WHITE PAPER

Electronic Trading System Performance

Introduction

High-performance trading has driven the infrastructure to deliver ever-lower latencies, resulting in a need for visibility into that infrastructure at correspondingly shorter time-scales. This visibility must be based on timestamps with a precision that is at least an order of magnitude or two finer than the latencies being measured. That is, the resolution of the measurements should be no more than 10%, and preferably under 1%, of the latencies in question.

In the case of our work with Nomura, their NXT Direct platform delivers latencies under 3µs (three millionths of a second). In order to characterize the performance of the system fully, Nomura needed a solution to deliver sub-microsecond precision time-stamping and latency measurements. For example, an optimization of their platform that increases performance by 10% would not be measurable at microsecond resolution.

High Performance Trading Systems

Corvil offers nanosecond-granularity latency measurements as part of its Streaming Analytics Platform in support of the relentless decrease in the latency of high-performance trading systems. It is interesting, however, to dig beyond this simple answer and explore some of the techniques and architectures that have allowed such systems to achieve the performance levels that they do.

Let us start by looking at the high-performance trading loop and survey its constituent systems and components. All such systems differ in their details, but there are some broad generalizations we can make. On the exchange side we have:

- The matching engines, where the market is actually implemented in the form of order books for different traded instruments.

All such systems differ in their details, but there are some broad generalisations we can make

- The gateways, which authenticate trading members and route their orders to the appropriate matching engines.
- The market data publishers, which track the live order books and publish their state as either snapshots or the changes in their state.

On the market participant side, there is a much greater diversity of trading systems:

- At one end of the spectrum there are highly scaled DMA systems offering sophisticated order-execution functionality through a network of EMSs, OMSs, and smart-order routers. The broker's clients typically access these DMA systems through FIX engines, which provide a uniform interface to those clients irrespective of the venue at which their orders are ultimately executed.
- At the other end of the spectrum, there are algorithmic trading engines that consume exchange market data feeds raw and issue trading instructions directly back to the exchange over a high-performance messaging protocol. These single-host systems are no less sophisticated than distributed DMA systems, but are stripped down to the bare essentials for ultimate performance.

In parallel to the order-entry path there is also the market data path with:

- Feed-handlers to arbitrate between redundant A- and B-feeds, and normalize the multitude of different exchange-native market data feed formats.

- Ticker-plants to filter and fan out the market data to all the subscribers that require it.

Finally, there are the networks that connect all these sub-systems together to create the full end-to-end trading loop. This crude summary necessarily ignores much of the supporting infrastructure such as power, cooling, and management. It noticeably neglects even storage and databases: this is because the low-latency path, followed by orders, executions, and market data ticks, typically does not touch storage. It is not that orders and ticks are not persisted; it is that storage systems are too slow to be part of the low-latency/high-performance data-path.

Intel and AMD CPUs provide an ever-growing level of performance driven by Moore's law

High Performance Architectures and Technologies

Having summarized the major components of trading systems by functional role, let us now turn to the technologies they use.

Software on CPUs

Much of the core functionality of these systems is implemented in software on general-purpose Intel or AMD CPUs. By and large, much of the latency budget is spent in this software, precisely because this is where the complexity lies. Intel and AMD CPUs provide an ever-growing level of performance driven by Moore's law.

Unfortunately, from a performance perspective, this has of late been provided in the form of increasing numbers of cores rather than increasing clock-speeds.

Nevertheless software latency is lower than it has ever been, not only because of hardware improvements but also because of improvements in the software's use of the hardware. Examples include:

- The use of cache-friendly algorithms – variations of traditional designs that may not appear ideal from a pure computational perspective, but achieve lower latency through lower cache-miss rates.

- Lock-free structures that minimize contention between different threads in the multi- threaded designs that are necessary to take advantage of multi-core systems.

Kernel - and stack-bypass

In our discussion of software performance so far, we have focused exclusively on aspects internal to the application of interest. However, any deployed application runs on top of an operating system and cohabits with any number of other applications and processes. The most effective way of ensuring the application performs well is to keep it as isolated as possible from the OS and other processes, just as we want to keep its own internal threads as independent as possible by avoiding locking.

In fact, it is much more difficult to avoid locking in the operating system: lock-free code requires all threads to cooperate and observe protocol, but such cooperation cannot be relied on across different processes. Where it is possible, the OS vendor may simply not have implemented it. We have already seen that the high-performance trading loop doesn't really touch storage, so the only area where it does transit the operating system is in network I/O.

The two principal approaches to lowering the latency of network I/O through the OS both bear the name "bypass": traditionally the

receipt and transmission of application data across the network is done over IP (either using TCP or UDP over IP) in which the OS handles both the lowest level of Ethernet communications with the network interface card (NIC) and also the mapping of the Ethernet frames to the messages being sent and received by the application. A hardware-independent layer of code known as the network stack handles this latter mapping.

- Kernel-bypass moves the network stack out of the kernel and into user-space. The two reasons for doing so are that a specialized stack that is better tuned to modern NICs can be used, and that all locking can be pushed out of the stack and down into the hardware. Examples of this approach include Solarflare's open-source stack called OpenOnload, which integrates into Linux, and Myrinet Express.
- Stack-bypass avoids the IP stack entirely, using a much more direct mechanism to transfer data from host to host. This almost always means some form of Remote Direct Memory Access (RDMA): RDMA (not to be confused with Direct Market Access) is a mechanism that allows devices in a host other than the CPU to read and write memory directly. It was originally developed to allow I/O devices to transfer data to and from memory without requiring intervention from the CPU; with RDMA, it has been extended to allow direct transfers between memory on one system to memory on another. RDMA is a lower latency process than traversing the IP stack twice (once on the sender and again on the receiver) but it does depend on reliable transport across the network – something not provided by ordinary Ethernet.

Kernel-bypass moves the network stack out of the kernel and into user-space

FPGAs (Field Programmable Gate Arrays)
No discussion of high-performance computing would be complete without mentioning FPGAs. These are components that lie somewhere between software and hardware, and one way of Kernel-bypass moves the network stack out of the kernel and into user-space describing what they are is to contrast CPUs and ASICs. General purpose processors that are the CPUs of standard servers and PCs read a stream of instructions from memory, whereas application-specific integrated circuits have their instruction set hard-wired into them when they are manufactured. CPUs provide ultimate flexibility, whereas ASICs sacrifice that flexibility for ultimate performance and low latency.

FPGAs are closer to ASICs in that their instruction sets are implemented in silicon rather than being read on the fly, but are programmable in that they can be rewired on demand. They are not as fast or efficient as ASICs, but their programmability is a huge advantage: the trading algorithms they implement may need to be updated every few months or weeks.

FPGAs can implement some algorithms and tasks with a much lower latency than the corresponding software on a CPU can. They also typically have a very deterministic latency profile: once the algorithm has been compiled to logic, it takes a known number of gates and cycles to complete the given task. That is, the bandwidth and latency of the processor is known and doesn't change under load or other conditions.

High-Performance Networking

The techniques used to reduce network latencies are somewhat different, as the primary causes of network latency are quite different to those of software latency. Those primary causes are:

- Propagation delay, caused by the finite speed of signals in fiber or electrical cables.
- Serialization delay, caused by the fact that large data packets must be written one bit at a time onto the transmission medium.
- Queuing delay, caused by the aggregation that is necessary to make networking physically scalable and economically viable.

The simplest and most direct way of tackling latency in networks has always been to increase bandwidths: this directly reduces serialization delay but, more importantly, it also reduces the likelihood of congestion and therefore queuing delay.

Today, most Ethernet networks in low-latency environments run at 10Gb/s speeds. In some cases, the move from 1Gb/s is driven purely by the desire to reduce serialization delay (from 12µs for an MTU to 1.2µs). However, market data rates are always growing, and today most larger feeds will burst above 1Gb/s at short time-scales and, at some point, 10Gb/s is usually needed to manage congestion on aggregation links.

Once there is 10Gb/s in the core, it makes sense to extend it all the way to the servers, because then you can take full advantage of cut-through switching: with 10Gb/s in and 10Gb/s out, a switch can start transmitting a packet almost as soon as it starts to receive it. The delay through the switch is minimal – just the switching time across its backplane. However, this is not possible if the servers are 1Gb/s attached: the switch must store the packets completely before switching them from 1Gb/s to 10Gb/s.

Interestingly, it is the simplest component of latency that has received the most attention in recent years, namely propagation delay. By taking space in an exchange colo or proximity hosting site, a trading firm can eliminate nearly all of the propagation delay between their algorithms and the matching engines. This provides a simple, clear, and quantifiable improvement in their latency profile, one which many firms clearly feel justifies the high premium exchanges and hosting sites charge for such proximity.

Infiniband

Infiniband is a high-bandwidth, low-latency, host- interconnect technology that is most often used in high-performance computing clusters. It is effectively a LAN technology, used as an alternative to Ethernet, and many of its advanced features have started to be adopted by modern Ethernet variants. Central to IB from a latency perspective is its flow-control: this effectively forces senders and receivers to negotiate bandwidth through the fabric. Once this has been done, any data transfer across the IB components is lossless and has a deterministic latency. Its architecture allows for very low-latency messaging, with host-to-host latencies of about half that achievable even with 10Gb/s Ethernet.

Interestingly, it is the simplest component of latency that has received the most attention in recent years, namely propagation delay

However, it also pushes extra complexity into the end systems, resulting in a network interface that does not map cleanly to the standard socket model.

Latency across the Trading Loop

We have identified the principal components in the high-performance trading loop, and have looked at some of the technologies they use to deliver low latency. It is worth reviewing the latencies that are achievable in each of these technologies and components.

At the software layer, modern CPUs can enable very high levels of performance; the complexity of the functionality implemented in software varies by many orders of magnitude, and so the latency across software components varies correspondingly. However, we can compare the latencies for two tasks on the trading loop of very different complexity:

- Decoding a FAST market data tick: 50ns

Many market data feeds are delivered in a bandwidth-optimized binary encoding known as FAST. Decoding it often requires a modest amount of state to be established, and a non-trivial amount of processing to turn the bytes on the wire into machine representations of the market data. However, the format does allow for some very high-performance decoding strategies; expert implementations can achieve sustained decoding rates of as high as 20 million ticks per second, corresponding to a decoding time of 50ns per tick.

> Over the wide-area, network latencies are dominated by propagation delay

- Matching a marketable equity order: 16µs

ALGO Technologies have showcased their equities-matching engine with measurements made using Corvil; these show that, from network to network, arriving order to departing execution-report, the average latency achieved was as low as 16µs.

On the network I/O front, kernel-bypass can reduce host-to-host latencies to as low as 4µs (OpenOnload on a Solarflare NIC). RDMA can deliver less than half that latency over 10Gb/s Ethernet, while Infiniband can lower that even further to under 1µs.

Within the network itself, latencies can vary enormously depending on topology. Over the wide-area network latencies are dominated by propagation delay; a typical refractive index of just under 1.5, propagation in fibre-optical cables costs about 5ns of latency per metre. Within local-area networks, serialization and switching delays are much more significant than propagation delay. Cut-through Ethernet switches can deliver latencies under 500ns (half a microsecond); however, as we noted above, a change in bandwidth, such as switching from 1Gb/s to 10G/b or vice versa, forces the switch to operate in store-and-forward mode. This will induce an extra latency of at least the serialization time on the lower bandwidth: an IP packet carrying 200 bytes of data will take over 2µs to be serialized onto a 1Gb/s Ethernet link Serialization delay on 10Gb/s is ten times lower, but will still cost over 1.2µs for a (non-jumbo) maximum-size 1500 bytes IP packet.

Latency Distribution

The low levels of latency that can be achieved by the technologies we have discussed are impressive, but there are a number of very important considerations that must be taken into account.

The most fundamental is that any comparison of latencies must define very clearly exactly what latencies are being compared and how they are measured. A very simple example of how confusion can arise is in the discussion of network I/O latencies: how long does it take to send a message across the network? The answer can be either "4µs" or "under 2µs" depending on how it is measured: the latter is the latency to get a message from the sending application out onto the wire, but to get that message into memory for use by the receiving application will take the same length of time again on the receiving host, as well as the network switching time.

The topology of networks nearly always requires significant aggregation

Another aspect of this consideration is that most attention gets paid to minimum or average latencies, whereas it is usually the maximum latency or the high percentiles of the latency distribution that are most important. In the latency measurements that Corvil makes in production systems, there is typically a huge difference between the minimum or average values and the high percentiles. These variations are typically due to queuing in one form or another. For example, the topology of networks nearly always requires significant aggregation; the resulting oversubscription of aggregation links is typically not a problem on average – trading networks are usually operated at low load – but microbursts can drive significant queuing in the buffers that protect the aggregation links against packet loss.

It is not that there are no latency spikes from application to application across IB, they just all happen inside the hosts rather than on the IB network

As we have seen, one of the attractive features of FPGAs in trading systems is the fact that they typically have a deterministic (or close to deterministic) latency profile. Infiniband fabrics also provide deterministic latency guarantees; however it is worth pointing out that they do so by providing bandwidth reservations from host to host across the fabric, and pushing all the queuing back into the senders. It is not that there are no latency spikes from application to application across IB, they just all happen inside the hosts rather than on the IB network.

The Need for Streaming Data Analytics to Optimize the Health and Performance of Trading Systems

In the high-performance trading world, streaming analytics, or real-time insights from network data, reveal that reliability and predictability of execution are often more important determiners of profitability than average latency. Consider for example the question of market data latency: it may be that the average latency across the feed-handler is quite low, but that it is sensitive to the load. High market activity, such as that driven by the announcement of economic indicators, will result in microbursts of market data; these may then drive spikes in the feed-handler latency precisely at the time that it is important to get timely updates of market activity.

Knowing the average latency across the feed- handler does not help in engineering the system for reliably low latency. What is required is the ability to timestamp all the events at the appropriate resolution, measure the latency of every hop, and correlate the latency across the full technology stack with the microbursts that drive the latency. This is known as Streaming Analytics, and Corvil is the only vendor that provides solutions for managing electronic trading system health and performance across the network and application in this unified manner.

Recap and Summary: Deploy Streaming Analytics Platform to Run Your Business in the NOW.

High-performance trading systems today have driven the adoption and development of a set of technologies that enable trade-execution and market data delivery and handling at the timescale of microseconds. Many of the key processes can be implemented within a handful of microseconds, which means that precision latency management must be capable of delivering an accuracy of hundreds or tens of nanoseconds.

At the same time, total system latencies can vary by orders of magnitude because of the effects of dynamic congestion. Effective latency management requires not just a capture of the complete distribution of latencies, but also an analysis of the causes of latency. That is, it is not sufficient to just measure the spikes in trading system latency, but we must also capture the microbursts and other infrastructure behavior that drives the latency spikes.

Corvil's Streaming Analytics platform is the only solution to provide nanosecond latency measurements and unified latency management across the whole application and network infrastructure.

For more information or to contact us, visit **www.corvil.com**

CWP-080814