UBM

Electronically reprinted from August 2012

Wall Street & Technology

Business Innovation Powered By Technology

Streamlining the Program Execution Path Is the Shortest Distance to Faster Trades

By Steve Graves, McObject

"Proximity" is the watchword for today's capital markets technology. Many organizations are hosting their trading algorithms as close as possible to securities exchanges — often in the exchanges' own colocation centers. The goal is to accelerate automated trades by reducing communication distance. Though the practice can sound far-fetched, given the speed of data packets traveling over fiber optic cables, it speaks volumes about the time scale of today's buying and selling and the competitive advantage of shaving microseconds from the trade cycle.

It is surprising, then, how rarely one hears discussion of another type of latency-inducing distance in trading: the execution path, or the length of the series of instructions that the CPU must complete to perform a task. Like physical distance in the example above, the time overhead imposed by CPU instructions appears insignificant on a small scale. Processor speeds are routinely measured in billions of instructions per second, after all.

Better Than Colocation

But automated trading and other real-time capital markets functions are computationally intensive — they burn through instructions by the millions or even billions. And compared to physical distance, execution path is supremely malleable. Once you colocate your system, you can't reduce physical distance further — and you enjoy no advantage over competitors that also have colocated. However, significant efficiency gains can result from many types of changes to software, ranging from fine-tuning bits of code (the efficiency gap between a switch/case and an if/then/else statement used in an algorithm can start to matter, when repeated thousands of times in a looping function) to complete redesigns of systems and applications.

Another advantage of a short execution path is that it increases the likelihood a code segment will already be in L1/L2 cache when needed by the CPU. When the system encounters an instruction that isn't already in the cache, it must retrieve that instruction from main memory — and this becomes a performance bottleneck. In a system with a 2.4 gigahertz CPU and a 400 megahertz front-side bus, for example, fetching from memory is the equivalent of speeding down the Autobahn at 240 kph, then having to slow down to 40 kph for an obstacle, then racing to the next obstacle at 240 kph.

To grasp the computational intensiveness of real-time capital markets, consider the risk management function within an algorithmic trading system. This code executes at the moment an order is placed and determines whether or not to allow the trade. Risk management algorithms sort and filter a flood of information, acquiring needed data points and applying complex math to assess risk along dozens of parameters. Because it must execute before a trade completes, risk management code is time-critical, and the amount of data that must be processed to accomplish its objective is growing, due to factors such as exchanges' expanding data broadcasts.

To continue with the example, let's say a firm's goal is to cut risk management latency in order to reduce order-processing time to below 1 millisecond. On hardware that can process 10 billion instructions per second — and excluding other factors that affect latency — the 1 millisecond limit sets the maximum allowable execution path for order-processing at 10 million instructions. Now, let's say that through clever programming, the firm improves software efficiency to achieve order-processing latency that is 35 percent faster than this limit — in other words, it cuts 350 microseconds from the processing time (or it can handle 3.5 million more instructions in the same time).

How does this compare to colocation's advantages? Packets travel over fiber optic cable at the speed of light, less some amount determined by the cable's index of refraction — 124,274 miles per second often is cited as a rule-of-thumb speed. At that rate, and all other factors being equal, the 35 percent gain in code efficiency equates to slicing about 43 miles from physical distance, which is well beyond the 25 mile minimum said to motivate firms' colocation decisions.

Results May Vary

Efficiency gains of 35 percent don't result from every application tweak or even redesign. But they do happen, often when a new approach is imported from some other realm of technology. For example, capital markets developers have successfully borrowed from the field of embedded systems, such as real-time aerospace and telecom systems. Embedded technology appeals to financial systems developers because it is fast (imagine what could happen if a jet's avionics weren't fast enough), deterministic (response time is predictable) and fault-tolerant. Embedded developers have long focused, sometimes obsessively, on trimming execution path in order to minimize CPU hits, both to gain performance and to accommodate the less powerful processors used in embedded systems.

An example of the transfer of ideas from embedded to financial technology comes from database systems, particularly the application programming interfaces (APIs) used by developers to call database functions from application code. Typically, if embedded software developers use an off-the-shelf database system, they choose one with a native language API that is navigational — that is, it consists of functions that are embedded in application code and work their way through physical data structures one record at a time, with application logic determining whether the current row is a member of the set of interest.

This approach contrasts sharply with the SQL API used widely in enterprise applications, including many financial systems. SQL provides a higher level of abstraction to programmers, which is convenient, but comes at the cost of a much longer execution path. Before physical data is even touched in a SQL operation, the command (the SQL statement) requires parsing and optimization phases. The latter is a particularly CPU-intensive step in which optimizer logic examines multiple ways of carrying out the operation, in order to determine the most efficient one.

The performance difference between the two kinds of APIs is dramatic and due almost entirely to a shorter execution path for the native API versus a longer one for SQL. In order to demonstrate the performance of SQL versus that of the native API, we performed identical queries against the same data (a three-table, 1.17 terabyte, 15.5 billion row database stored in main memory) using an in-memory database (IMDS) hosted on a SGI Altrix 4700 system with 80 dual-core, 1.6 Ghz Intel Itanium 2 processors (160 cores total) and 4 terabytes of NUMA RAM, running SUSE Enterprise Linux 9.

In the simplest query performed, the database system completed 28.21 million "SELECT" operations per second using the SQL API. The same query implemented with the native, navigational API yielded 87.78 million queries per second. In a more complex operation requiring a three-table join, the SQL API completed 4.36 million queries per second, compared to 11.13 million per second with the native API. Financial system developers are increasingly seeking out the latter type of interface and achieving higher performance via its use.

Execution path has proven just as relevant to minimizing latency as physical distance. To meet the competitive challenge of low-latency capital markets, trading organizations would do well to focus on both.

Steve Graves *co-founded McObject in 2001. As the company's president and CEO, he has both spearheaded McObject's growth and helped the company attain its goal of providing embedded database technology that makes embedded systems smarter, more reliable and more cost-effective to develop and maintain. Prior to McObject, Graves was president and chairman of Centura Solutions and VP of worldwide consulting for Centura Software. He also served as president and COO of Raima Corp.*