

# Microsoft's Jim Gray on Computing's Breakthroughs, Lessons, and Future

Dejan Milojicic



Jim Gray is a senior researcher in Microsoft's Scalable Servers Research Group and manager of the company's Bay Area Research Center. He specializes in databases and transaction processing systems, with current work focusing on building supercomputers with commodity components. Jim was awarded the ACM's A.M. Turing award for his contributions to information technology and the US Geological Survey's John Wesley Powell Award for his work on the TerraServer project. Recently Jim talked with Hewlett-Packard scientist and IEEE DSO editor Dejan Milojicic about the future of computer science.

***Dejan Milojicic: Which computing science breakthroughs have made the most difference? Can you predict or envision the next big steps (or the need for them), and when they might occur?***

Jim Gray: To spectators, it might seem that there are breakthroughs in computing, but folks on the inside know it generally takes decades for things to go from concept to widely adopted technology. The Internet took 35 years, relational databases took 20 years, the microprocessor took 15 years, and SGML (now HTML) took 20 years. So, your readers know that there are no breakthroughs. There are ideas, sometimes radical ideas, and then there's the work. Thomas Edison said it's 1 percent inspiration and 99 percent perspiration. And he was really inspired!

But you're really asking, "What are some of those inspirations?" Well, we have Babbage's computer, Bush's Memex, and Turing's thinking machine. After 100 years we've delivered the computer and we're within a decade of producing Memex, but the solution to the Turing test is still at least 20 years off. On a smaller scale, the microprocessor, procedural and nonprocedural programming (Fortran, Visicalc, SQL), GUIs, and networking all seem like big and nonobvious ideas. I'm less impressed with breakthroughs such as electronic-program guides and one-click checkout, both of which are generating huge patent revenue.

In terms of what comes next, we haven't delivered Memex or thinking machines, and the computers we have delivered sometimes create more work than they save. Many of us are suffering with information overload, and large organizations are suffering with the fact that computers are cheap to buy but expensive to own: "free like a puppy" has new poignancy for IT executives today.

So, we need to deliver these "breakthroughs" in the next 25 years:

1. Make computers self-managing so that they take care of themselves.
2. Deliver on Memex while not buying us an avalanche of information we don't want—that is, summarize and filter information so that the computer augments human intelligence and is a delight to use.
3. Create thinking machines in the sense that Turing suggested 50 years ago.
4. Something new that has arrived since the 1950s is the realization that life-is-software; DNA is digital software. So, much of biology is really software. My guess is that by 2050 we'll be programming DNA in really interesting ways to make energy, clean the environment, create food and materials, and of course cure disease. In the Wired article "Why the Future Doesn't Need Us" ([www.wired.com/wired/archive/8.04/joy.html](http://www.wired.com/wired/archive/8.04/joy.html)), Bill Joy eloquently described the downside of this, but I view it as inevitable—a Faustian bargain.
5. Herb Simon's intelligent universe is emerging where everything has intelligence. The doors, the lights, the cereal box, all know their role in life and communicate with

the Web to serve humankind. This is spooky. It signals the end of privacy. One of the IT challenges is to make the compromises of such a world acceptable. Bill Joy didn't warn us about the end of privacy, but I would add it to his list.

***What are the most important lessons learned in computing science and technology, and how can we leverage them for the future? Which important attributes lost relevance over the past, and which are increasing as the technology and market pendulum moves? Are there any attributes that will never lose relevance (for example, scalability and fault tolerance versus performance and availability)?***

We are in an IT winter (and a "telco winter") right now. This is a reaction to the irrational exuberance of 1995-2000. People of that era expected to get rich quick, wanted to have overnight revolutions. They forgot that 9 out of 10 startups fail completely and that only 1 in 30 are successful (two-thirds are absorbed by larger companies, so the investors get their money back). Those are 1980 statistics, and we've probably returned to that. This is fine by me; I went into IT for the excitement, the sense of accomplishment, and the creativity. Those lures are still there for the scientist and engineer. But the greedy ones should move on to real estate or law.

The lesson we've learned is that Moore's law gives us 100 times more each decade. This changes the trade-offs and opens new application areas. We need to imagine a world where we have 100 or 10,000 times more information or processing power and then imagine ways of using that power. It's hard to do that—we're all stuck in our paradigms. What if computers could really understand language? What if they were self-managing? What if storage and networking and processing were 10,000 times cheaper (that is, free)?

Your desktop computer has the processing power, storage capacity, and bandwidth of all the computers sold in 1970. If we were still doing IT in 1970s style, we'd have a single-computer, zero-billion-dollar industry. Not a pretty picture for the job market. What we do today will be a one-computer, zero-billion-dollar industry in 2030—there will be a complete turnover. Indeed, the rate of change is accelerating so it might happen much sooner than that.

In terms of things that have lost relevance: performance is much less pressing than it was when computers cost millions of dollars, and usability is much more pressing. Some things will probably never change. People will always be worried about security, privacy, integrity, and other attributes that are central in human interaction—these issues are universal in human-human interaction and so are likely to be universal in human-machine interaction.

***How would you distinguish the roles of researchers (academia and labs) from those of developers (industry, startups, open source)? Where does one stop, and where should the other take over? What's the role of standards bodies?***

Developers work on products. They have 24-month horizons. That means six months for invention and then 18 months for test. They must have a very clear idea about what they want to build in such an environment—they can't take many risks. Startups are different: they expect a 90 percent failure rate, they expect complete change in business plans in 18 months, and they're very agile. They build prototypes and hope someone buys them. Researchers typically take a longer view—the 10-year view. There's much wrong with the way we do research, but on the whole it produces students and some very good results that are beyond what product and startup groups can do.

Standards bodies codify what's understood. They are useful for portability and for interoperability. Seeing the horrors of a judge telling one company what extensions it can add to another company's standard programming language, I'm all for open standards. But, no one implements all of the SQL standard, the C++ template standard is incomprehensible, and the OSI standard died under its own weight. So, standards are important and are there to help with portability and interoperability, but they are post-facto. At best, they appear after the research and prototypes are done and before the products are widely deployed.

***What's your advice to researchers? You always seemed to have had a small hands-on projects of your own (for example, worldwide telescope), keeping you close to the technology. How do you achieve optimum balance between development and research, and is there a rule of thumb or is it entirely personal preference?***

JG: Some people think deductively: they look at the scene and see the answer. I think inductively. I look at  $n = 0$ ,  $n = 1$ ,  $n = 2$  and hope that eventually I see the pattern. I work from the specific to the general. I have to use a system to understand it—it must be a left-brain right-brain thing. Marvin Minsky said it best: "You have to think about thinking something in order to think about thinking." Other people can just grock thinking by thinking about it: top-down rather than my bottom-up approach.

People like me need to do things in order to understand. That's why I build systems. That's why I tinker. That's why I read so much.

***Who do you see as key drivers and catalysts in databases and transaction processing today? Traditional computing companies? Academia? Small companies and startups? Open source communities? Other industry including vertical markets?***

Traditional SQL systems as we know them will be commoditized within a decade. MySQL is good enough for simple tasks—and it's getting better. Web servers have already replaced all the classic TP monitors (CICS, ACMS, Pathway, Tuxedo). The current crop of app servers are also challenged by the synthesis of DBMS and runtimes.

But, there is a revolution. Microsoft calls it .Net, others have branded it in other ways, but a new distributed computing platform is emerging. It involves loosely coupled object systems with the DBMS providing object management services (invocation, persistence, security). This is a huge gob of software that the open-source community will be challenged to clone anytime soon. So the software companies like Microsoft, IBM, and Oracle have a credible future if they execute this plan well. There is also the higher-level information services business like Amazon, eBay, Google, Yahoo!, AOL, and MSN that organize information for you. That's a major trend in information management.

On the client side, organizing personal information stands as a real challenge. How will you organize all your digital stuff? In 2010, I expect you'll have about 10 Tbytes of personal information (mostly video). How will you find anything? That's the main thrust of the Longhorn effort that has thousands of Microsoft programmers working late tonight.

Data management, data analysis, and data visualization are growing issues not only at the business level but also at the personal level.

***Grid computing raised a lot of interest in industry recently. Do you think this is a promising technology or is it only the most recent fashion? Where do you think that grids might have most impact in the future, and how broadly can they get adopted, for example in the spectrum from subsuming Web services to remaining focused on technical computing?***

Grid computing can mean CPU-cycle harvesting for scientists, peer-to-peer information sharing (such as Napster), data grids (such as Google, eBay, Amazon, and MSN), or access grids (such as the next-generation telephone). Each of these is a real and important application. Grid is an umbrella term for all of them—going beyond the HTTP Web. My pals call it .Net, IBM calls it on demand, Sun calls it N1, and so on. These are all just umbrella names. There is no real "grid technology" that addresses all these areas. There are pieces of technology that are very cool. I love SETI@Home and am eager to see BOINC proliferate. I think XML, SOAP, and WSDL are real advances. So, there's some technology here and there's substance, but "grid" has become an umbrella term.

***You've explored computing architectures—for example, high-end systems, clusters, and cyberbricks. Which architecture is becoming dominant nowadays, and what are the most important problems preventing these architectures from being even more widely adopted? How do you see these problems being addressed and by whom? What trends are affecting the computing architectures of today (for example, mobile and wireless, p2p, server consolidation, outsourcing)?***

Cyberbricks have arrived—everyone builds their computers with cyberbricks—1U or 4U boxes

---

with commodity memory, commodity disks, and commodity networking. The NEC EarthSim is impressive, but it has half the processing power of SETI@Home, Google, or MSN. I don't think cyberbricks could be more widely adopted; they have 90 percent of the market. The only folks not using them are companies with legacy applications running on mainframes. The ROI on porting those apps is just not there—but no sane person puts a new app on anything but a cyberbrick.

The challenge now is that companies can afford to, and do, own a thousand cyberbricks. How do you manage them, how do you retire them?

***What's the center of gravity of future systems? Is it hardware (personal, Internet data centers, utility computing models), systems software (MS Windows, Linux, embedded OSs), applications, management tools, vertical markets?***

I'm a software guy. People buy solutions, not computers. They don't buy an OS or a CPU, they buy solutions. VisiCalc drove PCs into the office. The Internet drove PCs into the home, and Web servers and database servers into the data center. Applications will drive the next waves as well. I cannot predict the next killer app, but it's coming.

***If you had a chance to repeat some of your work from the past, what would you do differently? Can you extrapolate this question to examples of other big projects and research, in other words: what if?***

We've seen the effects of unintended consequences in the recent California election. I believe we live in the best of all possible worlds—and I'm glad of that. As a software guy, you quickly learn that when you change something, other things stop working or start working differently. Certainly, there are things I regret doing, and things I regret not doing—but most of those are at a personal level. At a professional level, I regret having worked for hardware companies for the first 25 years of my career. I'm a software guy, and we were always hobbled by a desire to make our companies' hardware more marketable. Working for a software company is much better for a software guy.

***Can you offer advice for researchers and developers at the beginning of their career? What are recipes and rules for success, if there are any at all? What are the promising areas that they should address and what are those that may not lead to success? How do you define success?***

My advice is personal. You need passion for what you are doing. If you don't have that, then find a different job or career. I think without passion, life is empty. We are fortunate to be working in areas where one can be passionate (I couldn't be passionate about serving burgers at a fast-food restaurant). If you have that passion, you'll probably have joy and pleasure and a full life. Without it, nothing else matters. We are gold-mining in cyberspace. There are nuggets all around. All you need to do is find a rich vein and mine it. You should have no problem finding these veins, they're

all around us. Depending on your temperament you can go for a home run, try something really radical that, if you're right, will make a huge difference—or you can play it safe and just go for singles and doubles.

***What are examples of brilliant technologies that failed in adoption, and how can we make sure to avoid this happening again (for example, Beta versus VHS, early RISC systems). What is your personal perspective on the "innovator's dilemma" regarding the discovery of new markets and the transfer of successful technologies through new products to new markets? What's the right balance between de facto standards, open-source efforts, and traditional standard efforts?***

Well, neither Beta nor VHS were brilliant-both were "good enough" at the time. Sony Betamax was better but they were greedy, and so Sony got nothing. I hope they learned a lesson from that. We can all learn a lesson from that. I can't think of a brilliant technology that failed-they all emerge, perhaps not in their original form.

Yes, Clayton Christensen's "Innovator's Dilemma" nicely states the dilemma-you have to put yourself out of business or someone else will. One of the attractions of working at Microsoft is that the entire company understands this and is constantly rejuvenating the products. The current set is disjoint from the products of 10 years ago, and the same thing will happen in the next decade. We have a big investment in research, and there's good communication between research and development. In many companies that relationship is dysfunctional, but so far Rick Rashid has done a wonderful job of integrating research with the product groups, and Bill Gates has done a wonderful job of fostering that mutual respect.

***Do you think that there are any revisions needed to Amdahl's laws? Can you envision it being broken any time in the future? What about other popular laws (Reed's law, Metcalfe's law, and so on)-are there any trends impacting the nature of these laws?***

The surprise is that Moore's law and Amdahl's laws are still valid 35 years after the fact-I covered this in some depth in "Rules of Thumb in Data Engineering" (a paper I coauthored with Prashant Shenoy). Carl Shapiro and Hal Varian's Information Rules: A Strategic Guide to the Network Economy (1998) adds a new economics dimension to the discussion. Both of those works discuss trends. The basic point is that all these laws are about ratios. When the ratio changes, the trade-off changes. Since hardware is now commodity, most of the value has moved to software in the broad sense of information. The hardware cost is largely management. For example, database vendors can't train database administrators (DBAs) fast enough; so they're having to simplify the products so that more customers can buy them or so that one DBA can manage more systems.

***Looking back, what are you most proud of as a researcher, developer, and technologist? What are your favorite mistakes, and what have you learned from them?***

Professionally, I'm most proud of the people I've mentored-students I've helped, coworkers I've encouraged. There have been good papers (transactions, concurrency, fault tolerance, distributed systems, data cubes, online science), but all that is much less important to me than the friends I've made and the people I've helped. My big mistakes were being too enthusiastic about a bad idea; my biggest successes were being enthusiastic about a bad idea. My current bad idea is to put all scientific data online and integrate all of it into one giant database. The original concept was crazy; but I learned, and the current approach seems very promising.