt, E., Lazovatsky, Y., DaSilva, A., Zhong, J., Stanyon, C.A., Finley, R.L., Jr, White, K., Braverman, M., Jarvie, T., Gold, S., Leach, M., Knight, J., Shimkets, R., McKenna, M., Chant, J., Rothberg, J. A protein interaction map of Drosophila melanogaster. *Science 302*, 5651 (2003), 1727–1736.
13. Hirsh, E., Sharan, R. Identification of conserved protein complexes based on a model of protein network evolution. *Bioinformatics 23* (2007), e170–e176.
14. IBM. IBM ILOG CPLEX V12.1 user's manual for CPLEX, 2009.
15. Ideker, T., Sharan, R. Protein networks in disease. *Genome Res. 18* (2008), 644–652.
16. Kalaev, M., Bafna, V., Sharan, R. Fast and accurate alignment of multiple protein networks. *J. Comput. Biol. 16* (2009), 989–999.
17. Kelley, B.P., Sharan, R., Karp, R.M., Sittler, T., Root, D.E., Stockwell, B.R., Ideker, T. Conserved pathways within bacteria and yeast as revealed by global protein network alignment. *Proc. Natl. Acad. Sci. U.S.A. 100*, 20 (Sep. 2003), 11394–11399.
18. Kelley, B.P., Yuan, B., Lewitter, F., Sharan, R., Stockwell, B.R., Ideker, T. PathBLAST: a tool for alignment of protein interaction networks. *Nucleic Acids Res. 32*, Web Server issue (Jul. 2004), W83–W88.
19. Klau, G.W. A new graph-based method for pairwise global network alignment. *BMC Bioinformatics 10*, Suppl 1 (2009), S59.
20. Li, S., Armstrong, C., Bertin, N., Ge, H., Milstein, S., Boxem, M., Vidalain, P., Han, J.D., Chesneau, A., Hao, T., Goldberg, D., Li, N., Martinez, M., Rual, J., Lamesch, P., Xu, L., Tewari, M., Wong, S., Zhang, L., Berriz, G., Jacotot, L., Vaglio, P., Reboul, J., Hirozane-Kishikawa, T., Li, Q., Gabel, H., Elewa, A., Baumgartner, B., Rose, D., Yu, H., Bosak, S., Sequerra, R., Fraser, A., Mango, S., Saxton, W., Strome, S., Heuvel, S.V.D., Piano, F., Vandenhaute, J., Sardet, C., Gerstein, M., Doucette-Stamm, L., Gunsalus, K., Harper, J., Cusick, M., Roth, F., Hill, D., Vidal, M. A map of the interactome network of the metazoan *C. elegans*. *Science 303*, 5657 (2004), 540–543.
21. Mayrose, I., Shlomi, T., Rubinstein, N.D., Gershoni, J.M., Ruppin, E., Sharan, R., Pupko, T. Epitope mapping using combinatorial phage-display libraries: a graph-based algorithm. *Nucleic Acids Res. 35*, 1 (2007), 69–78.
22. Mongioví, M., Sharan, R. Global alignment of protein-protein interaction networks. *Data Mining for Systems Biology*. H. Mamitsuka, C. DeLisi, and M. Kanehisa, eds. Springer, 2012, in press.
23. Ogata, H., Fujibuchi, W., Goto, S., Kanehisa, M. A heuristic graph comparison algorithm and its application to detect functionally related enzyme clusters. *Nucleic Acids Res. 28* (2000), 4021–4028.
24. Pellegrini, M., Marcotte, E., Thompson, M., Eisenberg, D., Yeates, T. Assigning protein functions by comparative genome analysis: protein phylogenetic profiles. *Proc. Natl. Acad. Sci. U.S.A. 96* (1999), 4285–4288.
25. Rain, J., Selig, L., Reuse, H.D., Battaglia, V., Reverdy, C., Simon, S., Lenzen, G., Petel, F., Wojcik, J., Schachter, V., Chemama, Y., Labigne, A., Legrain, P. The protein–protein interaction map of *Helicobacter pylori*. *Nature 409* (2001), 211–215.
26. Ross, C.A., Tabrizi, S.J. Huntington's disease: from molecular pathogenesis to clinical treatment. *Lancet Neurol. 10*, 1 (Jan. 2011), 83–98.
27. Rual, J.-F., Venkatesan, K., Hao, T., Hirozane-Kishikawa, T., Dricot, A., Li, N., Berriz, G.F., Gibbons, F.D., Dreze, M., Ayivi-Guedehoussou, N., Klitgord, N., Simon, C., Boxem, M., Milstein, S., Rosenberg, J., Goldberg, D.S., Zhang, L.V., Wong, S.L., Franklin, G., Li, S., Albala, J.S., Lim, J., Fraughton, C., Llamosas, E., Cevik, S., Bex, C., Lamesch, P., Sikorski, R.S., Vandenhaute, J., Zoghbi, H.Y., Smolyar, A., Bosak, S., Sequerra, R., Doucette-Stamm, L., Cusick, M.E., Hill, D.E., Roth, F.P., Vidal, M. Towards a proteome-scale map of the human protein–protein interaction network. *Nature 437*, 7062 (Oct. 2005), 1173–1178.
28. Salwinski, L., Miller, C.S., Smith, A.J., Pettit, F.K., Bowie, J.U., Eisenberg, D. The database of interacting proteins: 2004 update. *Nucleic Acids Res. 32*, Database issue (Jan. 2004), D449–D451.
29. Sayers, E.W., Barrett, T., Benson, D.A., Bolton, E., Bryant, S.H., Canese, K., Chetvernin, V., Church, D.M., DiCuccio, M., Federhen, S., Feolo, M., Fingerman, I.M., Geer, L.Y., Helmberg, W., Kapustin, Y., Landsman, D., Lipman, D.J., Lu, Z., Madden, T.L., Madej, T., Maglott, D.R., Marchler-Bauer, A., Miller, V., Mizrachi, I., Ostell, J., Panchenko, A., Phan, L., Pruitt, K.D., Schuler, G.D., Sequeira, E., Sherry, S.T., Shumway, M., Sirotkin, K., Slotta, D., Souvorov, A., Starchenko, G., Tatusova, T.A., Wagner, L., Wang, Y., Wilbur, W.J., Yaschenko, E., Ye, J. Database resources of the national center for biotechnology information. *Nucleic Acids Res. 39*, Database issue (Jan. 2011), D38–D51.
30. Sharan, R., Ideker, T. Modeling cellular machinery through biological network comparison. *Nat. Biotechnol. 24*, 4 (Apr. 2006), 427–433.
31. Sharan, R., Ideker, T., Kelley, B., Shamir, R., Karp, R. Identification of protein complexes by comparative analysis of yeast and bacterial protein interaction data. *J. Comput. Biol. 12* (2005), 835–846.
32. Sharan, R., Suthram, S., Kelley, R.M., Kuhn, T., McCuine, S., Uetz, P., Sittler, T., Karp, R.M., Ideker, T. Conserved patterns of protein interaction in multiple species. *Proc. Natl. Acad. Sci. U.S.A. 102*, 6 (Feb. 2005), 1974–1979.
33. Shlomi, T., Segal, D., Ruppin, E., Sharan, R. QPath: a method for querying pathways in a protein–protein interaction network. *BMC Bioinformatics 7* (2006), 199.
34. Singh, R., Xu, J., Berger, B. Global alignment of multiple protein interaction networks with application to functional orthology detection. *Proc. Natl. Acad. Sci. U.S.A. 105*, 35 (Sep. 2008), 12763–12768.
35. Stark, C., Breitkreutz, B.-J., Chatr-Aryamontri, A., Boucher, L., Oughtred, R., Livstone, M.S., Nixon, J., Auken, K.V., Wang, X., Shi, X., Reguly, T., Rust, J.M., Winter, A., Dolinski, K., Tyers, M. The BioGRID interaction database: 2011 update. *Nucleic Acids Res. 39*, Database issue (Jan. 2011), D698–D704.
36. Stelzl, U., Worm, U., Lalowski, M., Haenig, C., Brembeck, F., Goehler, H., Stroedicke, M., Zenkner, M., Schoenherr, A., Koeppen, S., Timm, J., Mintzlaff, S., Abraham, C., Bock, N., Kietzmann, S., Goedde, A., Toksoz, E., Droege, A., Krobitsch, S., Korn, B., Birchmeier, W., Lehrach, H., Wanker, E. A human protein–protein interaction network: a resource for annotating the proteome. *Cell 122* (2005), 957–968.
37. Uetz, P., Giot, L., Cagney, G., Mansfield, T., Judson, R., Knight, J., Lockshon, D., Narayan, V., Srinivasan, M., Pochart, P., Qureshi-Emili, A., Li, Y., Godwin B., Conover, D., Kalbfleisch, T., Vijayadamodar, G., Yang, M., Johnston, M., Fields, S., Rothberg, J. A comprehensive analysis of protein–protein interactions in *Saccharomyces cerevisiae*. *Nature 403*, 6770 (2000), 623–627.
38. Yosef, N., Sharan, R., Noble, W.S. Improved network-based identification of protein orthologs. *Bioinformatics 24*, 16 (Aug. 2008), i200–i206.
39. Yu, H., Braun, P., Yildirim, M.A., Lemmens, I., Venkatesan, K., Sahalie, J., Hirozane-Kishikawa, T., Gebreab, F., Li, N., Simonis, N., Hao, T., Rual, J.-F., Dricot, A., Vazquez, A., Murray, R.R., Simon, C., Tardivo, L., Tam, S., Svrzikapa, N., Fan, C., de Smet, A.-S., Motyl, A., Hudson, M.E., Park, J., Xin, X., Cusick, M.E., Moore, T., Boone, C., Snyder, M., Roth, F.P., Barabsi, A.-L., Tavernier, J., Hill, D.E., Vidal, M. High-quality binary protein interaction map of the yeast interactome network. *Science 322*, 5898 (Oct. 2008), 104–110.

Nir Atias (atias@post.tau.ac.il) is a Ph.D. candidate in the Blavatnik School of Computer Science, Tel Aviv University, Tel Aviv Israel.

Roded Sharan (roded@post.tau.ac.il) is a professor in the Blavatnik School of Computer Science, Tel Aviv University, Tel Aviv Israel.

The University of Tennessee at Chattanooga
Department of Computer Science and Engineering
Assistant/Associate Professor Position

The Department of Computer Science and Engineering (CSE) at the University of Tennessee at Chattanooga invites applications from exceptionally qualified candidates in Computer Science or Computer Engineering for a tenure-track faculty position at the Assistant or Associate Professor level beginning August 1, 2012. A Ph.D. is required and strongest consideration will be given to candidates with expertise in Networking (communications, protocols, large-scale multimedia systems, mobile and wireless, high-performance, security, social).

The CSE department (www.cs.utc.edu), part of the College of Engineering and Computer Science, offers an ABET accredited B.S. degree, a M.S. degree, and has received certification by the CNSS, NSA, and DHA as a National Center of Academic Excellence in Information Assurance Education. The College is also home to the SimCenter and its graduate programs (M.S./Ph.D.) in Computational Engineering.

To apply, please e-mail in Word or pdf format an application letter, resume, descriptions of teaching and research philosophies, a copy of your transcript listing the completion of your doctoral degree and 3 letters of recommendation to facultyvitae26@utc.edu. If you have questions please contact Dr. Mina Sartipi, Mina-artipi@utc.edu.

Screening of applicants who have provided complete information will begin immediately and continue until the position is filled. The University of Tennessee at Chattanooga is an equal employment opportunity/affirmative action/Title VI & IX/Section 504 ADA/ADEA institution, and, as such, encourages the application of qualified women and minorities.

research highlights

DOI:10.1145/2160718.2160739

Technical Perspective
Best Algorithms + Best Computers = Powerful Match

By William Gropp

SAY YOU WANT to simulate the motion over time of the stars in a galaxy to learn about how galaxies formed and why the universe appears as it does to us. This seems like a simple computational problem—the laws of motion are governed by Newton's laws (let's ignore relativity) and simply require computing the force of every star on every other. For N stars, that is N^2 operations, and for billions of stars, that is a billion billion operations. Is it feasible on a single processor? Sadly, even today's fastest single processor and one single moment in time, the answer is "no." Even worse, this computation must be repeated millions or billions of times, as the motion of each star is followed. (If you prefer biology or materials science, replace "stars" with "atoms.")

So is parallelism the answer? Unfortunately not. Even the biggest systems today do not provide enough processing power to handle these problems (supercomputers may be fast, but they are not infinitely fast).

What can we do?

The following paper by Lashuk et al. shows the way. Solving such a difficult problem requires taking advantage of everything possible. The authors start with the algorithm. While the obvious way of computing the mutual interaction of N bodies on each other takes quadratic time in the number of bodies, there is a clever method that, with rigorous error bounds, can compute those interactions to any specified accuracy in linear time! This algorithm, developed 25 years ago by Greengard and Rohklin, exploits a hierarchical representation of the forces. But even a linear-time algorithm on a single processor is not fast enough for the problems that scientists want to solve. So, having picked an excellent algorithm, the authors proceed to parallelize it. However, the choice of algorithm does not make that easy. Because the algorithm uses a hierarchical representation, it is natural that the data structures involved are trees, and because the distribution of bodies, be they stars or atoms, is nonuniform, the trees are unbalanced. This makes effective parallelization difficult because a major impediment to efficient parallelization is effective load balance. The paper then tackles parallelism at multiple levels, starting with distributed memory parallelism, then continuing to multicore processes and finishing with GPUs. A very pragmatic argument is made for the use of hybrid parallel programming models (specifically, MPI combined with OpenMP and/or CUDA): a single programming model is not always the best solution for real applications.

Why should you read the following paper? Because it provides a clear description of solving a difficult problem by effectively combining the best algorithms with the best computers, while making use of the most appropriate strategies for parallelization.

Why should you read the following paper? Because it provides a clear description of solving a difficult problem by effectively combining the best algorithms with the best computers, while making use of the most appropriate strategies for parallelization. Throughout this work you will find the use of simple yet effective complexity analysis to guide and explain the choices that are made. This is high-performance computing at its best—using the best algorithms with the biggest machines; using complexity analysis to *engineer* a high-performance solution, using a hierarchy of programming models, each providing high performance for that part of a parallel computer. The result is a tool that can be used by scientists to solve problems that cannot be solved by any other means—an example of how computing has become critical to the advancement of science. C

William Gropp (wgropp@illinoise.edu) is the Paul and Cynthia Saylor Professor of Computer Science and Deputy Director for Research at the Institute for Advanced Computing Applications and Technologies at the University of Illinois Urbana-Champaign, IL.

DOI:10.1145/2160718.2160740

A Massively Parallel Adaptive Fast Multipole Method on Heterogeneous Architectures

By Ilya Lashuk, Aparna Chandramowlishwaran, Harper Langston, Tuan-Anh Nguyen, Rahul Sampath, Aashay Shringarpure, Richard Vuduc, Lexing Ying, Denis Zorin, and George Biros

Abstract

We describe a parallel fast multipole method (FMM) for highly nonuniform distributions of particles. We employ both distributed memory parallelism (via MPI) and shared memory parallelism (via OpenMP and GPU acceleration) to rapidly evaluate two-body nonoscillatory potentials in three dimensions on heterogeneous high performance computing architectures. We have performed scalability tests with up to 30 billion particles on 196,608 cores on the AMD/CRAY-based Jaguar system at ORNL. On a GPU-enabled system (NSF's Keeneland at Georgia Tech/ORNL), we observed 30× speedup over a single core CPU and 7× speedup over a multicore CPU implementation. By combining GPUs with MPI, we achieve less than 10 ns/particle and six digits of accuracy for a run with 48 million nonuniformly distributed particles on 192 GPUs.

1. INTRODUCTION

N-body problems are the computational cornerstone of diverse problems in mathematical physics,[16] machine learning,[8] and approximation theory.[4] Formally, the problem is how to efficiently evaluate

$$f(x_i) = \sum_{j=1}^{N} K(x_i, u_j) s(y_j),\ i = 1, \ldots, N, \qquad (1)$$

which is essentially an $\mathcal{O}(N^2)$ computation of all pairwise interactions between N particles. We refer to f as the "*potential*," s as the "*source density*," x as the coordinates of the target particles, y as the coordinates of the source particles, and K as the interaction kernel. For example, in electrostatic or gravitational interactions $K(x, y) = K(r) = \frac{1}{|r|}$ with $r = x - y$ and $|r|$ its Euclidean norm. Equation (1) expresses an *N-body problem*.

The challenge is that the $\mathcal{O}(N^2)$ complexity makes the calculation prohibitively expensive for problem sizes of practical interest. For example, understanding blood flow mechanics[21] or understanding the origins of the universe[26] requires simulations in which billions of particles interact.

Given the importance of *N*-body problems, there has been a considerable effort to accelerate them. Here is the main idea: the magnitude and variance of $K(r)$ decay with $|r|$, and so when the particles or groups of particles are well separated, the interactions can be approximated with small computational cost. But the details are tricky. How can we perform the approximation? Can we provide rigorous error and complexity bounds? And how do we implement these approximations efficiently?

One of the first successful attempts toward fast schemes was the Barnes–Hut method[1] that brought the complexity down to $\mathcal{O}(N \log N)$. Greengard and Rokhlin[3, 10, 22] solved the problem. They introduced the fast multipole method (FMM) that uses a rapidly convergent approximation scheme that comes with rigorous error bounds and reduces the overall complexity (in three dimensions) to $\mathcal{O}(\log(1/\epsilon)^3 N)$, where ϵ is the desired accuracy of the approximation to the exact sum. This result is impressive: *we can evaluate the sum in* Equation (1) *in arbitrary precision in linear time*.

So how does FMM work? Here is a sketch. Given the positions of particles for which we want to compute all pairwise interactions, we build an octree (henceforth we just consider FMM in three dimensions) so that any leaf node (or octant) has approximately a prescribed number of particles; then we perform tree traversals to evaluate the sum. We perform a postorder (bottom-up) traversal followed by a preorder traversal (top-down). The postorder traversal consists of calculations that involve an octant and its children. The preorder traversal is more complex, as it involves several octants in a neighborhood around the octant being visited.

At this level of abstraction, we begin to see how we might parallelize the FMM. When the distribution of particles is uniform, the analysis is relatively straightforward: we perform the traversals in a level-by-level manner, using data parallelism across octants in each level. The longest chain of dependencies is roughly $\mathcal{O}(\log N)$. The challenge lies in the case of nonuniform particle distributions, in which even the sequential case becomes significantly more complicated.[3]

Summary of the method: We describe and evaluate an implementation of the FMM for massively parallel distributed memory systems, including heterogeneous architectures based on multicore CPU and GPU co-processors. We use the message passing interface (MPI) for expressing distributed memory parallelism, and OpenMP and CUDA for shared memory and fine-grain parallelism. There exist many variants of FMM, depending on the kernel and the

The original version of this paper has the same title and was published in the *2009 Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*. Some results in this paper also appeared in *Reference Rahimian et al*, 2010.

interaction approximation scheme. Our implementation is based on the *kernel-independent FMM*.[28, 29] However, this choice affects only the construction of certain operators used in the algorithm. It does not affect the overall "flow" of the algorithm or its distributed-memory parallelization. (It does affect single-core performance as vectorization and fine-grain parallelism depend on the FMM scheme.)

Our parallel algorithm can be summarized as follows. Given the particles, distributed in an arbitrary way across MPI processes, and their associated source densities, we seek to compute the potential at each point. (For simplicity, in this paper, we assume that source and target points coincide.) First, we sort the points in Morton order,[24] a kind of space-filling curve, and then redistribute them so that each process owns a contiguous chunk of the sorted array. Second, we create the so-called locally essential tree (LET, defined in Section 3), for each process in parallel. Third, we evaluate the sums using the LETs across MPI processes and using GPU and OpenMP acceleration within each process. All the steps of the algorithm are parallelized, including tree construction and evaluation.

Related work: There is a considerable literature on parallel algorithms for N-body problems. A work–depth analysis of the algorithm gives $\mathcal{O}(N)$ work and $\mathcal{O}(\log N)$ depth for particle distributions that result in trees that have depth bounded by $\mathcal{O}(\log N)$. Greengard and Gropp analyzed and implemented the algorithm using the concurrent-read, exclusive-write (CREW) PRAM model for the case of uniform particle distributions.[9] The implementation is straightforward using level-by-level traversals of the tree and using data parallelism. The complexity of the algorithm, omitting the accuracy-related term, is $\mathcal{O}(N/p)+\mathcal{O}(\log p)$, where p is the number of threads. This result extends to message passing implementations.

The implementation, parallel scalability, and complexity analysis become harder when one deals with nonuniform distributions of particles. Callahan and Kosaraju[2] propose an exclusive read exclusive write (EREW)-PRAM algorithm with $p=N$ processors which is work optimal, takes $\log p$ time, and does not depend on the distribution of the points. To our knowledge, that algorithm has not been used in practice. Our algorithm uses many ideas that first appeared in the seminal work of Warren and Salmon, who introduced space-filling curves using Morton-ordering for partitioning the points across processors, the notion of local essential trees (LET), and a parallel tree construction algorithm.[27] These ideas were widely adopted and analyzed in other works.[7, 23]

Teng was the first to provide a complexity analysis that accounts for communication costs for particle distributions whose tree depth is bounded by $\log N$ (in finite floating point precision, all particle distributions satisfy this assumption).[25] The key idea is to build a communication graph in which graph nodes correspond to tree octants and edges to interactions between octants. Teng shows that FMM communication graphs have a bisector whose edge cut (in three dimensions) is bounded by $\mathcal{O}(N^{2/3}(\log N)^{4/3})$; to compare, the bisector for a uniform grid is $\mathcal{O}(N^{2/3})$. This result shows that scalable FMM calculations are theoretically possible. Teng outlines a divide-and-conquer parallel algorithm to construct and partition the tree. Morton-order sorting is used to build the tree and repartitioning is done using a geometric graph partitioning method that guarantees low edge-cut and good load balancing.[18] Assuming (1) a parallel radix sort is used to partition the points and (2) the depth of the FMM tree grows logarithmically with N, *the overall algorithm scales as* $\mathcal{O}(N/p)+\mathcal{O}(\log p)$. The constants in the complexity estimates involve flop rate, memory latency, and bandwidth parameters of the underlying machine architecture.

Work that is focused more on actual implementations and performance analysis includes Kurzak and Pettitt[15] and Ogata et al.[19] Several groups have been working on GPU acceleration.[5, 11, 12, 20, 30] Finally, for nearly uniform distributions of particles, one can use fast Fourier transforms (particle-in-cell methods) to calculate the sums in $\mathcal{O}(N \log N)$ time.

Contributions: To our knowledge, there are no FMM implementations that demonstrate scalability on thousands of MPI processes for highly nonuniform particle distributions. Our main contribution is to produce and demonstrate the scalability of such an implementation that exploits all levels of parallelism. In Section 3, we discuss the distributed memory parallelism and introduce a novel FMM-specific all-reduce algorithm that uses hypercube routing. In Section 4, we introduce the shared memory parallelism of our implementation, and in Section 5, we discuss our experiments and the scalability results.

2. OUTLINE OF FMM

In this section, we describe the FMM data structures, the main algorithmic steps, and the parallel scalability of the method. Without loss of generality for the rest of the paper, we assume that the source particles y_j and the target particles x_i coincide. The FMM tree construction ensures that every leaf octant contains no more than q particles. The sequential algorithm is simple: the root octant is chosen to be a cube that contains all the particles; we insert the particles in the tree one by one; we subdivide an octant when it contains more than q particles. The parallel construction is more involved, and we describe it in Section 3.

After the tree construction, for each octant β we need to construct its *interaction lists*. These lists contain other octants in the tree. Each list represents a precisely defined spatial neighborhood around β, and for every list we have to compute "interactions" between β and the octants in that list. By "interactions," we refer to floating point operations that correspond to matrix–matrix or matrix–vector multiplications. Once the interaction lists have been constructed, the tree is traversed twice: fist bottom-up and then top-down. In each traversal for each octant, we perform calculations that involve other octants in its interaction lists.

One difficulty in FMM codes is the efficient mathematical representation and implementation of the octant–octant interaction matrices so that we achieve algorithmic efficiency without compromising accuracy. The size of the matrices varies, but typical dimensions are 100–1000, depending on the implementation. FMM algorithm designers seek to reduce the size of the matrices, to employ symmetries to reduce storage needs, to use precomputation,

and to use fast approximations (e.g., fast Fourier transforms or low-rank singular value decompositions). All we need to remember here is that once we fix the desired accuracy, the computation of the interactions between an octant β and the octants in its interaction lists can be completed in $\mathcal{O}(1)$ time.

Now let us give the precise definition of these lists. For each octant β in the tree, we create four lists of octants, the so-called U-, V-, W- and X-list.[3] The U-list is defined only for leaf octants. For a leaf octant β, the U-list of β consists of all leaf octants adjacent to β, including β itself. (We say that α is adjacent to β if β and α share a vertex, an edge, or a face.) For any octant β, its *colleagues* are defined as the adjacent octants that are in the same tree level. The V-list of an octant β (leaf or non-leaf) consists of those children of the colleagues of β's parent octant, $P(\beta)$, which are not adjacent to β. The W-list is only created for a leaf octant β and contains an octant α if and only if α is a descendant of a colleague of β, α is not adjacent to β, and the parent of α is adjacent to β. The X-list of an octant β consists of those octants α which have β on their W-list. These definitions are summarized in Table 1.

For each octant $\beta \in T$, we store its level ℓ and two vectors, u and d. The u vector is an approximate representation of the potential generated by the source densities inside β. This representation is sufficiently accurate only if the evaluation particle is *outside the volume covered by β and the colleagues of β*. For the u vector we will use the name "upward equivalent density" or simply "upward density." This terminology is motivated by Ying et al.[28, 29]

The d vector is an approximate representation of the potential generated by the source densities *outside the volume covered by β and colleagues of β*. This approximate representation is sufficiently accurate only if the evaluation particle is *enclosed by β*. For the d vector we will use the name "downward equivalent density" or simply "downward density." For leaf octants ($\beta \in L$), we also store x, s, and f. Given the above definitions, approximately evaluating the sum in Equation (1) involves an upward computation pass of T, followed by a downward computation pass of T, seen more specifically in Algorithm 1. Let us reiterate that the details of the `"interact"` step in Algorithm 1 depend on list type and the underlying FMM scheme.

Table 1. Notation. Here we summarize the main notation used to define the interaction lists for each octant in the FMM tree. (The symbol "\" denotes standard set exclusion.) Note that $\alpha \in \mathcal{W}(\beta)$ need not be a leaf octant; conversely, since $\mathcal{W}(\beta)$ exists only when $\beta \in L$, $\alpha \in \mathcal{X}(\beta)$ implies that $\alpha \in L$. We say that α is adjacent to β if β and α share a vertex, an edge, or a face. See Figure 1 for an example of the U, V, W, and X lists in two dimensions.

α, β	Octants in the FMM tree
ℓ	FMM tree level
q	Maximum number of particles/octant
Octant lists	
T	The FMM tree
L	All leaf octants
$P(\beta)$	Parent octant of β ($\emptyset$ for root of T)
$\mathcal{A}(\beta)$	All ancestor octants of β
$\mathcal{K}(\beta)$	The eight children octants of β ($\emptyset$ for $\beta \in T$)
$\mathcal{D}(\beta)$	All descendant octants of β in T
$\mathcal{C}(\beta)$ (colleagues)	Adjacent octants to β, same level
$\mathcal{J}(\beta)$	Adjacent octants to β (arbitrary level)
U-list: $\alpha \in \mathcal{U}(\beta)$	$\alpha \in L$ adjacent to β (note: $\beta \in \mathcal{U}(\beta)$)
V-list: $\alpha \in \mathcal{V}(\beta)$	$\alpha \in \mathcal{K}(\mathcal{C}(P(\beta)))\backslash\mathcal{C}(\beta)$
W-list: $\alpha \in \mathcal{W}(\beta)$	$\alpha \in \mathcal{D}(\mathcal{C}(\beta))\backslash\mathcal{J}(\beta)$, $P(\alpha) \in \mathcal{J}(\beta)$
X-list: $\alpha \in \mathcal{X}(\beta)$	iff $\beta \in \mathcal{W}(\alpha)$
$\mathcal{I}(\beta)$ (interaction)	$\mathcal{V}(\beta) \cup \mathcal{U}(\beta) \cup \mathcal{W}(\beta) \cup \mathcal{X}(\beta)$

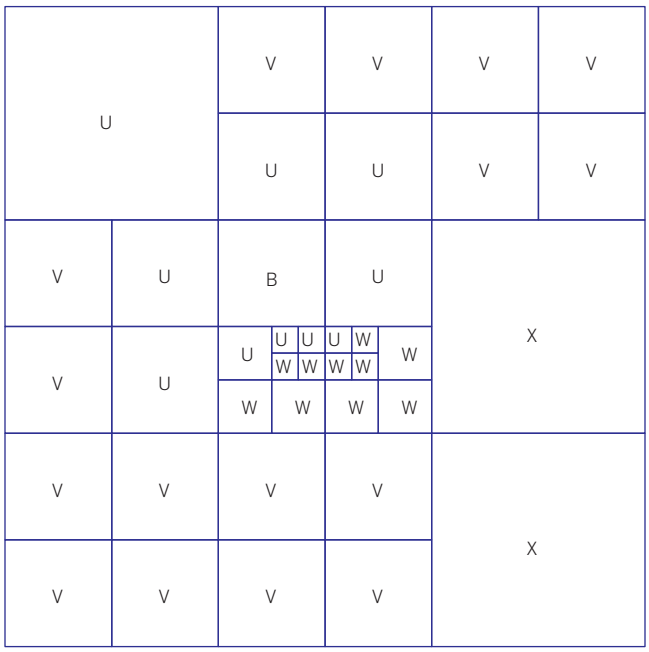
2.1. Parallelizing the evaluation phase

Understanding the dependencies in Algorithm 1 and devising efficient parallel variants can be accomplished by using distributed and shared memory programming models. These models can then be implemented using MPI, OpenMP, and GPU application programming interfaces.

There are multiple levels of concurrency in FMM: across steps (e.g., the S2U and ULI steps have no dependencies), within steps (e.g., a reduction on octants of an octant β during the VLI step), and vectorizations of the per-octant calculations. In fact, one can pipeline the S2U, U2U, and VLI steps to expose even more parallelism. We do not take advantage of pipelining because it reduces the modularity and generality of the implementation.

The generic dependencies of the calculations outlined in Algorithm 1 are as follows: The Approximate Interactions (except for steps (5a) and (5b)) and Direct Interactions parts can be executed concurrently. For the approximate interaction calculations, the order of the steps denotes dependencies, for example, step (2) must start after step (1) has been completed. Steps (3a) and (3b) can be executed in any order. However, concurrent execution of steps (3a) and (3b) requires a concurrent write with accumulation. This is also true for the steps (5a) and (5b), they are independent up to concurrent writes.

Figure 1. FMM lists. Here we show an example of the U, V, W, and X lists in two dimensions for a tree octant *B*. One can verify that $\mathcal{I}(B)$ is inside the domain enclosed by $\mathcal{C}(P(B))$.



Algorithm 1. $\{f_i\}_{i=1}^N = \text{FMM}\left(\{x_i, y_i, s_i\}_{i=1}^N, \text{octree}\right)$

// *APPROXIMATE INTERACTIONS*
// **(1) S2U: source-to-up step**
$\forall \beta \in L$: `source to up densities interaction`
// **(2) U2U: up-to-up step (upward)**
`Postorder traversal of` T
$\forall \beta \in T$: `interact` $(\beta, P(\beta))$
// **(3a) VLI : V-list step**
$\forall \beta \in T: \forall \alpha \in \mathcal{V}(\beta)$ `interact` (β, α) ;
// **(3b) XLI : X-list step**
$\forall \beta \in T: \forall \alpha \in \mathcal{X}(\beta)$: `interact` (β, α) ;
// **(4) D2D: down-to-down step (downward)**
`Preorder traversal of` T
$\forall \beta \in T$: `interact` (β, α) ;
// **(5a) WLI: W-list step**
$\forall \beta \in L: \forall \alpha \in \mathcal{W}(\beta)$: `interact` (β, α) ;
// **(5b) D2T : down-to-targets step**
$\forall \beta \in L$: `evaluate potential on` $x_i \in \beta$;

//*DIRECT INTERACTIONS*
// **ULI: U-list step (direct sum)**
$\forall \beta \in L: \forall \alpha \in \mathcal{U}_\beta$: `interact` (β, α) ;

Our overall strategy is to use MPI-based distributed memory parallelism to construct the global tree and to partition it into overlapping subtrees (the LETs) in order to remove dependencies between steps (3a)/(5a) and (3b)/(5b). We handle the concurrent writes explicitly and we use shared-memory-based parallelism on GPUs within each MPI process to accelerate the direct interactions and steps (1), (3), (5) of the indirect interactions in Algorithm 1. In the following sections, we give the details of our approach.

3. DISTRIBUTED MEMORY PARALLELISM

The main components of our scheme are (1) *the tree construction*, in which the global FMM tree is built and each process receives its locally essential tree and a set of leaves for which it assumes ownership and (2) *the evaluation*, in which each process evaluates the sum at the particles of the leaf octants it owns. This sum has two components: direct interactions (evaluated exactly) and indirect interactions (evaluated approximately).

The input consists of the particles and their source densities. The output of the algorithm is the potential at the particles. Before we describe the algorithm, we need the following definition:[27]

Locally essential tree (LET): Given a partition of L across processes so that L_k is the set of leaf-octants assigned to the process k (i.e., the potential in these octants is computed by process k), the LET for process k is defined as the union of the interaction lists of all owned leaves and their ancestors:

$$\text{LET}(k) := \cup_{\forall \beta \in [L_k \cup \mathcal{A}(L_k)]} \mathcal{I}(\beta).$$

The basic idea[27] for a distributed memory implementation of the FMM algorithm is to partition the leaves of the FMM tree across processes, construct the LET of each process, and then compute the N-body sum in parallel. There are two communication-intensive phases in this approach: the first phase is the LET construction and the second phase is an all-reduce during evaluation. Next, we discuss the main components in these two phases.

3.1. Tree construction

The inputs for the tree construction procedure are the particles. The output is the local essential tree on each process, which is subsequently used in the computation, along with geometrical domain decomposition of the unit cube across MPI processes. The latter is used throughout the algorithm. The tree construction involves (1) the construction of a distributed linear complete octree that contains only the leaf octants and (2) the construction of the per-process LETs.

We start by creating the distributed and globally Morton-sorted array containing all the leaves from the global tree. The algorithms for this task are known (we use the one described in Sundar et al.;[24] see also Hariharan and S. Aluru[13]), and their main ingredient is the parallel Morton-sort of particles. This sort determines the overall complexity of the tree construction.

The distribution of the leaves between processes induces a geometric partitioning of the unit cube: each process controls the volume covered by the leaves it owns. By Ω_k, we will denote the region "controlled" by process k. Each process stores the overall geometric partitioning: we use an `MPI_AllGather` to exchange the first and last octants of their region, which is all is needed to define the partition.

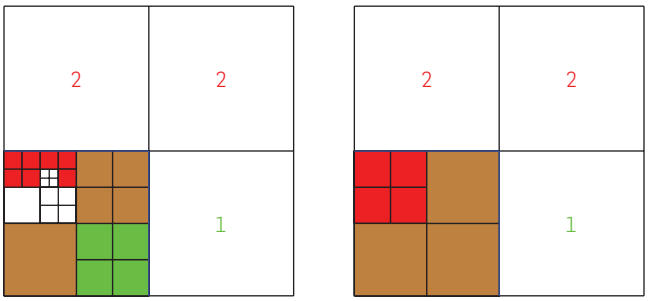
Next, each process adds all ancestor octants to its local leaves, thus creating a local tree. The task of exchanging information about "ghost" octants still remains to be completed. To accomplish that, let us introduce the following definitions:

- "*Contributor*" processes of an octant $\beta \in T$:
  $\mathcal{P}_c(\beta) := k \in 1 \ldots p : \beta$ overlaps with Ω_k
- "*User*" processes of an octant β:
  $\mathcal{P}_u(\beta) := k \in 1 \ldots p : P(\beta)$ or $\mathcal{C}(P(\beta))$ overlaps with Ω_k.

Let $I_{kk'}$ be the set of octants to which process k contributes and which process k' uses. Then, process k must send all octants in $I_{kk'}$ to MPI process k'. Figure 2 provides an illustration of this procedure. That is, each process contributor of each octant sends the octant data to all users of that octant. Once the exchange of the $I_{kk'}$ lists has been completed, all MPI processes insert received octants into their local trees. This concludes the construction of the per-process LETs. We summarize the construction of the local essential trees in Algorithm 2.

The correctness of the construction is based on the direct relation between LET and FMM. Consider the potential generated by the sources enclosed by some octant β. In order to evaluate this potential outside the volume covered by $\mathcal{C}(P(\beta))$, one does not need information regarding sources or upward densities associated with β, since the upward density of some ancestor of β would be used instead. This observation can be used to formalize the correctness of the LET construction. See Lashuk et al.[17] for a more detailed discussion.

Figure 2. Communication of ghost octants. Process 0 sends green octants to process 1, red octants to process 2, and brown octants to both 1 and 2. White octants in lower-left corner are "internal" to process 0 and not sent to anyone. The procedure is applied to both leaves and non-leaf octants. (a) Finer level of octree. (b) Coarser level of octree.



(a) (b)

Algorithm 2. LET construction

Input: *distributed set of particles x;*
Output: *LET on each process k*

1. L_k = `Points2Octree`(x) //**MPI**
2. $B_k = L_k \cup \mathcal{A}(L_k)$
3. $I_{kk'} := \{\beta \in B_k : P(\beta)$ or $\mathcal{C}(P(\beta))$ overlaps with $\Omega_{k'}\}$
4. $\forall k' : k' \neq k$
   Send $I_{kk'}$ *to process* k' //**MPI**
   Recv $I_{k'k}$ *from process* k' //**MPI**
   Insert $I_{k'k}$ *in* B_k
5. *Return Bk*

In the last step of Algorithm 2, every process independently builds U, V, W, and X lists for the octants in its LET which enclose the local particles (where the potential is to be evaluated). All necessary octants are already present in the LET, so no further communication is required in this step.

3.2. Load balancing

Assigning each process an equal chunk of leaves may lead to a substantial load imbalance during the interaction evaluation for nonuniform octrees. In order to overcome this difficulty, we use the following load-balancing algorithm.

After the LET setup, each leaf is assigned a weight, based on the computational work associated with its U, V, W, and X lists (which are already built at this particle). Then, we repartition the leaves to ensure that total weight of the leaves owned by each process is approximately equal. We use Algorithm 1 from Sundar et al.[24] to perform this repartitioning in parallel. Note that similarly to the previous section, each process gets a contiguous chunk of global (distributed) Morton-sorted array of leaves. In the final step, we rebuild the LET and the U, V, W, and X lists on each process.

Note that we repartition the leaves based solely on work balance ignoring the communication costs. Such an approach is suboptimal, but is not expensive to compute and works reasonably well in practice. In addition to leaf-based partitioning, the partitioning at a coarser level can also be considered;[25] we have not tested this approach.

3.3. FMM evaluation

Three communication steps are required for the potential evaluation. The first is to communicate the source densities for the U-list calculation. This communication is "local" in a sense that each process typically communicates only with its spatial neighbors. Thus, a straightforward implementation (say, using `MPI_Isend`) is acceptable.

The second communication step is to sum up the upward densities of all the contributors of each octant, the U2U step. During this step (bottom-up traversal), each process only builds *partial* upward densities of octants in its LET. The partial upward densities of an octant do not include contributions from descendants of the octant that belong to other MPI processes. For this reason, we need a third step, a communication step, to collect the upward densities to the users of each octant. This communication step must take place after the U2U step and before the VLI and XLI steps. Once this step is completed, every process performs a top-down traversal of its LET without communicating with other processes.

Algorithm 3 describes a communication procedure that combines the second and the third steps mentioned above. This algorithm resembles the standard "AllReduce" algorithm on a hypercube. What follows is the informal description of the algorithm.

At the start, each processor r forms a pool $\mathcal{S}$ of octants (and their upward densities) which are "shared", that is, not used by r alone. The MPI communicator (set of all MPI processes) is then split into two halves (for simplicity, we shall assume that the communicator size is a power of two) and each processor from the first half is paired with a "peer" processor from the other half.

Now consider those octants in $\mathcal{S}$ which are "used" by some processor from the other half of the communicator (not necessarily a peer of r). If such octants exist, r sends them (together with upward densities) to its "peer". According to the same rule, the "peer" will send some (possibly none) octants and densities to r. The received octants will be merged into $\mathcal{S}$, eliminating duplicate octants while *summing* densities for duplicate octants.

At this point, no further communication between the two halves is required, and the algorithm proceeds recursively by applying itself to each of the halves. We finally note that after each communication round, $\mathcal{S}$ is purged from the "transient" octants which are no longer necessary for communications and not used locally at r.

The time complexity of this algorithm is not worse than $\mathcal{O}(\sqrt{p})$. To be more specific, assuming that no process uses more than m shared octants and no process contributes to more than m shared octants, for a hypercube interconnect, the communication complexity of Algorithm 3 is $\mathcal{O}(t_s \log p + t_w m(3\sqrt{p}-2))$, where t_s and t_w are the latency and the bandwidth constants, respectively. See Lashuk et al.[17] for a proof.

After the three communication steps, all the remaining steps of Algorithm 1 can be carried out without further communication.

3.4. Complexity for uniform distributions of particles

We have all the necessary ingredients to derive the overall complexity of the distributed memory algorithm for uniform

distributions of particles. Let N be the number of particles and let p be number of processes. The number of octants is proportional to the number of particles. The first communication cost is associated with the parallel sort of the input particles. We have used a comparison sort instead of binning, so its time complexity is $\mathcal{O}\left(\frac{N}{p}\log\frac{N}{p}+p\log^2 p\right)$ (combination of sample sort and bitonic sort).[6] Exchanging the "ghost" octants has the same complexity as the reduce-broadcast algorithm described in Section 3.3, that is, $\mathcal{O}(\sqrt{p}m)$, where m is the maximal number of "shared" octants between two processes. For a uniform grid, m can be estimated as $\mathcal{O}((N/p)^{2/3})$ divided by p. The communication also includes the exchange of source densities. Assuming that the bandwidth of the system is reasonably high, we can neglect all lower order communication terms. In summary (assuming large enough bandwidth but assuming nothing about latency), the overall complexity of the setup phase is $\mathcal{O}\left(\frac{N}{p}\log\frac{N}{p}\right)+\mathcal{O}(p\log^2 p)$. For the evaluation, we have $\mathcal{O}\left(\frac{N}{p}\right)+\mathcal{O}(\log p)$.

Algorithm 3. Reduce and Scatter

Input: *partial upward densities of "shared" octants at "contributor" processes*
Input: *r (rank of current process); assume communicator size is* 2^d
Output: *upward densities of "shared" octants at "user" processes*

//Define shared octants (for each process):
$\mathcal{S}=\{\beta\in LET:\#(\mathcal{P}_u(\beta)\cup\mathcal{P}_c(\beta))>1\}$
//Loop over communication rounds (hypercube dimensions)
For $i := d-1$ *to* 0
  //Process s is our partner during this communication round
  1. $s := r$ *XOR* 2^i
  2. $u_s = s$ *AND* (2^d-2^i)
  3. $u_e = s$ *OR* (2^i-1)
  4. Send to s : $\left\{\beta\in\mathcal{S}:\mathcal{I}(\beta)\cap\left(\cup_{u_s}^{u_e}\Omega_s\right)\neq\emptyset\right\}$
  5. $q_s = r$ *AND* (2^d-2^i)
  6. $q_e = r$ *OR* (2^i-1)
  7. Delete $\left\{\beta\in\mathcal{S}:\mathcal{I}(\beta)\cap\left(\cup_{q_s}^{q_e}\Omega_s\right)\neq\emptyset\right\}$
  //Reduction
  8. Recv from s and append $\mathcal{S}$
  9. Remove duplicates for $\mathcal{S}$
  10. Sum up densities for duplicate octants.

For a nonuniform particle distribution, the complexity estimates for the setup and evaluation phases include an additional $t_w m\sqrt{p}$ term. Since we do not have a (nontrivial) bound on m, this result is worse than what is theoretically possible by Teng's algorithm. (He partitions the tree using both work and communication costs instead of just using the work at the leaf octants.)

4. INTRA-NODE PARALLELISM

A key ingredient to the efficiency of the overall algorithm is highly tuned intra-node performance. Many current supercomputing systems have heterogeneous nodes, meaning they have both conventional general-purpose multicore CPU processors and more specialized high-speed co-processors, such as GPUs. Consequently, our implementation can exploit both CPUs (via OpenMP) and GPUs (via CUDA).

Shared memory parallelization is common for either type of processor. For the S2U, D2T, ULI, WLI, VLI, and XLI steps, we can visit the octants in an embarrassingly parallel way. Moreover, all octant visits include octant-to-octant or octant-to-particle interactions expressed as matrix–vector multiplications. The matrices are dense except for the VLI calculations, which correspond to a diagonal translation. Thus, overall there are two levels of parallelism: across octants and across the rows of the corresponding matrix. The same approach to parallelism applies on both CPU and GPU co-processors, because they have essentially the same architectural features for exploiting parallelism: shared memory address spaces, a multilevel memory hierarchy, and vector units for regular data parallelism. The U2U and D2D remain sequential in our current implementation, though they could in principle be parallelized using rake and compress methods.[14]

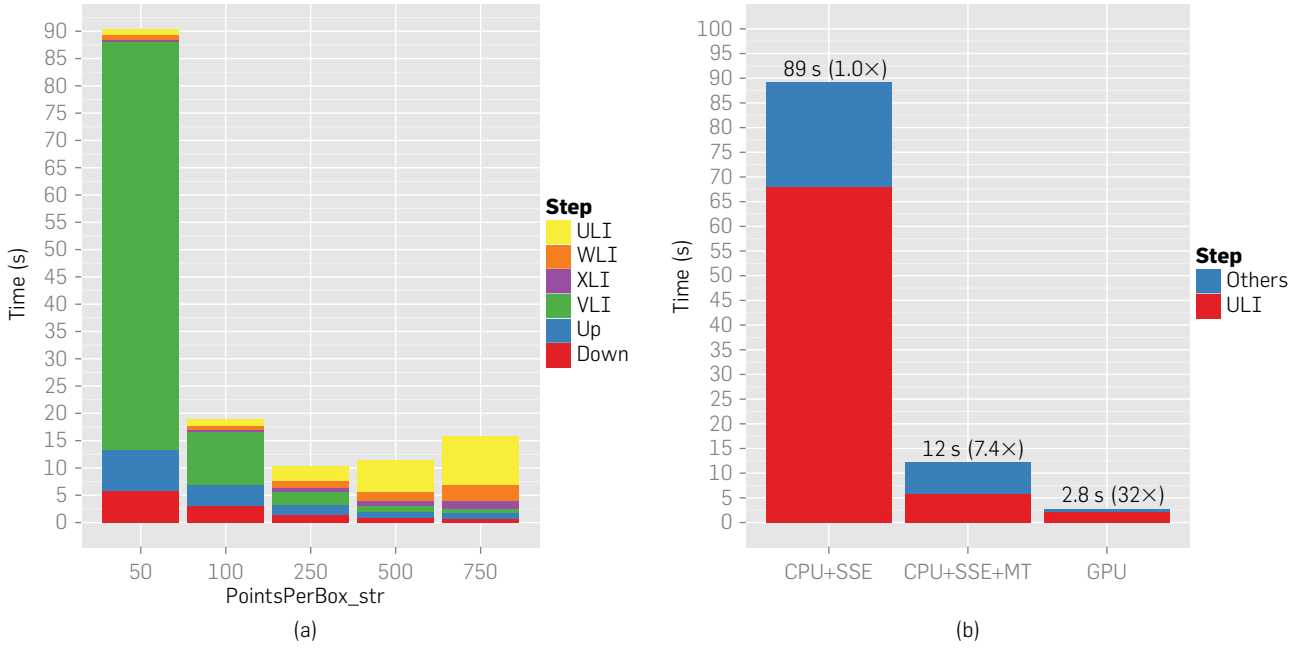
Although the architectural features are similar between CPUs and GPUs, differences in their *scale* directly affect the algorithm and implementation. The main algorithmic tuning parameter for the FMM is the maximum number of particles per octant, q. Increasing q makes the leaf octants larger and the overall tree shorter, which in turn increases the number of flops performed by the compute-bound ULI, WLI, and XLI steps, while decreasing the flop-cost of the memory-bound VLI and other phases. Figure 3(a) shows an example of this behavior. A processor's relative balance of performance and bandwidth will change these curves, thereby changing the optimal value of q.

Indeed, this phenomenon gives rise to the overall performance behavior shown in Figure 3(b), where we compare the single-socket execution times for three single-socket implementations: (i) single-threaded CPU with explicit vectorization using SSE intrinsics; (ii) multithreaded CPU with four threads and SSE; and (iii) GPU using CUDA. Note that combined threading and vectorization yields a 7.4× improvement over the single-threaded code. The GPU code is 32× faster than the single-threaded CPU code and 4.3× faster than the multithreaded CPU code, even including time to send/receive data from the GPU. Interestingly, the GPU win comes not just from parallelization and tuning but because faster processing enables better *algorithmic* tuning of q. In particular, the faster we can perform the compute-intensive ULI step, the larger we can make q thereby reducing the VLI and other memory-bound steps. In other words, there is a synergy between faster processing and algorithmic tuning in which we can trade more flops (ULI step) for less memory communication (e.g., VLI step). Such techniques are likely to apply more generally as many-core processors increase in use.

5. NUMERICAL EXPERIMENTS

This section evaluates the overall scalability of our implementation on two different architectures and in both strong and weak scaling regimes. (We have used the Stokes kernel instead of the Laplacian. $K(r)$ is defined as $1/|r|(1-r\otimes r)/$

Figure 3. Intra-node optimization. (a) Optimal particles per octant: increasing the leaf sizes increases the relative costs of ULI, WLI, and XLI while decreasing the costs of the other phases. (b) Implementation comparison: solid horizontal bars indicate the observed wall-clock time on each platform. In the hybrid CPU+GPU implementation, ULI runs on the GPU while the other steps run on the CPU, yielding an overall benefit from overlapping.



r^2), where $\otimes$ is the outer product between two three-dimensional vectors.)

Platforms and architectures: We tested our code on two supercomputing platforms, one based on conventional CPUs and the other based on hybrid CPU+GPU nodes. The CPU system is *Jaguar PF* at the National Center for Computational Sciences (UT/ORNL), (http://www.nccs.gov/computing-resources/jaguar/) a Cray XT5 with 224,256 cores (using 2.6-GHz hex-core AMD Opteron CPUs, 2GB/core), and a three-dimensional-torus network (9.6 GB/s link bandwidth). Jaguar is ranked third on the Top 500 List (www.top500.org) as of June 2011. The GPU system is the *Keeneland* initial delivery system, also at UT/ORNL (http://keeneland.gatech.edu). Keeneland is based on the Hewlett-Packard SL390 servers accelerated with NVIDIA Tesla M2070 GPUs. Keeneland has 120 compute nodes, each with dual-socket, six-core Intel X5660 2.8-GHz Westmere processors and 3 GPUs per node, with 24GB of DDR3 host memory. Nodes are interconnected with single rail QDR Infiniband.

MPI, strong scalability tests on Jaguar: Figure 4(a) summarizes the strong scaling experiments. The problem has 100 million unknowns. We use a nonuniform (line) distribution. On 24,576 cores, the evaluation phase takes just 1.7 s, which constitutes a speedup of 205× when increasing cores by a factor of 512×. The setup phase (tree construction) begins to dominate at this scale in the strong-scaling regime.

MPI, weak scalability tests on Jaguar: Figure 4(b) summarizes the weak scaling results. The problem size per core is kept fixed to approximately 672,000 unknowns. The FMM evaluation phase and setup phase (tree construction) times increase because of several factors. First, since we have highly nonuniform distributions (the max leaf-level is 23 and the min is 4—for a uniform tree, the leaf level would be 9), as we increase the problem size, the tree gets deeper and the cost per core increases (i.e., we observe a $\mathcal{O}(N \log N)$ scaling), as we have not reached the asymptotic $\mathcal{O}(N)$ phase of the FMM. Secondly, for nonuniform trees, it is difficult to load balance all phases of FMM. The solution to these scaling problems is to employ the hypercube-like tree-broadcast algorithm for all of the phases of FMM. (Currently it is used only in the postorder communication phase of the evaluation phase.) Finally, the setup phase is not multithreaded, which is the subject of future work. Nevertheless, we still attain a reasonably good utilization of compute resources: the evaluation phase sustains over 1.2 GFlopsper core (in double-precision).

GPU strong scalability tests on Keeneland: Figure 5 shows strong scaling for a 48 million particle problem on up to 192 GPUs of the Keeneland system. We use 1 MPI process per GPU and 3 GPUs per node (64 nodes total). In the best case, the evaluation completes in just 0.47 s. For comparison, we estimate that the equivalent purely direct ($\mathcal{O}(n^2)$) calculation on 192 GPUs, ignoring all communication, would require approximately 1000× more wall clock time, which confirms the importance of employing parallelism on an algorithmically optimal method.

6. DISCUSSION AND CONCLUSION

We have presented several algorithms that taken together expose and exploit concurrency at all stages of the fast multipole algorithms and employ several parallel programming

Figure 4. Strong and weak scalability on Jaguar PF. (a) Strong Scaling on Jaguar PF. The strong scalability result for 22M particles. Each bar is labeled by its absolute wall-clock time (seconds) and speedup relative to the 48-core case. There are six cores (and six OpenMP threads) per MPI process. The finest level of the octree is nine and the coarsest is three. The evaluation and setup phases show 205× and 48× speedups, respectively, when increasing the core count by 512×. (b) Weak Scaling on Jaguar PF. The weak scalability of the simulation to 196,608 cores, with approximately 672,000 unknowns per core. Each bar is labeled by wall-clock time (seconds) and parallel efficiency. We use one MPI process per socket and all of the six OpenMP threads in each MPI process. The finest to coarsest octree levels range from 24 to 4. Our implementation maintains parallel efficiency levels of 60% or more at scale.

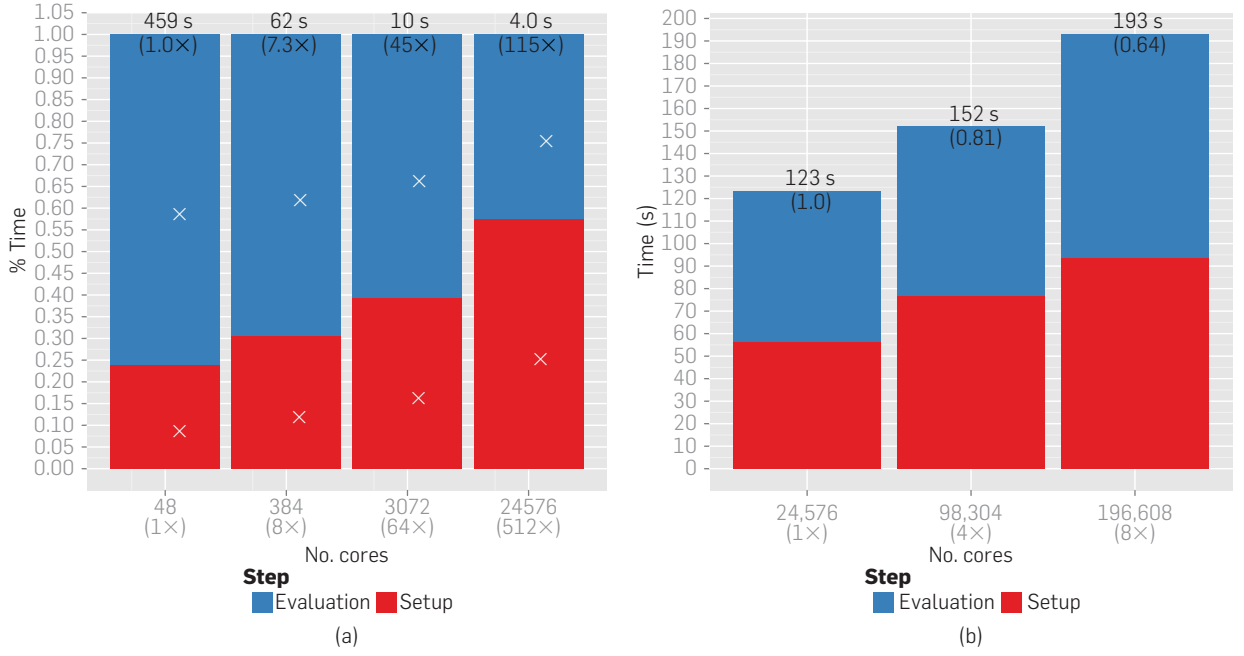
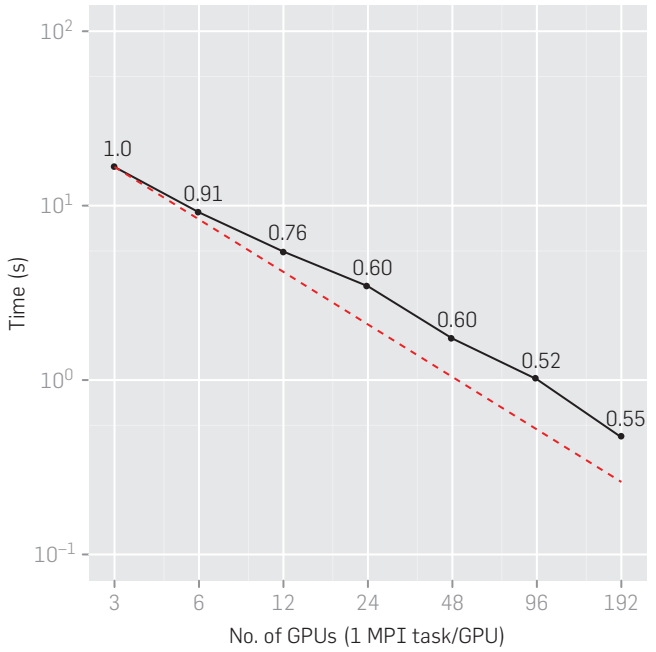


Figure 5. GPU strong scaling. We show strong scaling for FMM evaluation of 48 million particles on up to 192 GPUs of Keeneland system, which completes in as few as 0.47 s. The solid point-line shows the measured wall-clock time, and the dashed line shows the ideal time under perfect speedup. Each point is labeled by its parallel efficiency (ideal = 1).



paradigms. We showed that we can efficiently scale the tree setup with the major cost being the parallel sort, which in turn exhibits textbook scalability. We described a new reduction scheme for the FMM algorithm and we demonstrated overall scalability. We explored per-core concurrency using the streaming paradigm on GPU accelerators with excellent speedups. FMM is a highly nontrivial algorithm with several different phases, a combination of multiresolution data structures, fast transforms, and highly irregular data access. Yet, we were able to achieve significant speedups on heterogeneous architectures.

On our largest run on 196,608 cores, we observed about 220 TFlops/s in the evaluation, about 15% of the Linpack sustained performance. Assuming a 7× speedup on a hybrid machine of similar size means that our FMM would exceed one PetaFlop without further modifications. Our main conclusion is that it is possible to construct highly efficient scalable *N*-body solvers, at least up to hundreds of thousands of cores. But what about one million or many million cores?

One important lesson for us is that hybrid parallelism is absolutely necessary; once we reached several thousands of MPI processes, MPI resource management in the communication-intensive phases of the algorithm became an issue. Employing shared memory parallelism within the socket and combining with accelarators results in a more scalable code. The maintainability overhead, however, can be significant especially for the GPU case and for

a complex code like FMM. Currently, we are working on introducing shared-memory parallelism in the tree construction, sharing the workload between CPUs and GPUs (essentially treating the GPU as another MPI process), and introducing shared-memory parallelism in the upward and downward computations.

Acknowledgments

This work was supported in part by the U.S. National Science Foundation (NSF) grants CNS-0929947, CCF-0833136, CAREER-0953100, OCI-0749285, OCI-0749334, OCI-1047980, CNS-0903447 and Semiconductor Research Corporation (SRC) award 1981, U.S. Department of Energy (DOE) grant DEFC02-10ER26006/DE-SC0004915, and a grant from the U.S. Defense Advanced Research Projects Agency (DARPA). Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of NSF, SRC, DOE, or DARPA. Computing resources on the TeraGrid systems were provided under the TeraGrid allocation grants ASC-070050N, CCR-090024, ASC-100019, and MCA-04N026. We would like to thank the TeraGrid support staff, and also the staff and consultants at NCSA, TACC, and NICS from whom we have received significant assistance. C

References

1. Barnes, J., Hut, P. A hierarchical O(N logN) force-calculation algorithm. *Nature 324*, 4 (December 1986), 446–449.
2. Callahan, P.B., Kosaraju, S.R. Algorithms for dynamic closest pair and n-body potential fields. In *Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '95* (Philadelphia, PA, USA, 1995), Society for Industrial and Applied Mathematics, 263–272.
3. Carrier, J., Greengard, L., Rokhlin, V. A fast adaptive multipole algorithm for particle simulations. *SIAM J. Sci. Stat. Comput. 9*, 4 (1988), 669–686.
4. Cherrie, J., Beatson, R.K., Newsam, G.N. Fast evaluation of radial basis functions:methods for generalized multiquadrics in r^n. *SIAM J. Sci. Comput. 23*, 5 (2002), 1549–1571.
5. Darve, E., Cecka, C., Takahashi, T. The fast multipole method on parallel clusters, multicore processors, and graphics processing units. *Comptes Rendus Mécanique 339*(2–3) (2011), 185–193.
6. Grama, A., Gupta, A., Karypis, G., Kumar, V. An Introduction to Parallel Computing: Design and Analysis of Algorithms, 2nd edn, Addison Wesley, 2003.
7. Grama, A.Y., Kumar, V., Sameh, A. Scalable parallel formulations of the Barnes–Hut method for n-body simulations. In *Proceedings of the 1994 conference on Supercomputing, Supercomputing '94* (Los Alamitos, CA, USA, 1994), IEEE Computer Society Press, 439–448.
8. Gray, A., Moore, A. N-Body'problems in statistical learning. *Adv. Neural Inform. Process Syst.* (2001), 521–527.
9. Greengard, L., Gropp, W.D. A parallel version of the fast multipole method. *Comput. Math. Appl. 20*, 7 (1990), 63–71.
10. Greengard, L., Rokhlin, V. A fast algorithm for particle simulations. *J. Comput. Phys. 73* (1987), 325–348.
11. Gumerov, N.A., Duraiswami, R. Fast multipole methods on graphics processors. *J. Comput. Phys. 227*, 18 (2008), 8290–8313.
12. Hamada, T., Narumi, T., Yokota, R., Yasuoka, K., Nitadori, K., Taiji, M. 42 TFlops hierarchical N-body simulations on GPUs with applications in both astrophysics and turbulence. In *Proceedings of SC09*, The SCxy Conference Series (Portland, Oregon, November 2009), ACM/IEEE.
13. Hariharan, B., Aluru, S. Efficient parallel algorithms and software for compressed octrees with applications to hierarchical methods. *Parallel Comput. 31* (3–4) (2005), 311–331.
14. JáJá, J. An Introduction to Parallel Algorithms, Addison Wesley, 1992.
15. Kurzak, J., Pettitt, B.M. Massively parallel implementation of a fast multipole method for distributed memory machines. *J.Parallel Distr. Comput. 65*, 7 (2005), 870–881.
16. Greengard, L. Fast algorithms for classical physics. *Science 265*, 5174 (1994), 909–914.
17. Lashuk, I. et al. A massively parallel adaptive fast-multipole method on heterogeneous architectures. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, SC '09* (New York, NY, USA, 2009), ACM, 58:1–58:12.
18. Miller, G., Teng, S., Thurston, W., Vavasis, S. Separators for sphere-packings and nearest neighbor graphs. *J. ACM (JACM) 44*, 1 (1997), 1–29.
19. Ogata, S. et al. Scalable and portable implementation of the fast multipole method on parallel computers. *Comput. Phys. Comm. 153*, 3 (2003), 445–461.
20. Phillips, J.C., Stone, J.E., Schulten, K. Adapting a message-driven parallel application to GPU-accelerated clusters. In *SC '08: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing* (2008), 1–9.
21. Rahimian, A., Lashuk, I., Veerapaneni, S., Chandramowlishwaran, A., Malhotra, D., Moon, L., Sampath, R., Shringarpure, A., Vetter, J., Vuduc, R., Zorin, D., Biros, G. Petascale direct numerical simulation of blood flow on 200k cores and heterogeneous architectures. In *SC '10: Proceedings of the 2010 ACM/IEEE Conference on Supercomputing* (Piscataway, NJ, USA, 2010), IEEE Press, 1–12.
22. Rokhlin, V. Rapid solution of integral equations of classical potential theory. *J. Comput. Phys. 60* (1985), 187–207.
23. Sevilgen, F., Aluru, S., Futamura, N. A provably optimal, distribution-independent parallel fast multipole method. In *Proceedings of 14th International Parallel and Distributed Processing Symposium, 2000. IPDPS 2000* (2000), IEEE, 77–84.
24. Sundar, H., Sampath, R.S., Biros, G. Bottom-up construction and 2:1 balance refinement of linear octrees in parallel. *SIAM J. Sci. Comput. 30*, 5 (2008), 2675–2708.
25. Teng, S.H. Provably good partitioning and load balancing algorithms for parallel adaptive N-body simulation. *SIAM J. Sci. Comput. 19*, 2 (1998).
26. Teyssier, R. et al. Full-sky weak-lensing simulation with 70 billion particles. *Astron. Astrophys. 497*, 2 (2009), 335–341.
27. Warren, M.S., Salmon, J.K. A parallel hashed octtree N-body algorithm. In *Proceedings of Supercomputing*, The SCxy Conference Series (Portland, Oregon, November 1993), ACM/IEEE.
28. Ying, L., Biros, G., Zorin, D. A kernel-independent adaptive fast multipole method in two and three dimensions. *J. Comput. Phys. 196*, 2 (2004), 591–626.
29. Ying, L., Biros, G., Zorin, D., Langston, H. A new parallel kernel-independent fast multipole algorithm. In *Proceedings of SC03*, The SCxy Conference Series (Phoenix, Arizona, November 2003), ACM/IEEE.
30. Yokota, R., Bardhan, J., Knepley, M., Barba, L., Hamada, T. Biomolecular electrostatics using a fast multipole BEM on up to 512 GPUs and a billion unknowns. *Comput. Phys. Comm. 182*, 6 (Jun. 2011), 1272–1283.

Ilya Lashuk (lashuk2@llnl.gov), research scientist, Institute for Scientific Computing Research, Lawrence Livermore National Laboratory, Livermore, CA.

Aparna Chandramowlishwaran (aparna@cc.gatech.edu), graduate research assistant, Computational Science and Engineering Division, College of Computing, Atlanta, GA.

Harper Langston (harper@cc.gatech.edu), postdoctoral associate, Computational Science and Engineering Division, College of Computing, Atlanta, GA.

Tuan-Anh Nguyen (tuananh@cc.gatech.edu), graduate research assistant, Computational Science and Engineering Division, College of Computing, Atlanta, GA.

Rahul Sampath (rahul.sampath@gmail.com), postdoctoral associate, Oak Ridge National Laboratory, Oak Ridge, TN.

Aashay Shringarpure (aashay.shringarpure@gmail.com).

Richard Vuduc (richie@cc.gatech.edu), assistant professor, Computational Science and Engineering Division, College of Computing, Atlanta, GA.

Lexing Ying (lexing@math.utexas.edu), associate professor, Mathematics, The University of Texas at Austin, TX.

Denis Zorin (dzorin@cs.nyu.edu), professor, Courant Institute of Mathematical Sciences. New York University, New York, NY.

George Biros (gbiros@acm.org), W.A. "Tex" Moncrief, Jr. Simulation-Based Engineering Sciences Chair, Institute of Computational Engineering and Sciences, The University of Texas at Austin, TX.

DOI:10.1145/2160718.2160741

Technical Perspective
An Experiment in Determinism

By Steven Hand

THE ADVENT OF multicore processors means that complex software is becoming more complex: developers must turn to the use of multiple threads to exploit hardware parallelism, yet it is widely held that parallel programming is far more difficult and error prone than writing sequential code. In particular, the myriad allowable interleavings of thread execution mean that different runs of a program can produce different behaviors, including different outputs or—worse—bugs!

To address this concern, the recent past has seen considerable interest in *deterministic parallelism*: techniques to ensure that multiple executions of a program observe the same interleavings, regardless of variations in the hardware, the vagaries of the scheduler, the use of synchronization primitives, or the presence or absence of external workloads. The hope is that deterministic parallelism will make it easier to have confidence in testing software quality: for example, if a bug can manifest due to a particular ordering of read and writes to objects in memory, then deterministic parallelism will ensure that this either *always* happens—thus making it easier to debug—or ensure it *never* happens, avoiding the bug altogether.

Some previous work has investigated providing deterministic parallelism in the language (Deterministic Parallel Java), at the level of the compiler and/or runtime system (Kendo, CoreDet, dThreads), or within the hardware (DMP, RCDC). Most of these schemes do little to make parallel programming inherently simpler, and most are limited to multithreaded (rather than multiprocess) systems. In the following paper, however, Aviram et al. explore the idea of building an *operating system* that inherently enforces deterministic parallelism both within and across all hosted programs.

Most traditional operating systems provide interfaces that are nondeterministic; for example, creating a new process with `fork()` or opening a file with `open()` result in handles that depend on the set of prior operations performed, either systemwide or within the current process; similar behavior can be seen with memory allocations. The values can directly or indirectly affect control flow and general computation, thus leading to different behaviors even if there are no explicit 'race conditions.' And, of course, explicit nondeterminism is often exposed via operations such as `wait()` on a process or condition variable, or invoking `select()` on a set of file descriptors.

To avoid such issues, the authors take a clean-slate approach with their *Determinator* operating system. In Determinator, rather than threads sharing an address space, each independent thread of execution gets a private workspace. This is a *copy* of the nominal global state, and all of the thread's read and write operations are performed directly on the copy. At thread exit—or certain other well-defined points—the effects of the thread's execution are reconciled with the global state. This can be considered analogous to how modern revision control systems operate: a person "checks out" a private copy of the data, makes whatever changes he or she wishes, and then "checks in" the changes. The use of private workspaces eliminates any nondeterminism involved by fine-grained interleavings; in essence, read/write data races are impossible, since any thread will only ever read values that are canonical, or that it has itself written.

Much like with revision control systems, however, write/write conflicts are possible: two child threads could modify the same (logically shared) location during their execution. In Determinator this can be readily ascertained during the reconciliation phase, and causes a runtime exception. The authors argue that turning write/write conflicts into readily reproducible bugs simplifies the programming model, and avoids any reliance on "benign" data races.

Determinator also controls other aspects of nondeterminism. For example, identifiers returned by the kernel are thread-private rather than global, and all synchronization operations are deterministic, rather than first-come-first-served (like many mutual exclusion locks). This leads to a system that has a quite different "feel" to traditional operating systems, yet the authors demonstrate it is possible to run a number of parallel benchmark programs without any ill effect. They also describe an emulation layer that provides some familiar Unix APIs, albeit with occasionally different semantics. One good example is in the implementation of process synchronization; while `waitpid()` can be easily supported, the nondeterminism of `wait()` cannot be. Instead `wait()` turns into "wait for the child process with the lowest (thread-local) process id" that, in many cases, suffices.

Determinator is an interesting and thought-provoking piece of research. However, it is unclear whether or not future operating systems will be built this way. There are a number of challenges in terms of how to support I/O operations, particularly those that interact with users or remote systems; and in terms of how to support inherently nondeterministic computation, such as generating cryptographic keys.

This is not just a limitation of Determinator, but rather of most work in the field. Although deterministic parallelism has the potential to reduce the number of allowable behaviors a program (or system) can express, which should increase the efficacy of testing, it does not help with the fact that most useful programs behave differently given different *inputs*. Relying on extensive testing to validate the correctness of programs is intellectually unsatisfactory, particularly given the woefully inadequate coverage obtained, and the general lack of any specification. Nonetheless, it is possible that deterministic parallelism will make it all a smidge easier—and that can't be a bad thing, can it?

Steven Hand (Steven.Hand@cl.cam.ac.uk) is a Reader in computer systems in the Computer Laboratory at the University of Cambridge, U.K.

DOI:10.1145/2160718.2160742

Efficient System-Enforced Deterministic Parallelism

By Amittai Aviram, Shu-Chun Weng, Sen Hu, and Bryan Ford

Abstract

Deterministic execution offers many benefits for debugging, fault tolerance, and security. Current methods of executing *parallel* programs deterministically, however, often incur high costs, allow misbehaved software to defeat repeatability, and transform time-dependent races into input- or path-dependent races without eliminating them. We introduce a new parallel programming model addressing these issues, and use Determinator, a proof-of-concept OS, to demonstrate the model's practicality. Determinator's microkernel application programming interface (API) provides only "shared-nothing" address spaces and deterministic interprocess communication primitives to make execution of all unprivileged code—well-behaved or not—precisely repeatable. Atop this microkernel, Determinator's user-level runtime offers a *private workspace* model for both thread-level and process-level parallel programming. This model avoids the introduction of read/write data races, and converts write/write races into reliably detected conflicts. Coarse-grained parallel benchmarks perform and scale comparably to nondeterministic systems, both on multicore PCs and across nodes in a distributed cluster.

1. INTRODUCTION

The multicore revolution has made parallel programming both ubiquitous and unavoidable, but building reliable parallel software is difficult and error-prone. Mainstream parallel programming models introduce pervasive risks for timing-dependent *races*, leading to nondeterministic "heisenbugs." Slight synchronization bugs often cause nondeterministic races between threads accessing shared memory.[2, 20, 21] Concurrent processes can similarly race when passing messages or accessing files in a shared file system, yielding not just bugs but also security vulnerabilities.[25]

We would often prefer to run software *deterministically*, so that from a given input it always produces the same output.[20] Beyond mere convenience considerations, deterministic execution is also a basic virtualization tool: a foundation required to implement replay debugging,[19] fault tolerance,[11] and accountability mechanisms.[16] Methods of intrusion analysis[12] and timing channel control[4] further require the system to *enforce* determinism, even on malicious code that might be designed to evade analysis.

User-space techniques for parallel deterministic execution[8, 22] show promise but have limitations. First, by relying on a *deterministic scheduler* residing in the application process, they permit buggy or malicious applications to compromise determinism by interfering with the scheduler. Second, deterministic schedulers emulate conventional APIs by synthesizing a repeatable—but arbitrary—schedule of inter-thread interactions, often using an instruction counter as an artificial time metric. Therefore, data races remain, but their manifestation depends subtly on inputs and code path lengths instead of on "real" time. Third, the user-level instrumentation required to isolate and schedule threads' memory accesses can incur considerable overhead, even on coarse-grained code that synchronizes rarely.

To meet the challenges presented by ubiquitous parallelism, we may need to rethink standard nondeterministic parallel programming models instead of merely shoehorning them into artificial execution schedules. To this end, we present *Determinator*, a proof-of-concept OS designed to make determinism the norm,[20] avoiding data races at either memory access[21] or higher semantic levels.[2] Owing to its OS-level approach, Determinator supports existing languages, can enforce deterministic execution not only on a single process but on groups of interacting processes, and can prevent malicious user-level code from subverting the kernel's guarantee of determinism. In order to explore the design space freely, Determinator takes a "clean-slate" approach, offering only limited compatibility with basic Unix process, thread, and file APIs. To improve backward compatibility, Determinator's environment could be implemented in a "sandbox," extending a legacy kernel.[9]

Determinator's kernel denies user code direct access to hardware resources whose use can yield nondeterministic behavior, including real-time clocks, cycle counters, and writable shared memory. User code runs within a hierarchy of single-threaded, process-like *spaces*, each running in a private virtual address space and capable of communicating only with its immediate parent and children. Potentially useful sources of nondeterminism, such as timers, which the kernel represents as I/O devices, can be accessed by unprivileged spaces only via explicit communication with more privileged spaces. A supervisory space can thus mediate all nondeterministic inputs affecting a subtree of unprivileged spaces, logging true nondeterministic events for future replay or synthesizing artificial events, for example.

The original version of this paper was published in the *9th USENIX Symposium on Operating Systems Design and Implementation,* October 4–6, 2010, Vancouver, BC, Canada.

Atop this kernel API, Determinator's user-level runtime emulates familiar shared-resource programming abstractions. The runtime employs file replication and versioning[24] to offer applications a logically shared file system accessed via the Unix file API, and adapts distributed shared memory techniques[1] to emulate shared memory for multithreaded applications. Since this emulation is implemented in user space, applications can freely customize it, and runtime bugs cannot compromise the kernel's guarantee of determinism.

Rather than emulating conventional parallel memory models, Determinator explores a *private workspace* model.[3] In this model, each thread keeps a private virtual replica of all shared memory and file system state; normal reads and writes access and modify this working copy. Threads reconcile their changes only at program-defined synchronization points, much as developers use version control systems. This model eliminates read/write data races, because reads see only *causally prior* writes in the explicit synchronization graph, and write/write races become *conflicts*, which the runtime reliably detects and reports independently of any (real or synthetic) execution schedule.

Experiments with common parallel benchmarks suggest that Determinator can run coarse-grained parallel applications deterministically with both performance and scalability comparable to nondeterministic environments. Determinism incurs a high cost on fine-grained parallel applications, however, because of Determinator's use of virtual memory to isolate threads. For "embarrassingly parallel" applications requiring little inter-thread communication, Determinator can distribute the computation across nodes in a cluster mostly transparently to the application, maintaining usable performance and scalability. The prototype still has many limitations to be addressed in future work, such as a restrictive space hierarchy, limited file system size, no persistent storage, and inefficient cross-node communication.

2. A DETERMINISTIC PROGRAMMING MODEL

Determinator's goal is to offer a programming model that is *naturally* and *pervasively deterministic*. To be *naturally deterministic*, the model's basic abstractions should avoid introducing data races or other nondeterministic behavior in the first place, not merely provide ways to control, detect, or reproduce races. To be *pervasively deterministic*, the model should behave deterministically at all levels of abstraction: for example, for shared memory access, inter-thread synchronization, file system access, interprocess communication, external device or network access, and thread/process scheduling.

Many intermediate design points may yield useful tradeoffs: enforcing determinism only on synchronization and not on memory accesses can improve efficiency, for example.[22] For now, however, we explore the feasibility of a "purist" approach to pervasive determinism. To achieve this goal, we found we had to address at least four sources of nondeterminism in current systems: nondeterministic inputs that applications may require for operation; shared state such as memory and file systems; synchronization APIs that threads and processes use to coordinate; and namespaces through which applications use and manage system resources.

2.1. Explicit nondeterministic inputs

Many applications use nondeterministic *inputs*, such as incoming messages for a Web server, timers for interactive applications, and random numbers for cryptographic algorithms. We seek not to eliminate such nondeterministic inputs but to make relevant inputs explicit and controllable.

Mechanisms for parallel debugging,[19] fault tolerance,[11] accountability,[16] and intrusion analysis[12] all rely on the ability to replay a computation instruction-for-instruction, in order to replicate, verify, or analyze a program's execution history. Replay can be efficient when only I/O need be logged, as for a uniprocessor virtual machine,[12] but becomes much more costly if *internal* sources of nondeterminism due to parallelism must also be replayed.[13]

Determinator transforms useful sources of nondeterminism into explicit I/O, which applications access via controllable channels, and eliminates only internal nondeterminism resulting from parallelism. If an application calls `gettime-ofday()`, for example, a supervising process can intercept this I/O to log, replay, or synthesize these time "inputs."

2.2. A race-free model for shared state

Conventional systems give threads concurrent access to many forms of shared state, such as shared memory and file systems, yielding data races and heisenbugs if the threads are improperly synchronized.[20, 21] Replay debuggers[19] and deterministic schedulers[8, 22] make data races reproducible once they manifest, but do not change the inherently race-prone model in which developers write applications.

Determinator replaces the standard concurrent access model with a *private workspace* model,[3] which avoids data races in the first place. This model gives each thread a private virtual replica of all logically shared states, including shared memory and file system state. A thread's normal reads and writes affect only its private working state, and do not interact directly with other threads. Instead, Determinator accumulates each thread's changes to the shared state, and then *reconciles* these changes among threads only at program-defined synchronization points. This model is related to early parallel Fortran systems,[6] replicated file systems,[24] distributed version control systems,[15] and snapshot isolation in databases.[7] To our knowledge, however, Determinator is the first OS to introduce such a model for pervasive thread- and process-level determinism.

If one thread executes the assignment "$x = y$" while another concurrently executes "$y = x$," for example, these assignments race in the conventional model, but are race-free under Determinator and always swap x with y. Each thread's read of x or y always sees the "old" version of that variable, saved in the thread's private workspace at the last explicit synchronization point.

Figure 1 illustrates a more realistic example of a game or simulator, which uses an array of "actors" (players, particles, etc.) to represent some logical "universe," and updates all actors in parallel at each time step. To update the actors, the

Figure 1. C pseudocode for lock-step time simulation, which contains a data race in standard concurrency models but is bug-free under Determinator.

```
struct actor_state actor[nactors];
main()
    initialize all elements of actor[] array
    for (time = 0; ; time++)
        for (i = 0; i < nactors; i++)
            if (thread_fork(i) == IN_CHILD)
              // child thread to process actor[i]
              examine state of nearby actors
              update state of actor[i] accordingly
              thread_exit();
        for (i = 0; i < nactors; i++)
            thread_join(i);
```

main thread forks a child thread per actor, then synchronizes by joining all child threads. The child thread code to update each actor is "inline" within the **main()** function, a convenience of Unix `fork()` that Determinator extends to shared memory threads.

Each child thread reads the "prior" state of any or all actors in the array, then updates the state of its assigned actor in-place, without explicit copying or additional synchronization. With standard threads this code has a read/write race: each child thread may see an arbitrary mix of "old" and "new" states as it examines other actors in the array. Under Determinator, however, this code is correct and race-free. Each child thread reads only its private working copy of the actors array, which is untouched except by the child thread itself, since the main thread forked that child. As the main thread rejoins all its child threads, Determinator merges each child's actor array updates back into the main thread's working copy, for use in the next time step.

Traditional write/write races become *conflicts* in this model. If two child threads concurrently write to the same actor array element, for example, Determinator detects this conflict and signals a runtime exception when the main thread attempts to join the second conflicting child. In the conventional model, by contrast, either thread might "win" this timing-dependent race, silently propagating a likely erroneous value throughout the computation. Running this code under a conventional deterministic scheduler causes the "winner" to be decided based on a synthetic, reproducible time metric (e.g., instruction count) rather than real time, but the race remains and may still manifest or vanish because of slight changes in inputs or instruction path lengths.

2.3. A race-free synchronization API

Standard threads can behave nondeterministically even in a correctly locked program with no memory access races. Two threads might acquire a lock in any order, for example, leading to high-level data races.[2] This nondeterminism is inherent in the lock abstraction: we can record and replay or synthesize a lock acquisition schedule,[22] but such a schedule is still arbitrary and unpredictable to the developer.

Fortunately, many synchronization abstractions are naturally deterministic, such as fork/join, barriers, and futures. Deterministic abstractions have the key property that when threads synchronize, *program logic alone* determines the points in the threads' execution paths at which the synchronization occurs, and the threads that are involved. In fork/join synchronization, for example, the parent's **thread_join**(*t*) operation and the child's **thread_exit()** determine the respective synchronization points, and the parent explicitly indicates the thread *t* to join. Locks fail this test because one thread's **unlock()** passes the lock to an arbitrary successor thread's **lock()**. Queue abstractions such as semaphores and pipes are deterministic if only one thread can access each end of the queue[18] but nondeterministic if several threads can race to insert or remove elements at either end.

Since the multicore revolution is young and most application code is yet to be parallelized, we may still have a choice of what synchronization abstractions to use. Determinator therefore supports only race-free synchronization primitives natively, although it can emulate nondeterministic primitives via deterministic scheduling for compatibility.

2.4. Race-free system namespaces

Current operating system APIs often introduce nondeterminism unintentionally by exposing shared namespaces implicitly synchronized by locks. Execution timing affects the pointers returned by `malloc()` or `mmap()` or the file numbers returned by `open()` in multithreaded Unix processes, and the process IDs returned by `fork()` or the file names returned by `mktemp()` in single-threaded processes. Even if only one thread actually uses a given memory block, file, process ID, or temporary file, the assignment of these names from a shared namespace is inherently nondeterministic.

Determinator's API avoids creating shared namespaces with system-chosen names, instead favoring thread-private namespaces with application-chosen names. Applications, not the system, choose virtual addresses for allocated memory and process IDs for children. This principle ensures that naming a resource reveals no shared state information other than what the application itself provided. Since implicitly shared namespaces often cause multiprocessor contention, designing system APIs to avoid this implicit sharing may be synergistic with recent multicore scalability work.[10]

3. THE DETERMINATOR KERNEL

We now briefly outline the Determinator kernel's design and API. Applications normally do not use this API directly, but rather use the user-level runtime described in Section 4.

3.1. Spaces

Determinator executes application code within a hierarchy of *spaces*, illustrated in Figure 2. Each space consists of CPU register state for a single control flow, a virtual address space containing code and working data, and optionally a *snapshot* of a prior version of this address space. A Determinator space is analogous to a single-threaded Unix process, with important

differences; we use the term "space" to highlight these differences and avoid confusion with the "process" and "thread" abstractions that Determinator emulates at the user level.

A Determinator space cannot outlive its parent, and a space can directly interact *only* with its immediate parent and children via three system calls described below. The kernel provides no file systems, writable shared memory, or other abstractions that imply globally shared state.

Only the *root space* has direct access to nondeterministic inputs via I/O devices, such as console input or clocks. Other spaces access I/O devices only indirectly via parent/child interactions or via privileges delegated by the root space. A parent space can thus control all nondeterministic inputs into any unprivileged space subtree, for example, logging inputs for future replay. This space hierarchy also creates a performance bottleneck for I/O-bound applications, a design limitation we leave to address in future work.

3.2. System call API

Determinator spaces interact only as a result of processor traps and three system calls—Put, Get, and Ret, shown in Table 1. Arguments allow multiple related operations to be combined in one system call. A Put call can, for example, initialize a child's registers, copy a virtual memory range into the child, set permissions on the target range, snapshot the child's address space, and start the child executing.

Each space has a namespace of child spaces. User-level code manages this namespace, specifying any child number in a Get or Put; the kernel creates this child if it does not exist. If the child did exist and was running, the kernel blocks the parent until the child stops via Ret or a trap. These cooperative "rendezvous" semantics ensure that spaces synchronize only at well-defined points in both spaces' execution.

Figure 2. The kernel's hierarchy of *spaces*, each containing private register and virtual memory state.

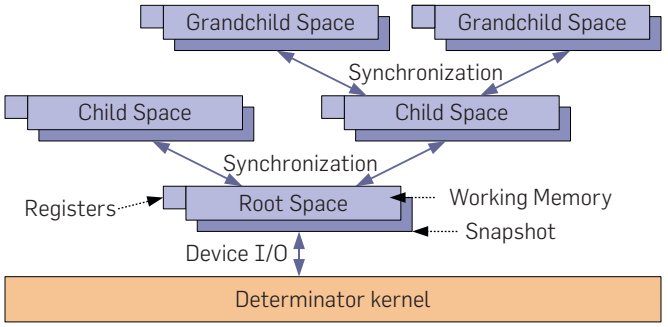


Table 1. System calls comprising Determinator's kernel API.

Call	Interacts with	Description
Put	Child space	Copy register state and/or virtual memory range into child, and optionally start child executing
Get	Child space	Copy register state, virtual memory range, and/or changes since the last snapshot out of a child
Ret	Parent space	Stop and wait for parent to issue a Get or Put. Processor traps also cause implicit Ret

User-level code can specify a Copy option in a Get or Put call to copy a range of virtual memory between the invoking space and a child. The kernel uses copy-on-write to optimize large copies and avoid physically copying read-only pages. A Snap option similarly uses copy-on-write to save a reference snapshot of a child space's entire virtual address space.

The Put call also supports a Merge option, which is like Copy, except that the kernel copies only bytes that *differ* between the child's current and reference snapshots into the parent space, leaving other bytes in the parent untouched. The kernel also detects conflicts: if a byte changed in *both* the child's and parent's spaces since the snapshot, the kernel generates an exception, treating a conflict as a programming error like an illegal memory access or divide-by-zero. Determinator's user-level runtime uses Merge to give multithreaded processes the illusion of shared memory, as described later.

Finally, the Ret system call stops the calling space, returning control to the space's parent. Exceptions such as divide-by-zero also cause an implicit Ret, providing the parent a status code indicating why the child stopped.

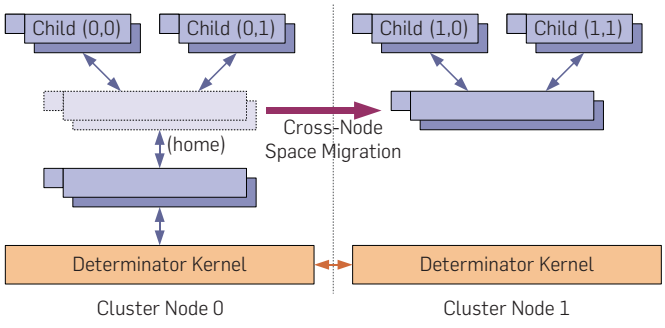
To facilitate debugging or protect against looping children, a parent can start a child with an *instruction limit*, forcing the child to trap back to the parent after executing a specific number of instructions. Counting instructions instead of "real time" preserves determinism, while enabling spaces to "quantize" a child's execution to implement deterministic scheduling at user level.[8]

3.3. Distribution via space migration

Space hierarchies can span not only multiple CPUs in a multiprocessor/multicore system but also multiple nodes in a homogeneous cluster. While distribution is semantically transparent to applications, an application may have to be designed with distribution in mind to perform well.

To support distribution, the kernel interprets the higher order bits in each space's child namespace as a node number. When a space invokes Put or Get, the kernel first migrates the calling space's state and control flow to the node specified in the child number argument, before creating or interacting with some child on that node indicated in the remaining child number bits. Figure 3 illustrates a space migrating between two nodes and managing children on each.

Figure 3. A space migrating between two nodes and starting child spaces on each node.



When the kernel migrates a space, it first transfers to the receiving kernel only the space's register state and address space metadata. The receiving kernel requests the space's memory pages on demand as the space accesses them on the new node. Each node's kernel avoids redundant cross-node page copying in the common case when a space repeatedly migrates among several nodes—for example, when a space starts children on each of several nodes, then returns later to collect their results. For read-only pages such as program code, each node's kernel reuses cached copies of these pages whenever the space returns to that node.

4. HIGH-LEVEL PARALLEL ABSTRACTIONS

Determinator's kernel API eliminates many convenient and familiar abstractions, but we can reproduce many of them deterministically in user space, with important trade-offs. This section describes how Determinator's user-level runtime infrastructure emulates traditional Unix processes, file systems, threads, and synchronization.

4.1. Processes and fork/exec/wait

We make no attempt to replicate Unix process semantics exactly, but wish to emulate traditional `fork/exec/wait` enough to support common uses in scriptable shells, build tools such as `make`, and multiprocess applications such as compilers.

Fork: A basic Unix `fork()` requires only one Put system call, to copy the parent's entire memory state into a child space, set up the child's registers, and start the child. The difficulty arises from Unix's global process ID (PID) namespace, a source of nondeterminism as discussed in Section 2.4. Since most applications use PIDs returned by `fork()` merely as an opaque argument to a subsequent `waitpid()`, our runtime makes PIDs local to each process. Each process allocates its own PIDs, which are unrelated to and may numerically conflict with PIDs in other processes. This change breaks Unix applications that pass PIDs among processes, and means that commands like "`ps`" must be built into shells for the same reason that "`cd`" already is. This approach works for compute-oriented applications following the typical fork/wait pattern, however.

Exec: A user-level implementation of Unix `exec()` must construct the new program's memory image, intended to replace the old program, while still executing the old program's runtime library code. Our runtime loads the new program into a "reserved" child space never used by `fork()`, then calls Get to copy that child's entire memory atop that of the (running) parent: this Get thus "returns" into the new program. The runtime also carries over some Unix process state, such as the PID namespace and file system state described later, from the old to the new program.

Wait: When an application calls `waitpid()` to wait for a specific child, the runtime calls Get to synchronize with the child's Ret and obtain the child's exit status. The child may Ret back to the parent to make I/O requests before terminating; the parent's runtime services the I/O request and resumes the `waitpid()` transparently to the application.

Unix's `wait()` is problematic as it waits for *any* ("the first") child to terminate, violating determinism as discussed in Section 2.3. The kernel cannot offer a "wait for any child" system call without compromising determinism, so our runtime simply waits for the child that was forked earliest.

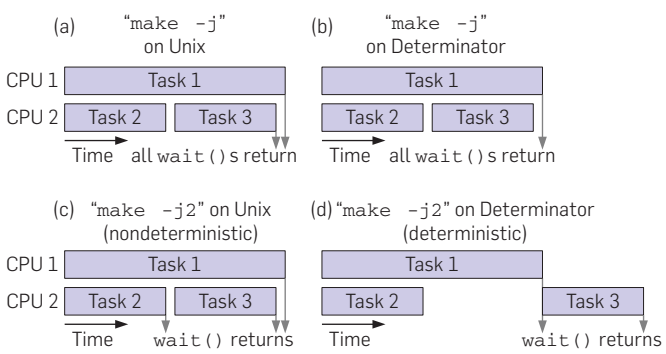
This behavior does not affect applications that fork several children and then wait for all of them to complete, but affects two common uses of `wait()`. First, interactive Unix shells use `wait()` to report when background processes complete; an interactive shell running under Determinator requires special "nondeterministic I/O privileges" to provide this functionality (and related functions such as interactive job control). Second, our runtime's behavior may adversely affect the performance of programs that use `wait()` to implement dynamic scheduling or load balancing in user space.

Consider a parallel `make` run with or without limiting the number of concurrent children. A plain "`make -j`," allowing unlimited children, leaves scheduling to the system. Under Unix or Determinator, the kernel's scheduler assigns tasks to available CPUs, as in Figure 4 (a) and (b). If the user runs "`make -j2`," however, then `make` initially starts only tasks 1 and 2, then waits for one to complete before starting task 3. Under Unix, `wait()` returns when the short task 2 completes, enabling `make` to start task 3 promptly as in Figure 4 (c). On Determinator, the `wait()` returns only when (deterministically chosen) task 1 completes, resulting in nonoptimal schedule in Figure 4 (d): determinism prevents the runtime from learning which of tasks 1 and 2 completed first. The unavailability of timing information with which to inform user-level scheduling decisions thus suggests a practice of leaving scheduling to the system in a deterministic environment (e.g., "`make -j`" instead of "`-j2`").

4.2. A shared file system

Unix's file system provides a convenient namespace and repository for staging program inputs, storing outputs, and holding intermediate results such as temporary files. Since our kernel permits no physical state sharing, user-level code must emulate shared state abstractions. Determinator's "shared-nothing" space hierarchy is similar to a distributed system consisting only of uniprocessor machines, so our user-level runtime borrows distributed file system principles to offer applications a shared file system abstraction.

Figure 4. Parallel `make` under Unix and Determinator: (a) and (b) with unlimited parallelism; (c) and (d) with a 2-worker quota imposed at user level.



Since our current focus is on emulating familiar abstractions and not on developing storage systems, Determinator's file system currently provides no persistence: it effectively serves only as a temporary file system.

Our runtime mimics a weakly consistent, replicated file system,[24] by maintaining a complete file system replica in each process's address space, as shown in Figure 5. When a process creates a child via `fork()`, the child inherits a copy of the parent's file system in addition to the parent's open file descriptors. Individual `open/close/read/write` operations in a process use only that process's file system replica, so different processes' replicas may diverge as they modify files concurrently. When a child terminates and its parent collects its state via `wait()`, the parent's runtime uses file versioning to propagate the child's changes into the parent.

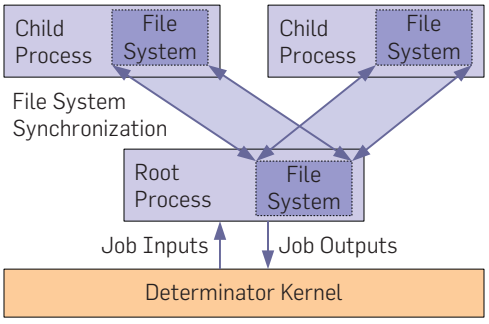
If a parallel `make` forks several compiler processes, for example, each child writes its output `.o` file to its own file system replica. The parent's runtime merges these `.o` files into the parent's file system as the parent `wait()`s on each child. This copying and reconciliation is not as inefficient as it may appear, due to the kernel's copy-on-write optimizations. Replicating a file system image among many spaces copies no physical pages until user-level code modifies them, so all copies of identical files consume only one set of pages.

As in any weakly consistent file system, processes that perform unsynchronized, concurrent writes to the same file may cause *conflicts*. When our runtime detects a conflict, it simply discards one copy and sets a conflict flag on the file; subsequent attempts to `open()` the file result in errors. This behavior is intended for batch compute applications for which conflicts indicate a bug, whose solution is to fix the bug and rerun the job. Interactive use would demand a policy that avoids losing data. The user-level runtime could implement stronger consistency and avoid unsynchronized concurrent writes, at higher synchronization costs.

4.3. Input/output and logging

Since unprivileged spaces can access external I/O devices only indirectly via parent/child interaction within the space hierarchy, our user-level runtime treats I/O as a special case of file system synchronization. In addition to regular files, a process's file system contains special *I/O files*, such as console input and output files. Unlike Unix device special files, Determinator's I/O files actually hold data in the process' file system image: for example, the console input file accumulates all characters that the process has received from the console, and the console output file holds all characters written to the console. In the current prototype, console or log files can eventually "fill up" and become unusable, though a suitable garbage-collection mechanism could address this flaw.

Figure 5. Each process maintains a private replica of the shared file system, using file versioning to reconcile replicas at synchronization points.



When a process does a console `read()`, the C library first returns unread data already in the process's local console input file. When no more data is available, instead of returning an end-of-file condition, the process calls Ret to synchronize with its parent and wait for more console input (or in principle any other form of new input) to become available. When the parent does a `wait()` or otherwise synchronizes with the child, it propagates any new input to the child. When the parent has no new input for any waiting children, it forwards all their input requests to its parent, and ultimately to the kernel via the root process.

When a process does a console `write()`, the runtime appends the new data to its internal console output file as it would append to a regular file. The next time the process synchronizes with its parent, file system reconciliation propagates these writes toward the root process, which forwards them to the kernel's I/O devices. A process can request immediate output propagation by explicitly calling `fsync()`.

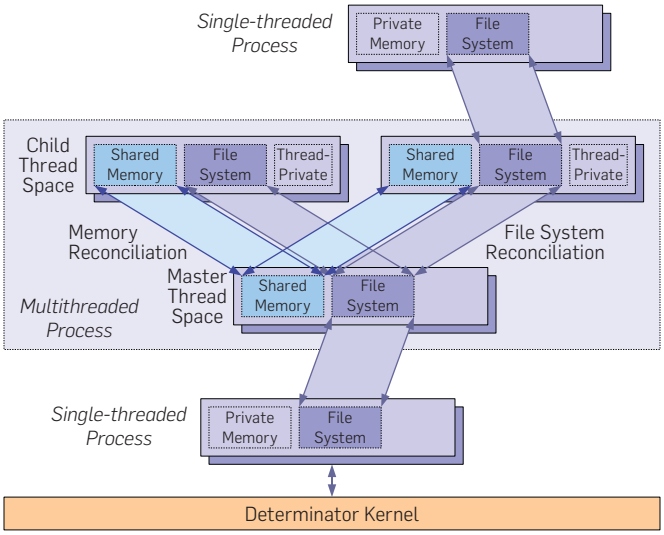
Reconciliation handles "append-only" writes differently from other writes, enabling concurrent writes to console or log files without conflict. During reconciliation, if both the parent and child have made append-only writes to the same file, reconciliation appends the child's latest writes to the parent's copy of the file, and vice versa. Each process's replica thus accumulates all processes' concurrent writes, though different processes may observe these writes in a different order. Unlike Unix, rerunning a parallel computation from the same inputs with and without output redirection yields byte-for-byte identical console and log file output.

4.4. Shared memory multithreading

Shared memory multithreading is popular despite its nondeterminism, in part because parallel code need not pack and unpack messages: threads simply compute "in-place" on shared variables and structures. Since Determinator gives user spaces no physically shared memory other than read-only sharing via copy-on-write, emulating shared memory involves distributed shared memory (DSM) techniques. Adapting the private workspace model of Section 2.2 to thread-level shared memory reuses ideas from early parallel Fortran machines[6] and release-consistent DSM systems,[1] although these prior designs did not offer determinism.

Our runtime uses the kernel's Snap and Merge operations (Section 3.2) to emulate shared memory in the private workspace model, using fork/join synchronization. To fork a child, the parent thread calls Put with the Copy, Snap, Regs, and Start options to copy the shared part of its memory into a child space, save a snapshot of that memory state in the child, and start the child running, as illustrated in Figure 6. The master thread may fork multiple children

Figure 6. A multithreaded process built from one space per thread, with a master space managing synchronization and memory reconciliation.



this way. To synchronize with a child and collect its results, the parent calls Get with the Merge option, which merges all changes that the child made to shared memory, since its snapshot was taken, back into the parent. If both parent and child—or the child and other children whose changes the parent has collected—have concurrently modified the same byte since the snapshot, the kernel detects and reports this conflict.

Our runtime also supports barriers, the foundation of data-parallel programming models like OpenMP.[23] When each thread in a group arrives at a barrier, it calls Ret to stop and wait for the parent thread managing the group. The parent calls Get with Merge to collect each child's changes before the barrier, then calls Put with Copy and Snap to resume each child with a new shared memory snapshot containing all threads' prior results. While the private workspace model conceptually extends to nonhierarchical synchronization,[3] our prototype's strict space hierarchy currently limits synchronization flexibility, an issue we intend to address in the future. *Any* synchronization abstraction may be emulated at some cost as described in the next section, however.

An application can choose which parts of its address space to share and which to keep thread-private. By placing thread stacks outside the shared region, all threads can reuse the same stack area, and the kernel wastes no effort merging stack data. Thread-private stacks enable a child thread to inherit its parent's stack and run "inline" in the same C/C++ function as its parent, as in Figure 1. If threads wish to pass pointers to stack-allocated structures, however, then they may locate their stacks in disjoint shared regions. Similarly, if the file system area is shared, then the threads share a common file descriptor namespace as in Unix. Excluding the file system area from shared space and using normal file system reconciliation (Section 4.2) to synchronize it yields thread-private file tables.

4.5. Emulating legacy thread APIs

For legacy code already parallelized using nondeterministic synchronization, Determinator's runtime can emulate the standard pthreads API via deterministic scheduling.[8] In this case, the process's initial *master space* never runs application code directly, but acts as a scheduler supervising one child space per application thread. The scheduler uses the kernel's instruction limit feature (Section 3.2) to *quantize* each thread's execution. After allowing each thread to run independently for one quantum, the scheduler uses Merge to collect the thread's shared memory changes, then restarts the thread from the merged state for another quantum.

To emulate pthreads synchronization, a thread can end its quantum prematurely to interact with the scheduler. Each mutex, for example, always has an "owner" thread, whether or not the mutex is locked. The owner can lock and unlock the mutex without scheduler interactions, but another thread needing the mutex must invoke the scheduler to obtain ownership. At the current owner's next quantum, the scheduler "steals" the mutex's ownership if the mutex is unlocked, otherwise placing the locking thread on the mutex's queue to be awoken once the mutex becomes available.

While deterministic scheduling offers compatibility with legacy code, it has drawbacks. The master space, required to serialize synchronization operations, may limit scalability unless execution quanta are large. Large quanta can increase the time threads waste waiting to interact with another, however, to steal an unlocked mutex, for example.

Further, since the deterministic scheduler may preempt a thread and propagate shared memory changes at any point in application code, the *programming model* remains nondeterministic. In contrast to our private workspace model, if one thread runs "$x = y$" while another runs "$y = x$" under the deterministic scheduler, the result may be repeatable but is no more predictable to the programmer than before. While rerunning a program with *exactly* identical inputs will yield identical results, if an input change or code recompilation affects the length of any instruction sequence, this change may cascade into a different execution schedule and trigger schedule-dependent if not timing-dependent bugs.

5. EVALUATION

This section evaluates the Determinator prototype, first informally, and then examines single-node and distributed parallel processing performance and, finally, code size.

Determinator is written in C with small assembly fragments, and currently runs on the 32-bit x86 architecture. Since our focus is on parallel compute-bound applications, Determinator's I/O capabilities are currently limited to text-based console I/O and a simple Ethernet-based protocol for space migration (Section 3.3). The prototype has no persistent disk storage: the runtime's shared file system abstraction currently operates in physical memory only.

5.1. Experience using the system

We find that a deterministic programming model simplifies debugging of both applications and user-level

runtime code, since user-space bugs are always reproducible. Conversely, when we do observe nondeterministic behavior, it can result only from a kernel (or hardware) bug, immediately limiting the search space.

Because Determinator's file system holds a process's output until the next synchronization event (often the process's termination), each process's output appears as a unit even if the process executes in parallel with other output-generating processes. Further, different processes' outputs appear in a consistent order across runs, as if run sequentially.

While race detection tools exist,[21] we find it convenient that Determinator detects conflicts under "normal-case" execution, without requiring a special tool. Determinator's model makes conflict detection as standard as detecting a division by zero or illegal memory access.

A subset of Determinator doubles as *PIOS*, "Parallel Instructional Operating System," used in Yale's OS course for the past 2 years.[14] While determinism is not a primary course topic, we found Determinator/PIOS to be a useful tool for teaching operating systems in the multicore era, due to its simple design and minimal kernel API. Partly derived from and inspired by JOS,[17] PIOS includes an instructional framework where students fill in missing pieces of a "skeleton." Students work in small groups to reimplement all of Determinator's core features: multiprocessor scheduling, virtual memory with copy-on-write and Snap/Merge, user-level threads with fork/join, the user-space file system with versioning and reconciliation, and cross-node space migration. In their final projects, students extend the OS with features such as graphics, pipes, and remote shells. The course is challenging and incurs a heavy programming load, but anonymous student reviews have been highly positive due to its educational value and perceived relevance.

5.2. Single-node multicore performance

Since Determinator runs user-level code "natively" on the hardware instead of rewriting user code,[8] its performance is comparable to that of existing systems when running coarse-grained parallel code, but incurs higher costs on fine-grained parallel code because of the system calls, context switches, and virtual memory operations required at synchronization events.

Figure 7. Determinator performance relative to pthreads under Linux using parallel benchmarks.

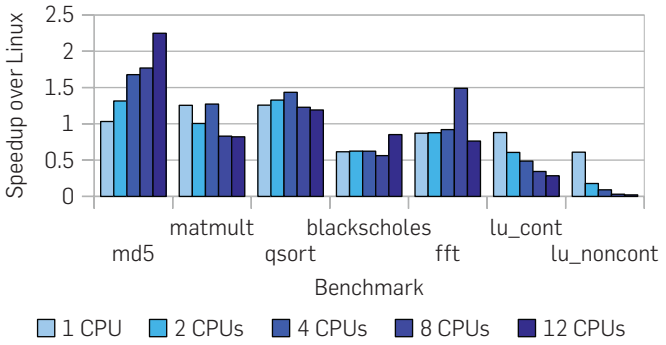


Figure 7 shows the performance of several shared-memory parallel benchmarks on Determinator, relative to the same benchmarks using standard pthreads on 32-bit Ubuntu Linux 9.10. The coarse-grained benchmarks *md5, matmult, qsort, blackscholes*, and *fft* perform comparably to nondeterministic execution under Linux, while the fine-grained *lu* benchmarks show a higher performance cost.

Figure 8 shows each benchmark's speedup relative to single-threaded execution on Determinator. The "embarrassingly parallel" *md5* and *blackscholes* scale well, *matmult* and *fft* level off after four processors (but still perform comparably to Linux), and the remaining benchmarks scale poorly.

5.3. Distributed computing performance

To evaluate cross-node space migration (Section 3.3), we changed the *md5* and *matmult* benchmarks to distribute workloads across up to 32 uniprocessor nodes, while preserving Determinator's shared memory programming model. Lacking a cluster suitable for native testing, we ran Determinator under QEMU,[5] on a shared-use Linux cluster.

Figure 9 shows parallel speedup versus single-node execution in Determinator, on a log-log scale. The *md5-circuit* benchmark serially migrates to each node to fork

Figure 8. Determinator parallel speedup relative to its own single-CPU performance.

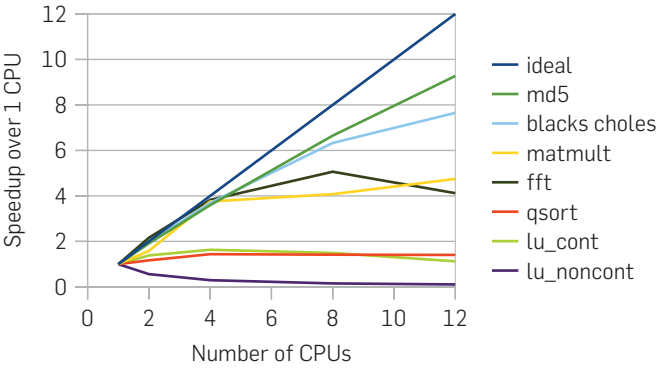
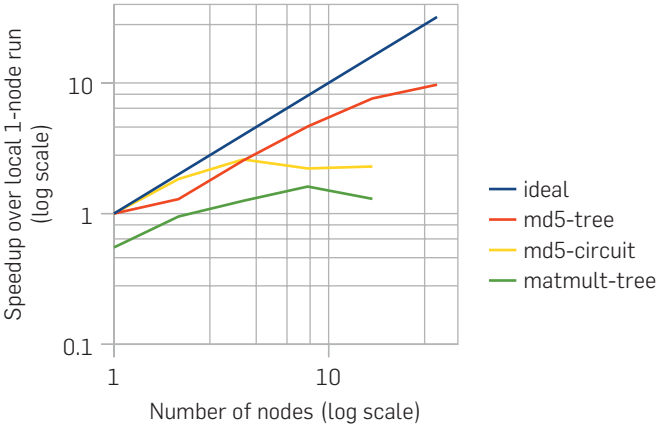


Figure 9. Speedup of deterministic shared-memory benchmarks on varying-size distributed clusters.



worker threads and collect results, while *md5-tree* and *matmult-tree* fork and join workers recursively in a binary tree structure. The "embarrassingly parallel" *md5-tree* performs and scales well with recursive work distribution, while *matmult-tree* suffers from the cost of large cross-node data transfers using Determinator's unoptimized page copying protocol.

As Figure 10 shows, the shared-memory benchmarks on Determinator perform comparably to nondeterministic, distributed-memory equivalents running on Linux. Illustrating the benefits of its shared memory API, the Determinator version of *md5* is 63% of the size of the Linux version (62 lines versus 99), which uses remote shells to coordinate workers. Determinator's *matmult* is 34% of the size of the Linux version (90 lines versus 263), which passes data using TCP.

5.4. Implementation complexity

To provide a feel for implementation complexity, Table 2 shows source code line counts for Determinator, as well as its PIOS instructional subset, counting only lines containing semicolons. The entire system is less than 15,000 lines, about half of which is generic C and math library code needed mainly for porting Unix applications easily.

6. CONCLUSION

While Determinator is only a proof of concept, it shows that operating systems can offer a pervasively, naturally deterministic environment. Determinator avoids introducing data races in shared memory and file system access, thread and process synchronization, and throughout its API. Experiments suggest that such an environment can efficiently run coarse-grained parallel applications, both on multicore machines and across clusters, though running fine-grained applications efficiently may require hardware evolution.

Figure 10. Deterministic shared-memory benchmarks versus distributed-memory equivalents for Linux.

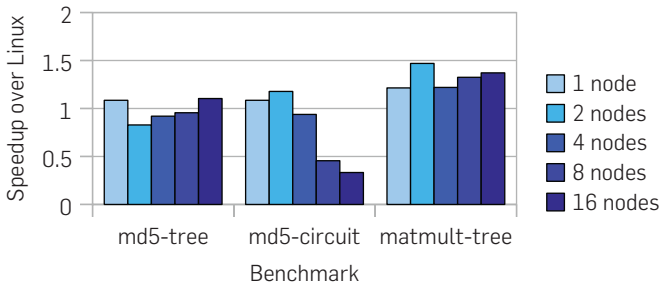


Table 2. Implementation code size of the Determinator OS and of PIOS, its instructional subset

Component	Determinator semicolons	PIOS semicolons
Kernel core	2044	1847
Hardware drivers	751	647
User-level runtime	2952	1079
Generic C library code	6948	394
User-level programs	1797	1418
Total	14,492	5385

Acknowledgments

We thank Zhong Shao, Ramakrishna Gummadi, Frans Kaashoek, Nickolai Zeldovich, Sam King, and the OSDI reviewers for their valuable feedback. We thank ONR and NSF for their support under grants N00014-09-10757 and CNS-1017206.

References

1. Amza, C. et al. TreadMarks: Shared memory computing on networks of workstations. *IEEE Computer 29*, 2 (Feb. 1996), 18–28.
2. Artho, C., Havelund, K., Biere, A. High-level data races. In *Workshop on Verification and Validation of Enterprise Information Systems (VVEIS)* (Apr. 2003), 82–93.
3. Aviram, A., Ford, B., Zhang, Y. Workspace consistency: A programming model for shared memory parallelism. In *2nd Workshop on Determinism and Correctness in Parallel Programming (WoDet)* (Mar. 2011).
4. Aviram, A., Hu, S., Ford, B., Gummadi, R. Determinating timing channels in compute clouds. In *ACM Cloud Computing Security Workshop (CCSW)* (Oct. 2010).
5. Bellard, F. QEMU, a Fast and Portable Dynamic Translator, In *USENIX Annual Technical Conference*, Anaheim, CA, April 10-15, 2005.
6. Beltrametti, M., Bobey, K., Zorbas, J.R. The control mechanism for the Myrias parallel computer system. *Comput. Architect. News 16*, 4 (Sept. 1988), 21–30.
7. Berenson, H. et al. A critique of ANSI SQL isolation levels. In *SIGMOD* (June 1995).
8. Bergan, T. et al. CoreDet: A compiler and runtime system for deterministic multithreaded execution. In *15th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)* (Mar. 2010).
9. Bergan, T. et al. Deterministic process groups in dOS. In *9th USENIX Symposium on Operating Systems Design and Implementation (OSDI)* (Oct. 2010).
10. Boyd-Wickizer, S. et al. Corey: An operating system for many cores. In *8th USENIX Symposium on Operating Systems Design and Implementation (OSDI)* (Dec. 2008).
11. Castro, M., Liskov, B. Practical byzantine fault tolerance. In *3rd USENIX Symposium on Operating Systems Design and Implementation (OSDI)* (Feb. 1999), 173–186.
12. Dunlap, G.W. et al. ReVirt: Enabling intrusion analysis through virtual-machine logging and replay. In *5th USENIX Symposium on Operating Systems Design and Implementation* (Dec. 2002).
13. Dunlap, G.W. et al. Execution replay for multiprocessor virtual machines. In *Virtual Execution Environments (VEE)* (Mar. 2008).
14. Ford, B. PIOS: Parallel instructional operating system, 2010. http://zoo.cs.yale.edu/classes/cs422/pios.
15. git: the fast version control system. http://git-scm.com/.
16. Haeberlen, A., Kouznetsov, P., Druschel, P. PeerReview: Practical accountability for distributed systems. In *21st ACM Symposium on Operating Systems Principles (SOSP)* (Oct. 2007).
17. Kaashoek, F. et al. 6.828: Operating system engineering. http://pdos.csail.mit.edu/6.828/.
18. Kahn, G. The Semantics of a Simple Language for Parallel Programming", In *Information Processing '74: Proceedings of the IFIP Congress* (1974), pp. 471-475.
19. Leblanc, T.J., Mellor-Crummey, J.M. Debugging parallel programs with instant replay. *IEEE Trans. Comput. C-36*, 4 (Apr. 1987), 471–482.
20. Lee, E. The problem with threads. *Computer*, *39*, 5 (May 2006), 33–42.
21. Musuvathi, M. et al. Finding and reproducing heisenbugs in concurrent programs. In *8th USENIX Symposium on Operating Systems Design and Implementation (OSDI)* (Berkeley, California, USA, 2008), USENIX Association.
22. Olszewski, M., Ansel, J., Amarasinghe, S. Kendo: Efficient deterministic multithreading in software. In *14th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)* (Mar. 2009).
23. OpenMP Architecture Review Board. OpenMP application program interface version 3.0, May 2008.
24. Parker, D.S. Jr. et al. Detection of mutual inconsistency in distributed systems. *IEEE Trans. Software Eng. SE-9*, 3 (May 1983).
25. Wei, J., Pu, C. TOCTTOU vulnerabilities in UNIX-style file systems: An anatomical study. In *4th USENIX Conference on File and Storage Technologies (FAST)* (Dec. 2005).

Amittai Aviram, Shu-Chun Weng, Sen Hu, and Bryan Ford, Yale University, Department of Computer Science, New Haven, CT.

DOI:10.1145/2160718.2160743

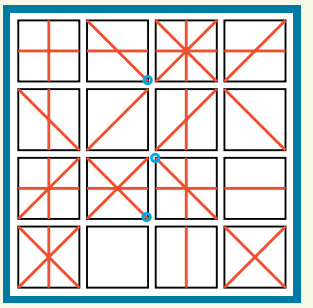
Peter Winkler

Puzzled
Designs on Square Grids

Welcome to, as usual, three new puzzles. However, unlike previous columns, where solutions to two were known (and included in the related Solutions and Sources in the next issue), this time expect to see solutions to all three in June.

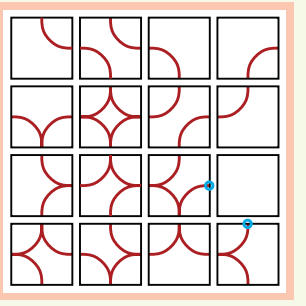
1. In composing this puzzle, writer/mathematician Barry Cipra was inspired by a work called "Straight Lines in Four Directions and All Their Possible Combinations" by the great American artist Sol Lewitt. You are given 16 unit squares, each containing a different combination of vertical crossing line, horizontal crossing line, SW-NE diagonal, and SE-NW diagonal. The object is to tile a 4×4 grid with these squares (without rotating them) in such a way that no line ends before it hits the edge of the grid or, alternatively, prove it cannot be done. Figure 1 shows a failed attempt, with dangling ends circled.

Figure 1.



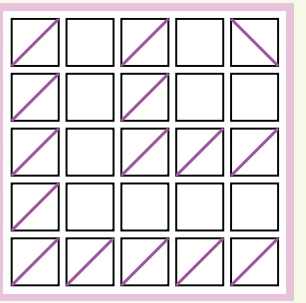
2. Cipra also composed an entertaining curved version. In it, each unit square contains one of the 16 possible combinations of four quarter circles, each of radius 1/2 and centered at a corner. Can you tile a 4×4 grid with these squares so no path ends before it hits the edge of the grid? Or, better still, can you arrange the tiles so an even number of quarter circles meet at each edge shared by two squares? Figure 2 shows a failed attempt, again with dangling ends circled.

Figure 2.



3. We now upgrade to a 5×5 grid in a puzzle passed to me by my Dartmouth colleague Vladimir Chernov. In it, each unit square is blank or has one diagonal drawn in. How many non-blanks can you pack in without any two diagonals meeting at a corner? Figure 3 shows a legal configuration with 15 diagonals. Can you beat it with 16 diagonals? Or prove 15 is the maximum possible?

Figure 3.



Readers are encouraged to submit prospective puzzles for future columns to puzzled@cacm.acm.org.

Peter Winkler (puzzled@cacm.acm.org) is William Morrill Professor of Mathematics and Computer Science at Dartmouth College, Hanover, NH.


Frederick P. BROOKS, JR.


Charles P. (Chuck) THACKER


Charles W. BACHMAN


Stephen A. COOK


Barbara LISKOV


Richard M. KARP


Leslie G. VALIANT


Marvin MINSKY


Judea PEARL


Adi SHAMIR


Richard E. STEARNS


John HOPCROFT


Robert E. KAHN


Vinton G. CERF


Robert TARJAN


Ivan SUTHERLAND


Butler LAMPSON


Juris HARTMANIS


Andrew C. YAO


Donald E. KNUTH


Dana S. SCOTT


Raj REDDY


Fernando J. CORBATÓ


Edward A. FEIGENBAUM


Alan C. KAY


William (Velvel) KAHAN


Frances E. ALLEN


E. Allen EMERSON


Niklaus WIRTH


Ken THOMPSON


Leonard ADLEMAN


Ronald RIVEST


Edmund CLARKE


Joseph SIFAKIS

34 TURING AWARD WINNERS TOGETHER?

WHAT DO YOU GET WHEN YOU PUT

THE ACM TURING CENTENARY CELEBRATION

A ONCE-IN-A-LIFETIME EVENT THAT YOU'LL NEVER FORGET... AND YOU'RE INVITED!

CHAIR:
Vint Cerf, *'04 Turing Award Winner, Chief Internet Evangelist, Google*

PROGRAM CO-CHAIRS:
Mike Schroeder, *Microsoft Research*
John Thomas, *IBM Research*
Moshe Vardi, *Rice University*

MODERATOR:
Paul Saffo, *Managing Director of Foresight, Discern Analytics* and *Consulting Associate Professor, Stanford*

WHEN:
June 15th & June 16th 2012

WHERE:
Palace Hotel, San Francisco CA

HOW:
Register for the event at turing100.acm.org

REGISTRATION IS FREE-OF-CHARGE!

acm | A.M. TURING

THE A.M. TURING AWARD, often referred to as the "Nobel Prize" in computing, was named in honor of Alan Mathison Turing (1912–1954), a British mathematician and computer scientist. He made fundamental advances in computer architecture, algorithms, formalization of computing, and artificial intelligence. Turing was also instrumental in British code-breaking work during World War II.

Join the Conversation

GAMES ART/DESIGN FILM/TV PRODUCTION RESEARCH
PRODUCT DEVELOPMENT EDUCATION STUDENT OTHER

Bring your artistic ability, scientific innovation, and everything in between to inspire and be inspired by the most diverse gathering in computer graphics and interactive techniques.

You Are SIGGRAPH2012

The **39th** International **Conference** and **Exhibition** on **Computer Graphics** and **Interactive Techniques**

Conference 5–9 August 2012
Exhibition 7–9 August 2012
Los Angeles Convention Center

Sponsored by ACM**SIGGRAPH**

www.siggraph.org/s2012