



# Reuters Market Data System

## RMDS 6.0

### Performance Test Results on Sun SPARC Enterprise T5120 with and without Solaris Containers (1GbE)

**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 1.0

Date of Issue: January 11, 2008

Amjad.Khan@sun.com

## Contents

1 General Information .....	4
1.1 Objective.....	4
1.1.1 Results Summary for Standard (“Traditional”) RMDS Configuration.....	4
1.1.2 Results Summary for RMDS with Solaris Containers Configuration.....	5
1.2 Sun Technologies Used.....	5
1.2.1 Sun SPARC Enterprise T5120 Server .....	5
1.2.2 Solaris Containers.....	5
1.3 Testing Methodology.....	6
1.4 Software Versions.....	7
1.4.1 RMDS.....	7
1.4.2 RMDS Test Tool.....	7
1.4.3 Operating Systems.....	7
1.5 Hardware.....	8
2 Preparation for Performance Test.....	8
2.1 Network.....	8
2.2 Hardware.....	8
2.3 Operating System Configuration.....	8
2.3.1 Solaris Kernel Parameters.....	8
2.3.2 TCP and UDP Buffers.....	9
2.3.3 Intel NIC (e1000g) Driver Settings.....	9
2.3.4 Additional Solaris Kernel and e1000g NIC Parameters for Network Latency.....	9
2.4 RMDS Configuration.....	9
2.5 Miscellaneous Notes.....	10
2.5.1 RMDS Traditional Configuration Tuning Notes.....	11
2.5.2 RMDS with Solaris Containers Configuration (Multiple Instances) Tuning Notes.....	12
3 Detailed Results for RMDS Traditional Configuration.....	14
3.1 RSSL/RWF Update Throughput.....	14
3.1.1 Standalone Source Distributor.....	14
3.1.2 P2PS/LAN.....	14
3.2 RSSL/RWF Update Throughput.....	15
3.2.1 RSSL/RWF.....	15
3.3 End-to-End RSSL/RWF Latency.....	16
3.3.1 RRCP Backbone Results.....	16
3.3.2 Rendezvous Backbone Results.....	17
4 Detailed Results for RMDS with Solaris Containers Configuration (Multiple Instances).....	18
4.1 RSSL/RWF Update Throughput.....	18
4.1.1 Standalone Source Distributor.....	18
4.1.2 P2PS/LAN.....	19
4.2 Update Throughput via P2PS/POP.....	20
4.2.1 RSSL/RWF.....	20

© Sun Microsystems, Inc. 2008. 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 6.0 on Sun SPARC Enterprise T5120 Server running with and without Solaris 10 containers. The test procedures are described in Reuters RMDS 6.0 Performance Test Procedures and Results document.

Two separate set of test runs were made, one using a standard (“Traditional”) RMDS benchmark procedure defined in “ Reuters RMDS 6.0 Performance Test Procedures and Results” document without any Solaris containers on T5120 and second using the exact same software and operating system, but running multiple instances of RMDS applications, Source Distributor (*src\_dist*) and Point-to-Pont Server (*p2ps*), on the same hardware using Solaris 10 containers, a virtualization technology. In the second set of tests, the standard test sequence was extended to demonstrate the T5120’s multi-threaded architecture.

The goal of these tests is to measure throughput and latency through RMDS 6.0 infrastructure components, specifically the Point-to-Pont Server (*p2ps*) and Source Distributor(*src\_dist*) on Sun SPARC Enterprise T5120 server with and without virtualization technology(Solaris Containers). The tests are grouped into two main categories:

- **RMDS Traditional Configuration**
  - Update throughput using RSSL/RWF data (see 3)
  - Update throughput via P2PS/POP (see 3)
  - End-to-end RSSL/RWF latency using embedded time-stamp (see 3)
- **RMDS with Solaris Containers Configuration (Multiple Instances)**
  - Aggregated update throughput via multiple instances of *src\_dist* or *p2ps* using RSSL/RWF data (see 4)
  - Aggregated update throughput via multiple instances of P2PS/POP (see 4)

#### 1.1.1 Results Summary for Standard (“Traditional”) RMDS Configuration

RMDS Components Configuration	RMDS Throughput Using RMDS Traditional Configuration on Ethernet Infrastructure
<i>P2PS-LAN, RRCP, Producer 50/50 Fanout (Cache Disabled)</i>	262,600 updates/sec
<i>P2PS-LAN, RVD, Producer 50/50 Fanout (Cache Disabled)</i>	277,750 updates/sec
<i>P2PS-POP, Producer 50/50 Fanout (Cache Enabled)</i>	257,550 updates/sec
<i>Src_dist, RRCP (Cache Disabled)</i>	126,000 updates/sec
<i>Src_dist, RVD (Cache Disabled)</i>	120,000 updates/sec
<i>P2PS-LAN, RRCP No Fanout (Cache Disabled)</i>	152,000 updates/sec
<i>P2PS-LAN, RVD No Fanout (Cache Disabled)</i>	130,000 updates/sec
<i>P2PS-POP, No Fanout (Cache Enabled)</i>	74,500 updates/sec

## 1.1.2 Results Summary for RMDS with Solaris Containers Configuration

RMDS Components Configuration	RMDS Aggregated Throughput Using Solaris containers on Ethernet Infrastructure
<i>P2PS-LAN, RRCP, Producer 50/50 Fanout (Cache Disabled)</i>	1,333,200 updates/sec
<i>P2PS-LAN, RVD, Producer 50/50 Fanout (Cache Disabled)</i>	1,363,500 updates/sec
<i>P2PS-POP, Producer 50/50 Fanout (Cache Enabled)</i>	1,333,200 updates/sec
<i>Src_dist, RRCP (Cache Disabled)</i>	630,000 updates/sec
<i>Src_dist, RVD (Cache Disabled)</i>	600,000 updates/sec
<i>P2PS-LAN, RRCP No Fanout (Cache Disabled)</i>	720000 updates/sec
<i>P2PS-LAN, RVD No Fanout (Cache Disabled)</i>	660,000 updates/sec
<i>P2PS-POP, No Fanout (Cache Enabled)</i>	402,000 updates/sec

## 1.2 Sun Technologies Used

### 1.2.1 Sun SPARC Enterprise T5120 Server

The T5120 is a world's first 64-thread, general purpose server powered by the UltraSPARC T2 processor that has 8 cores, each of which can support (in the hardware) 8 strands, so that the hardware appears to the operating system to have 64 processors. The term "strand" is used to avoid confusion with a software thread. A multi-threaded software application will use one or more hardware strands simultaneously during execution. In the extreme case, each thread will run on a separate strand. Details of Sun SPARC Enterprise T5120 and UltraSPARC T2 can be found at:

<http://www.sun.com/servers/coolthreads/t5120/> and

<http://www.sun.com/processors/UltraSPARC-T2/>

### 1.2.2 Solaris Containers

Solaris Containers were introduced with the Solaris 10 OS, and are available on all platforms that the Solaris 10 OS is supported on. Solaris Containers 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 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](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 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 6 Solaris containers(zones) were created, which allowed 6 independent **src\_dist** or **p2ps** applications to be run simultaneously within its own container on a single T5120. Each

*src\_dist* or *p2ps* instance was paired with a transport daemon, either *rrcpd* or *rvd*, running with it within each containers(zone).

Solaris Containers allow multiple instances of RMDS software to run on the T5120. Tests were run with 1, 2, 4, and 6 instances of the RMDS software running on the T5120, each instance in its own container, with results showing excellent scaling up to 6 instances of *src\_dist* or *p2ps* on a single T5120 system. No modifications were required to RMDS software to account for the use of a Solaris containers as opposed to a separate system.

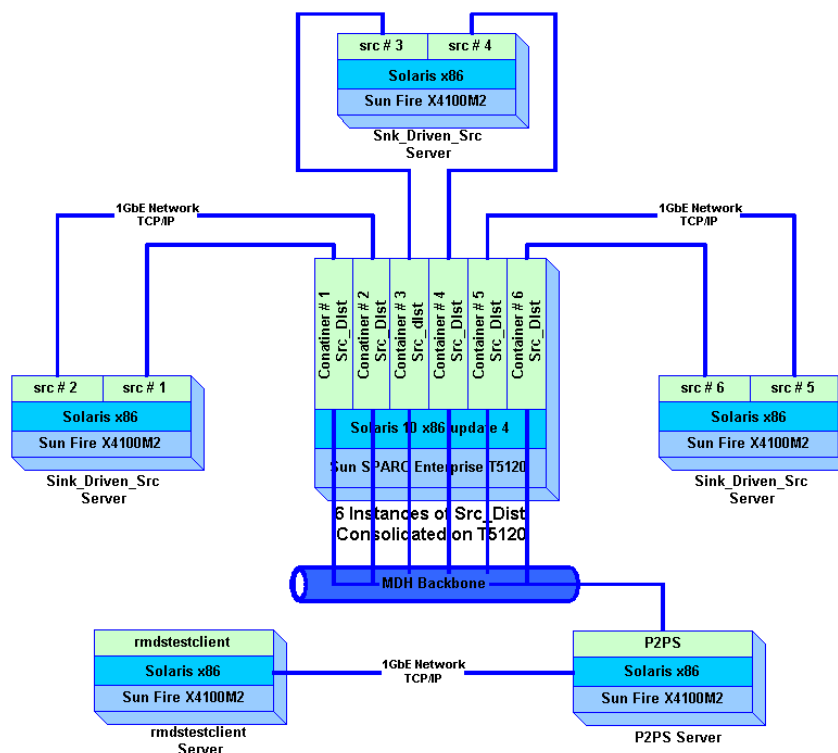
### 1.3 Testing Methodology

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.

For the RMDS multiple instances testing with Solaris containers, single, 2, 4, and 6 simultaneously active containers(zones) were installed on T5120 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:

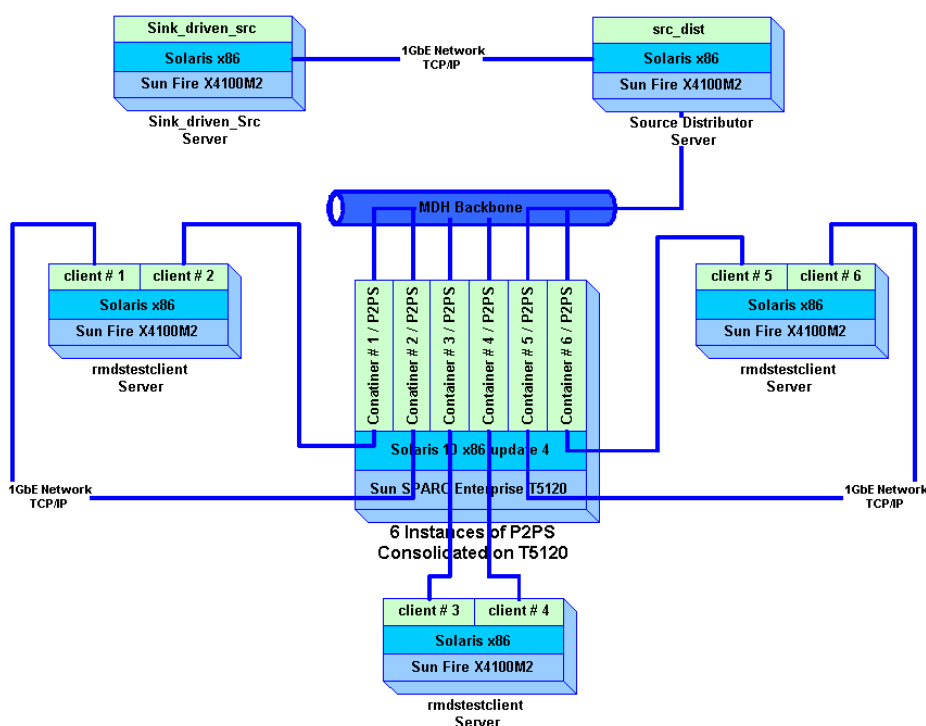
## RMDS 6.0 with Solaris Containers Config # 1

Up to 6 x SRC\_DIST Instances on a Single T5120



## RMDS 6.0 with Solaris Containers Config # 2

Up to 6 x P2PS Instances on a Single T5120



The embedded time-stamp approach was used to calculate end-to-end latency for Level 1 (Quotes and Trades) data. RMDS 6.0 end-to-end update latency is measured by using *sink\_driven\_src* as the publisher and *rmdstestclient* as the subscriber.

In the embedded time-stamp 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.

### 1.4 Software Versions

#### 1.4.1 RMDS

*src\_dist* ver. mdh6.0.2.L2  
*p2ps* ver. p2ps6.0.2.L2  
*rscp* as included in p2ps6.0.2.L2  
*rvd* 7.4.19

#### 1.4.2 RMDS Test Tool

*sink\_driven\_src* (from MDH load above)  
*rmdstestclient* (from P2PS load above)

#### 1.4.3 Operating Systems

*Solaris 10 8/07* (update 4)

## 1.5 Hardware

**Sun SPARC Enterprise T5120 Server** – 1.4GHz UltraSPARC T2 processor. The Sun SPARC Enterprise T5120 server is the first mainstream server to offer 64 compute threads in a single socket. With just one chip, it delivers record-breaking performance, maximum energy efficiency, and the industry's best performance per watt.

**Sun Fire X4100 M2** – 2 X 3.0GHz dual-core AMD Opteron 2222 processors (total 4 cores)

**Cisco Catalyst 4948** 1 Gbps Network Switch

## 2 Preparation for Performance Test

### 2.1 Network

All the performance tests were run where the machines were connected to a private network via 1 Gbps switch (Cisco Catalyst 4948). All the network cards and switch ports were set to Auto Negotiate. For testing purposes the e1000g0, e1000g1 and e1000g2 interfaces were used only.

Each zone configured on the T5120 was given 3 virtual IP addresses that corresponded to the 3 primary NICs used in standard single instance testing. 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.

### 2.2 Hardware

Due to the availability of only one T5120, the primary RMDS component (**src\_dist** or **p2ps**) in each throughput test was run on the T5120 while other components including the test infrastructure were run on Sun x4100m2 systems equipped with 3.0GHz AMD Opteron processors. Test results for **src\_dist** and **p2ps** were configured with the **src\_dist/p2ps** and transport daemon pair(s) running in each container on the T5120. On all of the system the Intel (e1000g) interfaces were used for MDH backbone and TCP network traffic.

### 2.3 Operating System Configuration

The Solaris 10 08/07 (update 4) was installed on all systems. The e1000g network driver was used for all 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](#).

The RMDS 6.0.2 software used in this testing is the standard Reuters distribution for Sun SPARC/Solaris. As such it was compiled with optimization specific to the UltraSPARC IIIi processor, not to the UltraSPARC T2. Some performance gains may be realizable with a custom compile for the T2 architecture.

#### 2.3.1 Solaris Kernel Parameters

Any settings changed from the defaults are noted below:

Step	Procedure		System File
1	OS	Enter the following lines in system file noted	/etc/system
	Solaris	set dld:dld_opt=2 (for throughput and latency) 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	<b>OS</b>	<b>Enter the following lines in system file noted</b>	<b>System File</b>
	Solaris	/usr/sbin/ndd -set /dev/tcp tcp_max_buf 4194204	/lib/svc/method/net-init
		/usr/sbin/ndd -set /dev/tcp tcp_recv_hiwat 1048576	
		/usr/sbin/ndd -set /dev/tcp tcp_xmit_hiwat 1048576	
		/usr/sbin/ndd -set /dev/udp udp_max_buf 4194204	
		/usr/sbin/ndd -set /dev/udp udp_xmit_hiwat 524288	
/usr/sbin/ndd -set /dev/udp udp_recv_hiwat 1048576			

### 2.3.3 Intel NIC (e1000g) Driver Settings

Any settings changed from the defaults are noted below:

Step	Procedure		
3	<b>OS</b>	<b>Enter the following lines in the file noted</b>	<b>System File</b>
	Solaris e1000g	/usr/sbin/ndd -set /dev/e1000g0 tx_interrupt_enable 0	/lib/svc/method/net-init
		/usr/sbin/ndd -set /dev/e1000g1 tx_interrupt_enable 0	
/usr/sbin/ndd -set /dev/e1000g2 tx_interrupt_enable 0			

### 2.3.4 Additional Solaris Kernel and e1000g NIC Parameters for Network Latency

Step	Procedure		
4	<b>OS</b>	<b>Enter the following lines in the file noted</b>	<b>System File</b>
	Solaris e1000g driver	set autoup=300	/etc/system
		/usr/sbin/ndd -set /dev/e1000g0 rx_bcopy_threshold 128	/lib/svc/method/net-init
		/usr/sbin/ndd -set /dev/e1000g1 rx_bcopy_threshold 128	
		Add the following two entries in the /kernel/drv/e1000g.conf file on Solaris 10 u4 and reboot the system:	/kernel/drv/e1000g.conf
intr_adaptive=0,0,0,0,0,0,0,0; intr_throttling_rate=0,0,0,0,0,0,0,0;			

## 2.4 RMDS 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>\$RMDS_SW/config</i> <i>on mdh and p2ps/LAN systems</i>
<i>pop.cnf.template</i>	Configuration file to do P2PS/POP test	<i>\$RMDS_SW/config</i> <i>on p2ps/POP system only</i>

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

Config File	Parameter	Value
<i>rmads.cnf</i>	*RRCP*udpRecvBufSize	4096
<i>rmads.cnf</i>	*RRCP*udpSendBufSize	4096
<i>rmads.cnf</i>	*RDFD*readBias	60 (add this entry)
<i>rmads.cnf</i>	*p2ps*maxOutputBuffers	1000
<i>rmads.cnf</i>	*p2ps*guaranteedOutputBuffers	800

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

Config File	Parameter	Value
<i>rmads.cnf</i>	*IDN_RDF*rrmpFlushInterval	0
<i>rmads.cnf</i>	*RDFD*rrmpFlushInterval	0
<i>rmads.cnf</i>	*p2ps*tcpNoDelay	True
<i>rmads.cnf</i>	*p2ps*timedWrites	False
<i>rmads.cnf</i>	*p2ps*flushInterval	0

## 2.5 Miscellaneous Notes

Any other significant deviations from the standard test procedures, or clarifications, are noted below (such as number/type of machines used, CPU binding policy, etc.):

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. When running with **rwd** in a LAN configuration, thread 7 (of 8) is the high runner. 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 **rwd** transport.

There were two different approaches were taken for the processor binding during these tests. For the RMDS traditional testing without any containers, a very tight processor binding was used since there is more flexibility without containers to create the processor set and binding of any processes to a processor set.

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”.

## 2.5.1 RMDS Traditional Configuration Tuning Notes

In a traditional RMDS configuration another core was used for the transport daemon (*rrcpd* or *rvd*). Bindings of threads to specific processor strands were done using a processor set created of a whole core (8 strands) for each *src\_dist* or *p2ps* instance. One strand out of this processor set was used for the high runner thread of the *src\_dist* or *p2ps*. In case of *p2ps*, the second, third and fourth strands of the same core was used for the other *p2ps* active threads.

Test	Deviation	Comments
<i>src_dist</i> – single instance tests (without Container)	CPU binding and interrupts disabling and CPU scheduling class assignment	<p>A processor set was created out of the core #2 (CPU strands 8,9, 10,11,12,13,14,15) and the high runner thread of <i>src_dist</i> was bound to this processor set. Another processor set was created out of core #2 (CPU strands 16, 17, 18, 19, 20, 21, 22, 23) and the respective transport daemon (<i>rrcpd</i> or <i>rvd</i>) was bound to this processor. Further more the high runner threads of transport was bound to CPU strands 16 and 18 within that processor set to take advantage of its multithreaded capability. Interrupts were disabled on any CPU core that was running an application thread.</p> <p>FX scheduling class was assigned to <i>src_dist</i> and <i>rrcpd</i> or <i>rvd</i> daemon processes:</p> <pre>% priocntl -s -c FX -m 60 -p 60 `pgrep src_dist` % priocntl -s -c FX -m 60 -p 60 `pgrep rrcpd` or % priocntl -s -c FX -m 60 -p 60 `pgrep rvd`</pre>
<i>p2ps(LAN/POP)</i> – single instance tests (without Container)	CPU binding and interrupts disabling and CPU scheduling class assignment	<p>A processor set was created out of the core #2 (CPU strands 8,9, 10,11,12,13,14,15) and the <i>p2ps</i> process was bound to this processor set. The high runner thread of <i>p2ps</i> was bound to CPU strand 8 and others threads were bound to other strands available in the same processor set. Another processor set was created out of core #2 (CPU strands 16, 17, 18, 19, 20, 21, 22, 23) and the respective transport daemon (<i>rrcpd</i> or <i>rvd</i>) was bound to this processor. Further more the high runner threads of transport was bound to CPU strands 16 and 18 within that processor set to take advantage of its multithreaded capability. Interrupts were disabled on any CPU core that was running an application thread.</p> <p>FX scheduling class was assigned to <i>src_dist</i> and <i>rrcpd</i> or <i>rvd</i> daemon processes:</p> <pre>% priocntl -s -c FX -m 60 -p 60 `pgrep p2ps` % priocntl -s -c FX -m 60 -p 60 `pgrep rrcpd` or % priocntl -s -c FX -m 60 -p 60 `pgrep rvd`</pre>
End-to-End Latency (without Container)	CPU binding and interrupts disabling and CPU scheduling class assignment	Please see the section 2.3.4 and above notes for setting up the <i>src_dist</i> and P2PS for network latency.

## 2.5.2 RMDS with Solaris Containers Configuration (Multiple Instances) Tuning Notes

For the RMDS testing with Solaris container there was no processor set used. All of the high runner threads of *src\_dist* or *p2ps* processes were bound within a specific core as mentioned below. **Since we were not able to use the processor set and tight processor binding in Solaris container environment hence the RMDS single instance throughput performance without container is little better than the same single instance test running in Solaris containers.** In multiple containers tests a single core was also shared between two instances of transports or applications.

Test	Deviation	Comments
<i>src_dist</i> – multiple instance tests (with Container)	CPU binding and interrupts disabling and CPU scheduling class assignment	<p>Interrupts were disabled on all strands in cores 1-7 (CPU strands 8 to 63). A single <i>src_dist</i> and transport daemon pair was run per container.</p> <p>For the tests up to 4 instances of <i>src_dist</i>, each <i>src_dist</i> instance high runner thread was bound to a strand of a core that is not shared with any other process on the system. Similarly, the higher runner threads of its associated transport were bound to first 4 strands of a different core that was also shared with transport of another <i>src_dist</i> instance running in another container and sharing the remaining last 4 strands of the same core. So 4 cores on the system were bound to the 4 instances of <i>src_dist</i> and 2 cores were shared between 4 instances of the transports; totalling to 6 cores. The “<i>pbind</i>” command was used from each container to bind the relevant high runner threads of <i>src_dist</i> and its associated transport. Bindings were chosen to assure that only one <i>src_dist</i> high runner thread was running on each core of the T5120.</p> <p>For the tests for 6 instances of <i>src_dist</i> a different processor binding approach was taken. In this case the each core was shared between the <i>src_dist</i> and its associated transport. <b>(NOTE: that's why we have seen some performance degradation for each <i>src_dist</i> instance in case of running 6 instances since the single core was shared between two processes).</b></p> <p>Each <i>src_dist</i> instance high runner thread was bound to a first strand of the core and the higher runner threads of its associated transport were bound to remaining 7 strands of the same core. There was total 6 cores used for this test and remaining two were being left for the kernel and other processes on the system. The “<i>pbind</i>” command was used from each container to bind the relevant high runner threads of <i>src_dist</i> and its associated transport.</p> <p>FX scheduling class was assigned to <i>src_dist</i> and <i>rrcpd</i> or <i>rvd</i> daemon processes for all the tests:  <pre>% priocntl -s -c FX -m 60 -p 60 `pgrep src_dist` % priocntl -s -c FX -m 60 -p 60 `pgrep rrcpd` or % priocntl -s -c FX -m 60 -p 60 `pgrep rvd`</pre> </p>

<p><b>p2ps(LAN/POP) – multiple instances tests (with Container)</b></p>	<p>CPU binding and interrupts disabling and CPU scheduling class assignment</p>	<p>Interrupts were disabled on all strands in cores 1-7 (CPU strands 8 to 63). A single <b>p2ps</b> and transport daemon pair was run per container.</p> <p>For the tests up to 4 instances of <b>p2ps</b>, each <b>p2ps</b> instance high runner threads were bound to first 4 strands of a core that is not shared with any other process on the system. Similarly, the higher runner threads of its associated transport were bound to first 4 strands of a different core that was also shared with transport of another <b>p2ps</b> instance running in another container and sharing the remaining last 4 strands of the same core. So 4 cores on the system were bound to the 4 instances of <b>p2ps</b> and 2 cores were shared between 4 instances of the corresponding transports; totalling to 6 cores. The “<b>pbind</b>” command was used from each container to bind the relevant high runner threads of <b>p2ps</b> and its associated transport. Bindings were chosen to assure that only <b>p2ps</b> high runner threads were running on each core of the T5120.</p> <p>For the tests for 6 instances of <b>p2ps</b> a different processor binding approach was taken. In this case the each core was shared between the <b>p2ps</b> and its associated transport. <b>(NOTE: that's why we have seen some performance degradation for each p2ps instance in case of running 6 instances since the single core was shared between two processes)</b>.</p> <p>Each <b>p2ps</b> instance high runner threads were bound to first 4 strands of the core and the higher runner threads of its associated transport were bound to remaining last 4 strands of the same core. There was total 6 cores used for this test and remaining two were being left alone for the kernel and other processes on the system. The “<b>pbind</b>” command was used from each container to bind the relevant high runner threads of <b>p2ps</b> and its associated transport.</p> <p>FX scheduling class was assigned to <b>p2ps</b> and <b>rrcpd</b> or <b>rvd</b> daemon processes for all the tests:</p> <pre>% priocntl -s -c FX -m 60 -p 60 `pgrep src_dist` % priocntl -s -c FX -m 60 -p 60 `pgrep rrcpd` or % priocntl -s -c FX -m 60 -p 60 `pgrep rvd`</pre>
---	---	---

### 3 Detailed Results for RMDS Traditional Configuration

Results for a single *src\_dist* and *p2ps* without any containers, as is typically tested, 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

Configuration Option	Transport	Max Throughput	Comments
Cache Disabled	RRCP	126,000	See section 2.5.1 for performance tuning details for <i>src_dist</i> with RRCP transport.
Cache Enabled	RRCP	67,000	See section 2.5.1 for performance tuning details for <i>src_dist</i> with RRCP transport.
Cache Disabled	Rendezvous	120,000	See section 2.5.1 for performance tuning details for <i>src_dist</i> with Tibco RV transport.
Cache Enabled	Rendezvous	66,000	See section 2.5.1 for performance tuning details for <i>src_dist</i> with Tibco RV transport.

##### 3.1.2 P2PS/LAN

Configuration Option	Mounts : Commonality	Transport	Max Throughput	Comments
Cache Disabled	No fan-out	RRCP	150,000	See section 2.5.1 for performance tuning details for P2PS with RRCP transport.
Cache Enabled	No fan-out	RRCP	78,000	See section 2.5 for performance tuning details for P2PS with RRCP transport.
Cache Disabled	100 mounts; Producer 50/50	RRCP	5,200 input 262,600 output	See section 2.5.1 for performance tuning details for P2PS with RRCP transport.

Cache Disabled	No fan-out	Rendezvous	130,000	See section 2.5.1 for performance tuning details for P2PS with Tibco RV transport.
Cache Enabled	No fan-out	Rendezvous	72,000	See section 2.5.1 for performance tuning details for P2PS with Tibco RV transport.
Cache Disabled	100 mounts; Producer 50/50	Rendezvous	5,400 input 277,750 output	See section 2.5.1 for performance tuning details for P2PS with Tibco RV transport.

## 3.2 RSSL/RWF Update Throughput

These tests were performed using a P2PS/POP with data provided by an upstream P2PS/LAN, with an RRCP transport.

### 3.2.1 RSSL/RWF

Configuration Option	Mounts : Commonality	Max Throughput	Comments
Cache Enabled	No fan-out	74,500	See section 2.5.1 for performance tuning details for p2ps/POP without any transport.
Cache Enabled	100 mounts; Producer 50/50	5,100 input 257,550 output	See section 2.5.1 for performance tuning details for p2ps/POP without any transport.

### 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. “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.

NOTES:

- 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

Update Rate [74-byte RWF messages]	Mean Latency (millisec)	Std Deviation (millisec)	Maximum Latency (millisec)	Minimum Latency (millisec)	Number of Latency Points
1,000	0.565	0.041	0.675	0.483	2730
5,000	0.793	0.043	0.946	0.703	2764
10,000	1.000	0.047	1.289	0.897	2760
20,000	1.188	0.081	2.097	0.931	2800
30,000	1.309	0.140	2.644	0.961	2730
40,000	1.427	0.204	3.337	0.974	2720
50,000	1.534	0.289	3.971	1.012	2770
60,000	1.642	0.381	4.690	1.000	2720
70,000	1.820	0.486	5.265	1.104	2720
80,000	1.900	0.633	5.961	1.046	2730
90,000	2.159	1.024	10.776	1.030	2730
100,000	2.601	1.549	13.247	1.028	2750

### 3.3.2 Rendezvous Backbone Results

Update Rate [74-byte RWF messages]	Mean Latency (millisec)	Std Deviation (millisec)	Maximum Latency (millisec)	Minimum Latency (millisec)	Number of Latency Points
1,000	0.621	0.045	1.305	0.539	2730
5,000	0.844	0.137	4.998	0.750	2760
10,000	1.083	0.135	4.664	0.984	2720
20,000	1.314	0.604	10.947	0.979	2720
30,000	1.522	0.703	11.082	1.026	2720
40,000	1.695	0.947	11.292	1.057	2730
50,000	1.932	1.230	12.678	1.059	2810
60,000	4.841	2.450	16.785	1.050	2730
70,000	5.270	2.380	17.941	1.095	2730
80,000	6.187	2.170	17.262	1.465	2740
90,000	7.514	2.988	23.796	2.586	2770

## 4 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:

### 4.1 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.

#### 4.1.1 Standalone Source Distributor

For the performance tuning details of the following tests please refer to the section 2.5.2:

Configuration Option	Transport	Max Throughput			
		1 zone	2 zones	4 zones	6 zones
Cache Disabled	RRCP	124,000	120,000	114,000	105,000 **
			240,000 Aggregate	456,000 Aggregate	630,000 Aggregate
Cache Enabled	RRCP	67,000	67,000	65,000	61,000
			134,000 Aggregate	260,000 Aggregate	366,000 Aggregate
Cache Disabled	Rendezvous	116,000	115,000	110,000	100,000 **
			232,000 Aggregate	440,000 Aggregate	600,000 Aggregate
Cache Enabled	Rendezvous	66,000	66,000	64,000	60,000
			132,000 Aggregate	256,000 Aggregate	360,000 Aggregate

\*\* The RRCP and RVD at the P2PS server was becoming the bottleneck. Otherwise there was about 8% cpu idle at the *src\_dist* in each zone.

#### 4.1.2 P2PS/LAN

For the performance tuning details of the following tests please refer to the section 2.5.2:

Config. Option	Mounts : Commonality	Transport	Max Throughput			
			1 zone	2 zones	4 zones	6 zones
Cache Disabled	No fan-out	RRCP	148,000	147,000 294,000 Aggregate	138,000 552,000 Aggregate	120,000 720,000 Aggregate
Cache Enabled	No fan-out	RRCP	78,000	76,000 152,000 Aggregate	74,000 296,000 Aggregate	68,000 408,000 Aggregate
Cache Disabled	100 mounts; Producer 50/50	RRCP	5,000 input 252,500 output	4,800 input 242,400 output 484,800 Aggregate	4,700 input 237,350 output 949,400 Aggregate	4,400 input 222,200 output 1,333,200 Aggregate
Cache Disabled	No fan-out	Rendezvous	126,000	122,000 244,000 Aggregate	118,000 472,000 Aggregate	110,000 660,000 Aggregate
Cache Enabled	No fan-out	Rendezvous	70,000	69,000 138,000 Aggregate	67,000 268,000 Aggregate	65,000 390,000 Aggregate
Cache Disabled	100 mounts; Producer 50/50	Rendezvous	5,200 input 277,750 output	5,000 input 252,500 output 505,000 Aggregate	4,900 input 247,450 output 989,800 Aggregate	4,500 input 227,250 output 1,363,500 Aggregate

## 4.2 Update Throughput via P2PS/POP

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 2.5.2:

### 4.2.1 RSSL/RWF

Configuration Option	Mounts : Commonality	Max Throughput			
		1 zone	2 zones	4 zones	6 zones
Cache Enabled	No fan-out	72,000	72,000	70,000	67,000
			144,000 Aggregate	280,000 Aggregate	402,000 Aggregate
Cache Enabled	100 mounts; Producer 50/50	5,000 input 252,500 output	4,900 input 247,450 output 494,900 Aggregate	4,800 input 242,400 output 969,600 Aggregate	4,400 input 222,200 output 1,333,200 Aggregate