

Reuters Market Data System

RMDS 6 Multiple Instances(Consolidated)

Performance Test Results in Solaris

Containers/Zones Environment on Sun Blade X6270

(Two-Socket Intel Nehalem - Quad-Core dual thread)

Server Module with 10GbE Ethernet Infrastructure

DISCLAIMER: The test results and recommendations contained in this report are made available for informational purposes only. By issuing this report, Sun Microsystems, Inc. does not guarantee similar performance results. This report is provided for your internal use only and may not be redistributed or published in any form without the prior written consent of Sun Microsystems, Inc. All information contained herein is provided on an "AS-IS" BASIS WITHOUT WARRANTY OF ANY KIND. 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. Sun Microsystems, Inc. is pleased to provide this report for our customers. We appreciate that each organization has unique requirements, and therefore may find this information insufficient for its needs. Customers wishing to obtain custom analysis for their systems are encouraged to contact their local Sun Microsystems, Inc. representative.

Issue 1.1

Date of Issue: May 11, 2009

Amjad.Khan@sun.com

Contents

1 General Information	3
1.1 Objective.....	3
1.2 Results Summary for RMDS using Solaris Containers Configuration.....	3
1.3 Solaris Containers/Zones.....	4
1.4 Testing Methodology.....	4
1.5 Software Versions.....	6
1.5.1 RMDS.....	6
1.5.2 RMDS Test Tool.....	6
1.5.3 Operating Systems.....	6
1.6 Hardware.....	6
2 Preparation for Performance Test.....	6
2.1 Network.....	6
2.2 Hardware.....	6
2.3 Operating System Configuration.....	7
2.3.1 Solaris Kernel Parameters.....	7
2.3.2 TCP and UDP Buffers.....	8
2.3.3 Additional Solaris Kernel Parameters for Network Latency.....	8
2.4 RMDS Configuration.....	9
2.5 Miscellaneous Notes.....	10
3 Detailed Results for RMDS with Solaris Containers Configuration (Multiple Instances).....	15
3.1 RSSL/RWF Update Throughput.....	15
3.1.1 Standalone Source Distributor and P2PS/LAN on a single Server Simultaneously.....	15
3.1.2 Multiple Standalone Source Distributor in Solaris Containers Environment on a single Server.....	16
3.1.3 Multiple P2PS/LAN in Solaris Containers Environment on a single Server.....	16
3.2 Update Throughput via Multiple P2PS/POP Solaris Containers Environment on a single Server.....	17
3.2.1 RSSL/RWF.....	17
3.3 End-to-End RSSL/RWF Latency	18
3.3.1 RRCP Backbone Results When Src_dist and P2PS are installed in two Containers....	19

© Sun Microsystems, Inc. 2009. All Rights Reserved.

Sun Microsystems, Inc., by publishing this document, does not guarantee that any information contained herein is and will remain accurate or that use of the information will ensure correct and faultless operation of the relevant service or equipment. Sun Microsystems, Inc., its agents and employees, shall not be held liable to or through any user for any loss or damage whatsoever resulting from reliance on the information contained herein.

This document contains information proprietary to Sun Microsystems, Inc. and may not be reproduced, disclosed, or used in whole or part without the express written permission of Sun Microsystems, Inc.

Any Software, including but not limited to, the code, screen, structure, sequence, and organization thereof, and Documentation are protected by national copyright laws and international treaty provisions. This manual is subject to U.S. and other national export regulations.

1 General Information

1.1 Objective

The objective of this document is to report the performance test results for RMDS 6 in Solaris Containers/Zones environment on the Sun Blade 6000 equipped with Sun Blade X6270 (Intel Xeon CPU X5570 - Quad-Core – dual hyper-thread per core @ 2.93GHz) Server Modules. The test procedures are described in “Reuters RMDS 6 Performance Test Procedures and Results” document.

Two separate set of tests were made, one installing single instance of Source Distributor (**src_dist**) and Point-to-Point Server (**p2ps**) together on a single Sun Blade X6270 server module within Solaris containers/zones, a free Sun virtualization technology, environment. The **src_dist** and **p2ps** were installed on its dedicated Solaris container/zone using a standard (“**Traditional**”) RMDS benchmark procedure. The latency were also measured when **src_dist** and **p2ps** were installed on a single server within two containers/zones. The second set of tests were executed running multiple instances of RMDS applications, either multiple instances of Source Distributor (**src_dist**) or Point-to-Point Server (**p2ps**), on the same hardware using multiple Solaris 10 containers/zones. In the second set of tests, the standard test sequence was extended to demonstrate the Solaris and Sun Blade X6270 server module scalability for an existing non-multi-threaded RMDS version. To understand how Solaris Containers/Zones helps in reducing the TCO for RMDS, please look at “[Sun RMDS Server Consolidation with Solaris 10 Containers White Paper](#)”.

Sun Blade X6270 server module is a x86 architecture based on the Intel next generation Nehalem (Xeon X5000 series) processor chip. The blades used in these tests had 2 processor chips with 4 cores per chip and each core supports 2 hyper-threads (Total 8 Cores – 16 Hyper-Threads) to execute applications in parallel.

The goal of these tests is to measure throughput and latency through RMDS 6 infrastructure components, specifically the Point-to-Point Server (P2PS) and Source Distributor in Solaris Container/Zone environment for RMDS server consolidation purposes without compromising on the performance and latency. The tests are grouped into three categories:

- Update throughput via Src_Dist & P2PS-LAN using RSSL/RWF data (see 3.1)
- Update throughput via P2PS/POP (see 3.2)
- End-to-end RSSL/RWF latency using embedded timestamp (see 3.3)

1.2 Results Summary for RMDS using Solaris Containers Configuration

RMDS Components Configuration	RMDS Aggregated Throughput & End-to-End Latency Using Solaris Containers on 10GbEthernet Infrastructure
Maximum Throughput at < 1ms RMDS end-to-end Latency with RVD	900,000 updates/sec (0.981 msec)
P2PS-POP, Producer 50/50 Fanout (Cache Enabled)	10,529,250 updates/sec
P2PS-LAN, RRCP, Producer 50/50 Fanout (Cache Disabled)	8,453,700 updates/sec
Src_dist, RRCP (Cache Disabled)	3,330,000 updates/sec
P2PS-LAN, RRCP No Fanout (Cache Disabled)	3,060,000 updates/sec
P2PS-POP, No Fanout (Cache Enabled)	2,100,000 updates/sec

Note: No modifications were required to RMDS software to account for the use of a Solaris Containers as opposed to a separate system.

1.3 Solaris Containers/Zones

Solaris Containers/Zones were introduced with the Solaris 10 OS, and are available on all platforms that the Solaris 10 OS is supported on. Solaris Containers/zones allow individual instances of Solaris 10 to be “installed” and “booted” within a single global instance of the Solaris 10 OS. Each of these non-global containers/zones appear as individual Solaris instances, each with their own identity, user namespace, IPC namespace and network namespace. They are security and fault contained, thus one non-global container does not have the ability to view the data or resources associated with another non-global container. Faults inside one non-global container cannot propagate into any other non-global container.

Resources may be dedicated to Solaris Containers. These resources include CPU, memory and network bandwidth. Each container also has the ability to run with a different default scheduling class, thus affecting runtime behaviour for only the applications running within that container. File systems can be dedicated to containers, and can also be configured as shared resources between containers. Details of Containers can be found at:

http://www.sun.com/software/solaris/Containers_learning_center.jsp and
<http://www.opensolaris.org/os/community/zones/>

The tests were run using the Solaris Containers/Zones as virtualization technology. This technology allows multiple copies of an operating system environment to be run on the same machine. In this testing, up to 3 Solaris containers(zones) were created, which allowed 3 independent *src_dist* or *p2ps* applications to be run simultaneously within its own container on a single Sun Blade X6270 Server Module. Each *src_dist* or *p2ps* instance was paired with a *rccpd* transport daemon, running with it within each container(zone). **No Tibco RV transport was used in Solaris Container environment because of the limited time and resources. However, Tibco RV transport has been successfully tested with RMDS in Solaris container environment.**

Solaris Containers allow multiple instances of RMDS software to run on the Sun Blade X6270 server module. Tests were run with 1, 2, and 3 instances of the RMDS software running on the X6270 blade, each instance in its own container, with results showing excellent scaling up to 3 instances of *src_dist* or *p2ps* on a single Sun blade system. The traditional RMDS standalone benchmark on the same Sun Blade X6270 server module can be accessed at <http://www.sun.com/reuters>.

1.4 Testing Methodology

For throughput testing, the *sink_driven_src* utility was used to generate update traffic, and the *rmdstestclient* utility was used to consume the updates. Level 1 data was used, with a Reuters Wire Format (RWF) update size of 74 bytes. Tests with no fanout of updates used a 100,000 item watchlist. The infrastructure is tuned for maximum throughput, and the update rate was increased until the CPU limit was reached with no errors reported. Where needed, and as noted, multiple Source Distributors or multiple P2PSs were used to create the load necessary to measure the component under test.

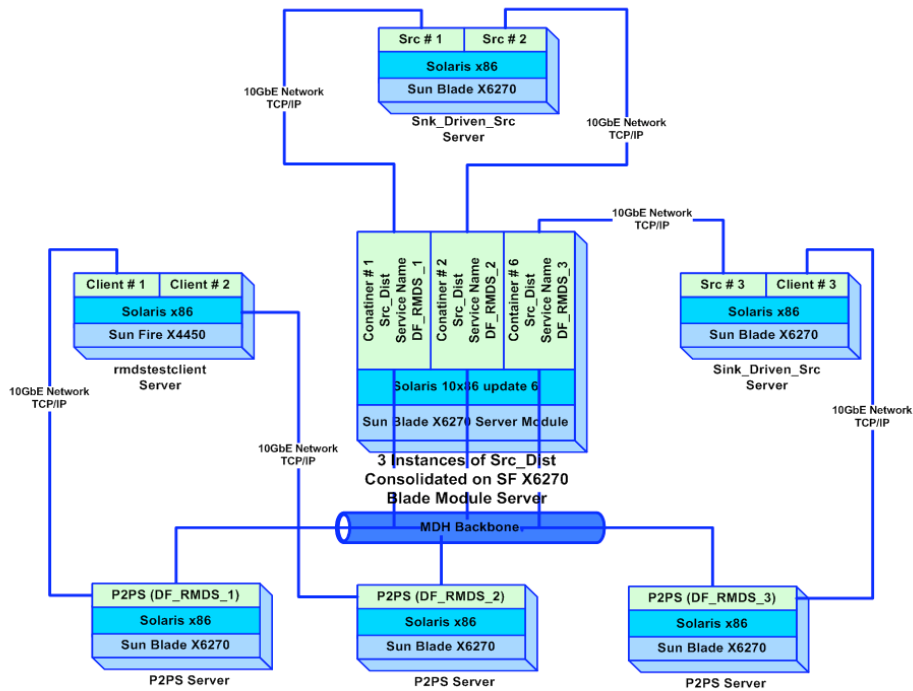
The embedded timestamp approach was used to calculate end-to-end latency for Level 1 (Quotes and Trades) data. RMDS 6 end-to-end update latency is measured by using *sink_driven_src* as the publisher and *rmdstestclient* as the subscriber. In the embedded timestamp approach, the publisher embeds timestamps 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.

For the RMDS multiple instances testing with Solaris containers, single, 2, and 3 simultaneously active containers(zones) were installed on X6270 blade system, each with a single *src_dist* or *p2ps* process. Each *src_dist* instance was connected with its own *sink_driven_src* application whereas in case of *p2ps* test, each *p2ps* instance was driven from a common RMDS Source

Distributor feed. Please look at the following diagram for further details on the configuration used for testing:

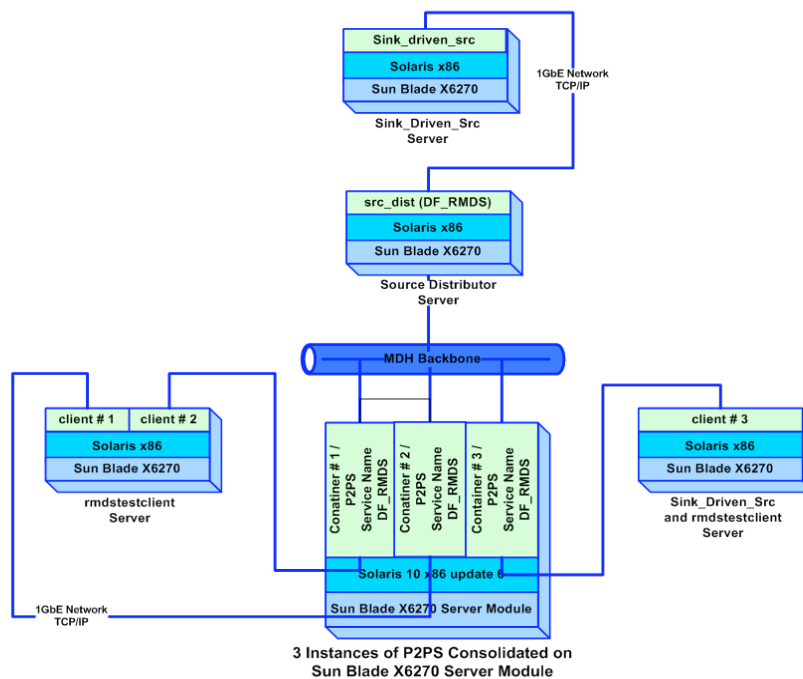
RMDS 6.0 with Solaris Containers Config # 1

Up to 3 x SRC_DIST Instances on a Single Sun Blade X6270 Server Module



RMDS 6.0 with Solaris Containers Config # 2

Up to 3 x P2PS Instances on a Single Sun Blade X6270 Server Module



1.5 Software Versions

1.5.1 RMDS

src_dist ver. mdh6.3.2.L1
p2ps ver. p2ps6.3.3.L1
rrcp as included in p2ps6.3.3.L1

1.5.2 RMDS Test Tool

sink_driven_src (from MDH load above)
rdmstestclient (from P2PS load above)

1.5.3 Operating Systems

Solaris 10 10/08 s10x_u6wos_07a X86 (update 6)
Patch Level Generic_137138-09

1.6 Hardware

- **Sun Blade X6270 Server Module** – Sun Blade Server Module powered by two-socket Quad-Core(dual thread) Intel Xeon X5500 series – **2 X 2.93GHz** Quad-Core Intel Xeon X5570 processors (total 8 cores / 16 threads)
- **Sun Dual 10GbE SFP+ PCIe Express Module** with Intel 82598 10 Gigabit Ethernet Controller Supports Pluggable SFP+ Transceivers (X1108A-Z)
- **Arista 7124S 10GbE Ethernet switch** 24-port 10 GbE 480 Gbps, 360 Mpps, L2/3/4, 1U Switch (SFP)

2 Preparation for Performance Test

2.1 Network

All the performance tests were run where the machines were connected to a private network via 10-Gigabit Ethernet(10GbE) switch Gbps. Each server blade had Sun Dual 10GbE SFP+ PCIe Express Module where each port in a blade connected to a separate backplane and switch. One was used for the UDP traffic and the other was used for TCP communications. Each zone configured on the X6250 was given 2 virtual IP addresses that corresponded to the 2 primary 10GbE NIC ports used. Each zone was separately addressable just as separate hardware systems would be via these IP addresses. No modifications were required to RMDS software to account for the use of a zone as opposed to a separate system.

All the network cards and switch ports were set to Auto Negotiate and the standard MTU of 1500 was used on all of these interfaces. There was no TOE used for throughput or latency testing over 10GbE network on Solaris. Solaris default 10GbE driver(**ixgbe**) was used for all of the testing.

2.2 Hardware

All RMDS components were run on the same class of machines. Five to Six Sun Blade X6270 server modules were available for these tests. For all tests, each X6270 blade module was installed with two to three Solaris containers/zones for server consolidation testing purposes in the Solaris container environment. In all throughput tests especially for src_dist tests for multiple instances installed on a single system, additional Sun Fire X4450 server was used for the rdmstestclient tools. On all of the systems the Sun Dual 10GbE SFP+ PCIe Express Module were used for MDH backbone and TCP network traffic shared across all the zones.

2.3 Operating System Configuration

Solaris 10 10/08 (update 6) was installed on all systems with the default Solaris 10 update 6 patch “Generic_137138-09”. The **ixgbe** network driver for 10GbE was used for all the testing.

The network parameters set via ndd commands may be set from the command line or via a Service Management Facility (SMF) manifest and method file (site/ndd) available from: <http://opensolaris.org/os/community/smf/manifests/>

For Solaris 10 update 3 and beyond, the udp_smallest_anon_port parameter should be set from the network-anon manifest and method files that are also available from the above [URL](#).

Further detail on tuning Solaris 10 and RMDS can be found at <http://www.sun.com/reuters> .

The combination of MDH 6.3.2 and P2PS 6.3.3 software used in this testing is the standard Reuters distribution for Sun x86 Solaris. As such RMDS 6 was compiled with optimization specific to the AMD Opteron processor, not to the Intel Xeon. Some additional performance gains may be realizable with a custom compile for the Intel Xeon architecture.

The following steps were taken to install the Solaris containers/zones on each of these servers:

- Default “**Sparse root**” models was used for non-global zone root file-system
- No “**Resource pool**” was used; No resource pool was associated to a zone
 - Processor sets were created by global zone and the src_dist or p2ps with its associated transport processes were bound to the psets. “**pbind**” was also used to further bind the hot runner threads of the RMDS processes within the pset. See section 2.5 for more details.
- No “**cpu capping**” or “**memory capping**” was used
- Default “**shared IP zones**” were used; No “**exclusive IP zones**” used

2.3.1 Solaris Kernel Parameters

Any settings changed from the defaults are noted below:

Step	Procedure		
1	OS	Enter the following lines in system file noted	System File
	Solaris	set dld:dld_opt=2	/etc/system
		set ddi_msix_alloc_limit=8	
		set pcplusmp:apic_multi_msi_max=8	
		set pcplusmp:apic_msix_max=8	
		set pcplusmp:apic_intr_policy=1	
set hires_tick=1 (Only on the sink_driven_src and rmdstestclient systems)			

2.3.2 TCP and UDP Buffers

Any settings changed from the defaults are noted below:

Step	Procedure		
2	OS	Enter the following lines in system file noted	System File
	Solaris	/usr/sbin/ndd -set /dev/tcp tcp_max_buf 8388608	/lib/svc/method/net-init
		/usr/sbin/ndd -set /dev/tcp tcp_recv_hiwat 4194304	
		/usr/sbin/ndd -set /dev/tcp tcp_xmit_hiwat 4194304	
		/usr/sbin/ndd -set /dev/udp udp_max_buf 8388608	
		/usr/sbin/ndd -set /dev/udp udp_xmit_hiwat 4194304	
		/usr/sbin/ndd -set /dev/udp udp_recv_hiwat 4194304	
/usr/sbin/ndd -set /dev/udp udp_do_checksum 0			

2.3.3 Additional Solaris Kernel Parameters for Network Latency

Step	Procedure		
4	OS	Enter the following lines in system file noted	System File
	Solaris	set autoup=300 (latency tests only) set tune_t_fsflushr=5 (latency tests only)	/etc/system

2.4 RMD5 Configuration

The configuration templates—*rmds.cnf.template* and *pop.cnf.template*—were customized for the tests.

Config File	Description	Path
<i>rmds.cnf.template</i>	Configuration file	<i>\$RMD5_SW/config</i> <i>on mdh and p2ps/LAN systems</i>
<i>pop.cnf.template</i>	Configuration file to do P2PS/POP test	<i>\$RMD5_SW/config</i> <i>on p2ps/POP system only</i>

Any non-required changes (i.e., IP addresses, hostnames, etc) for **throughput** are noted below:

Config File	Parameter	Value
<i>rmds.cnf</i>	*IDN_RDF*rrmpFlushInterval	20
<i>rmds.cnf</i>	*DF_RMD5*rrmpFlushInterval	20
<i>rmds.cnf</i>	*p2ps*tcpNoDelay	False
<i>rmds.cnf</i>	*p2ps*timedWrites	True
<i>rmds.cnf</i>	*p2ps*flushInterval	20
<i>rmds.cnf</i>	*RRCP*udpRecvBufSize	8192
<i>rmds.cnf</i>	*RRCP*udpSendBufSize	8192
<i>rmds.cnf</i>	*DF_RMD5*readBias	60 (add this entry)
<i>rmds.cnf</i>	*p2ps*maxOutputBuffers	32000
<i>rmds.cnf</i>	*p2ps*guaranteedOutputBuffers	800
<i>rmds.cnf</i>	*p2ps*poolSize	32000
<i>rmds.cnf</i>	*p2ps*latencyCalcMountBased	False
<i>rmds.cnf</i>	*p2ps*latencyCalcEnabled	False
<i>rmds.cnf</i>	*src_dist*latencyCalcServerBased	False
<i>rmds.cnf</i>	*src_dist*latencyCalcEnabled	False
<i>rmds.cnf</i>	*RRCP*maxPktPoolSize	200000
<i>rmds.cnf</i>	*RRCP*pktPoolLimitHigh	190000
<i>rmds.cnf</i>	*RRCP*pktPoolLimitLow	180000

Values modified for **network latency** testing are noted below:

Config File	Parameter	Value
<i>rmds.cnf</i>	*IDN_RDF*rrmpFlushInterval	0
<i>rmds.cnf</i>	*DF_RMD5*rrmpFlushInterval	0
<i>rmds.cnf</i>	*p2ps*tcpNoDelay	True
<i>rmds.cnf</i>	*p2ps*timedWrites	False
<i>rmds.cnf</i>	*p2ps*flushInterval	0
<i>rmds.cnf</i>	*p2ps*maxOutputBuffers	400
<i>rmds.cnf</i>	*p2ps*guaranteedOutputBuffers	200
<i>rmds.cnf</i>	*p2ps*poolSize	16000
<i>rmds.cnf</i>	*p2ps*latencyCalcMountBased	False
<i>rmds.cnf</i>	*p2ps*latencyCalcEnabled	False
<i>rmds.cnf</i>	*src_dist*latencyCalcServerBased	False
<i>rmds.cnf</i>	*src_dist*latencyCalcEnabled	False
<i>rmds.cnf</i>	*RRCP*maxPktPoolSize	80000
<i>rmds.cnf</i>	*RRCP*pktPoolLimitHigh	70000
<i>rmds.cnf</i>	*RRCP*pktPoolLimitLow	60000

2.5 Miscellaneous Notes

For RMDS 6, the **p2ps** application is a multi-threaded process where one thread does a substantial amount of work (this thread is referred to as the “high runner” below) with several other threads supporting it. When running with **rrcpd** in a LAN configuration, thread number 4 (of 5) is the high runner. Similarly when running in a POP configuration using RSSL/RWF, thread 4 (of 4) is the high runner.

Whereas, the **src_dist** application is a single threaded and the thread number 1 is always the high runner thread during all of the **src_dist** tests regardless of **rrcpd** or **rxd** transport.

There were two different approaches were taken for the processor binding during these tests. For up to two RMDS instances in containers environment, a very tight processor binding within processor sets since there were enough cores (8 cores) available to create a processor set out of 4 cores for each instance. However, for 3 instances no processor set was created and only the **pbind** command was used to bind the hot runner threads to different cores.

There are test cases where use of more strands may be desirable, and it is expected that continued development of the **src_dist** or **p2ps** application will also require changes to this strategy.

Aggregated output values are calculated as:

$$\text{Aggregate} = \text{input rate} * \text{instances}$$

Aggregated Fanout output values are calculated as:

$$\text{output} = \text{input rate} * \text{fanout} * \text{instances}$$

$$\text{Aggregate} = \text{output rate} * \text{instances}$$

where fanout is 50.5 as described in the Reuters document “RMDS 6.0 Performance Test Procedures and Results”.

Test	Deviation	Comments
All	Multicast vs Broadcast	The standard <code>rmcs.cnf.template</code> file does not enable multicast for RRCP, but does so for RV. All of the RRCP tests were run here with multicast as opposed to broadcast for UDP networking.

<p>src_dist and p2ps running on a single server</p>	<p>CPU binding, interrupts disabling and CPU scheduling class assignment</p> <p>src_dist and p2ps running in two Containers simultaneously on a single server</p>	<p>Interrupts were disabled on the cores 1-7 & 9 when hyperthreading was enabled on Nehalem processor. A single src_dist or p2ps paired with its transport daemon was run per container.</p> <p>Start the src_dist and p2ps processes from non-global-zones(NGZ) and then from the global zone(GZ), create and bind the hot runner threads of src_dist and p2ps and their associated transports as under:</p> <pre>% psrset -c -F 1 9 % psrset -c -F 2 3 % psrset -c -F 4 5 % psrset -c -F 6 7 % psradm -i 1-7 (Make sure the hyper-threading turned on)</pre> <pre>% psrset -b 1 <src_dist PID> % psrset -b 2 <src_dist RRCPD PID> % psrset -b 3 <p2ps PID> % psrset -b 4 <p2ps RRCPD PID></pre> <p>The following commands can be run through global zones or non-global zones as long as the <code>proc_prioctl</code> privilege is added to NGZs.</p> <pre>% pbind -b 1 <src_dist PID>/1 % pbind -b 9 <src_dist PID>/2 % pbind -b 2 <src_dist RRCPD PID>/7 % pbind -b 3 <src_dist RRCPD PID>/1-6,8-13</pre> <pre>% pbind -b 4 <p2ps PID>/4 % pbind -b 5 <p2ps PID>/1-3,5 % pbind -b 6 `pgrep rrcpd`/9 % pbind -b 7 `pgrep rrcpd`/1-8,10,13</pre> <p>FX scheduling class was assigned to src_dist and p2ps and its rrcpd daemon processes:</p> <pre>% prioctl -s -c FX -m 60 -p 60 <src_dist PID> % prioctl -s -c FX -m 60 -p 60 <p2ps PID> % prioctl -s -c FX -m 60 -p 60 <src_dist RRCPD PID> % prioctl -s -c FX -m 60 -p 60 <p2ps RRCPD PID></pre>
---	---	--

<p>src_dist – multiple instance tests with Solaris Container</p>	<p>CPU binding, interrupts disabling and CPU scheduling class assignment</p>	<p>A single src_dist and rrcp transport daemon pair was run per Solaris container/zone. Interrupts were disabled on the cores 1-7 when hyper-threading was enabled on Intel's Nehalem processor.</p> <p>For the tests up to 2 instances of src_dist, each src_dist instance process was bound to a processor set created out of a single core with its associated hyper-thread that is not shared with any other process on the system. Similarly, the higher runner threads of its associated transport were bound to a set of two cores processor set, different cores that was not shared with other processes on the system. So 2 cores with its associated hyper-threads on the system were used to the 2 instances of src_dist and 4 cores were used to run two instances of the rrcp transports; totaling to 6 cores. After starting the src_dist processes from the non-global zones(NGZ) the processor sets creation and the process binding to psets was done through the global zone (GZ). The "pbind" command was used from each NGZ to bind the relevant high runner threads of src_dist associated transport. With this tight process binding approach a very linear scalability was observed up to 2 instances of src_dist running on a single system.</p> <p>For the tests for 3 instances of src_dist a different processor binding approach was taken. In this case the some of the cores were shared between the src_dist associated rrcp transport. (NOTE: that's why we have seen some performance degradation for each src_dist instance in case of running 3 instances since some of the cores were shared between multiple processes).</p> <p>Each src_dist instance process was bound to a processor set created out of a single core with its associated hyper-thread that is not shared with any other process on the system. <i>Binding was chosen to assure that only one src_dist process was running on each core.</i> However, its associated rrcp transport were bound to a processor set created out of four cores with associated hyper-threads that were also shared with rrcp transport of other src_dist instances running in other zones on the same system. There was total 7 cores with associated hyper-threads used for this test and remaining one along with 1 hyper-thread were being left for the kernel and other processes on the system. The "pbind" command was used from each container/zone to bind the relevant high runner threads of src_dist associated rrcp transport.</p> <p>FX scheduling class was assigned to src_dist and rrcpd daemon processes: <pre>% priocntl -s -c FX -m 60 -p 60 <src_dist PIDs> % priocntl -s -c FX -m 60 -p 60 <src_dist RRCPD PIDs></pre></p>
---	--	---

<p>p2ps-LAN – multiple instances tests with Solaris Container</p>	<p>CPU binding and interrupts disabling and CPU scheduling class assignment</p>	<p>A single p2ps and rrcp transport daemon pair was run per Solaris container/zone. Interrupts were disabled on the cores 1-7 when hyper-threading was enabled on Intel's Nehalem processor.</p> <p>For the tests up to 2 instances of p2ps-LAN, each p2ps instance process was bound to a processor set created out of two cores that are not shared with any other process on the system. Similarly, the higher runner threads of its associated transport were bound to a set of two cores processor set, different cores that was not shared with other processes on the system. So 4 cores on the system were used to the 2 instances of p2ps and 4 cores were used to the two instances of the rrcp transports; totaling to 8 cores. The hyper-threads were used for the kernel and other processes on the system. After starting the p2ps processes from the non-global zones(NGZ) the processor sets creation and the process binding to psets was done through the global zone (GZ). The “pbind” command was used from each NGZ to bind the relevant high runner threads of p2ps associated transport. With this tight process binding approach a very linear scalability was observed up to 2 instances of p2ps running on a single system.</p> <p>For the tests for 3 instances of p2ps-LAN a different processor binding approach was taken. In this case the some of the cores were shared between the p2ps and its associated rrcp transports. (NOTE: that's why we have seen some performance degradation for each p2ps instance in case of running 3 instances since some of the cores were shared between multiple processes).</p> <p>Each p2ps-LAN instance process was bound to a processor set created out of a core and its associated hyper-thread that is not shared with any other process on the system. <i>Binding was chosen to assure that only one p2ps</i> process was running on each core and its associated hyper-thread. However, its associated rrcp transport were bound to a processor set created out of four cores that were also shared with rrcp transport of other src_dist instances running in other zones on the same system. There was total 7 cores and 3 hyper-threads used for this test and remaining one with 4 associated hyper-threads were being left for the kernel and other processes on the system. The “pbind” command was used from each container/zone to bind the relevant high runner threads of p2ps associated rrcp transport.</p> <p>FX scheduling class was assigned to p2ps and rrcpd daemon processes: <pre>% priocntl -s -c FX -m 60 -p 60 <p2ps PIDs> % priocntl -s -c FX -m 60 -p 60 <p2ps RRCPD PIDs></pre></p>
---	---	---

<p>p2ps-POP – multiple instances tests with Solaris Container</p>	<p>CPU binding and interrupts disabling and CPU scheduling class assignment</p>	<p>A single p2ps-POP instance was run per Solaris container/zone. Interrupts were disabled on the cores 2-7 when hyper-threading was enabled on Intel Nehalem processor.</p> <p>Since P2PS-POP does not require any transport to run hence there was only 2 cores per its instances required to achieve optimal results.</p> <p>For P2PS-POP 2 or 3 instances the similar processor binding approach was used. For the tests up to 3 instances of p2ps-POP, each p2ps instance process was bound to a processor set created out of two cores that are not shared with any other process on the system. So total 6 cores on the system were used to the 3 instances of p2ps-POP and remaining two cores with 8 hyper-threads were being left for the kernel and other processes on the system.</p> <p>After starting the p2ps-POP processes from the non-global zones(NGZ) the processor sets creation and the process binding to psets was done through the global zone (GZ). The “pbind” command was used from each NGZ to bind the relevant high runner threads of p2ps. With this tight process binding approach a very linear scalability was observed up to 3 instances of p2ps-POP running on a single system. There was still some head room to run more than 3 instances of p2ps-POP on this server.</p> <p>FX scheduling class was assigned to src_dist and rrcpd daemon processes: % priocntl -s -c FX -m 60 -p 60 <p2ps PIDs></p>
<p>End-to-End Latency when running src_dist and p2ps in Solaris Containers on the same system</p>	<p>CPU binding and interrupts disabling and CPU scheduling class assignment</p>	<p>Tuning techniques for src_dist and p2ps servers will remain same as described above when running both of these instances in Solaris containers on a single system. The following tuning is applied on the sink_driven_src and rmdstestclient server:</p> <pre>% psrset -c -F 4 % psrset -c -F 6 7 % psrset -f 1 % psrset -f 2 % psrset -e 1 ./sink_driven_src % psrset -e 2 ./rmdstestclient</pre> <p>FX scheduling class was assigned to sink_driven_src and rmdstestclient processes:</p> <pre>% priocntl -s -c FX -m 60 -p 60 `pgrep sink_driven_src` % priocntl -s -c FX -m 60 -p 60 `pgrep rmdstestclient`</pre> <p>Please see the section 2.3.4 and above notes for setting up the src_dist and P2PS for network latency.</p>

3 Detailed Results for RMDS with Solaris Containers Configuration (Multiple Instances)

Results for a single, 2, 4, and 6 simultaneously active containers(zones), each with a single *src_dist* or *p2ps* process are provided below:

3.1 RSSL/RWF Update Throughput

- All the throughput numbers quoted here are for Level 1 data.
- The data file used in these tests has 1 update, with an update (data, not including header) size of 74 bytes in RWF.
- All of the tests with no fan-out used 100,000 item watchlist.
- In most of the throughput tests the individual processes were bound to particular CPU(s).
- *sink_driven_src* and *rmdstestclient* were used as the publisher and consumer of data.
- In some Source Distributor tests, two P2PSs were used to create sufficient load.

3.1.1 Standalone Source Distributor and P2PS/LAN on a single Server Simultaneously

For the performance tuning details of the following tests please refer to the section 2.5:

Configuration Option	Transport	Max Throughput	Comments
Source Distributor (Cache Disabled)	RRCP	1,490,000 **	See section 2.5 for performance tuning details for <i>src_dist</i> with RRCP transport.
Source Distributor (Cache Enabled)	RRCP	640,000 **	See section 2.5 for performance tuning details for <i>src_dist</i> with RRCP transport.
P2PS-LAN (Cache Disabled)	RRCP	1,320,000 **	See section 2.5 for performance tuning details for <i>p2ps</i> with RRCP transport.
P2PS-LAN (Cache Enabled)	RRCP	620,000 **	See section 2.5 for performance tuning details for <i>p2ps</i> with RRCP transport.

** 3 cores were used for SRC_DIST/RRCPD and 4 cores were used for P2PS/RRCPD for a tight process binding.

3.1.2 Multiple Standalone Source Distributor in Solaris Containers Environment on a single Server

For the performance tuning details of the following tests please refer to the section 2.5:

Configuration Option	Transport	Max Throughput		
		1 zone	2 zones	3 zones
Cache Disabled	RRCP	1,480,000	1,440,000 ** / each instance 2,880,000 Aggregated	1,110,000 ** / each instance 3,330,000 Aggregated
Cache Enabled	RRCP	650,000	650,000 ** / each instance 1,300,000 Aggregated	540,000 ** / each instance 1,620,000 Aggregated

** 4 cores per SRC_DIST/RRCPD instance were being used for first two instances for a tight process binding. Very linear scalability was observed up to 2 instances of SRC_DIST/RRCPD in Solaris container environment. For the three instances of P2PS a different core binding approach was used with some sharing of cores with multiple processes. The Nehalem hyper-threading didn't help much for scaling beyond 3 instances of SRC_DIST/RRCPD in its associated zones.

3.1.3 Multiple P2PS/LAN in Solaris Containers Environment on a single Server

For the performance tuning details of the following tests please refer to the section 2.5:

Config. Option	Mounts : Commonality	Transport	Max Throughput		
			1 zone	2 zones	3 zones
Cache Disabled	No fan-out	RRCP	1,320,000	1,305,000 ** / each instance 2,610,000 Aggregated	1,020,000 ** / each instance 3,060,000 Aggregated
Cache Enabled	No fan-out	RRCP	620,000	620,000 ** / each instance 1,240,000 Aggregated	500,000 ** / each instance 1,500,000 Aggregated
Cache Disabled	100 mounts; Producer 50/50	RRCP	70,000 input 3,535,000 output	69,500 input ** 3,509.750 output / each instance 7,019,500 Aggregated	56,900 input ** 2,873,450 output / each instance 8,620,350 Aggregated

** 4 cores per P2PS/RRCPD instance were being used for first two instances for a tight process binding. Up to 2 instances of P2PS/RRCPD scales very linearly in Solaris container environment. For the three instances of P2PS no tight core binding was used. The Nehalem hyper-threading didn't help much for scaling beyond 3 instances of P2PS/RRCPD in its associated zones.

3.2 Update Throughput via Multiple P2PS/POP Solaris Containers Environment on a single Server

These tests were performed using a P2PS/POP with data provided by an upstream P2PS/LAN, with an RRCP transport. For the performance tuning details of the following tests please refer to the section 2.5:

3.2.1 RSSL/RWF

Configuration Option	Mounts : Commonality	Max Throughput		
		1 zone	2 zones	3 zones
Cache Enabled	No fan-out	720,000	710,000 ** / each instance 1,420,000 Aggregated	700,000 ** / each instance 2,100,000 Aggregated
Cache Enabled	100 mounts; Producer 50/50	70,000 input 3,535,000 output	69,800 input ** 3,524,900 output / each instance 7,049,800 Aggregated	69,500 input ** 3,509,750 output / each instance 10,529,250 Aggregated

** 2 cores per P2PS-POP instance were being used to bind its hot threads and it was able to scale up to three instances pretty linearly. **There was still some head room left to run more than 3 instances of p2ps-POP on this server as only 6 cores out of 8 cores and 8 hyper-threads were used for three instances.**

3.3 End-to-End RSSL/RWF Latency

Latency is defined as the time for a data item to propagate through one or more RMDS components. The following latency numbers are captured when **src_dist** and **p2ps** were running on a single server in their associated Solaris containers/zones.

The “End to end” 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 both the latency contribution from the API and the core infrastructure components.

- Caching was disabled in both the Source Distributor and the P2PS during these tests.
- Optimized binaries of the RMDS infrastructure components were used.
- NTP was disabled on the tools node, as any drifts in time will affect the reported latency.
- Tests were run with 100,000 item watchlist and RWF data update size of 74 bytes [Data file (**sample.xml**) was used].
- Latency tests were run at each update rate for at least 5 minutes, up to the maximum sustainable update rate for a given setup.
- Decode of data was turned on in these tests.

3.3.1 RRCP Backbone Results When Src_dist and P2PS are installed in two Containers

Update Rate [74-byte RWF messages]	Mean Latency (millisec)	Std Deviation (millisec)	Maximum Latency (millisec)	Minimum Latency (millisec)	Number of Latency Points
1,000	0.139	0.002	0.156	0.118	3150
5,000	0.174	0.003	0.193	0.152	3150
10,000	0.229	0.006	0.329	0.197	3150
20,000	0.280	0.021	0.359	0.224	3130
30,000	0.373	0.009	0.544	0.334	3150
40,000	0.377	0.012	0.652	0.309	3150
50,000	0.387	0.015	0.694	0.340	3130
60,000	0.386	0.014	0.730	0.304	3150
70,000	0.390	0.023	0.870	0.302	3150
80,000	0.391	0.022	0.935	0.330	3150
90,000	0.397	0.041	1.149	0.325	3150
100,000	0.396	0.035	1.104	0.341	3130
150,000	0.430	0.094	1.716	0.326	3130
200,000	0.462	0.065	1.029	0.354	3140
250,000	0.488	0.092	1.296	0.354	3140
300,000	0.517	0.110	1.398	0.344	3080
350,000	0.553	0.129	1.400	0.326	3130
400,000	0.549	0.138	1.723	0.320	3100
450,000	0.602	0.177	1.969	0.322	3120
500,000	0.666	0.217	2.221	0.351	3100
550,000	0.675	0.240	2.296	0.330	3130
600,000	0.720	0.275	2.710	0.317	3120
650,000	0.766	0.321	2.959	0.325	3110
700,000	0.789	0.337	3.078	0.316	3136
750,000	0.808	0.338	2.943	0.326	3130
800,000	0.827	0.363	3.026	0.302	3100
850,000	0.845	0.387	3.150	0.335	3090
900,000	0.981	0.471	3.419	0.318	3110
950,000	1.102	0.571	3.553	0.263	3110
1,000,000	1.140	0.664	4.903	0.280	3130
1,050,000	2.255	0.809	4.931	0.396	3120