Faster websites, more reliable data

October 14 2010, Larry Hardesty, MIT News

Today, visiting almost any major website -- checking your Facebook news feed, looking for books on Amazon, bidding for merchandise on eBay -- involves querying a database. But the databases that these sites maintain are enormous, and searching them anew every time a new user logs on would be painfully time consuming. To serve up data in a timely fashion, most big sites use a technique called caching. Their servers keep local copies of their most frequently accessed data, which they can send to users without searching the database.

But caching has an obvious problem: If any of the data in the database changes, the cached copies have to change too; moreover, any cached data that are in any way dependent on the changed data also have to change. Tracking such data dependencies is a nightmare for programmers, but even when they do their jobs well, problems can arise.

For instance, says Dan Ports, a graduate student in the Computer Science and Artificial Intelligence Lab, suppose that someone is bidding on an item on eBay. The names of the bidders could be cached in one place, the value of their bids in another. Making a new bid updates the database, but as that update propagates through the network of servers, it could reach the value cache before it reaches the name cache. The bidder would see someone else's name next to her bid and think she'd been beaten to the punch. "They might see their own bid attributed to somebody else," Ports says, "and wind up in a bidding war with themselves."

MIT researchers have developed a new caching system that eliminates this type of asymmetric data retrieval while also making database caches much easier to program. Led by Ports and his thesis advisor, Institute Professor Barbara Liskov, who won the 2008 Turing Award, the highest award in computer science, the research also involves associate professor Sam Madden, PhD student Austin Clements, and former master's student Irene Zhang. Ports presented the system on Oct. 5 at the USENIX Symposium on Operating Systems Design and Implementation in Vancouver.

Transact locally

Unlike existing database caching systems, Ports and Liskov's can handle what computer scientists call transactions. A transaction is a set of computations that are treated as a block: None of them will be performed unless all of them are performed. "Suppose that you're making a plane reservation, and it has two legs," says Liskov. "You're not interested in getting one of them and not the other. If you run this as a transaction, then the underlying system will guarantee that you get either both of them or neither of them. And it does this regardless of whether there are other concurrent accesses, or other users are trying to get seats on those flights, or there are machine failures, and so forth.

Transactions are a well-understood technique in computer science to achieve this kind of functionality.” Indeed, it’s the idea of transactions that gives the new system its name: TxCache, where “Tx” is a shorthand for “transaction.”

TxCache also makes it easier for programmers to manage caches. “Existing caches have the approach that they just make this cache and tell the programmer, ‘Here’s a cache: You can put stuff in it if you want; you can get stuff out of it if you want,’” says Ports. “But figuring out how to do that is entirely up to you.” TxCache, however, recognizes that a computer program already implicitly defines the relationships between stored data. For instance, a line of code might say that Z = X + Y, which is an instruction to look up X, look up Y, and store their sum as Z. With TxCache, the programmer would simply specify that that line of code — Z = X + Y — should be cached, and the system would automatically ensure that, whenever any one of those variables changed, the cached copies of the other two would be updated, everywhere. And, of course, it can perform the same type of maintenance with more complicated data dependencies, represented by more complicated functions.

Bean counting

According to Liskov, the key to getting TxCache to work was “a lot of bookkeeping.” The system has to track what data are cached where, and which data depend on each other. Indeed, Liskov says, it was the fear that that bookkeeping would chew up too many computing cycles that dissuaded the designers of existing caching systems from supporting transactions. But, she explains, updating the caches is necessary only when data in the database change. Modifying the data is a labor-intensive operation; the bookkeeping steps are comparatively simple. “Yes, we are doing more work, but proportionally it’s very small,” Liskov says. “It’s on the order of 5 to 7 percent.” In the researchers’ experiments, websites were more than five times as fast when running TxCache as they were without it.

“The trouble with large-scale services like Bing and Amazon and Google and the like is that they operate at such a high level of scalability,” says Solom Heddaya, a partner at Microsoft and infrastructure architect for Bing, Microsoft’s search engine. “On a single request from the user searching for something, there are many, many applications that get invoked in real time, and they together will use tens of thousands of servers.” On that scale, Heddaya says, some kind of caching system is necessary. But, he says, “until this paper came along, people building these systems said, ‘Hey, we will shift the burden to the programmer of the application. We will give you the convenience of caching, so that we bring the data closer to where the computation is, but we will make you worry about whether the cache has the right data.”

Heddaya cautions that, unlike some other caching systems, the MIT researchers’ offers significant performance improvements only for sites where reading operations — looking up data in the database — greatly outnumber writing operations — updating data in the databases. But according to Ports, “Adding support for using caching during read/write transactions is one of the things we're thinking about now. There aren't any major technical obstacles to doing so: It's mainly a question of how we can do so without introducing unexpected effects that make life more difficult for users and programmers.”
