SQL Databases v. NoSQL Databases

Author: Michael Stonebraker Massachusetts Institute of Technology
Published in: Communications of the ACM CACM Homepage archive
Volume 53 Issue 4, April 2010
Pages 10-11

BLOG@CACM

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

twitter
Follow us on Twitter at http://twitter.com/blogCACM

DOI:10.1145/1721654.1721659 http://cacm.acm.org/blogs/blog-cacm

SQL Databases v. NoSQL Databases

Michael Stonebraker considers several performance arguments in favor of NoSQL databases—and finds them insufficient.

From Michael Stonebraker's "The NoSQL Discussion has Nothing to Do With SQL"

http://cacm.acm.org/blogs/blog-cacm/50678

Recently, there has been a lot of buzz about NoSQL databases. In fact, there were at least two conferences on the topic in 2009, one on each coast. Seemingly, this buzz comes from people who are proponents of:

- document-style stores in which a database record consists of a collection of key-value pairs plus a payload. Examples of this class of system include CouchDB and MongoDB, and we call such systems *document stores* for simplicity;
- key-value stores whose records consist of key-payload pairs. Usually, these are implemented by distributed hash tables, and we call these *key-value stores* for simplicity. Examples include MemcacheDB and Dynamo.

In either case, one usually gets a low-level, record-at-a-time database management system (DBMS) interface instead of SQL. Hence, this group identifies itself as advocating NoSQL.

There are two possible reasons to move to either of these alternate DBMS technologies: performance and flexibility.

The performance argument goes something like the following: I started with MySQL for my data storage needs and over time found performance to be inadequate. My options were:

1. "Shard" my data to partition it across several sites, giving me a serious headache managing distributed data in my application or
2. Abandon MySQL and pay big licensing fees for an enterprise SQL DBMS or move to something other than a SQL DBMS.

The flexibility argument goes something like the following: My data does not conform to a rigid relational schema. Hence, I can't be bound by the structure of a RDBMS and need something more flexible.

This blog posting considers the performance argument; a subsequent posting will address the flexibility argument.

For simplicity, we will focus this discussion on the workloads for which NoSQL databases are most often considered: update- and lookup-intensive online transaction processing (OLTP) workloads, not query-intensive, data-warehousing workloads. We do not consider document repositories or other specialized workloads for which NoSQL systems may be well suited.

There are two ways to improve OLTP performance; namely, provide automatic sharding over a shared-nothing processing environment and improve per-server OLTP performance.

In the first case, one improves performance by providing scalability as nodes are added to a computing environment; in the second case, one improves the performance of individual nodes.

Every serious SQL DBMS—such as Greenplum, Aster Data, Vertica, ParAccel, and others—written in the last 10 years has provided shared nothing scalability, and any new effort would be remiss if it did not do likewise. Hence, this component of performance should be "table stakes" for any DBMS. In my opinion, nobody should ever run a DBMS that does not provide automatic sharding over computing nodes.

As a result, this posting continues about the other component; namely, single-node OLTP performance. The overhead associated with OLTP databases in traditional SQL systems has little to do with SQL, which is why "NoSQL" is such a misnomer.

Instead, the major overhead in an OLTP SQL DBMS is communicating with the DBMS using ODBC or

JDBC. Essentially *all* applications that are performance-sensitive use a stored-procedure interface to run application logic inside the DBMS and avoid the crippling overhead of back-and-forth communication between the application and the DBMS. The other alternative is to run the DBMS in the same address space as the application, thereby giving up any pretense of access control or security. Such *embeddable* DBMSs are reasonable in some environments, but not for mainstream OLTP, where security is a big deal.

Using either stored procedures or embedding, the useful work component is a very small percentage of the total transaction cost for today's OLTP databases, which usually fit in main memory. Instead, a recent paper[1] calculated that total OLTP time was divided almost equally between the following four overhead components:

Logging

Traditional databases write everything twice—once to the database and once to the log. Moreover, the log must be forced to disk, to guarantee transaction durability. Logging is, therefore, an expensive operation.

Locking

Before touching a record, a transaction must set a lock on it in the lock table. This is an overhead-intensive operation.

Latching

Updates to shared data structures, such as B-trees, the lock table, and resource tables, must be done carefully in a multithreaded environment. Typically, this is done with short-term duration latches, which are another considerable source of overhead.

Buffer Management

Data in traditional systems is stored on fixed-size disk pages. A buffer pool manages which set of disk pages is cached in memory at any given time. Moreover, records must be located on pages and the field boundaries identified. Again, these operations are overhead intensive.

If you eliminate any one of the above overhead components, you speed up a DBMS by 25%. Eliminate three and your speedup is limited by a factor of two. You must get rid of all four to run a DBMS a lot faster.

Although the NoSQL systems have a variety of different features, there are some common themes. First, many NoSQL systems manage data that is distributed across multiple sites, and provide the "table stakes" noted above. Obviously, a well-designed multisite system, whether based on SQL or something else, is way more scalable than a single-site system.

Second, many NoSQL systems are disk-based and retain a buffer pool as well as a multithreaded architecture. This will leave intact two of the four sources of overhead noted above.

Concerning transactions, there is often support for only single record transactions and an eventual consistency replica system, which assumes that transactions are commutative. In effect, the "gold standard" of ACID transactions is sacrificed for performance.

However, the net-net is that the single-node performance of a NoSQL, disk-based, non-ACID, multithreaded system is limited to be a modest factor faster than a well-designed stored-procedure SQL OLTP engine. In essence, ACID transactions are jettisoned for a modest performance boost, and this performance boost has nothing to do with SQL.

However, it is possible to have one's cake and eat it too. To go fast, one needs to have a stored procedure interface to a runtime system, which compiles a high-level language (for example, SQL) into low-level code. Moreover, one has to get rid of all of the above four sources of overhead.

A recent project[2] clearly indicated that this is doable, and showed blazing performance on TPC-C. Watch for commercial versions of these and similar ideas with open source packaging. Hence, I fully expect very high speed, open-source SQL engines in the near future that provide automatic sharding. Moreover, they will continue to provide ACID transactions along with the increased programmer productivity, lower maintenance, and better data independence afforded by SQL. Hence, high performance does not require jettisoning either SQL or ACID transactions.

In summary, blinding performance depends on removing overhead. Such overhead has nothing to do with SQL, but instead revolves around traditional implementations of ACID transactions, multithreading, and disk management. To go wildly faster, one must remove all four sources of the overhead discussed above. This is possible in either a SQL context or some other context.

References

1. S. Harizopoulos, et. al., "OLTP Through the Looking Glass, and What We Found There," *Proc. 2008 SIGMOD Conference*, Vancouver, B.C., June 2008.
2. M. Stonebraker, et. al., "The End of an Architectural Era (It's Time for a Complete Rewrite)," *Proc. 2007 VLDB Conference*, Vienna, Austria, Sept. 2007.

Disclosure: Michael Stonebraker is associated with four startups that are either producers or consumers of database technology. Hence, his opinions should be considered in this light.

Reader's comment

You seem to leave out several other subcategories of the NoSQL movement in your discussion. For example: Google's BigTable (and clones) as well as graph databases. Considering those in addition, would that change your point of view?

—Johannes Ernst

Blogger's Reply

I am a huge fan of "One size does not fit all." There are several implementations of SQL engines with very different performance characteristics, along with a plethora of other engines. Besides the ones you mention, there are array stores such as Rasdaman and RDF stores such as Freebase. I applaud efforts to build DBMSs that are oriented toward particular market needs.

The purpose of the blog entry was to discuss the major actors in the NoSQL movement (as I see it) as they relate to bread-and-butter online transaction processing (OLTP). My conclusion is that "NoSQL" really means "No disk" or "No ACID" or "No threading," i.e., speed in the OLTP market does not come from abandoning SQL. The efforts you describe, as well as the ones in the above paragraphs, are not focused on OLTP. My blog comments were restricted to OLTP, as I thought I made clear.

—Michael Stonebraker

Michael Stonebraker is an adjunct professor at the Massachusetts Institute of Technology.