A Conversation with Jim Gray

From *Queue* Storage Vol. 1, No. 4 - June 2003
by

DAVE PATTERSON What is the state of storage today?

JIM GRAY We have an embarrassment of riches in that we're able to store more than we can access. Capacities continue to double each year, while access times are improving at 10 percent per year. So, we have a vastly larger storage pool, with a relatively narrow pipeline into it.

We're not really geared for this. Having lots of RAM helps. We can cache a lot in main memory and reduce secondary storage access. But the fundamental problem is that we are building a larger reservoir with more or less the same diameter pipe coming out of the reservoir. We have a much harder time accessing things inside the reservoir.

DP How big were storage systems when you got started?

JG Twenty-megabyte disks were considered giant. I believe that the first time I asked anybody, about 1970, disk storage rented for a dollar per megabyte a month. IBM leased rather than sold storage at the time. Each disk was the size of a washing machine and cost around $20,000.

Much of our energy in those days went into optimizing access. It's difficult for people today to appreciate that, especially when they hold one of these $100 disks in their hand that has 10,000 times more capacity and is 100 times cheaper than the disks of 30 years ago.

DP How did we end up with wretched excess of capacity versus access?

JG First, people in the laboratory have been improving density. From about 1960 to 1990, the magnetic material density improved at something like 35 percent per year—a little slower than Moore's Law. In fact, there was a lot of discussion that RAM megabyte per dollar would surpass disks because RAM was following Moore's Law and disks were evolving much more slowly.

But starting about 1989, disk densities began to double each year. Rather than going slower than Moore's Law, they grew faster. Moore's Law is something like 60 percent a year, and disk densities improved 100 percent per year.

Today disk-capacity growth continues at this blistering rate, maybe a little slower. But disk access, which is to say, "Move the disk arm to the right cylinder and rotate the disk to the right block," has improved about tenfold. The rotation speed has gone up from 3,000 to 15,000 RPM, and the access times have gone from 50 milliseconds down to 5 milliseconds. That's a factor of 10. Bandwidth has improved about 40-fold, from 1 megabyte per second to 40 megabytes per second. Access times are improving about 7 to 10 percent per year. Meanwhile, densities have been improving at 100 percent per year.

DP I hadn't thought about it the way you explained it. It isn't that the access times have been improving too slowly; it's that the capacity has been improving too quickly. Access is on the old schedule, but the density is astronomical. What problems does that present going forward? How do we design storage systems differently?

JG The first thing to keep in mind is it's not over yet. At the FAST [File and Storage Technologies] conference about a year-and-a-half ago, Mark Kryder of Seagate Research was very apologetic. He said the end is near; we only have a factor of 100 left in density—then the Seagate guys are out of ideas. So this 200-gig disk that you're holding will soon be 20 terabytes, and then the disk guys are out of ideas. The database guys are already out of ideas!

What do you do with a 200-gig disk drive? You treat a lot of it as tape. You use it for snapshots, write-anywhere file systems, log-structured file systems, or you just zone frequent stuff in one area and try to waste the other 190 GB in useful ways. Of course, we could put the Library of Congress holdings on it or 10,000 movies, or waste it in some other way. I am sure that people will find creative ways to use all this capacity, but right now we do not have a clear application in mind. Not many of us know what to do with 1,000 20-terabyte drives—yet, that is what we have to design for in the next five to ten years.

DP You might look back at the old research papers before there was a disk and how you had to manage tape systems to look for ideas for the future.

JG Certainly we have to convert from random disk access to sequential access patterns. Disks will give you 200 accesses per second, so if you read a few kilobytes in each access, you're in the megabyte-per-second realm, and it will take a year to read a 20-terabyte disk.

If you go to sequential access of larger chunks of the disk, you will get 500 times more bandwidth—you can read or write the disk in a day. So programmers have to start thinking of the disk as a sequential device rather than a random access device.

DP So disks are not random access any more?

JG That's one of the things that more or less everybody is gravitating toward. The idea of a log-structured file system is much more attractive. There are many other architectural changes that we'll have to consider in disks with huge capacity and limited bandwidth.

On the one hand, these disks offer many opportunities. You can have a file where all the old versions are saved. The unused part of the disk can be used as tape or as archive. That's already happening with people making snapshots of the disk every night and offering a version of the file system as of yesterday or as of a certain point in time. They can do that by exploiting the disk's huge capacity.

DP Another use of tape has been archival and transport. I know you had some experience trying to use disks as a tape replacement. How has that turned out?

JG We are at a stage now where disk media and tape media have approximate price parity, which is to say it's about $1 a gigabyte per disk and per tape cartridge. So, you can think about writing to disk and then pulling the disk out and treating it as a tape cartridge.

The disk has properties that the tape doesn't have. Disk has higher bandwidth and is more convenient to access. You can just plug in the disk. You don't need a tape drive and you don't need a bunch of software that knows how to read tapes. You're actually mounting a file system. You've got no extra software, no extra concepts. You don't have to find the part of the tape that has your file, and you do not need those funny tape management systems.

I've been working with a bunch of astronomers lately and we need to send around huge databases. I started writing my databases to disk and mailing the disks. At first, I was extremely cautious because everybody said I couldn't do that—that the disks are too fragile. I started out by putting the disks in foam. After mailing about 20 of them, I tried just putting them in bubble wrap in a FedEx envelope. Well, so far so good. I have not had any disk failures of mailed disks.

The biggest problem I have mailing disks is customs. If you mail a disk to Europe or Asia, you have to pay customs, which about doubles the shipping cost and introduces delays.

DP Wouldn't that also be true with tape?

JG It's the same for tape and DVDs, but not for Internet file transfers. No customs duties on FTP—at least not yet.

The nuisance factor of moving the disks around can be a problem. It requires some intelligence to be able to plug a random IDE disk into a computer and get it to work, but the folks I am working with find disks more convenient than tape.

DP Will serial attached disks make it easier?

JG That should be a huge help. And Firewire is a help. Another option is to send whole computers. I've been sending NTFS disks (the Windows file system format), and not every Linux system can read NTFS. So lately I'm sending complete computers. We're now into the 2-terabyte realm, so we can't actually send a single disk; we need to send a bunch of disks. It's convenient to send them packaged inside a metal box that just happens to have a processor in it. I know this sounds crazy—but you get an NFS or CIFS server and most people can just plug the thing into the wall and into the network and then copy the data.

DP That makes me want to get mail from you.

JG Processors are not that expensive, and I count on people sending the computer back to me or on to the next person who wants the data.

DP What's the difference in cost between sending a disk and sending a computer?

JG If I were to send you only one disk, the cost would be double—something like $400 to send you a computer versus $200 to send you a disk. But I am sending bricks holding more than a terabyte of data—and the disks are more than 50 percent of the system cost. Presumably, these bricks circulate and don't get consumed by one use.

DP Do they get mailed back to you?

JG Yes, but, frankly, it takes a while to format the disks, to fill them up, and to send around copies of data. It is easier than tape, however, both for me and for the people who get the data.

DP It's just like sending your friends a really great movie or something.

JG It's a very convenient way of distributing data.

DP Are you sending them a whole PC?

JG Yes, an Athlon with a Gigabit Ethernet interface, a gigabyte of RAM, and seven 300-GB disks—all for about $3,000.

DP It's your capital cost to implement the Jim Gray version of "Netflicks."

JG Right. We built more than 20 of these boxes we call TeraScale SneakerNet boxes. Three of them are in circulation. We have a dozen doing TeraServer work; we have about eight in our lab for video archives, backups, and so on. It's real convenient to have 40 TB of storage to work with if you are a database guy. Remember the old days and the original eight-inch floppy disks? These are just much bigger.

DP "Sneaker net" was when you used your sneakers to transport data?

JG In the old days, sneaker net was the notion that you would pull out floppy disks, run across the room in your sneakers, and plug the floppy into another machine. This is just TeraScale SneakerNet. You write your terabytes onto this thing and ship it out to your pals. Some of our pals are extremely well connected—they are part of Internet 2, Virtual Business Networks (VBNs), and the Next Generation Internet (NGI). Even so, it takes them a long time to copy a gigabyte. Copy a terabyte? It takes them a very, very long time across the networks they have.

DP When they get a whole computer, don't they still have to copy?

JG Yes, but it runs around their fast LAN at gigabit speeds as opposed to the slower Internet. The Internet plans to be running at gigabit speeds, but if you experiment with your desktop now, I think you'll find that it runs at a megabyte a second or less.

DP Megabyte a second? We get almost 10 megabytes sustained here.

JG That translates to 40 gigabytes per hour and a terabyte per day. I tend to write a terabyte in about 8 to 10 hours locally. I can send it via UPS anywhere in the U.S. That turns out to be about seven megabytes per second.

DP How do you get to the 7-megabytes-per-second figure?

JG UPS takes 24 hours, and 9 hours at each end to do the copy.

DP Wouldn't it be a lot less hassle to use the Internet?

JG It's cheaper to send the machine. The phone bill, at the rate Microsoft pays, is about $1 per gigabyte sent and about $1 per gigabyte received—about $2,000 per terabyte. It's the same hassle for me whether I send it via the Internet or an overnight package with a computer. I have to copy the files to a server in any case. The extra step is putting the SneakerNet in a cardboard box and slapping a UPS label on it. I have gotten fairly good at that.

Tape media is about $3,000 a terabyte. This media, in packaged SneakerNet form, is about $1,500 a terabyte.

DP What about the software compatibility issues of sending a whole computer?

JG By sending a whole computer, the people just need CIFS or NFS. They can see the file system and pull files if they want. It completely nullifies a bunch of compatibility issues. Think of how easy it is to bring a wireless laptop into your network. This is the same story—except the system is wired.

Of course, this is the ultimate virus.

In the old days, when people brought floppy disks around, that was the standard way of communicating viruses among computers. Now, here I am mailing a complete computer into your data center. My virus can do wonderful things. This computer is now inside your firewall, on your network.

There are some challenges about how to secure this. The simple strategy is to say, "Look, if you were FTP'ing from this computer, it would be outside your firewall. So I'll mail it to you, and then you plant it outside your firewall, but LAN-connected, so that you're not paying the phone bill."

DP Run it through a firewall?

JG That's one strategy. The other strategy is to say, "I trust Dave. Dave sent me this computer."

DP What are storage costs today?

JG We have been talking about packaged storage so far. Packaged storage can be bought for $2,000 to $50,000 per terabyte, depending on where you shop. When it is "RAIDed," the price approximately doubles. If you go for Fibre Channel rather than direct-attached, then the fabric can add 30 percent to that. The simple answer is $3,000 per terabyte to $100,000 per terabyte, or $1,000 per year to $30,000 per year annualized. But the real cost of storage is management. Folks on Wall Street tell me that they spend $300,000 per terabyte per year administering their storage. They have more than one data administrator per terabyte. Other shops report one admin per 10 TB, and Google and the Internet Archive seem to be operating at one per 100 TB. The cost of backup/restore, archive, reorganize, growth, and capacity management seems to dwarf the cost of the iron. This stands as a real challenge to the software folks. If it is business as usual, then a petabyte store needs 1,000 storage admins. Our chore is to figure out how to waste storage space to save administration. That includes things like RAID, disk snapshots, disk-to-disk backup, and much simpler administration tools.

DP What is the tension between direct-attached disk, storage area network (SAN), and network-attached storage (NAS)?

JG Direct-attached disk is simple and cheap, but many argue that storage should be consolidated in storage servers so that it is easier to manage and can be shared—consolidation and virtualization. This creates a dance-hall design where CPUs are on one side of the picture and data is on the other side. The VaxCluster worked that way, the IBM Sysplex works that way, and now many large Unix data centers are moving that direction. When you look inside the storage servers, they are really servers with an operating system and a network fabric. This all seems really weird to me. These high-end storage servers are exporting a very low-level get-block put-block protocol. It creates a lot of traffic. File-oriented servers such as NetApp have a higher level of abstraction and so should have much better performance and virtualization. But up till now the relatively poor performance of TCP/IP has saddled Net-App with lower performance than a Fibre Channel SAN solution. Gigabit Ethernet, TCP offload engines (TOEs), and the NetApp direct-access storage protocol taken together give performance comparable to any Fibre Channel solution. These same technologies enable high-performance iSCSI. Once we have Gigabit Ethernet and TOE, I think the migration to a NAS solution will be much preferable to the get-block put-block interface of iSCSI.

DP Let's talk about the higher-level storage layers. How did you get started in databases? You were at IBM when the late Ted Codd formulated the relational database model. What was that like?

JG Computers were precious in those days. They were million-dollar items. Ease of use was not the goal. The mind-set was that labor was cheap and computers were expensive. People wrote in assembly language, and they were very, very concerned about performance.

Then along came Ted Codd, saying, "You know, it would be a lot easier to program in set theory." He observed that in the old days people didn't write Fortran, they wrote data-flows for unit-record equipment that processed sets of cards. Each device did a simple task. One would sort the set of cards or it would copy a set of cards, and so on. You could set up a plug board that would copy the cards or would throw away all the cards that didn't have a certain property or would duplicate all the cards that had a certain property. Programming consisted of configuring the machines and their plug boards and then running decks of cards through.

To some extent you can think of Codd's relational algebra as an algebra of punched cards. Every card is a record. Every machine is an operator. The operators are closed under composition: decks-of-cards in and decks-of-cards out. It was a chore to program those systems, but people programmed at a fairly high level of abstraction.

The cool thing was that once the operators were configured, it was easy to do just about anything. The architecture has both partition and pipeline parallelism, and it is very simple to conceptualize and debug since everything is in "cards."

It was a form of data-flow programming. Ted had experience in all these areas. He said, "We should figure out a way of doing things like that in this more modern world where, in fact, everything is on disk and accessible and the files are much, much bigger."

The reaction at the time was that it's going to be inefficient, and it's going to use more instructions and more disk I/Os than IBM's Information Management System (IMS), the first commercial hierarchical, structured database management system. In addition, the industry was moving from batch-transaction processing that dealt with sets of records to online transaction processing that dealt with individual records ("give me that bank account").

Ted's ideas were extremely controversial. The core issues were about how computers would be used in the future and what was important to optimize. This was not a PC world. This was a mainframe world of business data processing.

DP The controversy was whether anybody would pay for such inefficient use of the computer?

JG Right, and whether we were headed toward a world of online transaction processing where nobody actually wanted to go through all the records. Everything was going to be online, everything was going to be incremental, and there wouldn't be this need for batch reporting.

DP You wouldn't need to go back and look at the old data to have a summary of the current data instantly available?

JG The online advocates assumed that we would keep running totals, and they minimized the fact that the manager might want to balance the books. In fact, despite the advance of online transaction-processing systems, the batch-processing systems continue to be there and continue to evolve, and they have to deal with larger and larger data sets.

Paradoxically, the relational model ended up being really good for reporting in the decision-support part of batch processing.

Nonetheless, the relational guys took the challenge and said, "If we're going to be successful, we're going to have to perform as well as the IMS on the bread-and-butter transactions." A great deal of energy went into this, and I think it is fair to say that the relational implementations did okay. Today, all the best TPC-C [Transaction Processing Performance Council Online Transaction Processing Benchmark] results you see are with relational systems. The IMSes of the world are not reporting TPC-C results because, frankly, their price performance isn't very good.

It was this evolution. The database guys had to get their price-performance story together first. At a certain point, most people who bought the relational stuff bought it for the usability, not the price performance. They were getting new applications, and they wanted to get their applications up quickly.

You see this today. Two groups start; one group uses an easy-to-use system, and another uses a not-so-easy-to-use system. The first group gets done first, and the competition is over. The winners move forward and the other guys go home.

That situation is now happening in the Web services space. People who have better tools win.

DP What do you think is happening with databases in terms of open source? What is the Linux of databases?

JG I think it's exciting. Very small teams built the early database systems. A small team at Oracle built the original Oracle, and there were small teams at Informix, Ingress, Sybase, and IBM.

Twenty-five people can do a pretty full-blown system, and ship it, and support it, and get manuals written, and test it. The Postgress and MySQL teams are on that scale and likely represent the leading open-source DBMSes out there. Maybe the teams are getting larger at this point. A few years ago the DBMSes lacked transactions, optimization, replication, and lots of other cool features, but they are adding these features now.

The lack of a common code base is one of the things that has held back the database community and has been a huge advantage for the operating systems community. The academic world has had a common operating system that everybody can talk about and experiment with.

It has the downside of creating a mob culture. But the positive side is everybody has a common language and a common set of problems they are working on. Having MySQL as a common research vehicle is going to accelerate progress in database research. People can do experiments and compare one optimizer against another optimizer. Right now, it is very difficult for one research group to benefit from the work of others.

The flip side is this: What does this mean for the database industry as a whole? What does this mean for Oracle and Microsoft and DB2 and whoever else wants to make a database system?

So far, MySQL is very primitive and very simple. It will add features, and the real question is, can it evolve to be competitive with Oracle, Microsoft, and DB2?

Those companies spend a huge amount of energy on quality control, support, documentation, and a bunch of things that are thinner in the open-source community. But it could be that some company will step forward and MySQL.com will displace the incumbent database vendors.

The challenge is similar to the challenge we see in the OS space. My buddies are being killed by supporting all the Linux variants. It is hard to build a product on top of Linux because every other user compiles his own kernel and there are many different species. The main hope for Oracle, DB2, and SQLserver is that the open-source community will continue to fragment. Human nature being what it is, I think Oracle is safe.

DP Is MySQL.com trying to be the Red Hat of MySQL?

JG It could be that they will step forward and provide all of those things that IBM, Microsoft, and Oracle provided, and do it for a much lower price. I think the incumbent vendors will have to be innovative to make their products more attractive.

One thing that works in the incumbents' favor is fear, uncertainty, and doubt (FUD). If you base your company on a database, you are risking a lot. You want to buy the best one. People are usually pretty cautious about where they want to put their data. They want to know that it's going to have a disaster recovery plan, replication, good code quality, and in particular, lots and lots and lots of testing.

The thing that slows Oracle, IBM, and Microsoft down is the testing, and making sure they don't break anything—supporting the legacy. I don't know if the MySQL community has the same focus on that.

At some point, somebody will say, "I'm running my company on MySQL." Indeed, I wish I could hear Scott McNealy [CEO of Sun Microsystems] tell that to Larry Ellison [CEO of Oracle].

DP The whole corporation?

JG Right. Larry Ellison announced that Oracle is now running entirely on Linux. But he didn't say, "Incidentally we're going to run all of Oracle on MySQL on Linux." If you just connected the dots, that would be the next sentence in the paragraph. But he didn't say that, so I believe that Larry actually thinks Oracle will have a lot more value than MySQL has. I do not understand why he thinks the Linux problems are fixable and the MySQL problems are not.

DP To change the subject, I think one of the reasons you received the Turing Award was for your contributions in transactions, or for making the databases work better. Is that a fair characterization?

JG It's hard to know. There is this really elegant theory about transactions, having to do with concurrency and recovery. A lot of people had implemented these ideas implicitly. I wrote a mathematical theory around it and explained it and did a fairly crisp implementation of the ideas. The Turing Award committee likes crisp results. The embarrassing thing is that I did it with a whole bunch of other people. But I wrote the papers, and my name was often first on the list of authors. So, I got the credit.

But to return to your question, the fundamental premise of transactions is that we needed exception handling in distributed computations. Transactions are the computer equivalent of contract law. If anything goes wrong, we'll just blow away the whole computation. Absent some better model, that's a real simple model that everybody can understand. There's actually a mathematical theory lying underneath it.