TimesTen

In-Memory Databases

The Catalyst Behind Real-Time Trading Systems

Infrastructure software to support real-time applications, such as securities trading, is now commercially available. The In-Memory Database (IMDB) is a key part of this infrastructure. Unlike the custom-built variants they replace, commercial products based on IMDB technology go beyond just high performance, adding message-processing interfaces, industry-standard APIs, transactions, fault-tolerant failover and recovery, event publishing, and connections to back-office RDBMSs.

Today's downsized development teams have enough on their hands simply dealing with application-level changes. They no longer need to code "below the application", nor is that a prudent strategy compared with today's proven commercial options.

In-Memory Databases

The Catalyst Behind Real-Time Trading Systems

Infrastructure software to support real-time applications, such as securities trading, is now commercially available. The In-Memory Database (IMDB) is a key part of this infrastructure. Unlike the custom-built variants they replace, commercial products based on IMDB technology go beyond just high performance, adding message-processing interfaces, industry-standard APIs, transactions, fault-tolerant failover and recovery, event publishing, and connections to back-office RDBMSs.

Today's downsized development teams have enough on their hands simply dealing with application-level changes. They no longer need to code "below the application", nor is that a prudent strategy compared with today's proven commercial options.

Introduction: The Insatiable Need for Speed

For securities trading systems, the prolonged bear market has done nothing to lower the volume of trades processed. Of course, monetary trading volume is way down, as are the average spreads in the U.S. markets post-decimalization. But the systems are working as hard as they ever have to route, match, and track trade orders. Indeed, Nasdaq's reported statistics indicate that we're still operating at about the same share trading volume as the heady days of late 2000 (see Figure 1).

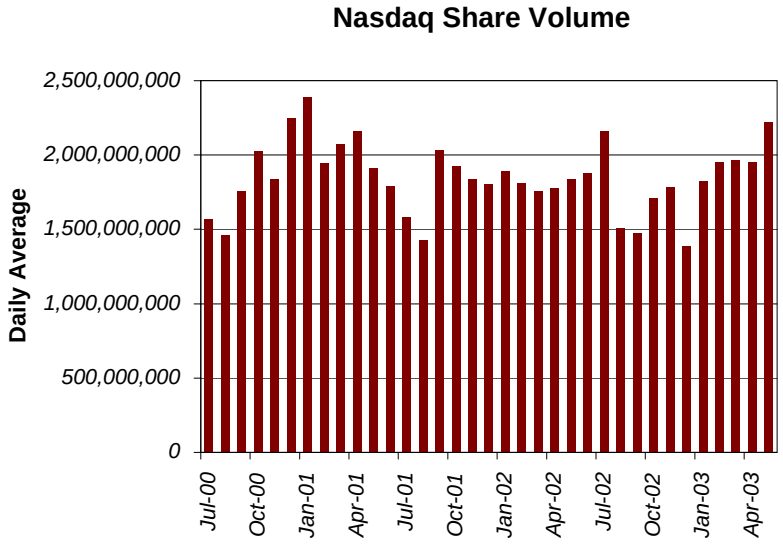

Figure 1
Trading Systems are Working as Hard as Ever
Despite the down market, the volume of trades completed today is almost as high as it has ever been, as shown by this chart of Nasdaq's daily trading volume from mid-2000 through mid-May 2003. Monetary trading volume has decline by 50 – 70%, making profitability the key goal for each trade. How quickly a trade is executed and at what price have become all-important to investors. Consequently, so are the speed and quality of the trading systems handling their business.

What's behind this are some significant changes in trading strategies and habits, including the explosive popularity of hedge funds and a dramatic increase in program trading. Many investors have resorted to short term buy then sell tactics, reflecting the uncertainty of the markets. How quickly a trade is executed and at what price have become all-important. Consequently, so are the speed and quality of the trading systems handling their business.

How quickly a trade is executed and at what price have become all-important. Consequently, so are the speed and quality of the trading systems.

So what happens when markets return to some semblance of "normalcy"? Many scenarios would suggest that trading volumes are going to increase. Today, and more so tomorrow, global investment banks and securities exchanges must be prepared for highly volatile markets, where *trading volumes could surge to over 1,000 trades per second*. Because of these extreme volumes, combined with the competitive urgency to provide immediate response, and the constant drive for differentiation, many of the largest trading firms are committed to developing their trading applications in-house.

The Build-It-All Dilemma

Until recently, commercial infrastructure software to support such demanding applications did not exist, forcing project teams to develop software "below the application" in order to achieve high performance without sacrificing reliability. These efforts almost always included the staging of time-critical data in memory, to avoid the delays inherent in even the fastest RDBMS.

In-memory data drives functions such as **pricing, order routing, order matching, position-keeping, trader alerts, program trading**, and **risk analysis**. A firm's competitiveness depends on its ability to keep pace with the market through these key functions. Without optimum performance, trading strategies are marginalized and price improvement is challenging.

It's easy to understand why trading firms felt obliged to develop in-memory data management technology. There were no commercial options and, until a few years ago, profitable trading operations easily funded such development and maintenance. Even so, this was challenging work, far different from applications development. Not only did the infrastructure need to perform, it had to be rock-solid reliable and never lose a trade. As such, these implementations were modest in functionality and "hard-wired" into the applications to minimize complexity. They worked, but weren't easily or quickly changed, and came at a high cost.

The "Do More With Less" Era

"Economics are pushing banks away from proprietary development, to using more vendor-based products and finally to consolidating their vendor relationships around unique and strategic vendors."

—Larry Tabb, CEO, The Tabb Group
Window on the future of fintech, (May, 2003)

The securities industry has been rocked with change since the start of this century. Recession and decimalization have conspired to undermine the spread-based business model. Among other changes, new regulations, new trading strategies, and new trade execution venues have forced continual enhancements to trading systems. Down-sized development teams are being asked to rebuild both applications and infrastructure on tight schedules. For most firms, this equation doesn't balance.

The stakes are high, for the economics of securities trading dictates consolidation – both for economies of scale and natural migration – toward those firms that provide the lowest trading cost, the richest choice of services, and the greatest likelihood of price improvement.

Financial firms can no longer afford to build infrastructure software, nor can their systems be competitive with the basic features that accompany home-grown technologies. Today's trading operation demands more – **flexibility, rock-solid reliability, replicated sites that are synchronized for disaster recovery, pre-trade analytics, event-driven alerts, real-time position keeping**. These are fundamental issues of competitiveness.

Commercial In-Memory Database (IMDB) products, with the additional functionality that surrounds them, can substantially reduce the risk and time to completion for these projects. Developers can focus on retooling their application logic, and take advantage of infrastructure software that's purpose-built for real-time, highly-reliable systems. The support of common industry-

standard interfaces by IMDB products enables easier integration and future flexibility. Down-sized development teams now have a good chance of a successful outcome.

Infrastructure Software for Real-Time Applications

The dilemma just described isn't confined to capital markets applications. Other industries, such as telecommunications, travel and logistics, and factory automation, have also had performance-demanding applications that required the development of supporting infrastructure software.

The fundamental problem is that infrastructure software has historically been commercialized for the "enterprise" mass market. The goal of enterprise software is to be "good enough" for as many companies and applications as possible. Demanding applications, such as securities trading, aren't the focus of these products. Over time, this gap inevitably expands, as vendors overload their products with checklist features designed for even broader applicability. This is the case with the RDBMS and Application Server product categories today. These products are rarely used in the front-office trading infrastructures of the largest brokerages and exchanges (see Figure 2).

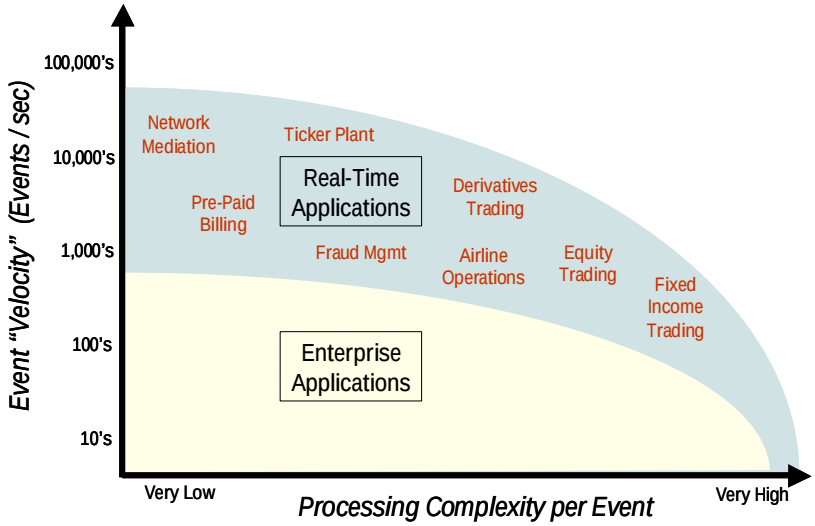

Figure 2
Demanding Applications Require a Different Supporting Infrastructure
Applications that must process a thousand or more simple business events per second (or several hundred complex events) require a supporting infrastructure that goes beyond the level of conventional enterprise infrastructure software. In capital markets applications, ticker plants represent the highest volume of simple events, whereas fixed-income trading represents the most complex processing. Equities and derivatives trading systems fall in between those extremes.

Keep it Focused

The ideal commercial infrastructure software for supporting trading systems would contain key portions of an RDBMS (for data management) and an Application Server (for integration and fault tolerance) – focused on maximizing performance and availability, and enabling the developer to concentrate on writing business logic only. In addition, this infrastructure should be suitable for the popular and emerging hardware platforms, from multi-processor systems running Unix, Linux, or Windows, to new building-block blade server configurations.

The In-Memory Database was developed as a core technology in response to the need for real-time infrastructure software. Many of the functions desired in a real-time infrastructure software solution involve data management functionality, or are a natural extension of it. The balance of this paper is a survey of the attributes and applications of the In-Memory Database, and the evolution of additional functionality that, over time, should result in a generally complete solution for real-time infrastructure software.

The In-Memory Database (IMDB)

The profound improvements in computer architectures over the last decade gave rise to In-Memory Database technology. CPU performance (as measured by circuit density) has been doubling, on average, every year and a half. Memory chips have doubled in capacity at half the cost in the same intervals. Today, a gigabyte of physical memory for a server costs around $400. Relatively small servers come with 32 gigabytes of standard memory, with the capacity to exceed 100 gigabytes. Ten years ago it would have been difficult, perhaps impossible, to find a computer with such memory capacity, and the price would have been out of reach for all but a few (see Figure 3).

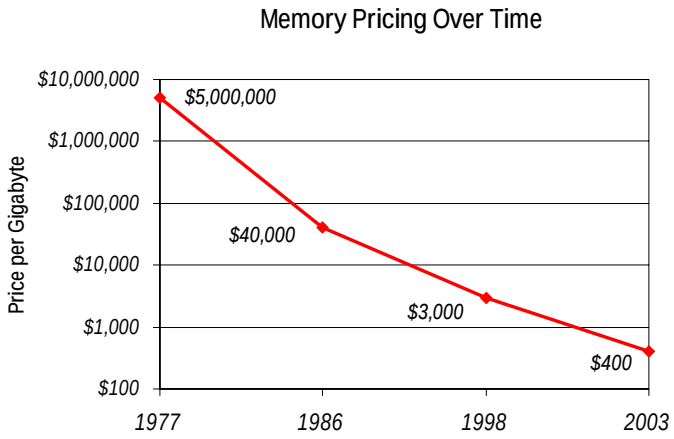

Figure 3
Memory Prices are no Longer an Issue
Memory pricing and density have followed the microprocessor technology curve over time. The graph at right shows actual prices for 1 gigabyte of RAM for a server computer on the year specified. Today's price for a gigabyte of RAM for an Itanium2 server (and almost any server) is $400. Once prohibitive, the cost of RAM is no longer a factor in the consideration of production-scale In-Memory Databases for front-office applications. Back-office systems and data warehouses are still largely the domain of disk-based databases, which may store hundreds of gigabytes of historical data that's rarely accessed.

Magnetic disk performance has increased much more slowly. Thus, the performance gap between CPUs and disks has widened substantially. That's not news to most computer users. It's well known that data management software, such as an RDBMS, will attempt to hold as much of the data in memory as it can, in order to avoid the performance penalty of disk access.

What most people don't realize is that the conventional RDBMS products *have been* successful in their quest to remove the disk bottleneck. Over the years, RDBMS products have become very sophisticated at predicting what data should stay in memory the longest, and when new data should be pulled from disk into memory. These algorithms are generally correct 75% or more of the time, so there's rarely a performance penalty due to applications waiting on disk drives. And if the database is small enough to fit entirely in memory, an RDBMS will let you do that too.

In adding logic to outsmart the disk problem, the RDBMS vendors have made CPU bandwidth their bottleneck.

So if disk I/O isn't the performance issue, why shouldn't an RDBMS be suitable for real-time applications? As software professionals know, there's always a performance bottleneck somewhere. In adding logic to outsmart the disk problem, the RDBMS vendors have made CPU bandwidth their bottleneck. Ever wonder why an RDBMS needs its own server platform, instead of sharing one with applications and other software products? Even when the entire database is in memory, an RDBMS will still crank through the same CPU-consuming algorithms as always, because it's central to their design.

This dilemma was not lost on the software researchers. In the 1990s, they invented new designs for database systems, focused on using the smallest amount of CPU necessary. Knowing that hardware systems would have liberal memory, they designed these architectures with memory-resident data, which removed the need for CPU-consuming logic designed to get around the disk bottleneck. By relying on the data always being in memory, these researchers were able to redesign data and index structures in ways that further reduced the

processing required. Disks were still used in these designs, simply to provide persistence and recovery of the data in case of system failures, much like tape drives previously backed up disk drives.

The end result of In-Memory Database technology is a significant reduction in the amount of CPU needed to complete standard database operations.

The end result of In-Memory Database technology is a significant reduction in the amount of CPU needed to complete standard database operations – as compared to a fully-cached RDBMS. The actual difference depends on exactly what work is being done. Read operations (e.g. reference data lookups) exhibit the greatest speedup. Write operations, which probably entail some logging of changes to disk to guarantee recoverability, could be slowed down somewhat by the disk operation. In practice, applications are almost never only reads or only writes, but rather a mixture. Processing a trade order involves a mixture of reads, updates, and inserts, for example. It is not unusual for an IMDB to perform 10 times faster (or said differently, to use 1/10 the CPU) as compared to the same application using a cached RDBMS.

To be clear, an IMDB, though able to access large amounts of in-memory data compared to a few years ago, is still nowhere the equal of an RDBMS in terms of database capacity. In many cases where an IMDB is appropriate, a back-office RDBMS will exist as well, either providing reference data for the IMDB, and/or receiving completed transactions from the IMDB (see Figure 4).

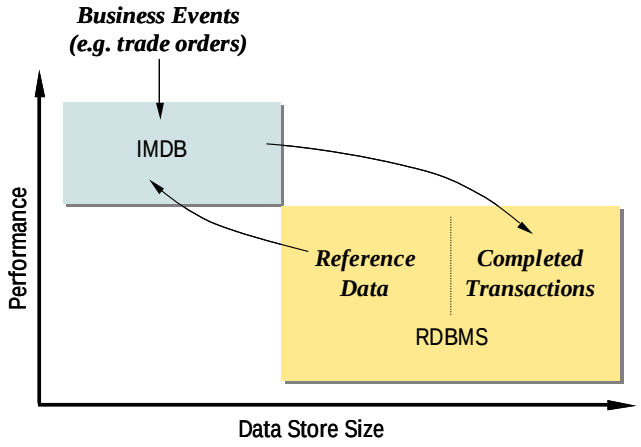

Figure 4
The Relationship Between an IMDB and RDBMS in a Trading Application
An IMDB is often deployed in conjunction with an RDBMS. The IMDB is used to capture and process transactions in real-time, perhaps using reference data pre-loaded from the RDBMS for validation checking. Completed transactions are eventually moved from the IMDB to the RDBMS for long-term storage.

It wasn't until the late 1990's that commercial products based on In-Memory Database technology became available, and not until more recently that they reached the level of customer usage and functional completeness that would warrant widespread consideration. Inevitably, as computing technologies evolve, so should software architectures. It's surprising that large industries such as capital markets and telecom were unable to procure commercial solutions until now, and that it's taken a recession and structural upheaval to force companies to evaluate their make vs. buy options. *Technology self-sufficiency created a self-fulfilling proposition, it would seem.*

The comparison of an IMDB with an RDBMS is instructive, but in the case of large trading systems, not the relevant comparison. As previously mentioned, custom caches and in-memory data structures are used in such systems already. Assuming they are well implemented, they should perform adequately, albeit in a limited context (primarily read operations, or minor updating).

Compared to custom in-memory solutions, application development is where a commercial IMDB product has a clear advantage. IMDB products provide standard programming interfaces, just like an RDBMS, which insulate the application code from the workings of the IMDB. As a result, there is an abundance of development talent that would be productive quickly with an

IMDB. Custom in-memory solutions rarely include a programming layer that insulates one application from the other, or even insulates the application code from the in-memory logic. Applications and data structures tend to be hard-wired together, making them hard to disentangle later on if features need to be added. Applications that use an IMDB product can be easily modified and new ones added, without affecting other applications. *This degree of flexibility is invaluable in today's changing securities climate.*

Data integrity and functional richness are the other major advantages of an IMDB product. It's one thing to develop a read-only in-memory cache, but quite another to develop a read/write transactional version, with multi-user locking, logging, recovery, and change replication for high-availability. Indeed, the cost of an IMDB product may be less than the cost of a few trades lost to a fragile custom implementation.

The Larger Picture

A complete infrastructure for real-time trading systems goes beyond data management. Ideally, developers should only have to write the business logic of their trading applications, and the infrastructure would take care of everything else.

For most applications, that means integration with messaging middleware, transaction scheduling and execution, transparent connections to a back-office RDBMS, outbound data publishing (for trader alerts, program trading, or real-time data snapshots), and automatic failover and recovery with guarantees of no lost data or messages. A tall order, but certainly achievable when starting with a foundation of well-conceived IMDB technology. In the near future, there should be little reason for financial firms to code "below the application" in order to support their business applications.

Assessing the Value

An investment in a commercial IMDB can generate a tremendous ROI, if it's used proactively to help shape a firm's business model.

An investment in a commercial IMDB product can generate a tremendous ROI, if it's used proactively to help shape a firm's business model. Creating and launching new products and services, aggressively seeking out price improvement, automating alerts and varying program trading strategies – such elements of an aggressive trading strategy are easily supported by an IMDB product.

Calculate the value of a few percentage increases in market share because your organization has introduced unique and powerful new services ahead of the market. The value can be surprising, even with a small trading operation Add to that the extra assurance of using commercially-hardened software, with a dedicated support organization and a regular stream of enhancements. Then consider the application improvements you can make using the engineers who would otherwise be developing or supporting custom-built infrastructure software.

This is not a question of make versus buy cost comparisons. It's really a market share and market positioning question. Do you want to take market share or relinquish it? In today's trading environment, that's the fundamental strategic question.

TimesTen, Inc. **www.timesten.com**

Corporate Headquarters:
800 West El Camino Real ◦ Mountain View, CA 94040 ◦ USA
800.970.1248 ◦ 650.526.5100 ◦ fax 650.526.5199

EMEA Headquarters:
500 Chiswick High Road ◦ London, W4 5RG ◦ UK
+44 (0) 20 8956 2532 ◦ fax +44 (0) 20 8956 2536