EVERSPIN® TECHNOLOGIES

App Note

Accelerating Fintech Applications with Lossless and Ultra-Low Latency Synchronous Logging using nvNITRO

Introduction

The financial industry is continually reinventing itself as each new generation of technology becomes available, making trading systems faster and more efficient. This ongoing cycle of improvement has even led to a new term for an emerging financial services sector revolution called "FinTech."

But as more financial systems try to increase their transaction throughput, this trend has also led to uncontrolled market scenarios, such as the "Flash Crash" in 2010. This crash resulted in further tightening of regulations and control measures which require the participating clients to log details of transaction orders and any transaction status changes. Additionally, traders are also required to maintain these records for at least 5 years. The created transaction logs are critical for ensuring investor transparency and protection from any foul play. In the case of compliance audits or any question regarding a transaction, these systems enable transactions to be inspected and replayed back.

This application note explores how Everspin nvNITRO™ technology can improve FinTech performance without creating additional risks from compliance.

The Challenge of Regulations on Trading Performance

FINRA (Financial Industry Regulatory Authority), MiFID II (Markets in Financial Instruments Directive) and many other financial regulatory bodies or requirements are now demanding that transacting clients log & timestamp (global) order details, transaction communications, as well as log all order status changes such as:

Accepted for bidding	Calculated	Canceled	Done for day
Expired	Filled	New	Partially filled
Pending cancel	Pending new	Pending replace	Rejected
Suspended	Stopped		

Question: What happens if order execution information and transaction data (logs) are lost?

This is a very real situation that could happen, but it all depends on the architectural decisions that are made with regards to logging and storing order information.

If the company gets audited for the transactions and the auditor finds the company has incomplete or missing transaction log data due to a failure to implement data protection "by design and by default," it may result in:

- Monetary fines up to or in excess of millions of dollars
- Suspension from trading in the market for some days or months, which itself could result in the loss of millions to hundreds of millions of dollars depending on the size of the company
- Other types of trade or monetary sanctions

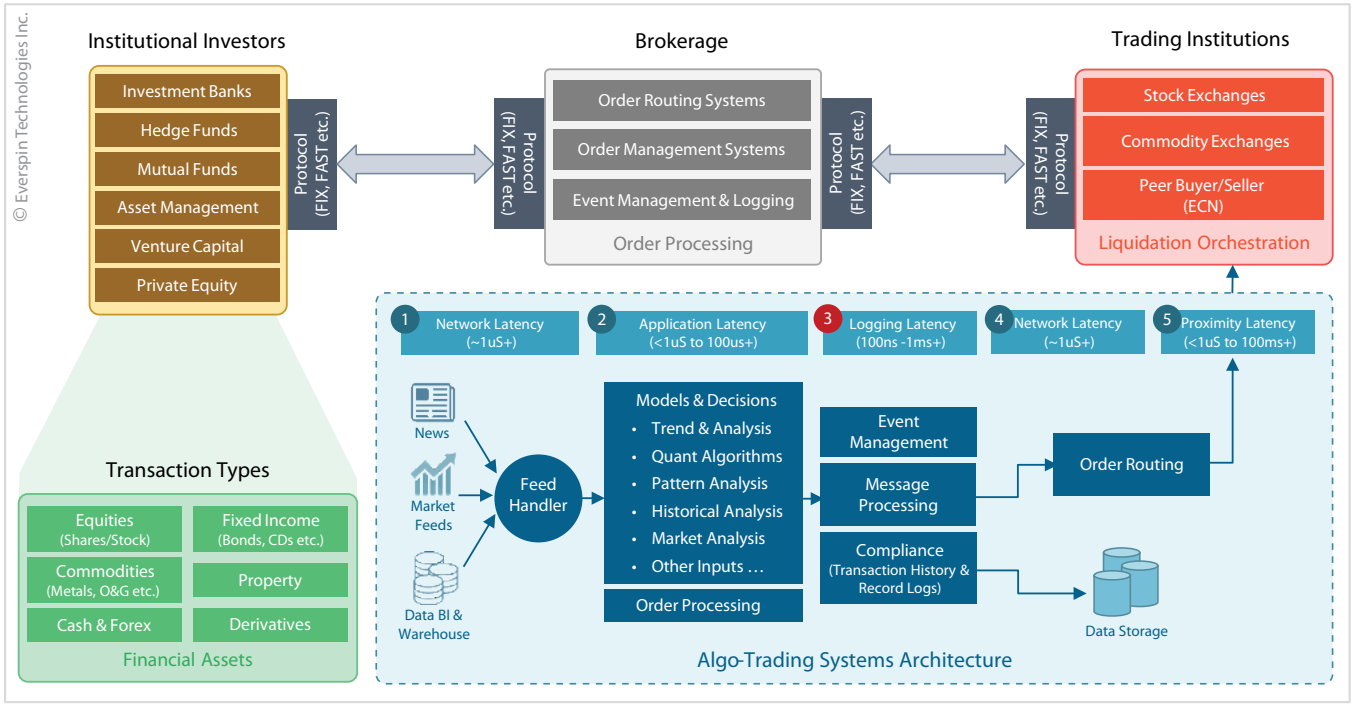

Figure 1: End-to-end overview of typical financial trading system

Transaction Logging Architecture

Trading systems use existing or custom logging service platforms such as Log4J, Log4J2, Logback, or Chronicle. For a moment let's focus on the section 3 of the entire trading system from Figure 1. The two most common ways to implement order detail and transaction event logging are synchronous logging and asynchronous logging.

Synchronous Logging: In this case the system waits until the logging data is committed to persistent storage media such as disk drives, SSDs or other non-volatile devices before beginning the next transaction. For synchronous logging the biggest advantage is that even in the case of a power failure or any other adverse situation, the log data is lossless; it is guaranteed to be retained after power is restored. In the case of the transaction not completing or application state needing to be reviewed, the transaction can be played back. But a big disadvantage of synchronous logging is that the latency of writing to a storage I/O device limits the performance of the entire transaction. The fastest storage media devices available are flash-based SSDs which have average 100% random write latencies of 30µs or more but typically exhibit latency in the 100µs range for 4KB block sizes and queue depth (QD) of 1. For high frequency trading (HFT) or many algorithmic (algo) trading scenarios (especially where trading institutions are located in close proximity), 30µs of latency will dominate the entire transaction, ultimately impacting overall trading performance. Because of the performance advantages of lower latency, many HFT-type systems are physically located next to trading exchanges in co-location centers, connected with the fastest network links in order to keep latency absolutely minimal.

Asynchronous Logging: In this case, the system does not have to wait until the logging data is committed to a persistent storage media device in order to start the next transaction. Trades can be performed asynchronously (in parallel) with data logging work. Typically, there is a ring buffer or memory mapped file (mmap) that is stored

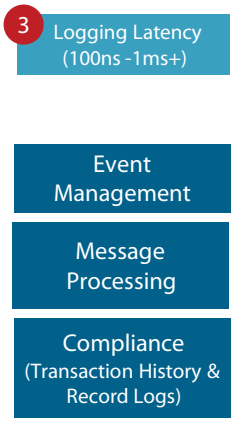

Figure 2: Logging transaction details

in system memory. A separate thread writes the logging data as fast as possible to the memory and another thread pulls data out from shared memory and writes to persistent media. The sequential ring buffer data or mmap file is pulled out in larger chunks and flushed to filesystem or persistent media using techniques like fsync, msync or o_sync when opening the file.

This method has a large advantage in latency as writing log data to memory is in the 100ns range and the system can run as fast as possible, at the speed of the system memory. But there is a large drawback to an asynchronous logging implementation: power failures can result in all transactions and order data in the memory being lost. Since the mmap'ed file or ring buffer could contain 1000's of records, a company could lose data for many transactions due to an unexpected failure. Interestingly this data is the most critical information that is needed to reconstruct the transactions after failure when the system is restored, but in an asynchronous logging scenario where the data was never written to a persistent location in time, that information is lost forever.

Architectural Decisions

Many companies who are executing financial trades close to speed of light, e.g. HFT type applications, may opt to choose "performance over persistence." The architectural choices that these companies make are designed to provide persistence for memory through sub-optimal means, utilizing strategies like:

- Uninterruptible power supplies (UPS) backed server compute racks
- Capacitor or battery backed NVRAM cards
- Battery backed NVDIMM's or NVDIMM-N

In these architectural cases, when a power failure happens, the system will keep backup power for the memory subsystem available until all memory contents have been transferred to a non-volatile device such as NAND flash. This process may last for only a few seconds. When system power is restored, the data will be transferred to a more permanent storage media. In the case of a catastrophic failure of a datacenter, UPS, batteries or super-capacitors, the transaction information that was still in memory and not committed to storage media will be lost. In such cases, the company may fail a regulatory audit, facing penalties, and possibly may have to rebuild transaction history from exchange logs. But customer orders that were in memory at the time of the failure that had not been received by the exchanges may possibly never be recovered.

It is a balancing act by designers and engineers to gauge risk versus reward. In the case of synchronous logging, today's fastest NAND-based SSDs can deliver low 20-60µs average latencies. Even though, for many algo trading scenarios, sub microsecond latencies may not be needed, the designers may still opt for asynchronous logging because of the latency distribution curves of flash-based SSDs (which have a long latency tail). For example, 99.99% latency for SSD media could be up to 100x slower and too far out, sometimes extending into the millisecond (ms) range. This is not good for many trading scenarios under high volume load or other cases where repeatable and deterministic execution performance is critical.

STT-MRAM Technology Changes Status Quo

Until now, the memory technology available to designers has been volatile, meaning after power is removed, the data contents in the memory are lost. But with the introduction of 256Mb STT-MRAM by Everspin Technologies, systems can now have memory which has high performance like DRAM but provides a persistent data storage that is non-volatile.

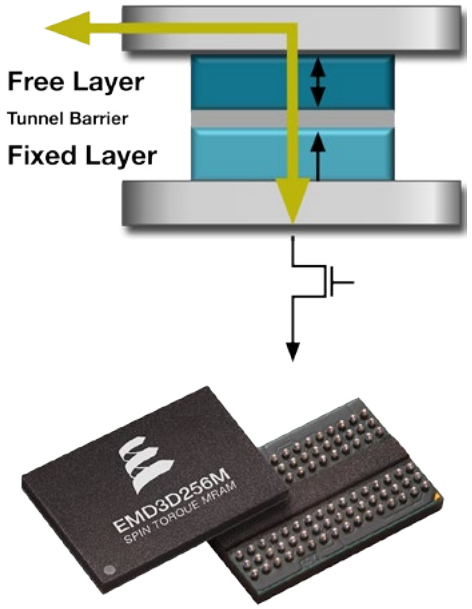

Figure 3: STT-MRAM

STT-MRAM stands for spin-transfer torque magnetoresistive random access memory. Any data written to STT-MRAM devices is natively persistent and does not require any batteries or supercapacitors. Data is captured by writing to a memory array that manipulates electron spin with a polarizing current. STT-MRAM performs like DRAM but requires no refresh. The current interface available is ST-DDR3 which is very similar to standard JEDEC DDR3. ST-DDR4 interfaces will be also available in the future, bringing higher speed and higher capacity.

Some of the benefits of STT-MRAM technology are:

- Non-volatile data
- Performance characteristics similar to DRAM memory
- Billion+ cycles of data endurance
- DDR3 compatible footprints and future DDR4 compatible footprints
- ST-DDR3 interface and future ST-DDR4 interface
- Byte writes and reads (no blocks)

nvNITRO Storage Accelerator (NVMe)

To facilitate system designers and engineers who want to use this revolutionary technology in their applications right away, Everspin has developed nvNITRO technology which enables a persistent memory storage accelerator card to be built with STT-MRAM. Everspin is the creator of the technology, but the accelerator cards themselves will be marketed and sold by SMART Modular Technologies under the name MRAM NVM Express Card. nvNITRO is a great solution for acceleration of lossless and ultra-low latency (ULL) synchronous logging. These products can span multiple industries where transaction processing and recording are critical.

Figure 4: nvNITRO storage accelerators

Highlights of nvNITRO storage accelerators include:

- Interfaces & Form factor
  - PCIe x8 Gen3 Half Height LP
  - U.2 x4 Gen3
- Capacity: 1GB and 2GB STT-MRAM
- Latency: Ultra-low < 7.2µs*
- Deterministic latency distribution with virtually no tail
- Performance: Up to 1,500,000 IOPS
- Durable, persistent memory
- Two access modes:
  - Block based (NVMe)
  - Memory mapped I/O
- Endurance: 1,000,000,000+ cycles
- Peer-To-Peer
  - PeerDirect (OFED compatible)
  - Fully offloaded NVMe over fabric
- Standard Windows/Linux NVMe drivers
- No need for batteries or supercaps

* 4KB block, 100% random writes, completion latency, Queue Depth (QD) = 1

nvNITRO - Low Latency and Lossless Circular Write Buffer

nvNITRO enables a system application to write or log large amounts of data at the full performance of incoming data while maintaining deterministic and extremely low latencies under 10µs. STT-MRAM's high performance, persistent memory enables a customer to utilize a single nvNITRO card as a front end connected to less expensive back end storage devices, which ultimately lowers the overall solution cost.

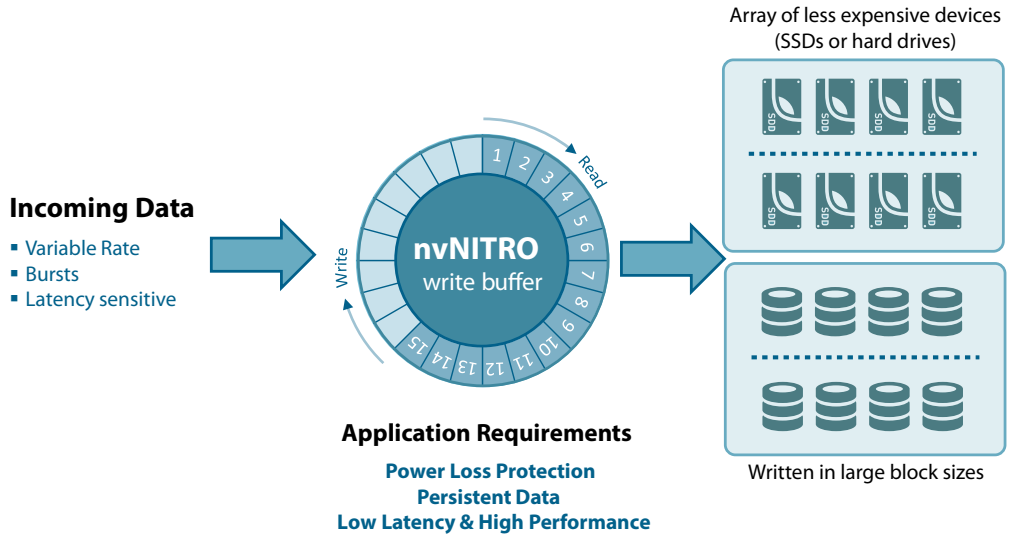

Figure 5: Implementing a lossless and low latency circular buffer

Refer to Figure 5 which shows an implementation of a 1GB nvNITRO accelerator as a very high performance, low latency, and lossless write buffer in front of an array of economically priced SSDs or hard disk drives. Highly random bursts of incoming high-speed data are fed to a circular (ring) buffer implemented in the nvNITRO NVMe storage device. No special driver is needed to utilize nvNITRO capabilities. By default, the nvNITRO presents itself as NVMe device and uses industry-standard Linux or Windows NVMe drivers.

The application software uses two threads with one thread writing the data to the buffer at full speed and the other thread emptying contents of the circular buffer in large batches of few kilobytes (KB) or more. Since nvNITRO uses STT-MRAM, data written to it is safe as soon as it written, enabling the writer thread to instantly post back a completion and begin processing the next work item. The entire system does not have to wait until the data is written to the disk drives or SSDs. The reader (drain) thread is very efficient because it can take larger chunks of data in the buffer and stream them to storage devices such as disk drives or SSDs. The storage devices, including rotating disks, are very efficient in handling sequential and large streams of data so customer applications using nvNITRO can use more cost-effective storage media devices.

This implementation allows for writing terabytes (TB) to petabytes (PB) of data and logging information with a very economical solution while maintaining lowest latency, high performance and power loss protection for data. No UPS, batteries, NVDIMMs, or supercapacitors are required because nvNITRO is inherently power fail safe through its use of STT-MRAM persistent memory technology. If architected correctly one may not even need expensive enterprise class SSDs because power loss protection (PLP) is already provided by the nvNITRO, but a minimum of FTL protection will still be required for those SSDs (which is present in virtually all enterprise SSDs in the market today).

Real-world Synchronous Logging Applications – Log4j, Java or C++

As we discussed early in this application note, most trading systems are required to log transaction details and their changing status. For this discussion, we will leave aside HFT for a moment which uses asynchronous logging to keep up with nanosecond responses. Most algo trading and other types of trading systems can benefit from the simplicity and robustness of a high performance, lossless and low latency logging design that can be implemented with a circular buffer using nvNITRO as detailed in the previous section.

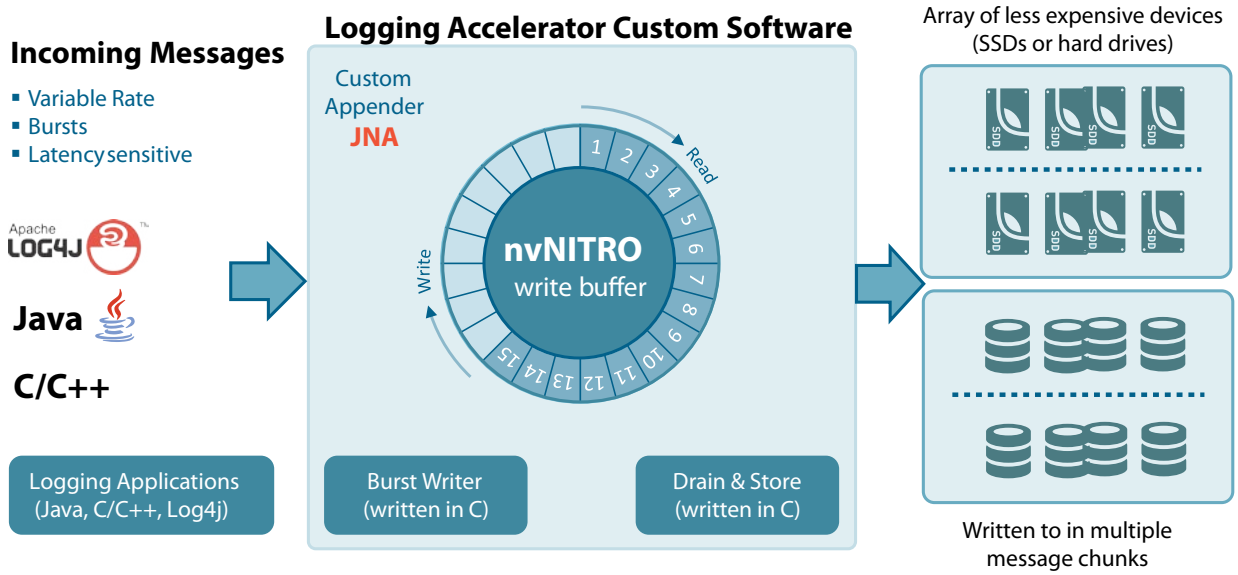

Figure 6: Lossless, low-latency and high performance Log4j acceleration

A very popular and extensive logging framework used by many companies is Log4j or Log4j2 in Java. Refer to Figure 6, which shows an application that was developed to accelerate Log4j synchronous logging using nvNITRO as a circular buffer.**Key solution goals** for this lossless and low latency logging software were as follows:

- Support any Java or C/C++ application with native APIs
- Provide custom appender plugin for plug-n-play working with Log4j

Design considerations are as follows:

- nvNITRO will be used as a raw NVMe device, i.e. no file system will be mounted on the nvNITRO
- To prevent data loss in between stages, any writes to nvNITRO or local storage media will be passed through without any caching
- Use of an independent writer thread which writes into the circular buffer and reader thread (draining) from the circular buffer
- Use of shared memory for buffer offset coordination to avoid race conditions between the writer and drain threads which effectively throttles both sides using busy waits
- The message records written to the storage media by the drain thread can be written in raw format or standard file system
- Message record structure is designed for optimal performance by avoiding multiple accesses of data in the buffer
- The message records in the buffer are variable length strings with a minimum size equal to the block size of the device
- Basic metadata information will be included in the head of the message record – read/write flag, message length, message sequence number, etc.

Key components to this architecture shown in Figure 6 and 7 are as follows:

Application: Any customer application written in Java, C/C++ or using Log4j framework that requires a lossless, low latency and high performance synchronous data logging capability.

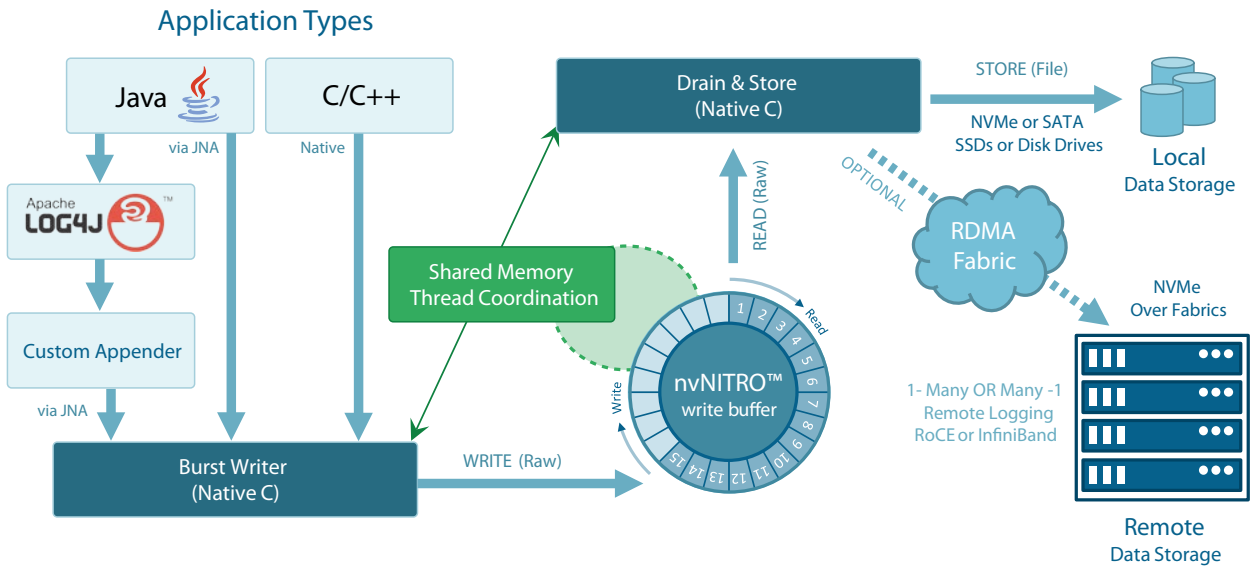

Figure 7: Lossless, low latency and high performance synchronous logging architecture using nvNITRO

Burst Writer (Raw): This process is written in C and is responsible for handling and writing incoming variable length log messages to the nvNITRO circular buffer. The log data is written in raw format to the circular buffer. In rare cases, the application may also experience backpressure when the reader thread has not drained enough of the messages to make room for writer thread to write into the circular buffer. With a properly designed system that maximizes incoming message rates, one should not run into this situation.

Drain & Store (Raw/File): This process is also written in C and is responsible for handling reading from the nvNITRO circular buffer in fixed batch sizes. The process then stores that data in a slower array of persistent storage media such as SSDs or disk drives as a large streamed data. The log data is read in raw format from the circular buffer but can be written to the storage array or devices with filesystem or in raw format. In case there are no new write log messages in the circular buffer, the reader thread just waits.

Shared Memory (Thread Co-ordination): This is a very small shared memory section in the system RAM to manage writer and reader thread offsets to handle race conditions and busy waits. Power failure does not have an impact even if the shared memory contents are lost as the message data is already present in the circular buffer inside the nvNITRO and each message metadata already has details about which data that has already been read or not. In case the application wants to go further, the application developer could possibly make use of MMIO region in nvNITRO.

Persistent Memory Mapped I/O Regions

In addition to NVMe block mode access to the device, nvNITRO also supports a MMIO region which is mapped as a BAR (Base Address Register) in the PCIe address space. nvNITRO can be partitioned in such a way that both regions are available to the user: block and memory mapped region. One caveat to using the MMIO region is that users must develop their own driver to access this region. Since any data written to the device is persistent, the memory mapped data is also persistent. Hence users could potentially implement the shared memory in the MMIO region. The proof of concept system that was developed and showcased in this application note used system memory for shared memory. The MMIO region and NVMe block region can be set in such a way that any percentage of total available space can be partitioned for one or the other.

Results

The tests were run by simulating high-speed incoming messages for different application scenarios where nvNITRO was used as a circular burst buffer before the data was stored in the larger storage media such as SSDs and disk drives:

1. Java application using JNA to interface with "Burst Writer."
2. Java application using Log4J. In this case a custom Java appender was used with JNA to interface with "Burst Writer," but no change in customer application code is required.
3. Finally, a high-performance Enterprise NVMe SSD was used to compare logging performance when "Burst Writer" does not use nvNITRO and data is written directly to the enterprise NVMe SSD.

Considerations

Note that the latency results will be dictated by completion latencies of the nvNITRO burst buffer device along with the code path latency and submission latency. In the case of Java applications, the latency will be higher than the C/C++ applications because of additional JNA code. In the case of Java applications that uses Log4J framework, even higher latency will be present due to the Log4J code path as well as the custom Java appender layer.

No extra efforts were made to further tune the operating system (OS) and BIOS for latency improvements. Hyper-threading was turned ON, but for relative measurements, this was not considered an issue.

In the case of Java applications, garbage collection (GC) will also impact latency jitter. During JVM warmup time, higher latencies will be observed. Hence, the results shown here will consider data points after the JVM is warmed up and is in a stable state.

Centos 7.x with kernel version 3.10 was used as the operating system and the drain thread copied the data from the nvNITRO burst buffer to the Enterprise class NVMe SSD.

Case 1: Java application using JNA to interface with "Burst Writer"

Figure 8 shows a per message latency chart of all messages, but the metrics summary shown below for latency distribution only uses the last 1 million messages in order to allow for JVM warmup.

As can be seen from the results, the average latency of the logging solution through the entire stack including the Java layer, the application layer, the JNA, the NVMe driver and the nvNITRO accelerator was only 12µS. However, what is really outstanding is that through the entire Java application stack, the 99.99% latency was only 26µS. The latency distribution of this application case is excellent.

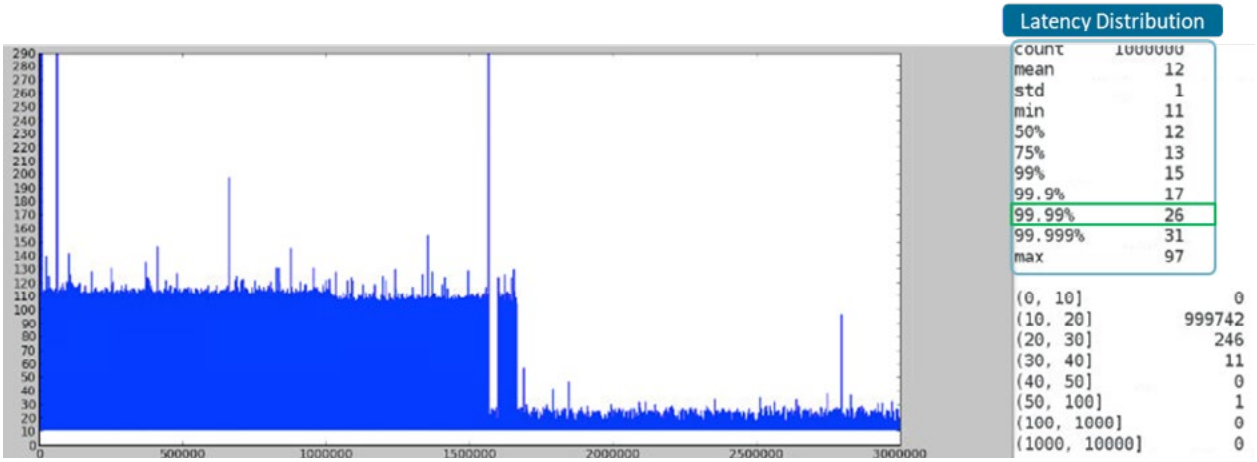

Figure 8: Java Application: Results of per message latency for all messages

Case 2: Java application using Log4J and Java appender w/ JNA to interface with "Burst Writer"

Figure 9 shows per message latency chart of all messages, but the metrics summary shown below for the latency distribution only uses the last 1 million messages in order to allow for JVM warmup.

As can be seen from the results, the average latency of the logging solution through the entire stack including the Java layer, the Log4J framework, the application layer, the custom Java appender, the JNA, the NVMe driver and the nvNITRO accelerator was only 13µS. Similar to the first case, what is truly amazing is that through the entire Java application stack and Log4J framework, the 99.99% latency was only 28µS. The latency distribution of this application case for synchronous logging is unmatched in the industry. Also note that the customer does not have to modify any part of their code; it is all plug and play with Log4J. The custom appender with the JNA interface to the "Burst Writer" library does all of the work under the hood to make this happen and uses the nvNITRO storage accelerator as a burst buffer. Compared to the Java application case where the user has to interface to JNA for the "Burst Writer" library, only 2µS of additional latency were added for the average and 99.99% distribution deviations.

Case 3: Compare With & Without nvNITRO "Circular Buffer" in a Log4J-based application

Figure 10 shows a comparison of synchronous logging latency of an application using Log4J framework:

With an nvNITRO circular burst buffer application (in green)

Without an nvNITRO circular burst buffer and instead writing directly to a NVMe enterprise SSD (in red)

For a moment, if we ignore the green latency results and focus on the result shown in red on the chart, we can see that for a solution using a very fast NVMe SSDs without an STT-MRAM based burst buffer, latency will be up to 9X higher. It is also very evident from just looking at the distribution of the red chart that it has a large spread and the latency tail actually keeps going to the right.

Note that latency distribution curve is very tight and the spikes for the "With nvNITRO" case were so high and narrow that the chart was cropped on the top. In the "Without nvNITRO" case, the latency tail was so long to the right that the chart was cropped on the right. From the data, it showed that 99.9% latency of the enterprise NVMe SSD was up to 80X worse than the case where nvNITRO was used for a circular burst buffer.

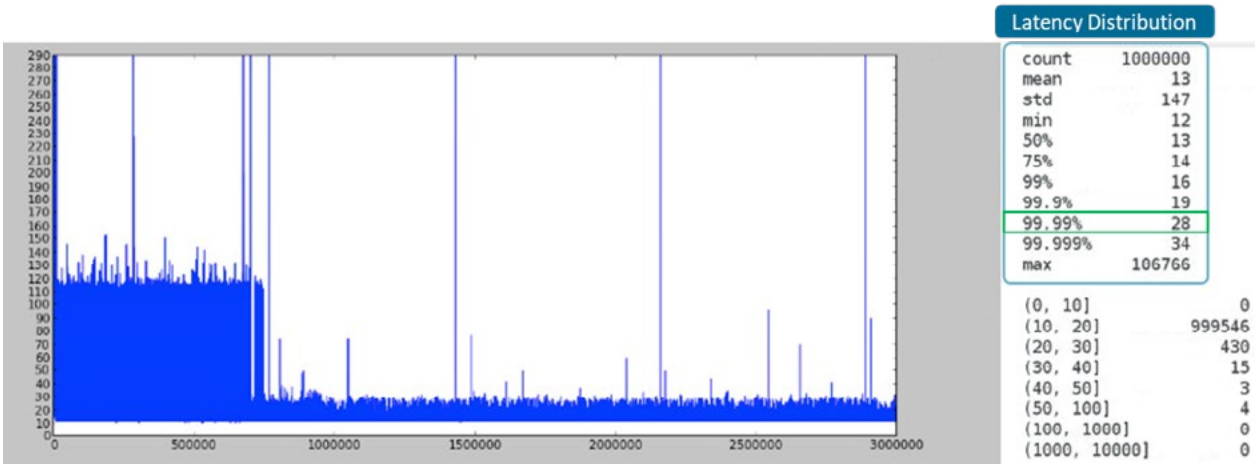

Figure 9: Log4J framework: Results of per message latency for all messages

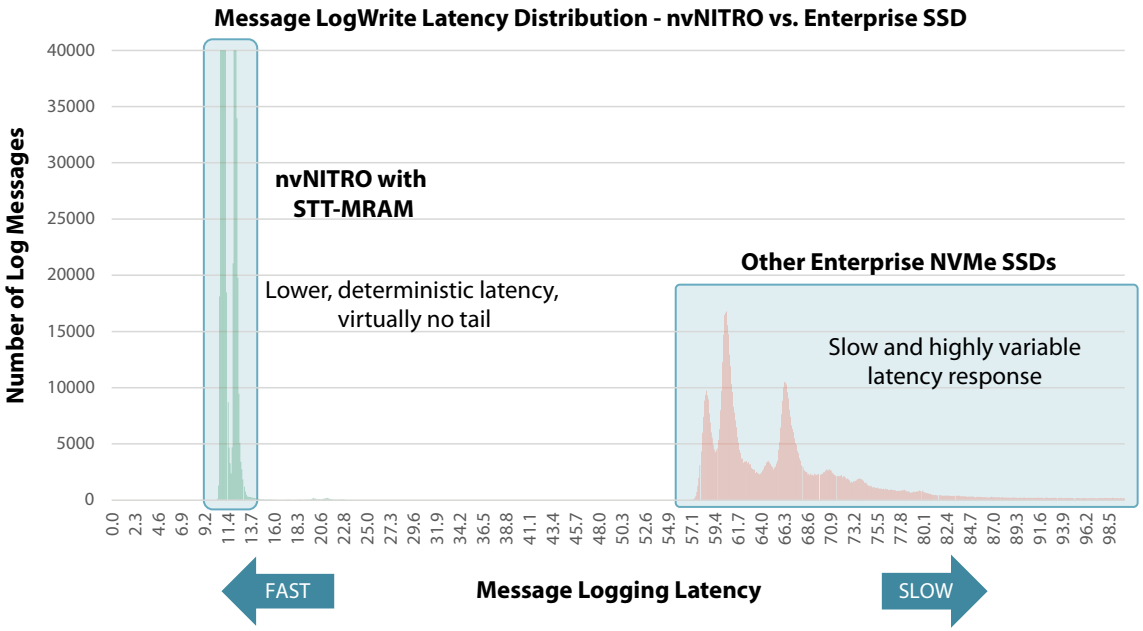

Figure 10: Compare Log4J Performance: With nvNITRO circular burst buffer vs. direct logging to an enterprise SSD without nvNITRO burst buffer

What does this all imply? By simply using an nvNITRO storage accelerator in front of the same high performance SSD or array of such SSDs, customers can completely change the dynamics of applications, performing synchronous logging nine times faster. Imagine the benefits to a financial application such as algo-trading with a 90% reduction in latency for synchronous logging.

Benefits

Up to 9x Performance – Improvement of logging latency over other fast enterprise NVMe SSDs.

Lossless – Fully persistent and synchronous logs.

Deterministic Performance – Very tight latency distribution curve with virtually no tail enables trading systems to perform consistently well, even under heavy load with an up to 80x latency improvement at 99.9%.

Failsafe protection of data – All data written into the nvNITRO is immediately persistent. No lag between posted writes and a commit.

Overall Economical Solution – By using nvNITRO as a burst circular buffer, customers do not have to purchase expensive SSDs or disk arrays for their backend log storage.

Embedded STT-MRAM burst buffers – It is possible to get benefits similar to nvNITRO but by using STT-MRAM directly in FPGA-based trading systems and running burst buffers right on the board, next to the FPGA at the speed of memory, while maintaining persistence of logs.

Future - Remote Logging over RDMA Fabrics

In addition to storing the log data locally, many companies want to consolidate the log data from multiple trading systems into a centralized storage system. There could be situations where the same trading system needs to log data to multiple different storage systems depending on the log data type and its criticality. Either of these cases, (1-to-many OR many-to-1) can be achieved in a variety of ways. nvNITRO has been verified to work successfully in an RDMA (remote direct memory access) fabric and in networks such as InfiniBand or RoCE (RDMA over converged Ethernet) as a block or memory device.

NVMe over Fabrics (NVMeOF) - By using a Mellanox 100Gb ConnectX-5 based InfiniBand fabric, which provides an RDMA connection between systems (or possibly any other compliant RoCE based network devices), nvNITRO was successfully used as an NVMe target device in a remote system. Even though this application note only details a local logging system, it is very possible to connect to another nvNITRO in remote machine as an NVMeOF target and map it to the local system, draining the log data remotely. There is potentially another method where the drain and source are also handled in the remote device. By deploying nvNITRO in the trading systems locally as well as in remote targets, very flexible and high performing systems can be achieved.

nvNITRO was able to achieve a full 1.5Million IOPs, 6GB/s BW over a 100Gb IB link for 4KB block sizes in a 100% random write traffic pattern as a NVMeOF target device. The end-to-end latency for QD1 was less than 20µS across that RDMA fabric.

Remote Persistent Memory - By using Mellanox 40Gb ConnectX-3 based InfiniBand fabric and OFED drivers, nvNITRO was successfully used as a remote persistent memory. Standard OFED drivers were used to map remote memory regions in an RDMA network. Standard OFED tests for bandwidth and latency were used. Going one step further, nvNITRO was used as a Peer-to-Peer device with a Mellanox IB device using a custom-compiled PeerDirect driver to enable full bypass of CPU and host memory in the target system. Data could be written and read from the remote persistent memory region in nvNITRO over the 40Gb RDMA fabric with end-to-end latencies lower than 3µs.

Conclusion

This application note outlines how nvNITRO, built with STT-MRAM technology, can enable a lossless, low latency, deterministic, and high performance synchronous logging solution for the financial industry - all at a lower overall solution cost.

The results have shown that the nvNITRO storage accelerator can supercharge FinTech applications by removing bottlenecks from applications when used as a circular burst buffer. Most importantly, the persistence of STT-MRAM means that log data is protected, reducing the risk of penalties for lack of compliance, which can have a significant impact on the overall profitability of a trading company.

Contact Information:

Author:
Pankaj Bishnoi
Director of System Applications
WW Sales Group

How to Reach Us:
www.everspin.com

E-Mail:
support@everspin.com
orders@everspin.com
sales@everspin.com

USA/Canada/South and Central America
Everspin Technologies
5670 W. Chandler Road, Suite 100
Chandler, Arizona 85226
+1-877-347-MRAM (6726)
+1-480-347-1111

Europe, Middle East and Africa
support.europe@everspin.com

Japan
support.japan@everspin.com

Asia Pacific
support.asia@everspin.com

Everspin Technologies, Inc.