BLOG@CACM

The *Communications* Web site, http://cacm.acm.org, features 13 bloggers in the BLOG@CACM community. In each issue of *Communications*, we'll publish excerpts from selected posts, plus readers' comments.

DOI:10.1145/1562164.1562169 http://cacm.acm.org/blogs/blog-cacm

Saying Good-bye to DBMSs, Designing Effective Interfaces

Michael Stonebraker discusses the problems with relational database management systems and possible solutions, and Jason Hong writes about interfaces and usable privacy and security.

From Michael Stonebraker's "The End of a DBMS Era (Might be Upon Us)"

Relational database management systems (DBMSs) have been remarkably successful in capturing the DBMS marketplace. To a first approximation they are "the only game in town," and the major vendors (IBM, Oracle, and Microsoft) enjoy an overwhelming market share. They are selling "one size fits all"; i.e., a single relational engine appropriate for all DBMS needs. Moreover, the code line from all of the major vendors is quite elderly, in all cases dating from the 1980s. Hence, the major vendors sell software that is a quarter century old, and has been extended and morphed to meet today's needs. In my opinion, these legacy systems are at the end of their useful life. They deserve to be sent to the "home for tired software."

Here's why.

If we examine the nontrivial-sized DBMS markets, it turns out that current relational DBMSs can be beaten by approximately a factor of 50 in most any market I can think of. What follows are a few examples.

In the data warehouse market, a column store beats a row store by approximately a factor of 50 on typical business intelligence queries. The reason is because column stores read only the columns of interest to the query and not all of them. In addition, compression is more effective in a column store. Since the legacy systems are all row stores, they are vulnerable to competition from the newer column stores.

In the online transaction processing (OLTP) market, a lightweight main memory DBMS beats a row store by a factor of 50. Leveraging main memory and the fact that no DBMS application will send a message to a human user in the middle of a transaction allows an OLTP DBMS to run transactions to completion with no resource contention or locking overhead.

In the science DBMS market, users have never liked relational DBMSs and want a non-relational model and query facility. (This was the topic of my last CACM blog, "DBMSs for Science Applications: A Possible Solution.")

If you are storing Resource Description Framework (RDF) data, which is popular in the bio community and elsewhere, then column stores are very good at certain RDF workloads. In addition, other ideas, such as RDF-3X, will beat conventional DBMSs in other situations. Lastly, native RDF engines (e.g., Virtuoso, Sesame, and Jena) may well gain traction. The point is that something else will beat conventional row stores in this market.

Text applications have never used relational DBMSs. This was pointed out to me most clearly by Eric Brewer nearly 15 years ago in the early days of Inktomi. He wanted to use a relational DBMS to store the results of Web crawling, but found relational DBMSs to be two orders of magnitude slower than a home-brew system. All the major Web-search engines use home-brew text software to serve us search results. None use relational DBMSs.

Even in XML, where the current major vendors have spent a great deal of energy extending their engines, it is claimed that specialized engines, such as Mark Logic or Tamino, run circles around the major vendors, according to a private communication by Dave Kellogg.

In summary, one can leverage at least the following ideas to get superior performance:

A non-relational data model. If the

user's data is naturally something other than tables and if simulating his natural data model on top of tables is awkward, then chances are that a native implementation of the natural data model will significantly outperform a conventional relational DBMS. This is certainly true in scientific data.

A different implementation of tables. If something other than a row store accelerates the user's queries, then a direct implementation of the relational model using non-row store technology will run circles around a conventional relational DBMS. This is true in the data warehouse marketplace.

A different implementation of transactions. Current row stores give you a "one size fits all" implementation of transactions. This can be radically beaten if a user has lesser requirements or if the system can take advantage of workload-specific features. This is true in the OLTP marketplace.

One of these characteristics is true in every market I can think of. Hence, in my opinion, the days of a "one size fits all" monolithic DBMS are at an end. The replacement will be a collection of vertical market-specific engines, with much higher performance.

You might ask, "What if I don't care about performance?" The answer: Run one of the open source relational DBMSs. They are mature, reliable, and, best of all, free.

You might also ask, "I am dug in deep with my current vendor(s). What do I do?" The answer: Take some portion of your DBMS budget and allocate it to new solutions. Over time, you will move onto better technology.

Reader's comment

It is very true that relational DBMSs are overhyped for not so valid reasons. The current trends also showcase that there are viable alternatives to relational DBMSs, which can beat them at their own game. Also, the emergence of distributed key-value stores, such as Cassandra and Voldemort, proves the efficiency and cost effectiveness of their approaches.

Also, the recently concluded NoSQL conference discussed at length how distributed, non-relational databases work, along with overview of the emerging alternatives in this space.

—Pavan Yara

From Jason Hong's "Designing Effective Interfaces for Usable Privacy and Security"

I often cringe when I hear highly technical engineers talk about people.

I usually hear broad generalizations tossed about, like "people are lazy, that's why they can't use the system" or "people don't understand security." The worst is "people are just stupid."

With this kind of attitude, it's no surprise there are so many complicated user interfaces in the world, let alone in privacy and security. Failing to try to understand things from the user's point of view is the cardinal sin in user interface design.

With this in mind, I thought it would be good to shift focus in this blog entry away from individual case studies of usable privacy and security, and look at the bigger picture of how to design better user interfaces.

Now, how to craft an effective user interface is a very involved topic that one can study for years, and there are lots of great Web sites and books out there. Effective user interface design combines our understanding of aesthetics, technology, and human behavior to develop artifacts that are useful, usable, and desirable for a specific target audience.

What makes usable privacy and security different from designing other interfaces is that privacy and security are often secondary tasks. People don't go to an e-commerce site explicitly wanting to protect their credit cards and email addresses; they go there to buy things. Security and privacy are obvious things they want while accomplishing their main goal, in the same manner that they want the Web site to also be fast and usable.

Roughly, there are three broad strategies for usable privacy and security (note that these aren't mutually exclusive):

- make the interface invisible
- make the interface more understandable
- train the users

A good example of better security by making the interface invisible is Secure Sockets Layer. End users don't have to do anything special, and all of their network traffic is transparently encrypted.

Oftentimes, we just need to make the user interface more understandable to end users. This might be accomplished through better layout, simplified task flows, better visualizations, or more appropriate metaphors (why do we sign digital documents using keys, anyway?).

Finally, some user interfaces may also require training the users. One common misconception about user interfaces is that they should be "intuitive" (a description that always raises a red flag with me). If you're a Star Trek fan like I am, you might remember that famous scene in *Star Trek IV* where Montgomery Scott, the ship's engineer, tries to use a Macintosh computer. After attempting to talk to the computer and getting no response, he picks up the mouse and tries talking into it. Intuitive indeed.

Applications are always designed for a specific context, for specific purposes, and for a specific target audience. The best designs will empower people and let them get started quickly, while also providing a way for them to get better.

As such, some applications will require some level of training. The training might range from a basic understanding of how to zoom in and out on the iPhone (which Apple cleverly trained people how to do, with their television ads), all the way to learning how to drive a car (something we actually start training our children to do since birth, given how ingrained cars are in society).

Now, this doesn't mean that you can get away with a disastrous user interface and expect people to have to train how to use it, but it also doesn't mean that all user interfaces should be walk up and use either. You have to balance ease-of-use with power and flexibility *for your specific audience and your specific goals.* As Silicon Valley pioneer Doug Engelbart once noted, if ease of use was all that mattered, we'd all still be riding tricycles. C

Michael Stonebraker is an adjunct professor at the Massachusetts Institute of Technology. **Jason Hong** is an assistant professor at Carnegie Mellon University.