solidDB and the secrets of speed

How the IBM in-memory database redefines high performance

Antoni Wolski
Sally Hartnell

22 January 2010

A look at the technical secrets inside IBM solidDB

A relational in-memory database, IBM solidDB is used worldwide for its ability to deliver extreme speed and extreme availability. As the name implies, an in-memory database resides entirely in main memory rather than on disk, making data access an order of magnitude faster than with conventional, disk-based databases. Part of that leap is due to the fact that RAM simply provides faster data access than hard disk drives.

But solidDB also has data structures and access methods specifically designed for storing, searching, and processing data in main memory. As a result, it outperforms ordinary disk-based databases even when the latter have data fully cached in memory. Some databases deliver low latency but cannot handle large numbers of transactions or concurrent sessions. IBM solidDB provides throughput measured in the range of tens-to-hundreds of thousands of transactions per second while consistently achieving response times (or latency) measured in microseconds. This article explores the structural differences between in-memory and disk-based databases, and how solidDB works to deliver extreme speed.

Some RDBMS history

When the first data management systems emerged in the 1960s, disk drives were the only place to store and access large amounts of data in a reasonable time. RDBMS designers concentrated on optimizing I/O and tried to align the data access patterns with the block structure imposed by the drives. Design strategy frequently centered on a shared buffer pool where data blocks were kept for reuse, while advances in access methods produced solutions like the renowned B+ tree, which is a block-optimized index.

Meanwhile, query optimization tactics focused on minimizing page fetches wherever possible. In the fierce battle for performance, disk I/O was often the deadliest enemy, and processing efficiency was sacrificed to avoid disk access. For example, with typical page sizes of 8 KB or 16 KB, in-page processing is inherently sequential and less CPU-efficient than random data access. Nevertheless, it remains a popular way to reduce disk access.

When the era of abundant memory arrived, many DBAs increased their buffer pools until they were large enough to contain an entire database-thus creating the concept of a fully cached database. But within the RAM buffer pools, the DBMSs were still hostage to all the structural inefficiencies of the block-oriented I/O strategy that had been created to deal with hard disk drives.

Moving past the blocks

One of the most noticeable differences of an in-memory database system is the absence of large data block structures. IBM solidDB eliminates them. Table rows and index nodes are stored independently in memory, so that data can be added without reorganizing big block structures. In-memory databases also forgo the use of large-block indexes, sometimes called bushy trees, in favor of slim structures where the number of index levels is increased and the index node size is kept to a minimum to avoid costly in-node processing. The most common in-memory database index strategy is called T-tree. IBM solidDB instead uses an index called trie (or prefix tree), which was originally created for text searching but turns out to be perfect for in-memory indexing. A trie (the name comes from the word retrieval) is made up of a series of nodes where the descendants of a given node have the same prefix of the string associated with that node. For example, if the word "dog" were stored in a trie as a node, it would descend from the node containing "do," which would descend from the node containing "d."

Trie indexes increase performance by reducing the need for key value comparisons and practically eliminating innode processing. The index contains a node that is a small array of pointers to the lower level. Instead of using the whole key value to walk the tree by way of comparisons, the key value is cut into small chunks of a few bits. Each chunk is a direct index to the pointer array of the corresponding level: the first left-hand-side chunk to the first-level nodes, the second chunk to the nodes of the second level, and so on. Thus, the entire search can be performed with just a few array element retrievals. Also, each index node is a small data block (approximately 256 bytes in solidDB), which is beneficial because the blocks fit precisely into modern processor caches, increasing processing efficiency by promoting efficient cache use. Small data arrays like these are the most efficient data structure in modern processors, and solidDB uses them frequently to maximize performance.

Checkpoints and durability: Paths to speed

IBM solidDB uses several additional techniques to accelerate database processing, starting with a patented checkpointing method that produces a snapshot-consistent checkpoint without blocking normal transaction processing. A snapshotconsistent checkpoint allows the database to restart from a checkpoint only. Other database products do not normally allow that-the transaction log files must be used to recalculate the consistent state (solidDB allows transaction logging to be turned off, if desired). The solidDB solution is made possible by the ability to allocate row images and row shadow images (different versions of the same row) without using inefficient block structures. Only those images that correspond to a consistent snapshot are written to the checkpoint file, and the row shadows allow the currently executing transactions to run unrestricted during checkpointing.

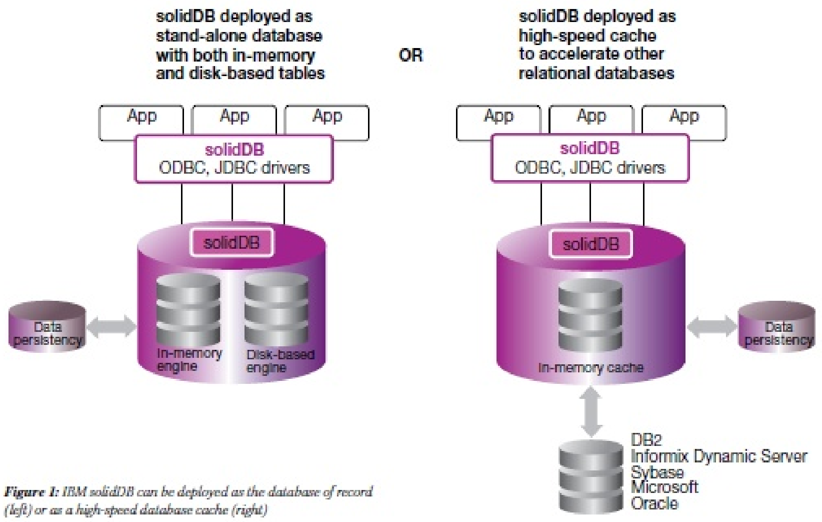
solidDB deployed as stand-alone database with both in-memory and disk-based tables
OR
solidDB deployed as high-speed cache to accelerate other relational databases
App
App
App
solidDB ODBC, JDBC drivers
solidDB
Data persistency
In-memory engine
Disk-based engine
In-memory cache
DB2
Informix Dynamic Server
Sybase
Microsoft
Oracle

Figure 1: *IBM solidDB can be deployed as the database of record (left) or as a high-speed database cache (right)*

Further, the solidDB query optimizer recognizes the different nature of the in-memory tables by estimating execution costs in a new way. Query optimization focuses on CPUbound execution paths, while a fully cached database will still be preoccupied with optimizing page fetches to mass storage that are no longer an issue.

Another technique IBM solidDB uses is the relaxation of transaction durability. In the past, databases always supported full durability, guaranteeing that the written data is made persistent the moment the transaction is committed. The problem is that full durability inflicts synchronous log writes, and thus it consumes resources and reduces response times. In many situations, accepting less durability for some tasks for the sake of faster response times is a perfectly acceptable trade-off. With solidDB, transaction durability can be relaxed at run time for a given database session or even for a single transaction.

IBM solidDB also increases database performance by helping developers avoid process context switches in client/ server interactions. By using a database access driver provided with solidDB that contains the full query execution code, a developer can effectively link the application with the DBMS code and use shared memory to share the data among the applications.

When all of these measures are applied and the application load is of a type that would inflict significant I/O in a traditional database, the increased throughput using solidDB can be an order of magnitude higher. Further, response time improvements are even more dramatic: latencies for query transactions are usually 10 to 20 microseconds and latencies for update transactions are generally less than 100 microseconds. In a traditional disk-based database, the corresponding times are typically measured in milliseconds.

solidDB speed and power

Beyond these performance advantages, solidDB also provides several additional benefits. It combines a fully transactional in-memory database and a powerful, disk-based database into a single, compact solution with the ability to transparently host part of the same database in memory and part on disk. And IBM solidDB is the only product on the market that can be deployed as a high-speed cache in front of almost any other relational, disk-based database (see Figure 1). Finally, solidDB delivers extreme availability, going beyond the typical five nines to 99.9999 percent uptime. In other words-if you're looking for extreme speed, you'll find it, but that's just the beginning for IBM solidDB.