

The Automated Trader **Technology Forum**



Application Latency

With:

- **Ary Khatchikian**, president and chief technology officer, Portware
- **Amir Prescher**, vice president, business development, Voltaire
- **Matt Meinel**, global director, business development, 29West
- **Gena Ioffe**, CEO, EGAR Technology
- **Ali Pichvai**, managing director, Quod Financial
- **Jonas Lindqvist**, head of system development, NeoNet
- **Yuriy Shterk**, vice president, product development, CQG
- **Nimrod Gindi**, business development manager, Mellanox Technologies

As growing competition places ever greater emphasis on the fine-tuning of automated and algorithmic trading programmes, AT asks leading solutions providers for their views on the tweaks that can minimise application latency and maximise performance.

Application vs. network latency: which is more important in improving algorithmic /automated performance?

Khatchikian: Improving application latency is more important without question. There is always a limit to what you can optimise regarding networks. With applications it's more complex because they are always changing.

A few lines of bad code can bring an algorithm to a crawl if you're not paying attention. In some cases, applications have been optimised on the algo level to overcome network latencies where cost was a serious issue.

Prescher: To achieve overall end-to-end latency reduction, both application latency and network latency must be addressed. Application latency has a few common causes. Code 'bloat' is one, but we find this to be the cause less often than most people would think. Trading firms do a lot of due diligence on their algorithmic trading applications and optimise those applications as much as possible.

The majority of the time, we see electronic trading infrastructures

running out of CPU cycles (even though they are running on the latest and greatest hardware). High market volumes and the resulting high packet rates are to blame for a large part of the high CPU utilisation problem. InfiniBand-based solutions can help because they offer network processing offload technology which helps to significantly increase packet per second rates, as well as lowering the CPU utilisation that occurs when processing the high packet rates. This is very different from traditional TCP offload engines (TOEs). Since market data feed packets and general FIX packets tend to be between 60 and 200 bytes, traditional TOEs are not useful. InfiniBand-based solutions can deliver latency that is 20-500 per cent lower (depending on the applications that are being used) than what is achievable with Ethernet-based solutions.

Meinel: Obviously, if the application has unnecessary latency when it is designed to be hitting databases halfway around the world, then application latency needs to be addressed first. However, most algorithmic trading shops have solved the obvious application latency issues and are focused on improving network latency at this point.

Ioffe: Network latency depends on the market. In FX markets, for example, volume of quotes is low and therefore bandwidth requirements are low as well. Conversely, US equity options markets are very liquid and network latency is a big issue. In most cases, bandwidth is readily available and there are vendors now who can provide a reliable network pretty much anywhere in the

continental US, so it is pretty much a question of having enough budget to get a reliable network.

Application performance is the biggest challenge. Simple arbitrage opportunities are pretty much gone in most markets, so applications become increasingly complex. Of course hardware is getting more powerful as well, but there are always hardware limitations, so developers have to come up with optimised code and smart models, which requires talent and expertise.

Pichvai: Improving latency is in fact a three-part process focusing not just on network latency and application performance, but also the main application architecture itself. The former is the low-hanging fruit. But the architecture and application performance is where the most improvement can be made, for instance by redesigning market data to make it low latency, or designing applications that are event-driven with a high throughput.

Lindqvist: Without low network latency, no application will be fast enough. So although both aspects are important, low network latency is needed to benefit from low application latency.

Shterk: Going forward, I believe that trading models whose performance is related to some relative competitive advantage in network latency will give way to the smartness behind the model itself. As technology continues to improve, network latency issues will reduce. The best technology is becoming more affordable and markets are becoming more liquid. This all means reduced profitability for traders relying on some sort of latency advantage.



Yuriy Shterk

“... trading models whose performance is related to some relative competitive advantage in network latency will give way to the smartness behind the model itself.”

Gindi: They go hand in hand. We will see algorithmic/automated performance improve for three reasons: applications will be more efficient; more CPU cycles will be free to handle application processing instead of network protocol processing; external impacts of the network, including overall network latency, network congestions and predictability of the variance of latency will be lower.

What are the biggest challenges in using multi-threaded applications as a means of reducing application latency in algo/auto trading?

Khatchikian: The biggest challenge is the complexity of the application. When implementing a multi-threaded application you must deal with threads that share common data. In addition, you must determine whether your bottleneck is either CPU or I/O bound. If you're looking to



Amir Prescher

“High market volumes and the resulting high packet rates are to blame for a large part of the high CPU utilisation problem.”

improve the rate of market data consumption or route more orders to the marketplace then it is I/O bound. If your application spends most of the time processing complex mathematical computations then it is CPU bound. Unfortunately, most algorithms are both and in each case you must determine whether the performance gained outweighs the complexity of implementation.

Meinel: One big task is finding the appropriately talented programmers who understand both how to write/debug multi-threaded applications and how the nuances of the specific financial products and specific markets operate. Without these skills existing in one person, human communication becomes the bottleneck in keeping the algo engine tuned to the market.

Ioffe: The majority of applications need to run calculations while at the same time processing multiple input sources, for example feeds of

market data and executions. They also keep vast amounts of data in memory. That creates a lot of problems in synchronising data and calculations and balancing optimal load at the same time. The next biggest issue is that many Intel boxes run Windows, which is not a very good environment for multi-threaded applications. Industry standard application servers are designed to handle transaction processing not high performance calculations; to optimise their performance, developers have had to write lots of code at the system level which is expensive and error-prone.

Pichvai: Multi-threading is a must because it allows scalability as volumes increase. Trading however puts some constraints on the use of multi-threading to its full extent. Execution messages for example – FIX or market/exchange-driven – have a certain sequencing, and even when they are processed very quickly, still have to be re-sequenced according to the standard. The limiting factor therefore is often not the application itself but its surrounding counterparties and applications.

Shterk: The challenge is to design and implement systems that take advantage of the benefits available from the latest developments in hardware and operating systems. Whether it's displaying the data, networking, the proper use of thread pools, understanding the implementation of these new technologies is the key issue. Applications are being developed that do not exploit the new technologies available and thus are not as fast or as robust as they could be. One specific example

would be developers over-designing the hardware and applications. For example, say there is a lot of contact switching between different threads, even though you are running a multi-threaded application, it may still be running slower because you are wasting resources.

Do you see any specific software advances being made in parallel processing that are particularly relevant to reducing application latency (e.g. to handle multiple simultaneous inputs)?

Khatchikian: The major software advances we have seen in parallel processing are to do with grid computing. Both computational grids and data grids can prove to be beneficial in reducing application latency. A computational grid can be used to process and distribute calculations to servers solely responsible for



Matt Meinel

“... we expect to see the core of most systems written in software, with hardware acceleration used for data feed handlers, core numeric calculations or logging.”

order routing. A data grid can be used to share and manage large amounts of tick data.

Prescher: Trading firms used to look at real-time JVM (Java Virtual Machine) and similar technology before eventually giving up and going back to C++ (speed/latency being the primary drivers). These days, we are seeing the next stage of software evolution where people are beginning to code the network/transport logic to native InfiniBand APIs (instead of BSD sockets) in order to completely eliminate TCP from the latency equation. Since native InfiniBand APIs are complex, typically customers outsource the porting of the transport portion of their application to their InfiniBand provider. Providers that have invested in the development of software libraries specific to the financial services industry make the porting process easier to reduce overall development time.

Pichvai: At this stage, in pure trading, parallelisation is being limited by the constraint I mentioned earlier (the sequencing of the messages), so I would say that any computing breakthroughs in parallel processing can't yet be fully leveraged in trading.

Lindqvist: The easier handling of multiple processors using an API such as OpenMP is definitely interesting, but much like using SSE to work on multiple values in parallel, you need to design your code in certain ways in order to make proper use of it.

Shterk: Processing simultaneous multiple inputs is one advance, as well as the handling of large amounts of data coming into your applications. The latest advances in software technology are making it easier to display large amounts of data on the screen while the data is being processed for other uses including the networking among the various threads.

And what about advances in hardware, e.g. high-performance video cards or field programmable gate arrays?



“In this area, you are doing a massive number of small calculations...”

Khatchikian: We have seen a trend where network appliances are taking on more application-specific duties. In our arena, there are several vendors that are currently providing appliances that handle FIX protocol communication.

Prescher: FPGAs are very interesting, but super-complicated. A software upgrade for an FPGA

solution requires an FPGA firmware upgrade – which is very complex compared to traditional software upgrades. Since very few trading firms employ VHDL (VHSIC Hardware Description Language) developers, most firms look to outsource FPGA development and maintenance. As always, it is important to understand where the bottlenecks lie before embarking on a long FPGA project. In our view, use of current network processing offload and kernel bypass capabilities, along with faster server hardware and better performing off-the-shelf market data software, will provide a sufficient performance boost for the great majority of firms. A firm that rolls out its own FPGAs after deploying InfiniBand-based ticker plants and electronic trading clusters – and after having refreshed server hardware and market data software – could potentially see single digit performance gains. However, the rule of being as slow as the slowest component holds very true for FPGAs.

Meinel: We've seen a lot of interest in hardware solutions for improving Java performance, fixed income or derivatives calculations or disk performance. Clearly, there seems to be a match between algorithmic trading and hardware solutions, but the ability to tweak algorithms quickly is still an important requirement. So we expect to see the core of most



“Where do you stand on the Java vs C++ debate?”

systems written in software, with hardware acceleration used for data feed handlers, core numeric calculations or logging.

Pichvai: The algorithmic trading infrastructure can be divided into upstream and downstream. The upstream is the analytics part, which provides the decision making. If based on pure mathematical modelling, it requires a lot of processing power. The downstream is the trading itself broken down into order trading and market data. The upstream can benefit from the latest advances, while the downstream is less a problem of pure processing. Indeed, all the latest hardware ideas are well adopted in both risk management and pricing. In this area, you are doing a massive number of small calculations, so a purpose-built

system can be applied in the upstream (real-time) quantitative part of the trading, but not on the down-stream trading part.

Shterk: For auto-trading, the advances in hardware technology in the past couple of years by manufacturers have been in the speed of data access, data analysis, data writing, data processing and data storage.

Gindi: Hardware accelerators, including CPU off-loading FPGAs, as well as network adapters that provide superior network protocol processing offload, can provide hundreds of percent in performance advantage. These more capable network adapters can achieve multiple tasks from lowering CPU processing overhead to speeding up application time to the line/wire.

From an application latency reduction perspective, which programming languages are most suited to developing algo/auto trading models. Where do you stand on the Java vs C++ debate?

Khatchikian: The language that is most suited for algorithmic development is determined by the team implementing the solution. We have seen clients develop algorithms in Excel, VB, S+, Fortran, C, C# and Java. Clients like to work with languages that they are comfortable with and select languages based on how quickly they can implement and enhance their algorithms. We decided to use Java for the following reasons:

- Platform independence. With a single codebase, we have run our application on some of the best performing 32/64 bit platforms.



Gena Ioffe

“Industry standard application servers are designed to handle transaction processing not high performance calculations.”

- Productivity. Java allows us to effectively manage the most complex applications with teams large and small.
- Dynamic optimisation and loading. Not only are applications optimised in real-time but are capable of deploying application logic without the need to restart systems.
- Stability. With built-in memory management and exception handling we are not exposed to memory overwrites or leaks that can either shutdown an application unexpectedly or bring it to a crawl if not properly managed in the development phase.
- Widespread market adoption and acceptance. Not only is there a wealth of third-party applications which allow for greater productivity, Java has been validated in the financial

services space and some of the most successful providers of FIX engines use Java as their language of choice.

Prescher: We use C++ for the most part with very careful and deliberate use of Java where it provides a true value-add (with minimal or no sacrifice to performance).

Meinel: For algorithms in equity and FX markets, definitely C/C++. For fixed income and interest rate derivatives, Java has the edge.

Ioffe: We combine both. C++ is better for developing high performance calculations, Java is much better at handling simultaneous inputs and delivering data across the enterprise.

Pichvai: We still believe that a compiled code such as C++ is more efficient than Java. The operating system is another factor worth considering. We have noticed that when the Intel-based clock is higher (i.e. processor clock speed), we attain better performance.

Lindqvist: C++ can be very slow if you are not careful, particularly in comparison with Java. C++ demands more from the developer, but in turn offers the opportunity to achieve faster code. You need to find where the bottlenecks are. Python or Perl may well be perfectly fine languages for automated trading strategies that do not require fast real-time calculations. However, when calculating something complex you would probably want

to implement that specific code in something faster.

Shterk: It all depends on the knowledge and skills that the application developers have. While different languages are better suited for different needs, if the team is better skilled in C++, then they should stick to C++. All of the languages have strengths and weaknesses. My background is C++, but experts in C++, C#, Java or Visual Basic will likely do the best job within the technology they are used to.



Nimrod Gindi

“... with the new real-time JVMs and their direct relationship to the network, Java is becoming ... at least equal to C++ in performance.”

Gindi: Common sense says that the lower you'll go into the programming languages/levels, the less overhead you'll experience and from that perspective assembler optimisations will provide the best improvement. Gut reaction might say that Java is not on an even field ▶



with C++, but with the new real-time JVMs and their direct relationship to the network, Java is becoming a more prevalent programming language and at least equal to C++ in performance.

To what extent can efficient application-level memory management reduce application latency?

Khatchikian: The implementation of a distributed cache can greatly reduce application latencies. The most expensive part of an application revolves around I/O. Any opportunity to avoid constant reads and writes to disk will improve performance dramatically.

Ioffe: Application-level memory management can reduce application latency to a great degree, but it involves very complex programming in a multi-threaded environment. Very good results can be achieved by utilising database algorithms if



Jonas Lindqvist

“It is possible to pre-calculate data for different scenarios and use it directly when needed.”

done intelligently. Many major databases cache data extensively on their own. Specialist database development companies will achieve better results than application developers using their own memory management techniques.

Meinel: Having one’s application explicitly handle memory management definitely allows less variability in latency. It may or may not reduce overall latency, but it certainly allows the system to endure latency at less inopportune times. If you are talking about Java, we have customers report spikes in latency from sub-millisecond to a 100 millisecond when garbage collection runs.

Pichvai: No high performance application can have totally efficient memory management. The mistake that Java-based developers often make is to ignore this fact. Memory management is not just about limiting leakage, but also about how you can release the memory once processing has occurred.

Lindqvist: Allocating and reallocating memory is expensive, so it should be done with care. Further, the memory can get fragmented, and when the computer decides to merge free blocks, you can get a non-deterministic performance hit similar to that of garbage collectors, even when the programme is written in C/C++. By handling memory in an efficient manner, you can save some time, but not maybe as much as many assume.

Shterk: The memory management errors that developers make are causing a large amount of performance issues. Common mistakes include unnecessary memory swapping, unnecessary allocation and de-allocation, and over allocation. All of this leads to an application that is using up a lot of resources requiring an unnecessary amount of time.



Ali Pichvai

“Memory management is not just about limiting leakage, but also about how you can release the memory once processing has occurred.”

Are there any specific ways in which the calculation of common analytics (e.g. statistical functions) can be improved to reduce application latency?

Khatchikian: Event Stream Processing or CSP environments can reduce the latency of common analytics. Rather than simply retrieving raw market data from a provider, these analytics could be

published alongside primitive market data elements.

Prescher: Improvements to infrastructure go a long way here. Things like running applications on the newest server hardware and making applications as multi-threaded as possible will help.



Ary Khatchikian

“While on the surface minimising external library calls could reduce latency, focusing resources and energies on other areas may be more productive.”

Converting algorithmic trading and market data environments to InfiniBand can reduce network latency by as much as 500 per cent and reduce end-to-end application latency between 20-200 per cent (depending on the application). A side benefit is the ability of those applications to process larger workloads due to the fact that resources that were consumed by network processing are now available to the application.

Lindqvist: By making simplifications to models, you can

sometimes get a faster calculation and a result that is good enough to be useful. However, you have to be careful to make assumptions that are really valid. It is possible to pre-calculate data for different scenarios and use it directly when needed.

Is there an argument for minimising external library calls in algo/auto trading applications, given the highly competitive nature of today’s financial markets?

Khatchikian: While on the surface minimising external library calls could reduce latency, focusing resources and energies on other areas may be more productive. Reducing latency with respect to algorithms is so broad and can require expertise in areas from application development, to network design and implementation.

Pichvai: The distinction needs to be made between what is common knowledge and what is actual in-house knowledge. Utilising external libraries in common knowledge provides lower cost and sharing of information – for instance VWAP is now a common algorithm so the usage of some external libraries can be recommended. But even for the external libraries, the way they are implemented can become very proprietary, and therefore create high value.

Shterk: There is always an argument for avoiding unnecessary steps. The library calls definitely introduce latency, but the question is: how critical is it for your application? You should not rely on

reducing microseconds; you should focus on developing a smart system.

Has grid computing yet been harnessed to its fullest extent to support algo/auto performance?

Khatchikian: Grid computing hasn’t yet been able to benefit algos to the fullest extent for two key reasons. First, grids require better communication between nodes in order to improve algo performance. Second, efficient sharing of data - along with broadcasting the health of each node - would make a serious impact on the performance of algorithms.

Prescher: Hardware commoditisation and clustering are happening all across the financial services industry. Using a high performance, low latency InfiniBand interconnect as the cluster fabric is critical for supporting trading application performance.

Lindqvist: No. Dividing a calculation between several computers will be used more. Calculating several ‘what if’ scenarios in parallel can improve the time needed for a programme to make the right decision.

Gindi: It has yet to be fully harnessed. We can see that it is being used to an extent and I would say in general today it is being used up to 50 per cent of its capability. Market leaders understand this and are already on the move to have the full potential used in 2007 production systems. 