TECHNICAL PERSPECTIVE:

THE DATA CENTER IS THE COMPUTER

by David A. Patterson

Internet services are already significant forces in searching, retail purchases, music downloads, and auctions. One vision of 21st century IT is that most users will be accessing such services over a descendant of the cell phone rather than running shrink-wrapped software on a descendant of the PC.

There are dramatic differences between of developing software for millions to use as a service versus distributing software for millions to run their PCs. First, services must be always available, so dependability is critical. Second, services must have tremendous bandwidth to support many users, but they must also have low latency so as not annoy customers who can easily switch to competing services. Third, the companies can innovate more quickly because their software is only run inside the company.

These requirements have led to distributed data centers. They are distributed to prevent a site disaster resulting in loss of power or networking from stopping the service, and to reduce latency to a worldwide customer base.

Companies like Google are starting to hire computer architects. When I asked Luiz Barroso of Google why, he said, "The data center is now the computer." Hence, computer architects are now designing and evaluating data centers.

This is certainly a provocative notion. However, if the data center is the computer, it leads to the even more intriguing question "What is the equivalent of the ADD instruction for a data center?"

Jeffrey Dean and Sanjay Ghemawat offer one answer in their paper. Like the early days of computing in the mid 20th century, the early developers of services at Google had to worry about a myriad of gritty details. For Google it includes partitioning data sets, communicating between independent computers, scheduling tasks to run simultaneously, and handling hardware and software failures. Given the pain of that experience, and the desire to let many develop new services inside Google, Dean and Ghemawat chose to raise the level of abstraction. Like their 20th century predecessors, the art was in hiding minutia while still delivering good performance and a useful programming interface.

This brings to mind three questions:

- What are useful programming abstractions for such a large system?
- How do thousands of computers behave differently from a small system?
- What must you do differently to run the abstraction on thousands of computers?

They decided to offer a two-phase primitive. The first phase maps a user supplied function onto thousands of computers. The second phase reduces the returned values from all those thousands of instances into a single result. Note that these two phases are highly parallel yet simple to understand. Borrowing the name from a similar function in Lisp, they christened the primitive "MapReduce.

Heterogeneity is one difference between running MapReduce on a single computer versus thousands. Companies don't buy thousands of computers in one fell swoop, so a single data center will have generations of computers of varying speed processors and different amounts of DRAM and disk. In addition to hardware variety, even identical equipment will behave differently. Some of it will break before and perhaps while running your program. There will also be several computers that are limping along, making much slower progress than their siblings do.

What should MapReduce do in light of this challenging environment? Here are a few highlights. First, the scheduler accommodates dynamic variation by assigning tasks based on how well a computer has done recently rather than by a static priority. Second, to cope with laggard computations, it was much faster to re-execute them on the fast nodes than to wait for the slow ones to complete. Third, the Google File System allows programs to access files efficiently from any computer, so functions can be mapped everywhere.

To test performance, they ran a program across a data center that sorts 10 billion randomly-generated records in 10 to 15 minutes, despite node failures. That's an impressive 10 million records a second. Such performance allowed Google to replace the old ad hoc programs that regenerate Google's index of the Internet with faster and simpler code based on MapReduce.

In addition to production use of MapReduce, it empowered novice programmers. In a half hour they can now write, run, and see output from their programs run on thousands of computers. The original paper had just a few months of experience by novice users, so I was delighted to read the updated paper to learn what happened over the last two years.

The beauty of MapReduce is that any programmer can understand it, and its power comes from being able to harness thousands of computers behind that simple interface. When paired with the distributed Google File System to deliver data, programmers can write simple functions that can do amazing things.

I predict MapReduce will inspire new ways of thinking about the design and programming of large distributed systems. If MapReduce is the first instruction of the "data center computer," I can't wait to see the rest of the instruction set, as well as the data center programming language, the data center operating system, the data center storage systems, and more.

Biography

David Patterson (pattrsn@eecs.berkeley.edu) is the Pardee Professor of Computer Science at U. C. Berkeley, and is a Fellow and Past President of ACM.