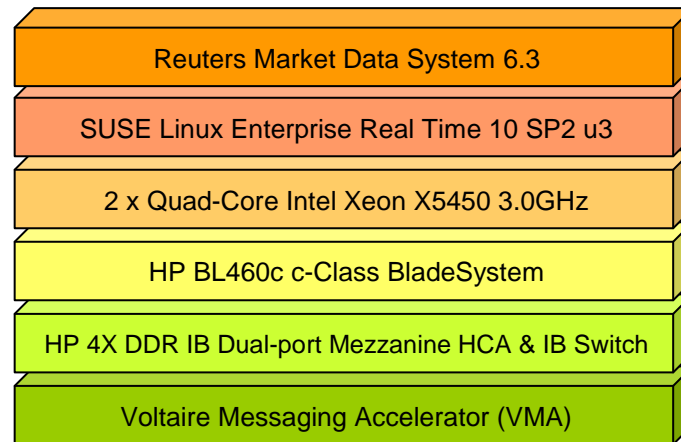


RMDS 6.3 with Novell SUSE Linux Enterprise Real-Time on HP c-Class Blades, Intel Xeon, and Voltaire InfiniBand

Issue 1.0, 11 Nov 2008

Technology Stack Under Test



Key Results

- ➔ Lowest mean latency reported to date with RMDS:
 - Less than 0.67 milliseconds at up to 750,000 updates per second with SLERT/InfiniBand
 - SLERT/InfiniBand yielded a 61% reduction on average compared to SLES/GigE
- ➔ SLERT reduced maximum latencies at higher update rates:
 - 3.01 milliseconds max latency at 750,000 updates per second observed with SLERT/InfiniBand
 - SLERT/InfiniBand reduced maximum latency values at rates over 200kups by 35% compared with SLES/InfiniBand and by 49% on average compared with SLES/GigE
- ➔ Lowest standard deviation of latency reported to date in latency-optimized RMDS configuration:
 - Less than 0.5 milliseconds through 750,000 updates per second with SLERT/InfiniBand
 - SLERT/InfiniBand reduced standard deviation of latency by an average of 33% compared to SLES/GigE and 47% compared to SLES/InfiniBand
- ➔ Highest throughput reported to date in the “Producer 50/50” fanout test for a two-socket server:
 - 10.1 million updates per seconds achieved with SLERT/InfiniBand
 - SLERT/InfiniBand increased throughput by over 670% compared to SLES/GigE

NOTE: The tests in this STAC Report are not based on STAC Benchmark specifications. STAC Benchmark specifications are currently under development by the STAC Benchmark Council. For more information, see www.STACresearch.com/council.

This document was produced by the Securities Technology Analysis Center, LLC (STAC®), an independent provider of performance measurement services, tools, and research to the securities industry. To be notified of future reports, or for more information, please visit www.STACresearch.com. Copyright © 2008 Securities Technology Analysis Center, LLC. “STAC” and all STAC names are trademarks or registered trademarks of the Securities Technology Analysis Center, LLC.

Disclaimer

The Securities Technology Analysis Center, LLC (STAC[®]) prepared this report at the request of Novell, Hewlett-Packard, Intel, and Voltaire, Inc. It is provided for your internal use only and may not be redistributed, retransmitted, or published in any form without the prior written consent of STAC. All trademarks in this document belong to their respective owners.

The test results contained in this report are made available for informational purposes only. STAC does not guarantee similar performance results. All information contained herein is provided on an "AS-IS" BASIS WITHOUT WARRANTY OF ANY KIND. STAC has made commercially reasonable efforts to adhere to Reuters' published test procedures and otherwise ensure the accuracy of the contents of this document, but the document may contain errors. STAC explicitly disclaims any liability whatsoever for any errors or otherwise.

The evaluations described in this document were conducted under controlled laboratory conditions. Obtaining repeatable, measurable performance results requires a controlled environment with specific hardware, software, network, and configuration in an isolated system. Adjusting any single element may yield different results. Additionally, test results at the component level may not be indicative of system level performance, or vice versa. Each organization has unique requirements and therefore may find this information insufficient for its needs.

Customers interested in a custom analysis for their environment are encouraged to contact STAC.

Contents

- 1. Background 6
- 2. Description of Tests 7
 - 2.1 Methodology 7
 - 2.1.1 Throughput testing 7
 - 2.1.2 Latency Testing 9
 - 2.2 System Specifications 11
 - 2.2.1 Servers 11
 - 2.2.2 Networking 11
 - 2.2.3 Network Interface Configurations 12
 - 2.2.4 Operating System 12
 - 2.2.5 TCP and UDP Buffer – key parameters 12
 - 2.2.6 RMDS Software 13
 - 2.2.7 RMDS Configuration 13
 - 2.2.8 Priorities and other settings 14
- 3. Results 18
 - 3.1 Throughput 18
 - 3.2 Mean and Max Latency 18
 - 3.2 Standard Deviation 24
- About STAC 28

Summary

The rapid growth of data traffic in the capital markets continues to be a major concern for industry technologists. As markets become more volatile, huge volumes of traffic can overwhelm systems, increase latency unpredictably, and throw off application algorithms. In fact, some algorithmic trading applications are more sensitive to the predictability of latency than they are to the average latency (within limits).

Novell believes that its real-time operating system, SUSE Linux Enterprise Real Time (SLERT), improves latency while maintaining high throughput. Novell and its partners asked STAC to measure the performance of a market data stack based on the Reuters Market Data System (RMDS) when running on SLERT and compare that to the same stack using Novell's general purpose operating system product, Novell SUSE Linux Enterprise Server (SLES). The platform used the latest Intel processors and HP blade servers. The networks used were Voltaire InfiniBand in some cases and gigabit Ethernet in others. Using this solution stack, the goals of the project were to:

- Measure RMDS latency in the latency-optimized RMDS configuration
- Compare the standard deviation of latency (jitter) between SLERT and SLES
- Find the maximum throughput of the P2PS in a multiplex configuration of the Producer 50/50 test, using the throughput-optimized RMDS configuration
- Compare the SLERT with InfiniBand environment to a typical environment in use today, SLES with 1GigE

To summarize, we found:

- SLERT with InfiniBand increased throughput by 8% compared to SLES with InfiniBand.
- SLERT with InfiniBand reduced latency on average almost 25% when compared to SLES with InfiniBand at the same data rates and reduced latency 47% when compared to SLERT with GigE at the same data rates.
- SLERT with InfiniBand yielded a 61% reduction on average in mean latency compared to a typical environment deployed today, SLES with GigE
- SLERT with InfiniBand reduced maximum latency values at rates over 200kups by 35% on average compared with SLES with InfiniBand and by 49% on average compared with SLES with GigE.
- SLERT reduced the standard deviation of latency by an average of 33% compared to SLES with 1GigE at the same data rates and by an average of 47% compared to SLES with

InfiniBand.

- SLERT with InfiniBand yielded a 38% reduction on average in the standard deviation of latency compared to today's typical environment, SLES with GigE

The following table summarizes the specific highlights in each test environment.

	Voltaire InfiniBand		GigE	
	SLERT 10 SP2u3	SLES 10 SP2	SLERT 10 SP2u3	SLES 10 SP2
Mean latency in latency-optimized RMDS configuration	Less than 0.67 millisecond at up to 750,000 updates per second	Less than 1 millisecond at up to 600,000 updates per second	Less than 1 millisecond at up to 500,000 updates per second	Less than 1 millisecond at up to 200,000 updates per second
Standard deviation of latency in latency-optimized RMDS configuration	Less than 0.5 milliseconds through 750,000 updates per second	Less than 0.5 milliseconds through 600,000 updates per second	Less than 0.5 milliseconds through 500,000 updates per second	Less than 0.5 milliseconds through 300,000 updates per second
Output in the "Producer 50/50" fanout test of a multiplexed P2PS	10.1 million updates per seconds	9.34 million updates per second	1.30 million updates per second	1.31 million updates per second

1. Background

Market data latency has a huge impact on the overall speed with which a trading firm can execute a transaction in response to new information. In some markets, firms can profit from as little as one millisecond of advantage over competitors, which drives them to find sub-millisecond optimizations of the systems fueling their trades. The latency obsession has resulted from the spread of automated trading to nearly every geography and asset class, and the resulting imperative to exploit—or defend against—new latency arbitrage opportunities.

Another consequence of automated trading is a ballooning of market data traffic volumes, which complicates the latency race, thanks to a well-established tradeoff between throughput and latency. Update-rate increases of 2 to 6 times in a single year are not uncommon for today's exchanges. Automated trading drives this traffic by both increasing transaction volumes and increasing the ratio of quotes and cancellations to actual trades. On top of this, large sell-side institutions often generate enormous amounts of real-time data internally, which they pump onto their internal market data system. The traffic from internal content sometimes exceeds that of information coming in from external sources.

This combination of forces keeps market data technologists on the lookout for new technologies that can shift the performance tradeoffs in the right direction. One layer of the technology stack that receives ongoing scrutiny is the operating system. This plays an important part in the performance equation, as it controls the interaction of the application with physical resources such as CPU, memory, disk, and network. Firms are particularly interested in the extent to which an operating system can reduce latency or improve throughput by improving resource scheduling.

Novell has recently released new versions of its general purpose and real time operating systems, SUSE Linux Enterprise Server 10 (currently Service Pack 2) and SUSE Linux Enterprise Real Time 10 (currently Service Pack 2 update 3).

SUSE Linux Enterprise Server (SLES) is an enterprise-quality server operating system, developed and backed by Novell. An open, scalable, high performance data center solution that spans a full range of hardware architectures, SLES can be deployed as a general purpose server, or it can be tailored to run a variety of specialized workloads reliably and securely.

SUSE Linux Enterprise Real Time (SLERT) is a fully supported real time operating system, specifically engineered to reduce the latency and increase the predictability of time-sensitive mission-critical applications.

We tested both SLES and SLERT by benchmarking two market data system configurations: the first emulating point-to-point distribution of data to hundreds of users, and the second following a configuration for low-latency distribution to heavy-duty applications. The key metric for the first configuration is max throughput—by convention, measured as total output from the server—while the key metrics for the second are both max throughput and latency at specific throughput levels.

In both cases, we ran the Reuters Market Data System (RMDS), using Reuters standard benchmarking procedures. We chose these procedures in order to enable easy comparison to other tests we have run with RMDS and because the emerging industry-standard STAC Benchmarks for market data middleware (STAC-M2) are still in development.

2. Description of Tests

2.1 Methodology

An HP c-Class BladeSystem was used to host the blades used in the tests. Each blade was configured with two Intel Xeon E5450 (codename “Harpertown”) processors running Novell’s SLES and SLERT operating systems.

The tests followed the procedures set forth by Reuters for hardware vendors and used test data supplied by Reuters.

2.1.1 Throughput testing

The P2PS “Producer 50/50” test is an extreme test of the fanout capability of a P2PS machine. It is oriented toward environments in which many users are connected to the P2PS and users have a high degree of commonality in their watchlists (meaning that for most of the updates that the P2PS receives from the backbone, it must forward each update to many users).

The sink_driven_src utility was used to generate update traffic from the sample files provided with the Source Distributor (sample.xml), and the rmdstestclient utility was used to consume the updates. The RMDS infrastructure was tuned for maximum throughput as per the Reuters RMDS 6.0 Performance Test Procedures, and the update rate was increased until data was lost, the system failed, or throughput lagged.

To maximize fanout performance, we chose a multiplex topology in which a single RRCP feeds multiple P2PS instances and each of those P2PS instances feeds multiple client apps. A single publishing app and a single source distributor supplied the data. This sort of “stacked” topology effectively co-locates multiple P2PS instances that would otherwise run on separate servers. The test harness is diagrammed in Figure 2-1.

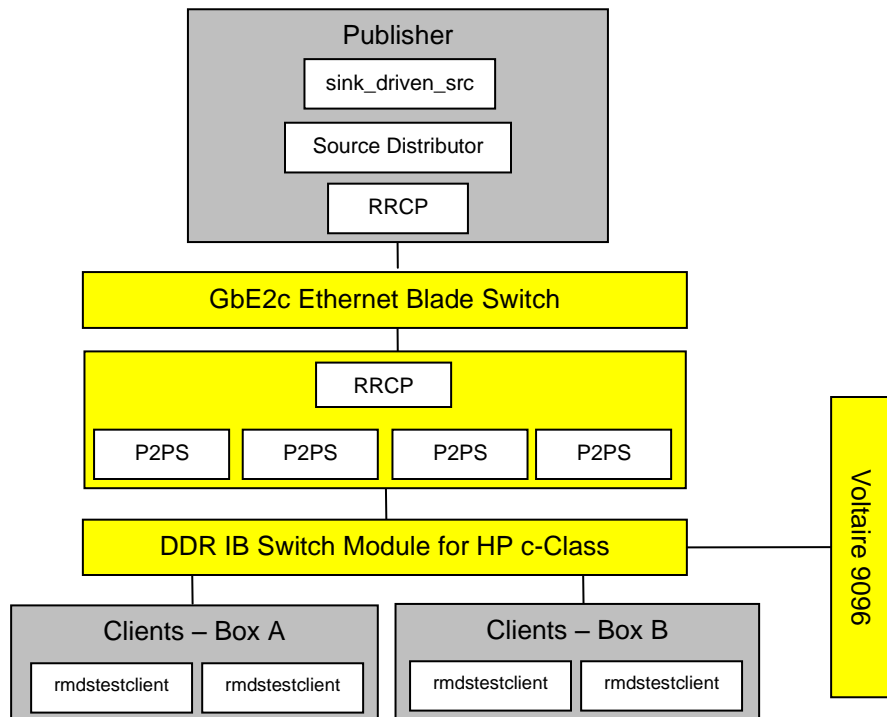


Figure 2-1: Stack under Test

For the InfiniBand throughput tests, the server running the P2PS instances had one HP 4X DDR IB Dual-port Mezzanine HCA InfiniBand interface; however only one of these ports was used during the test. Each server running rmdstestclient instances on the client LAN also had one 4X DDR IB Dual-port Mezzanine HCA InfiniBand interface card, but only one of these ports was used during the test. Both InfiniBand cards were connected to an internal, 4X DDR IB Switch Module for HP c-Class Blade System in the blade chassis. The InfiniBand fabric was managed by a Voltaire ISR 9096 switch. A dedicated network was used for the TCP traffic between the P2PS instances and the client applications. The SDP (Sockets Direct Protocol) library was used for the TCP traffic. The UDP multicast traffic between the source distributor box and P2PS box used a separate 1Gb/s interface port and network, as the traffic into the P2PS box was well under 1Gb/s.

For the Gigabit Ethernet throughput tests, the server had two GigE ports. All traffic between the P2PS instances and the client applications was over a single GigE interface. The UDP multicast traffic between the source distributor box and P2PS box used a separate GigE interface. This test harness is diagrammed in Figure 2-2.

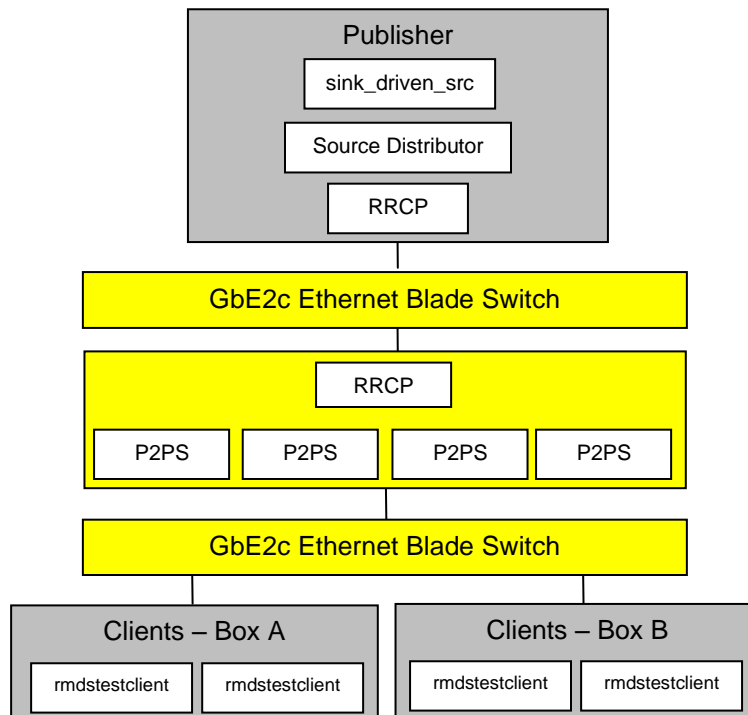


Figure 2-2: Stack under Test

2.1.2 Latency Testing

As specified by the Reuters test procedures, the embedded timestamp approach was used to calculate end-to-end latency for level 1 data, as shown in Figures 2-3 and 2-4. Latency was measured using ***sink_driven_src*** as the publisher and ***rmdstestclient*** as the subscriber. In the embedded timestamp approach, the publisher embeds timestamps into selected updates which the subscriber uses for latency calculations. In this scenario, the publisher and subscriber must be running on the same node for accurate timestamps. Latency tests were run on infrastructure tuned for low latency as per the Reuters RMDS 6.0 Performance Test Procedures.

Latency tests on InfiniBand used the VMA (Voltaire Messaging Accelerator) library for UDP traffic and the SDP library was used for TCP traffic. A virtual interface was used in order to have a differentiated network IP scheme, between the TCP and UDP traffic. On Gigabit Ethernet, the UDP multicast traffic on the RMDS backbone was put on a separate network from the TCP traffic.

End-to-end Infrastructure Latency Test

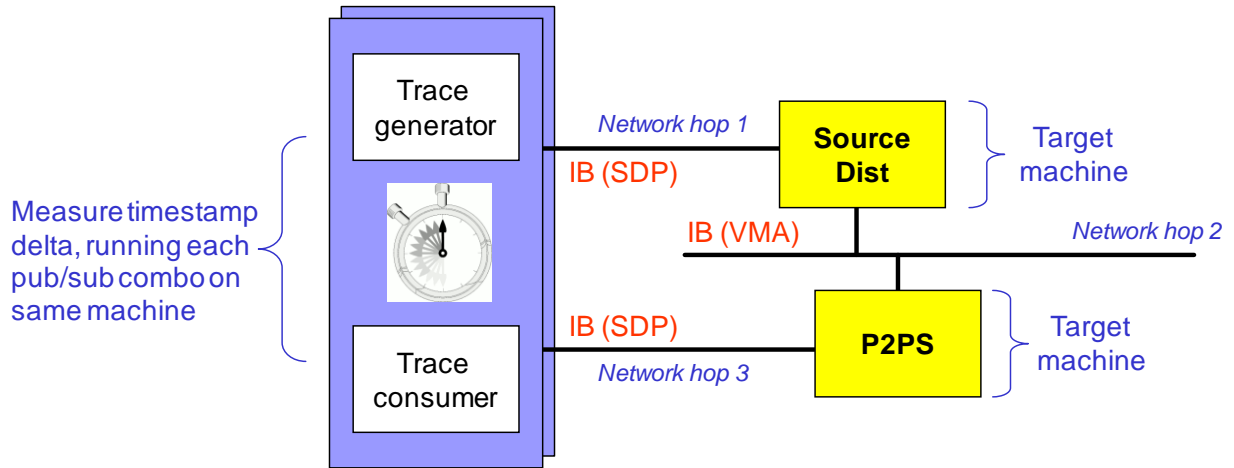


Figure 2-3 – Latency test on Voltaire InfiniBand

End-to-end Infrastructure Latency Test

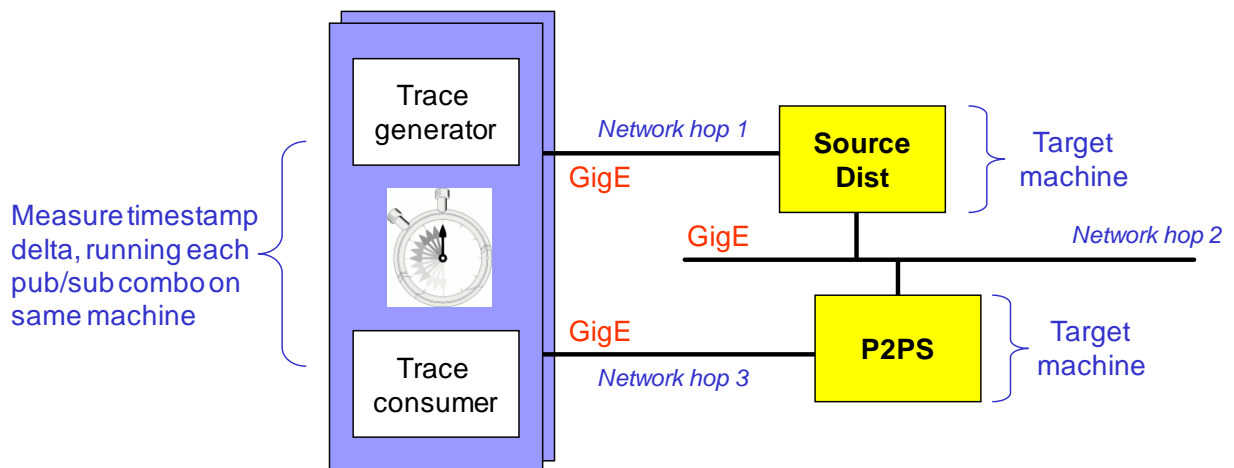


Figure 2-4 – Latency test on GigE

2.2 System Specifications

2.2.1 Servers

An HP c-Class BladeSystem chassis contained 4 blades (bay 1 to 4), each with the following configuration:

Vendor Model	BladeSystem c7000 Enclosure
Blade Bays	4 for Throughput tests and 3 for Latency tests
Rack Units	9U
Power Supplies	4

Each of the servers in the test harness had the following specifications:

Vendor Model	HP ProLiant BL460c G1 - Server Blade
Processors	2
Processor Type	Quad-Core Intel® Xeon® Processor X5450 3.00 GHz (codename "Harpertown")
Cache	12 MB Integrated L2 Cache split between 4 cores (2 x 6MB)
Bus Speed	1333 MHz
Memory	16 GB RAM, 667 Mhz
eth0 & eth1	Broadcom Corporation, NetXtreme II BCM5708S Gigabit Ethernet (rev 12)
eth2 & eth3	HP 4X DDR IB Dual-port Mezzanine HCA
NIC Note 1	All RMDS backbone traffic was directed over eth1 for Throughput tests
BIOS	I15 - 07/14/2008
Disks	SAS – 72 and 146 GB HP Smart Array E200i Controller
Rack Units	9 rack units for the blade chassis
Bays	4 different bays were used in the tests (4 for Throughput and 3 for Latency)

2.2.2 Networking

Switch #1	4X DDR IB Switch Module for HP c-Class BladeSystem
NIC	HP 4X DDR IB Dual-port Mezzanine HCA Mellanox Technologies MT25418 [ConnectX PCIe 2.0 2.5GT/s] (rev a0)
NIC driver	mlx4_core version 1.0
NIC firmware	2.5.0
SDP Version	libsdp-1.1.99-1.7
VMA Version	libvma-2.1.6-0
Switch #2	GbE2c Ethernet Blade Switch

NIC	Broadcom Corporation NetXtreme II BCM5708S Gigabit Ethernet (rev 12)
NIC Driver	bnx2, version 1.6.7
NIC firmware	1.9.6
Switch #3	Voltaire ISR 9096 switch used for IB Fabric Management

2.2.3 Network Interface Configurations

Any settings changed from the defaults are noted below:

The following values were set on specified Ethernet interfaces (eth0, eth1, ... <ethX>) used for RMDS traffic:	Command or Comment
Txqueuelen	ifconfig <ethX> txqueuelen 5000

2.2.4 Operating System

Version	SUSE Linux Enterprise Server 10 SP2 – 64 bit Kernel - 2.6.16.60-0.21-smp SUSE Linux Enterprise Real Time 10 SP2 – 64 bit Kernel - 2.6.22.19-20081007_SLERT10_SP2_BRANCH-rt ¹
General OS Services	All OS daemons were stopped with the exception of : dbus, fbset, haldaemon , kbd, kdump, microcode, network, openibd, random, resmgr, slert, splash, splash_early, sshd, syslog

2.2.5 TCP and UDP Buffer – key parameters

	Values were those specified by the Reuters guidelines. The following lines were entered into the System File (/etc/sysctl.conf):	System File
Setup-specific changes noted	net.core.wmem_max = 16777216	/etc/sysctl.conf for SLES /etc/slert/setup for SLERT
	net.core.wmem_default = 8388608	
	net.core.rmem_max = 16777216	
	net.core.rmem_default = 8388608	
	net.ipv4.tcp_rmem = 4096 262144 16777216	
	net.ipv4.tcp_wmem = 4096 8388608 16777216	
	net.ipv4.tcp_mem = 393216 524288 786432	
net.ipv4.ip_local_port_range = 32768 61000		

¹ This was a pre-release version of the kernel. It has since been released as version 2.6.22.19-0.18-rt.

2.2.6 RMDS Software

RMDS Binaries	mdh6.3.0.L4 p2ps6.3.1.L2 rrcp as included in p2ps6.3.1.L2
RMDS Test Tools	sink_driven_src (from mdh6.3.0.L4) rmdstestclient (from p2ps6.3.1.L2)

2.2.7 RMDS Configuration

	Ensure the following settings in <i>rmds.cnf</i> :
Common to all tests	*p2ps*rsslMsgPacking : True
	*p2ps*hashTableSize = 200000
	*usePointToPointData = False
	*RRCP*maxPktPoolSize : 80000
	*RRCP*pktPoolLimitHigh : 70000
	*RRCP*pktPoolLimitLow : 60000
	*RRCP*userQLimit : 32768
	*RRCP*udpRecvBufSize : 4096
	*RRCP*udpSendBufSize : 4096
	*<serviceName>*cacheLocation : srcApp
	*p2ps*enableCache : False
Throughput test	*p2ps*timedWrites : True
	*p2ps*flushInterval : 20
	*p2ps*tcpNoDelay : False
	*<serviceName>*rrmpFlushInterval : 20
	*p2ps*tcpSendBufSize : [commented out]
	*p2ps*guaranteedOutputBuffers : 800
	*p2ps*maxOutputBuffers : 5000
	*p2ps*poolSize : 32000
Latency test	*p2ps*timedWrites : False
	*p2ps*flushInterval : 0
	*p2ps*tcpNoDelay : True
	*<serviceName>*rrmpFlushInterval : 0
	*p2ps*tcpSendBufSize : 64240
	*p2ps*guaranteedOutputBuffers : 200
	*p2ps*maxOutputBuffers : 400
	*p2ps*poolSize : 16000
	*src_dist*route*numIpcInputBuffers : 10

	*src_dist*route*numIpcOutputBuffers : 100
	*src_dist*server*ipc*transmissionBus*guaranteedOutputBuffers : 200
	*src_dist*server*ipc*transmissionBus*numInputBuffers : 3
	*src_dist*server*ipc*transmissionBus*poolSize : 1600

2.2.8 Priorities and other settings

Settings indicated in the chart were used on runs for both SLES and SLERT, unless otherwise noted.

Throughput Tests InfiniBand	p2ps	P2PS 1 was bound with "taskset -c 0" P2PS 2 was bound with "taskset -c 2" P2PS 3 was bound with "taskset -c 4" P2PS 4 was bound with "taskset -c 6" All P2PS priorities were set with "chrt -f 55"
	src_dist	The src_dist priority was set with "chrt -f 55"
	rrcpd	Both src_dist and P2PS RRCP were bound with "taskset -c 1,5,3" The priority of both RRCP daemons was set with "chrt -f 55"
	rdmstestclient	client 1 on Box A and B was bound with "taskset -c 0" client 2 on Box A and B was bound with "taskset -c 4" All clients priorities were set with "chrt -f 49" ²
	SinkDrivenSrc	The source priority was set by "chrt -f 55"

² If the priority was set higher than 49, the rdmstestclient could not be used on SLERT unless it was strace'd.

	Operating System	<ul style="list-style-type: none"> • All system IRQ's were assigned to core 7 • The interrupt for the IB driver was assigned to cores 0,2,4,6 • The interrupt for the eth1 driver was assigned to cores 1,3,5 <p>For SLERT only:</p> <ul style="list-style-type: none"> • Cores 0 to 6 were shielded • The IRQ thread for the IB driver was assigned to cores 0,2,4,6 and a taskset value of -f 55 was assigned • The IRQ thread for the eth1 driver was assigned to cores 1,3,5 and a taskset value of -f 55 was assigned • A kernel module was loaded to remove all SMI's • The IPMI module was disabled <p>The priority of real-time "events" processes were increased with taskset value of -f 55</p>
Throughput Tests 1GigE	p2ps	<p>P2PS 1 was bound with "taskset -c 0" P2PS 2 was bound with "taskset -c 2" P2PS 3 was bound with "taskset -c 4" P2PS 4 was bound with "taskset -c 6" All P2PS priorities were set with "chrt -f 55"</p>
	src_dist	<p>The src_dist was bound with "taskset -c 1,5" The src_dist priority was set by "chrt -f 55"</p>
	rrcpd	<p>Both src_dist and P2PS RRCP were bound with "taskset -c 1,5,3" The priority of both RRCP daemons was set with "chrt -f 55"</p>
	rmdstestclient	<p>client 1 on Box A and B was bound with "taskset -c 0" client 2 on Box A and B was bound with "taskset -c 4" All clients priorities were set with "chrt -f 49"</p>
	SinkDrivenSrc	<p>The source was bound with "taskset -c 4" The source priority was set by "chrt -f 55"</p>

	Operating System	<ul style="list-style-type: none"> All system IRQ's were assigned to core 7 eth0 affinity set for core 1 (used for RRCP traffic) eth1 affinity set for core 0 (used for client traffic) <p>For SLERT only:</p> <ul style="list-style-type: none"> Cores 0 to 6 were shielded The IRQ thread for the eth0 driver was assigned to core 1 and a taskset value of -f 55 was assigned The IRQ thread for the eth1 driver was assigned to core 0 and a taskset value of -f 55 was assigned A kernel module was loaded to remove all SMI's The IPMI module was disabled The priority of real-time "events" processes were increased with taskset value of -f 55
Latency Tests InfiniBand	p2ps	The P2PS was bound with "taskset -c 1,5" The P2PS priority was set with "chrt -f 55"
	rrcpd	Both src_dist and P2PS RRCP were bound with "taskset -c 0,4,2,6" The priority of both RRCP daemons was set with "chrt -f 55"
	rmdstestclient	The client was bound with "taskset -c 1" The client priority was set with "chrt -f 55"
	SinkDrivenSrc	The source was bound with "taskset -c 4" The source priority was set by "chrt -f 55"
	Operating System	<ul style="list-style-type: none"> All system IRQ's were assigned to core 7 The interrupt for the IB driver was assigned to Core 0 <p>For SLERT only:</p> <ul style="list-style-type: none"> Cores 0 to 6 were shielded The IRQ thread for the IB driver was assigned to core 0 and a taskset value of A kernel module was loaded to remove all SMI's The IPMI module was disabled The priority of real-time "events" processes were increased with taskset value of -f 55
Latency Tests 1GigE	p2ps	The P2PS was bound with "taskset -c 1,5" The P2PS priority was set by "chrt -f 55"
	src_dist	The src_dist was bound with "taskset -c 1,5" The src_dist priority was set by "chrt -f 55"
	rrcpd	The source and P2PS RRCP was bound with "taskset -c 0,4,2,6" The priority of both RRCP daemons was set with "chrt -f 55" PRRCPCHRT="chrt -f 55"

	rmdstestclient	The client was bound with "taskset -c 4" The client priority was set by "chrt -f 55"
	SinkDrivenSrc	The source was bound with "taskset -c 5" The source priority was set by "chrt -f 55"
	Operating System	<ul style="list-style-type: none">• All system IRQ's were assigned to core 7• eth0 affinity set for core 1• eth1 affinity set for core 0 <p>For SLERT only:</p> <ul style="list-style-type: none">• Cores 0 to 6 were shielded• The IRQ thread for the eth0 driver was assigned to core 1 and a taskset value of -f 55 was assigned• The IRQ thread for the eth1 driver was assigned to core 0 and a taskset value of -f 55 was assigned• A kernel module was loaded to remove all SMI's• The IPMI module was disabled• The priority of real-time "events" processes were increased with taskset value of -f 55

3. Results

3.1 Throughput

Table 1 shows the aggregate updates/second of P2PS output in the Producer 50/50 test. SLERT with InfiniBand was able to achieve over 8% more throughput compared to SLES with InfiniBand. Compared to GigE, InfiniBand was able to achieve 613% and 677% on SLES and SLERT respectively. Throughput on GigE was limited by the available bandwidth on the GigE link.

System	Throughput rate inbound/outbound to P2PS box (updates/sec)
SLES/GigE	26,000 inbound/1.31 million outbound
SLERT/GigE	25,700 inbound/1.30 million outbound
SLES/InfiniBand	185,000 inbound /9.34 million outbound
SLERT/InfiniBand	200,000 inbound /10.1 million outbound

Table 1 – RMDS Producer 50/50 Throughput

3.2 Mean and Max Latency

“End-to-end” RMDS latency is defined as the delta between the time an update is posted by the publisher application to its API and the time the same update is received by the consuming application from its API, i.e. it includes the latency contribution from both the API and the core infrastructure components. Tables 2 through 5 record the latency statistics for tests run on SLES and SLERT with both 1GigE and InfiniBand. A chart of the mean latency for all of these combinations is presented in Figure 3-1 below.

In summary, these system configurations demonstrated the lowest mean latencies ever reported, in the context of previous RMDS benchmarks run by STAC. As expected, InfiniBand showed lower overall mean latency compared to GigE. On SLES with InfiniBand, the mean latency remained less than 1 ms through 600kups. SLERT demonstrated benefit here because it could support higher update rates at lower latencies. On SLERT with InfiniBand, the mean latency remained less than 0.670 ms through 750kups. When comparing latencies between environments, SLERT with InfiniBand reduced latency on average almost 25% when compared to SLES with InfiniBand and reduced latency 47% compared to SLERT with GigE. SLERT with InfiniBand yielded a 61% reduction in mean latency when compared to SLES with GigE, which most sites use today.

Figures 3-2 and 3-3 present the maximum latencies for these configurations. We observed generally lower maximums in the SLERT environment compared to the SLES environment when both systems used the same network environment. The reduction in maximum latencies is more apparent at higher update rates, when the system becomes more loaded. For example, at 750,000 updates per second

SLERT achieved 3.01 milliseconds of maximum latency. SLERT with InfiniBand reduced maximum latency values at rates over 200kups by 35% on average compared with SLES with InfiniBand and by 49% on average compared with SLES with GigE. In the SLERT environment, a special kernel module was available that removed all System Management Interrupts (SMI) from occurring. There was not enough time available in the project to investigate how much this optimization was responsible for reducing the maximum latencies.

Update Rate [74-byte RWF messages/sec]	Mean Latency (milliseconds)	Std Deviation (milliseconds)	Maximum Latency (milliseconds)	Minimum Latency (milliseconds)	Number of Latency Points
1,000	0.132	0.008	0.217	0.109	3000
5,000	0.178	0.073	4.145	0.140	3000
10,000	0.206	0.019	0.267	0.168	3000
20,000	0.241	0.023	0.355	0.189	3000
30,000	0.265	0.029	0.365	0.200	3000
40,000	0.281	0.031	0.655	0.207	3000
50,000	0.294	0.060	2.945	0.202	3000
60,000	0.302	0.042	0.950	0.207	3000
70,000	0.319	0.071	1.024	0.208	3000
80,000	0.326	0.086	4.076	0.209	3000
90,000	0.346	0.098	1.219	0.207	3000
100,000	0.346	0.064	0.575	0.213	3000
150,000	0.429	0.168	1.889	0.205	3000
200,000	0.478	0.134	1.070	0.220	3000
250,000	0.523	0.161	1.231	0.214	3000
300,000	0.579	0.201	1.506	0.215	3000
350,000	0.654	0.247	2.239	0.216	3000
400,000	0.731	0.291	2.097	0.212	3000
450,000	0.767	0.318	2.409	0.213	3000
500,000	0.833	0.351	2.569	0.210	3000
550,000	0.894	0.403	2.883	0.214	3000
600,000	0.938	0.430	3.003	0.215	3000
650,000	1.291	0.742	5.343	0.219	3000
700,000	1.483	0.728	5.060	0.223	3000
750,000	1.171	0.593	3.967	0.215	3000
800,000	1.843	2.617	17.724	0.210	3000

Table 2 – RMDS Latency on SLES with InfiniBand

Update Rate [74-byte RWF messages/sec]	Mean Latency (milliseconds)	Std Deviation (milliseconds)	Maximum Latency (milliseconds)	Minimum Latency (milliseconds)	Number of Latency Points
1,000	0.140	0.006	0.209	0.130	3000
5,000	0.156	0.004	0.238	0.142	3000
10,000	0.171	0.007	0.306	0.157	3000
20,000	0.205	0.010	0.390	0.188	3000
30,000	0.235	0.012	0.472	0.214	3000
40,000	0.254	0.026	1.251	0.224	3000
50,000	0.262	0.040	1.253	0.217	3000
60,000	0.272	0.036	0.766	0.223	3000
70,000	0.293	0.040	1.044	0.231	3000
80,000	0.302	0.051	1.090	0.233	3000
90,000	0.314	0.059	1.154	0.241	3000
100,000	0.303	0.045	1.052	0.231	3000
150,000	0.333	0.111	1.783	0.229	3000
200,000	0.344	0.062	1.201	0.229	3000
250,000	0.371	0.077	1.146	0.227	3000
300,000	0.386	0.083	1.249	0.232	3000
350,000	0.414	0.102	1.452	0.230	3000
400,000	0.433	0.127	1.596	0.226	3000
450,000	0.455	0.144	1.830	0.229	3000
500,000	0.480	0.177	2.016	0.230	3000
550,000	0.512	0.213	2.313	0.234	3000
600,000	0.545	0.246	2.349	0.233	3000
650,000	0.572	0.290	2.494	0.231	3000
700,000	0.617	0.370	2.879	0.235	3000
750,000	0.663	0.423	3.010	0.235	3000
800,000	1.560	3.763	25.292	0.232	3000

Table 3 – RMDS Latency on SLERT with InfiniBand

Update Rate [74-byte RWF messages/sec]	Mean Latency (milliseconds)	Std Deviation (milliseconds)	Maximum Latency (milliseconds)	Minimum Latency (milliseconds)	Number of Latency Points
1,000	0.255	0.003	0.278	0.240	3000
5,000	0.360	0.010	0.415	0.295	3000
10,000	0.462	0.007	0.490	0.432	3000
20,000	0.513	0.009	0.541	0.428	3000
30,000	0.542	0.021	0.787	0.454	3000
40,000	0.546	0.023	0.615	0.452	3000
50,000	0.576	0.026	0.680	0.464	3000
60,000	0.624	0.054	1.142	0.464	3000
70,000	0.656	0.054	0.780	0.470	3000
80,000	0.689	0.085	1.401	0.473	3000
90,000	0.723	0.096	1.359	0.481	3000
100,000	0.744	0.106	0.928	0.474	3000
150,000	0.878	0.249	2.319	0.467	3000
200,000	0.952	0.260	1.462	0.458	3000
250,000	1.075	0.332	2.116	0.474	3000
300,000	1.214	0.425	2.572	0.464	3000
350,000	1.332	0.507	2.898	0.467	3000
400,000	1.446	0.576	3.292	0.531	3000
450,000	1.604	0.665	3.896	0.461	3000
500,000	2.507	1.146	7.402	0.523	3000

Table 4 – RMDS Latency on SLES with 1 GigE

Update Rate [74-byte RWF messages/sec]	Mean Latency (milliseconds)	Std Deviation (milliseconds)	Maximum Latency (milliseconds)	Minimum Latency (milliseconds)	Number of Latency Points
1,000	0.244	0.004	0.263	0.227	3000
5,000	0.277	0.004	0.375	0.264	3000
10,000	0.322	0.006	0.457	0.308	3000
20,000	0.390	0.028	1.441	0.373	3000
30,000	0.448	0.011	0.656	0.424	3000
40,000	0.498	0.015	0.865	0.465	3000
50,000	0.525	0.037	1.514	0.446	3000
60,000	0.551	0.029	1.126	0.477	3000
70,000	0.552	0.032	1.179	0.461	3000
80,000	0.565	0.036	1.211	0.470	3000
90,000	0.573	0.056	1.382	0.471	3000
100,000	0.577	0.067	1.598	0.468	3000
150,000	0.603	0.095	2.036	0.474	3000
200,000	0.624	0.060	1.680	0.484	3000
250,000	0.672	0.075	1.716	0.477	3000
300,000	0.723	0.082	1.706	0.487	3000
350,000	0.761	0.124	1.867	0.475	3000
400,000	0.816	0.171	2.134	0.493	3000
450,000	0.875	0.226	2.592	0.485	3000
500,000	0.999	0.349	2.739	0.494	3000

Table 5 – RMDS Latency on SLERT with 1 GigE

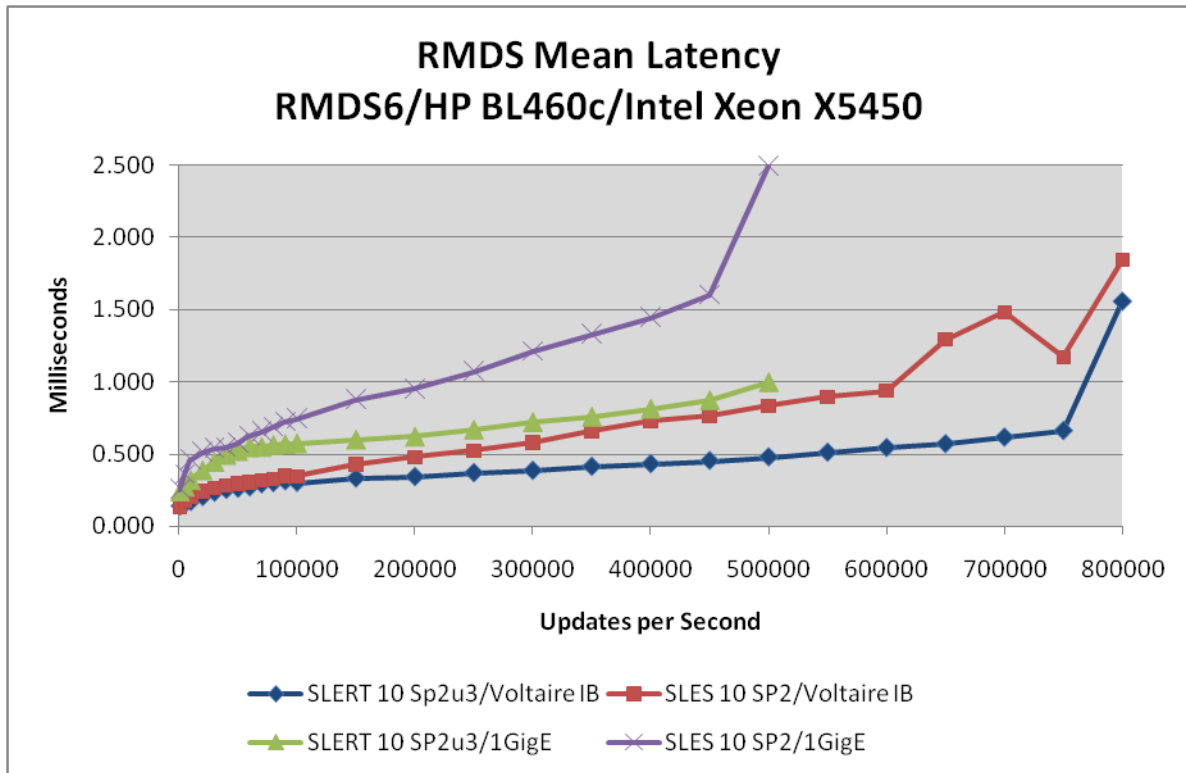


Figure 3-1

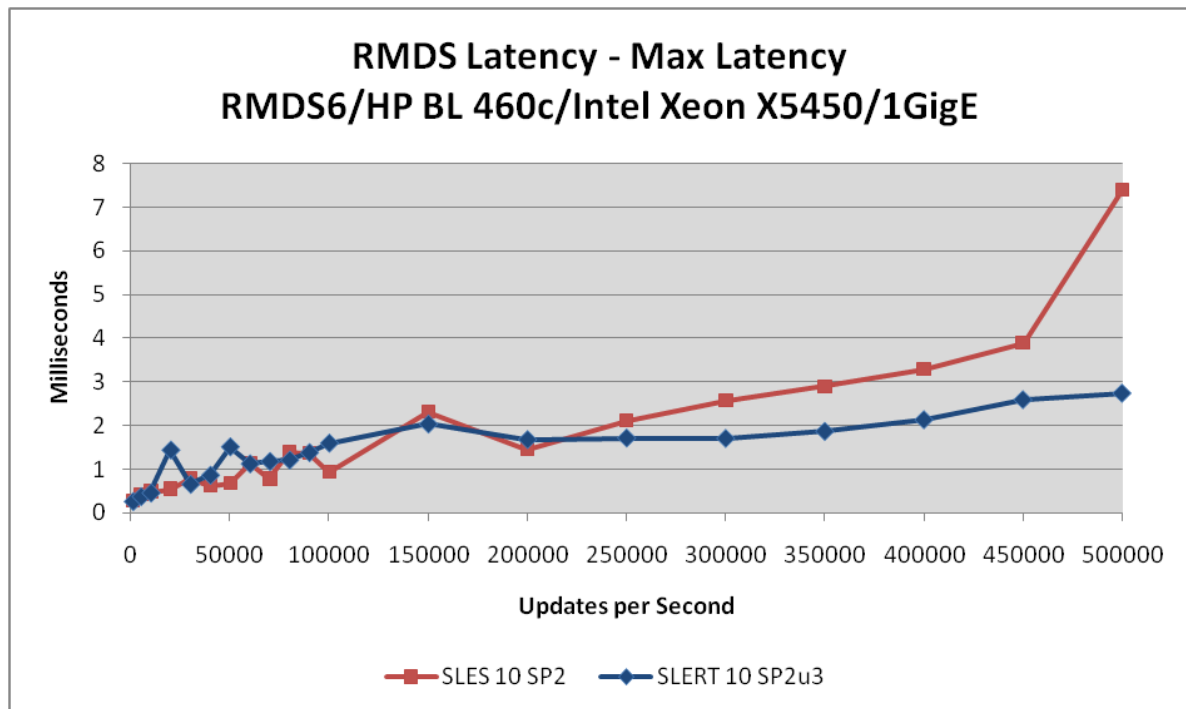


Figure 3-2

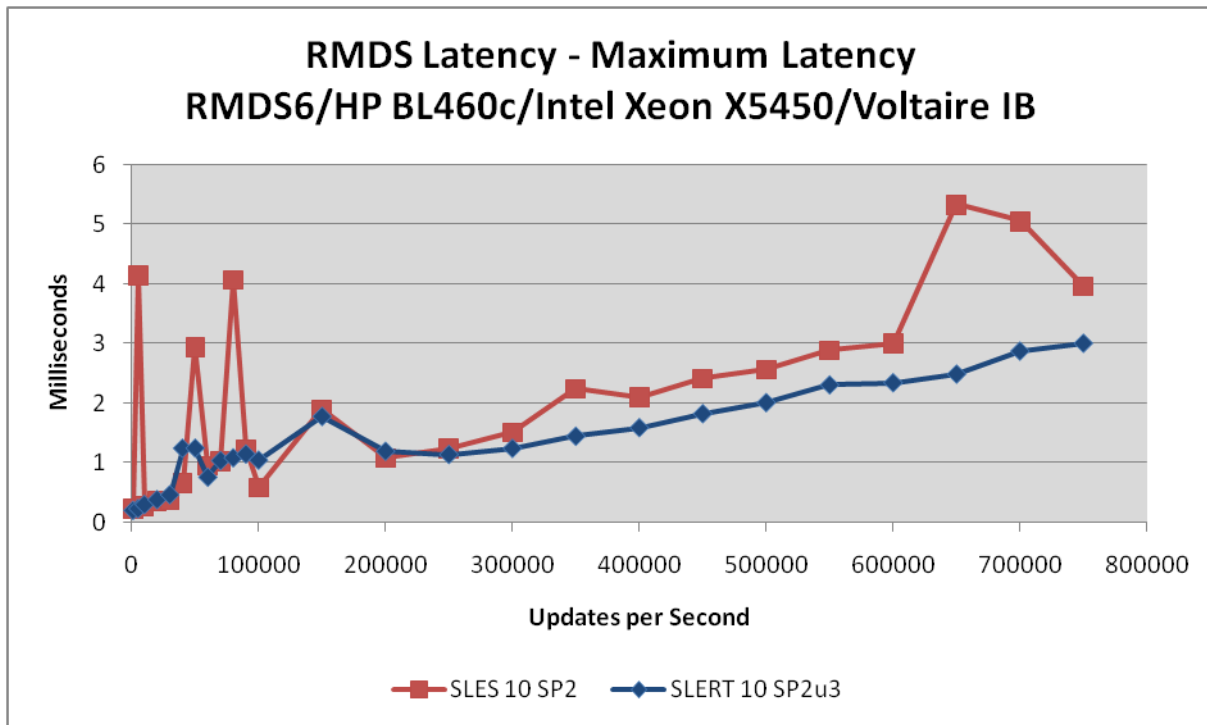


Figure 3-3

3.2 Standard Deviation

Latency dispersion or jitter – in other words, how spread out the latency values are – is just as important as the mean latency. Like the man who drowned in a river that was six inches deep on average, trader’s don’t care a lot about mean latency if the quote or order they cared the most about was delayed far beyond the mean. Moreover, if mean latency is low enough, reducing jitter can actually be more important than reducing mean latency, since dispersion befuddles trading algorithms with unpredictability.

Dispersion is an abstract concept that is captured in a variety of statistics, with standard deviation being a common one. In Figure 3-4, we plot the standard deviations across all of the update rates for the various operating system and network combinations used in the tests. In Figures 3-5 through 3-8, we also plot the latency histograms at specific rates, comparing the performance of SLES to SLERT on InfiniBand. The histograms show the frequency of latency observations for 10 microsecond buckets.

According to Novell, one of the main reasons to consider a real-time operating system is the promise of more deterministic latencies (i.e., lower jitter). Whereas on SLES, the standard deviation remained below 0.50 ms through 600,000 updates per second, on SLERT it remained below 0.50ms through 750,000 updates per second.

We see in Figure 3-4, that SLERT's standard deviation compares favorably to that in SLES for both 1GigE and InfiniBand environments. In these tests, SLERT reduced the standard deviation of latency by an average of 33% compared to SLES with 1GigE and by an average of 47% compared to SLES with InfiniBand. Figures 3-5 through 3-8 show how SLERT was able to keep the latencies both lower and in a tighter range.

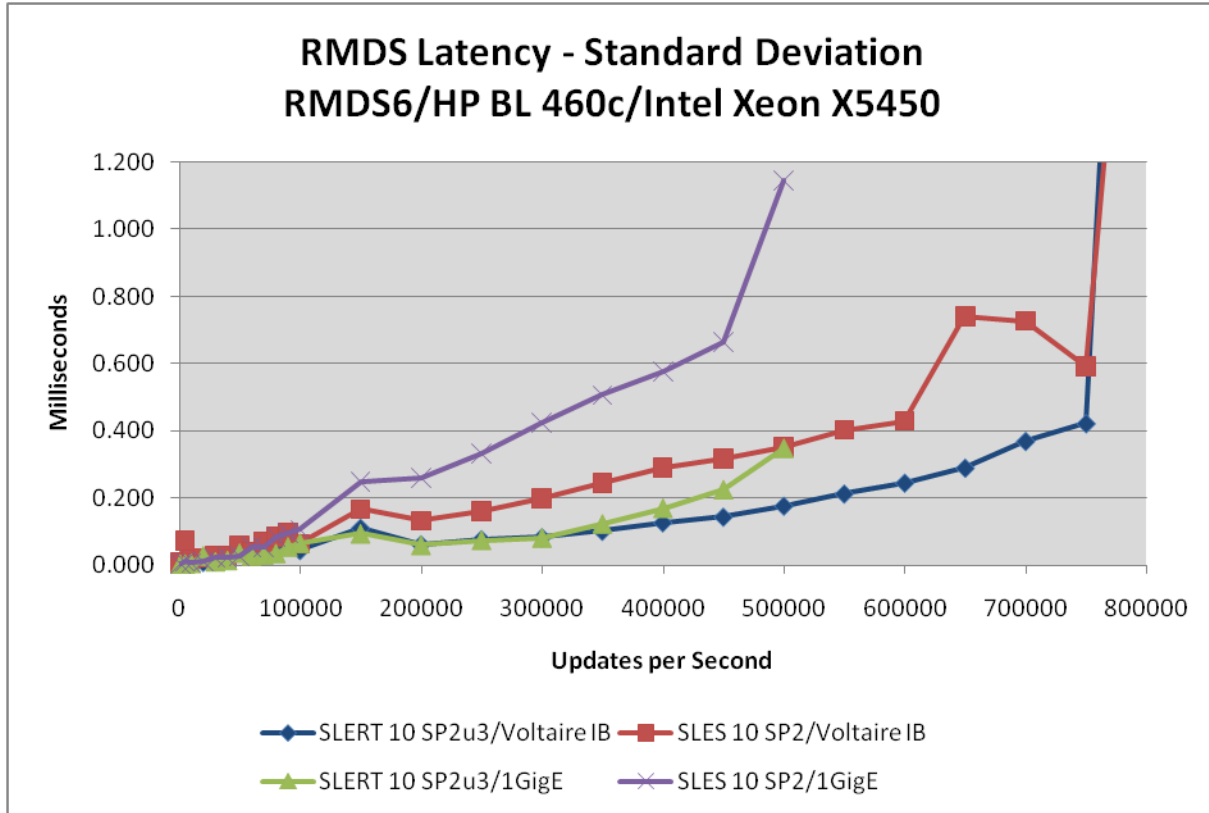


Figure 3-4

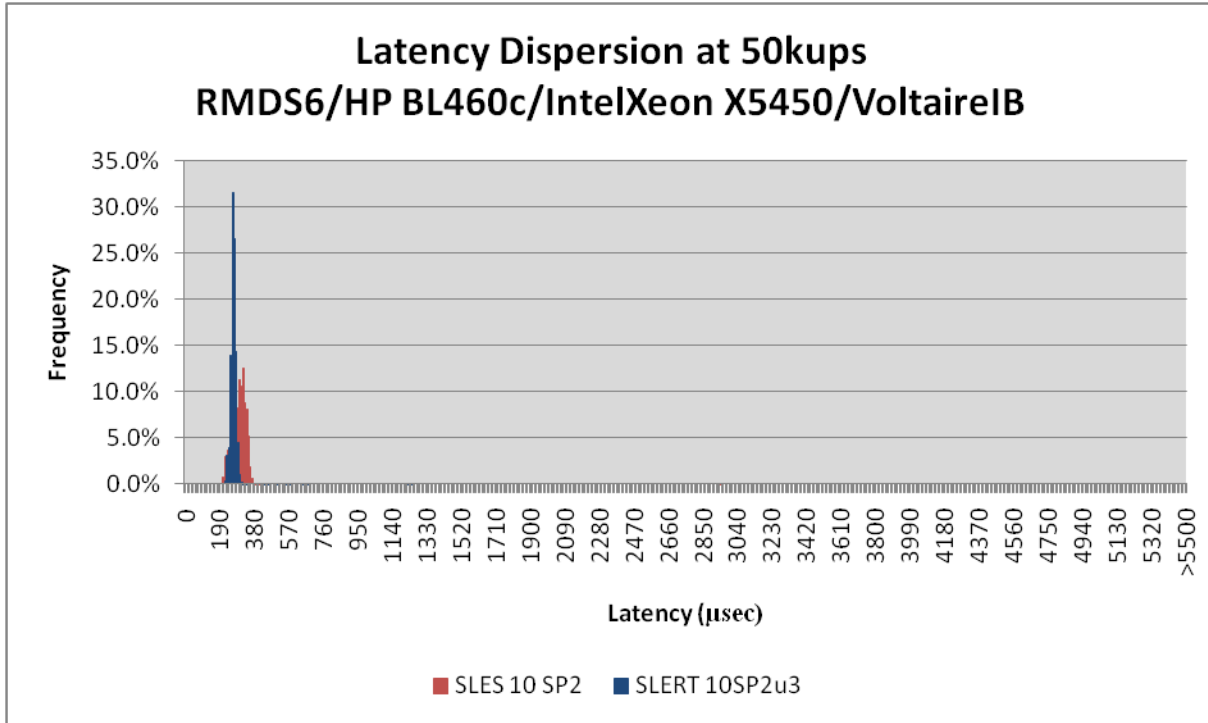


Figure 3-5

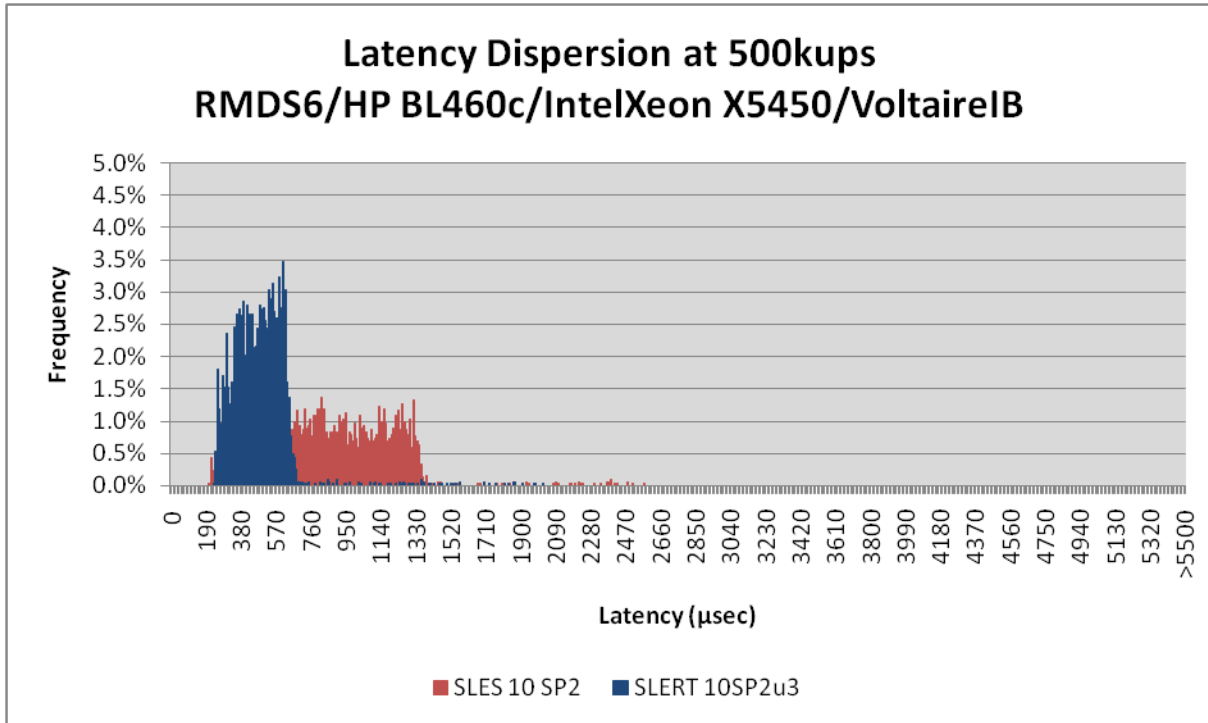


Figure 3-6

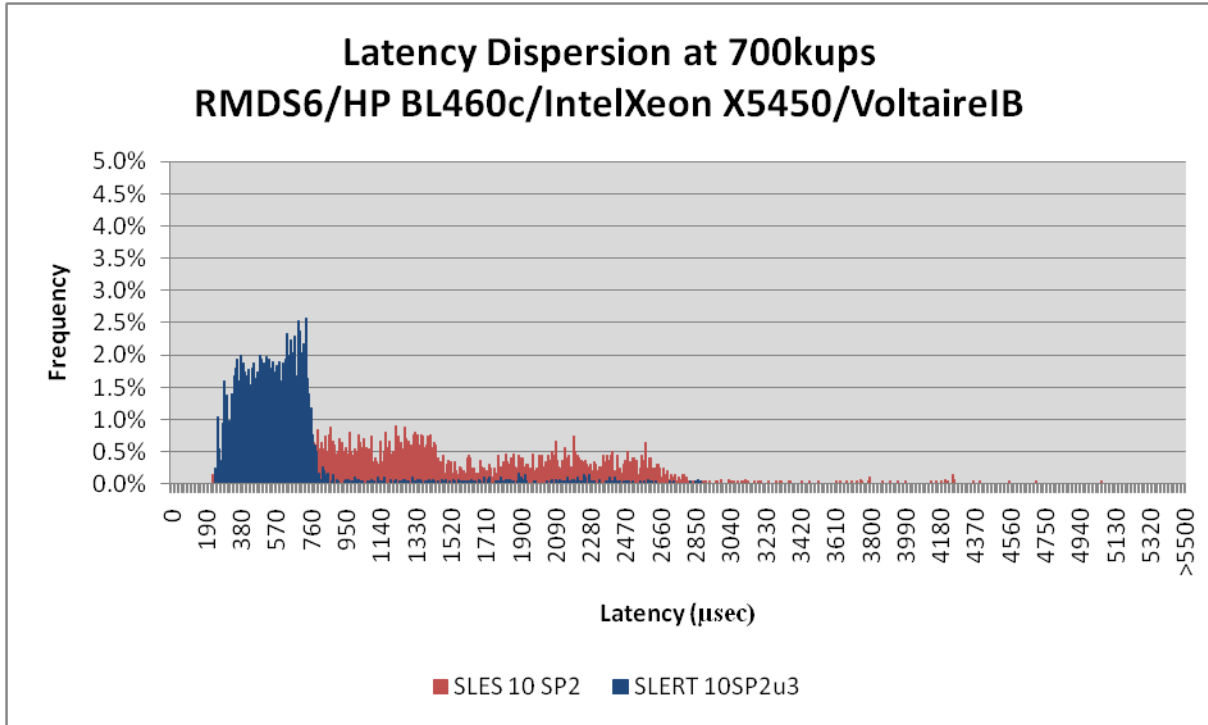


Figure 3-7

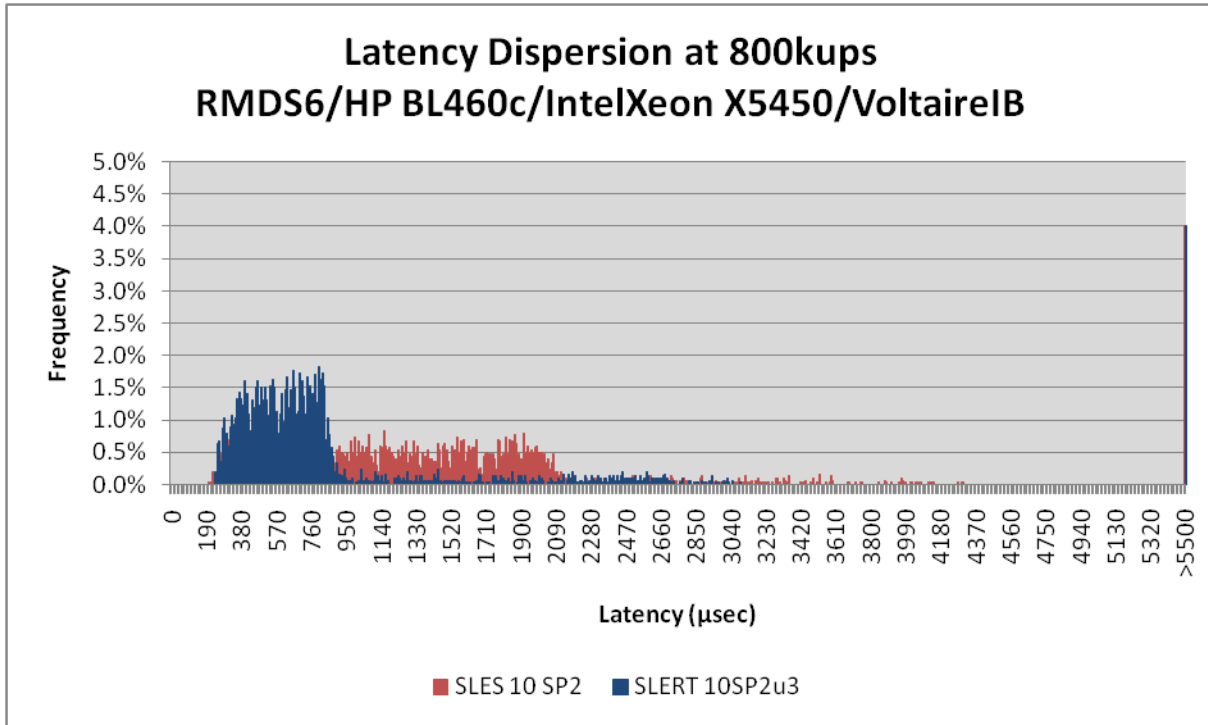


Figure 3-8

About STAC



The Securities Technology Analysis Center, or STAC, conducts private and public hands-on research into the latest technology stacks for capital markets firms and their vendors. STAC provides performance measurement services, advanced tools, and simulated trading environments in STAC Labs. Public STAC Reports, available for free at www.STACresearch.com, document the capability of specific software and hardware to handle key trading workloads such as real-time market data, analytics, and order execution.

STAC also facilitates the STAC Benchmark Council (see www.STACresearch.com/council), an organization of leading trading firms and vendors that specify standard ways to measure the performance of trading solutions.

To be notified when new STAC Reports like this one are issued, or to learn more about STAC, see our web site at www.STACresearch.com.