SPC BENCHMARK 1™
FULL DISCLOSURE REPORT

NETAPP, INC.

NETAPP FAS6240 *(CLUSTER)*

SPC-1 V1.12

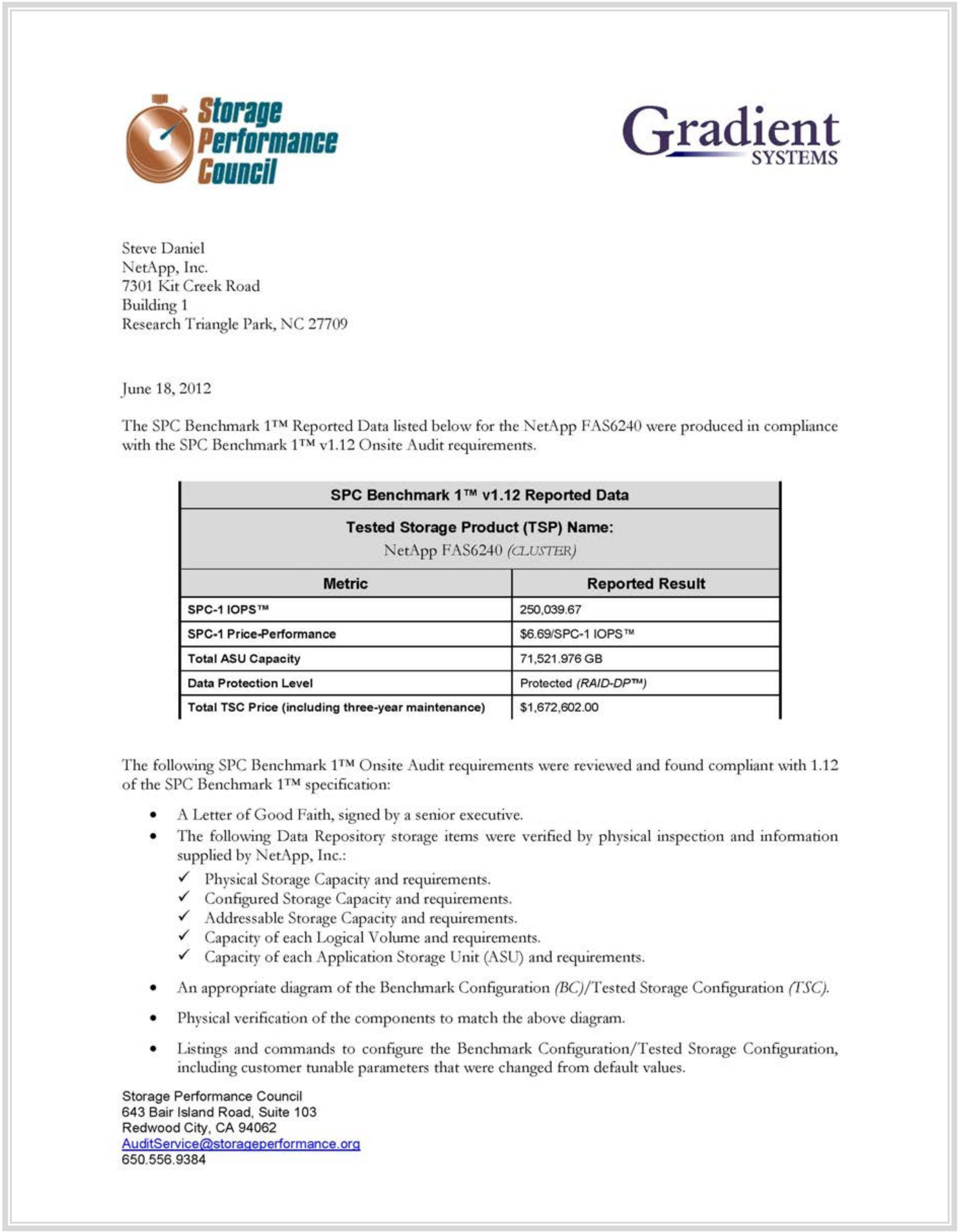
Submitted for Review: June 18, 2012
Submission Identifier: A00115

First Edition – June 2012

Trademarks

Table of Contents

AUDIT CERTIFICATION



Steve Daniel
NetApp, Inc.
7301 Kit Creek Road
Building 1
Research Triangle Park, NC 27709

June 18, 2012

The SPC Benchmark 1™ Reported Data listed below for the NetApp FAS6240 were produced in compliance with the SPC Benchmark 1™ v1.12 Onsite Audit requirements.

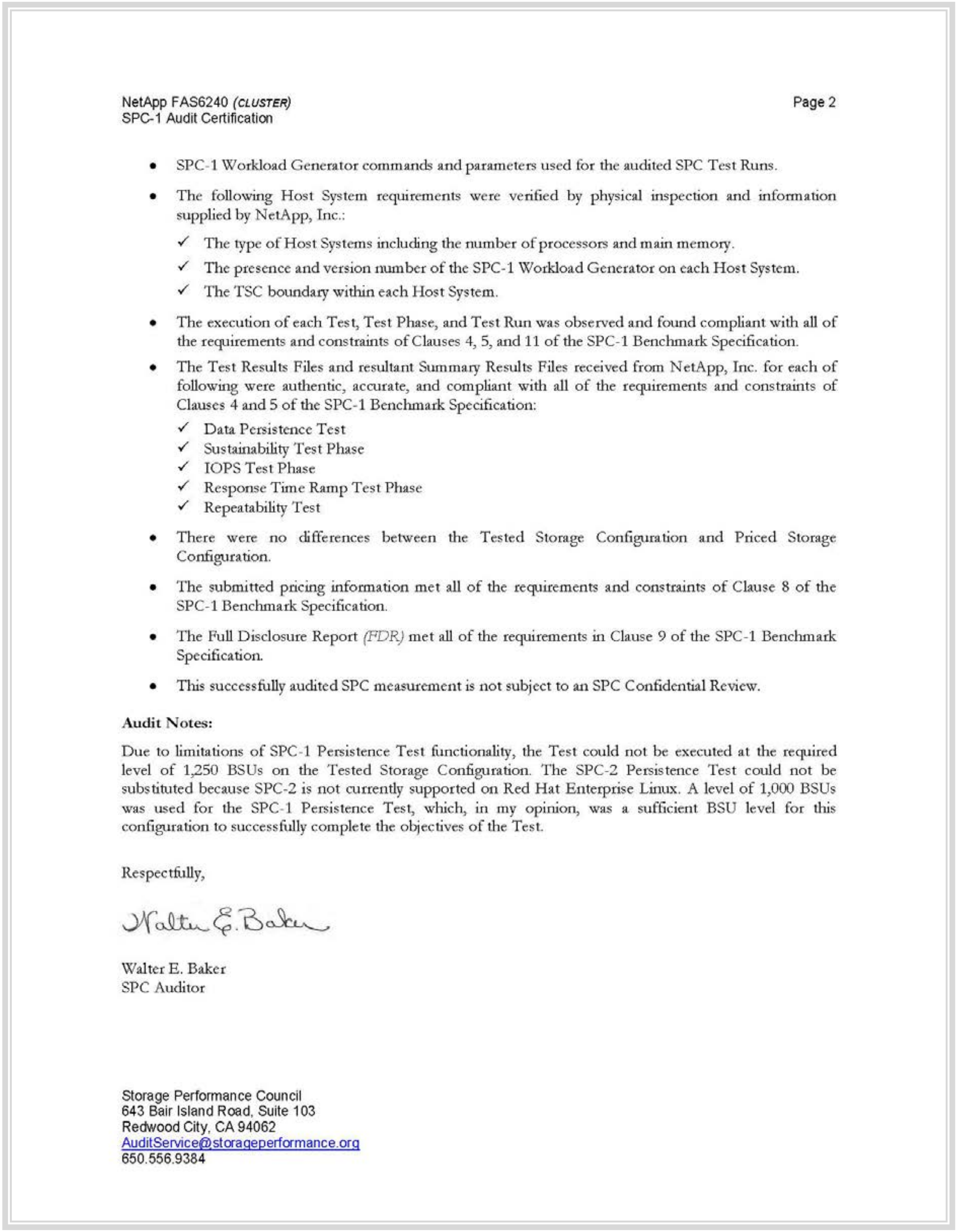
SPC Benchmark 1™ v1.12 Reported Data	
Tested Storage Product (TSP) Name: NetApp FAS6240 *(CLUSTER)*	
Metric	**Reported Result**
SPC-1 IOPS™	250,039.67
SPC-1 Price-Performance	$6.69/SPC-1 IOPS™
Total ASU Capacity	71,521.976 GB
Data Protection Level	Protected *(RAID-DP™)*
Total TSC Price (including three-year maintenance)	$1,672,602.00

The following SPC Benchmark 1™ Onsite Audit requirements were reviewed and found compliant with 1.12 of the SPC Benchmark 1™ specification:

- A Letter of Good Faith, signed by a senior executive.
- The following Data Repository storage items were verified by physical inspection and information supplied by NetApp, Inc.:
  - ✓ Physical Storage Capacity and requirements.
  - ✓ Configured Storage Capacity and requirements.
  - ✓ Addressable Storage Capacity and requirements.
  - ✓ Capacity of each Logical Volume and requirements.
  - ✓ Capacity of each Application Storage Unit (ASU) and requirements.
- An appropriate diagram of the Benchmark Configuration *(BC)*/Tested Storage Configuration *(TSC)*.
- Physical verification of the components to match the above diagram.
- Listings and commands to configure the Benchmark Configuration/Tested Storage Configuration, including customer tunable parameters that were changed from default values.

Storage Performance Council
643 Bair Island Road, Suite 103
Redwood City, CA 94062
AuditService@storageperformance.org
650.556.9384

AUDIT CERTIFICATION *(CONT.)*



NetApp FAS6240 *(CLUSTER)*
SPC-1 Audit Certification

Page 2

- SPC-1 Workload Generator commands and parameters used for the audited SPC Test Runs.
- The following Host System requirements were verified by physical inspection and information supplied by NetApp, Inc.:
  - ✓ The type of Host Systems including the number of processors and main memory.
  - ✓ The presence and version number of the SPC-1 Workload Generator on each Host System.
  - ✓ The TSC boundary within each Host System.
- The execution of each Test, Test Phase, and Test Run was observed and found compliant with all of the requirements and constraints of Clauses 4, 5, and 11 of the SPC-1 Benchmark Specification.
- The Test Results Files and resultant Summary Results Files received from NetApp, Inc. for each of following were authentic, accurate, and compliant with all of the requirements and constraints of Clauses 4 and 5 of the SPC-1 Benchmark Specification:
  - ✓ Data Persistence Test
  - ✓ Sustainability Test Phase
  - ✓ IOPS Test Phase
  - ✓ Response Time Ramp Test Phase
  - ✓ Repeatability Test
- There were no differences between the Tested Storage Configuration and Priced Storage Configuration.
- The submitted pricing information met all of the requirements and constraints of Clause 8 of the SPC-1 Benchmark Specification.
- The Full Disclosure Report *(FDR)* met all of the requirements in Clause 9 of the SPC-1 Benchmark Specification.
- This successfully audited SPC measurement is not subject to an SPC Confidential Review.

Audit Notes:

Due to limitations of SPC-1 Persistence Test functionality, the Test could not be executed at the required level of 1,250 BSUs on the Tested Storage Configuration. The SPC-2 Persistence Test could not be substituted because SPC-2 is not currently supported on Red Hat Enterprise Linux. A level of 1,000 BSUs was used for the SPC-1 Persistence Test, which, in my opinion, was a sufficient BSU level for this configuration to successfully complete the objectives of the Test.

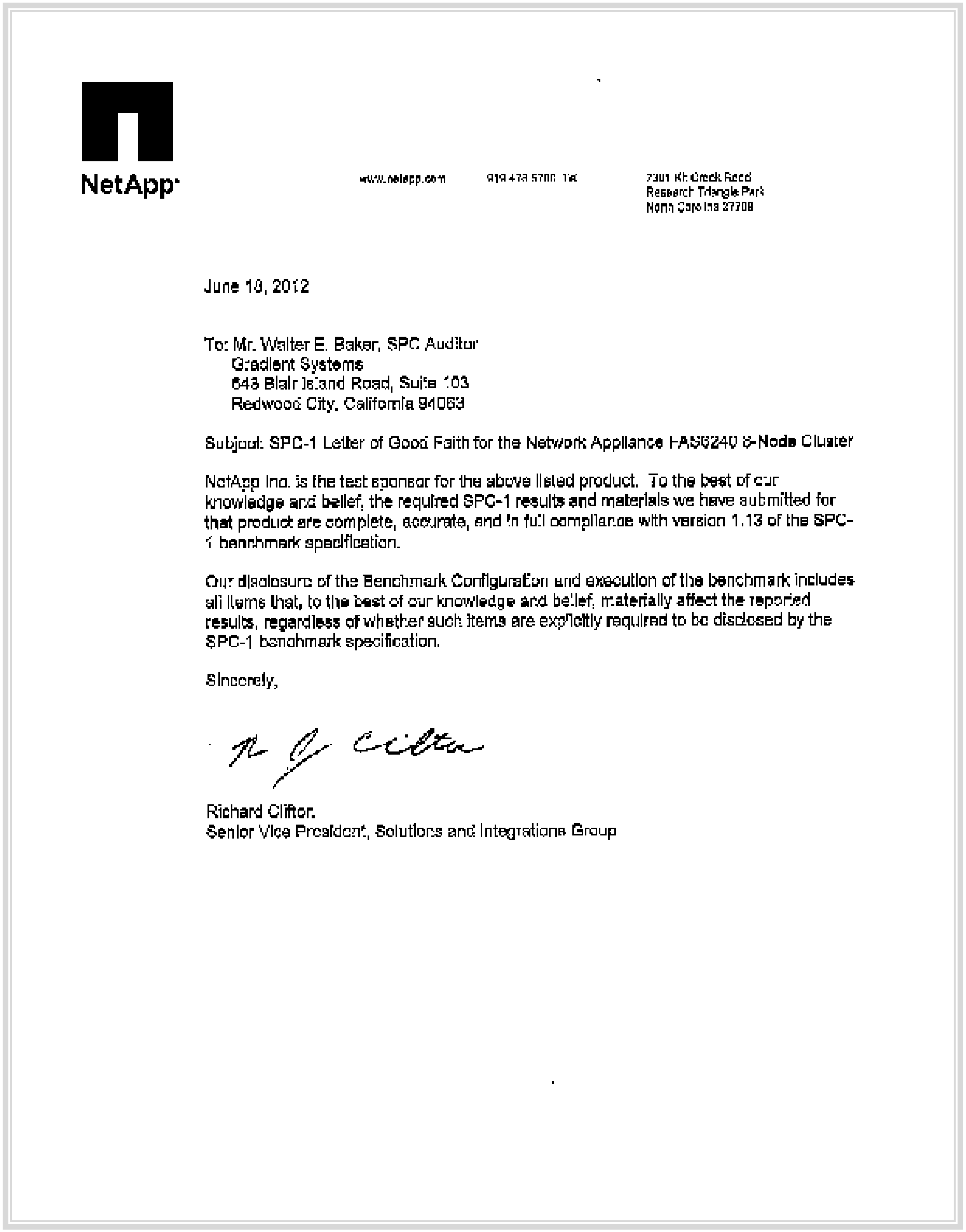
Respectfully,

Walter E. Baker

Walter E. Baker
SPC Auditor

Storage Performance Council
643 Bair Island Road, Suite 103
Redwood City, CA 94062
AuditService@storageperformance.org
650.556.9384

LETTER OF GOOD FAITH



NetApp

www.netapp.com 919 476 5700 Tel

7301 Kit Creek Road
Research Triangle Park
North Carolina 27709

June 18, 2012

To: Mr. Walter E. Baker, SPC Auditor
Gradient Systems
643 Blair Island Road, Suite 103
Redwood City, California 94063

Subject: SPC-1 Letter of Good Faith for the Network Appliance FAS6240 8-Node Cluster

NetApp Inc. is the test sponsor for the above listed product. To the best of our knowledge and belief, the required SPC-1 results and materials we have submitted for that product are complete, accurate, and in full compliance with version 1.13 of the SPC-1 benchmark specification.

Our disclosure of the Benchmark Configuration and execution of the benchmark includes all items that, to the best of our knowledge and belief, materially affect the reported results, regardless of whether such items are explicitly required to be disclosed by the SPC-1 benchmark specification.

Sincerely,

[illegible]

Richard Clifton
Senior Vice President, Solutions and Integrations Group

EXECUTIVE SUMMARY

Test Sponsor and Contact Information

Test Sponsor and Contact Information	
Test Sponsor Primary Contact	NetApp, Inc. – http://www.netapp.com Stephen Daniel – daniel@netapp.com 7301 Kit Creek Road Building 1 Research Triangle Park, NC 27709 Phone: (919) 476-5726 FAX: (919) 476-4272
Test Sponsor Alternate Contact	NetApp, Inc. – http://www.netapp.com Saad Jafri – saad@netapp.com 7301 Kit Creek Road Building 1 Research Triangle Park, NC 27709 Phone: (919) 476-5541 FAX: (919) 476-4272
Auditor	Storage Performance Council – http://www.storageperformance.org Walter E. Baker – AuditService@StoragePerformance.org 643 Bair Island Road, Suite 103 Redwood City, CA 94063 Phone: (650) 556-9384 FAX: (650) 556-9385

Revision Information and Key Dates

Revision Information and Key Dates	
SPC-1 Specification revision number	V1.12
SPC-1 Workload Generator revision number	V2.3.0
Date Results were first used publicly	June 18, 2012
Date the FDR was submitted to the SPC	June 18, 2012
Date the Priced Storage Configuration is available for shipment to customers	currently available
Date the TSC completed audit certification	June 18, 2012

Tested Storage Product (TSP) Description

The NetApp FAS6200 Series is an enterprise-class storage system combining powerful scalability, availability, and performance. The FAS6240 is the middle of three high-end models *(FAS6210, FAS6240, FAS6280)*, which are differentiated by performance and scalability. These systems help customers to confidently meet service levels, drive operational excellence, and respond to future growth.

With NetApp® Data ONTAP® 8 operating in Cluster-Mode, FAS6200 systems deliver storage that is always on and massively scalable, with superior operational efficiency to help manage data, application, and infrastructure growth. Plus, NetApp's unified storage architecture with Data ONTAP's industry-leading storage efficiency enable the efficient consolidation of SAN, NAS, primary, and secondary storage on a single platform, helping lower costs. These FAS6200 systems can handle demanding business and technical applications as well as rapidly changing virtualized and cloud environments.

Summary of Results

SPC-1 Reported Data	
Tested Storage Product (TSP) Name: NetApp FAS6240 *(cluster)*	
Metric	**Reported Result**
SPC-1 IOPS™	250,039.67
SPC-1 Price-Performance™	$6.69/SPC-1 IOPS™
Total ASU Capacity	71,521.976 GB
Data Protection Level	Protected *(RAID DP™)*
Total TSC Price (including three-year maintenance)	$1,672,602.00

SPC-1 IOPS™ represents the maximum I/O Request Throughput at the 100% load point.

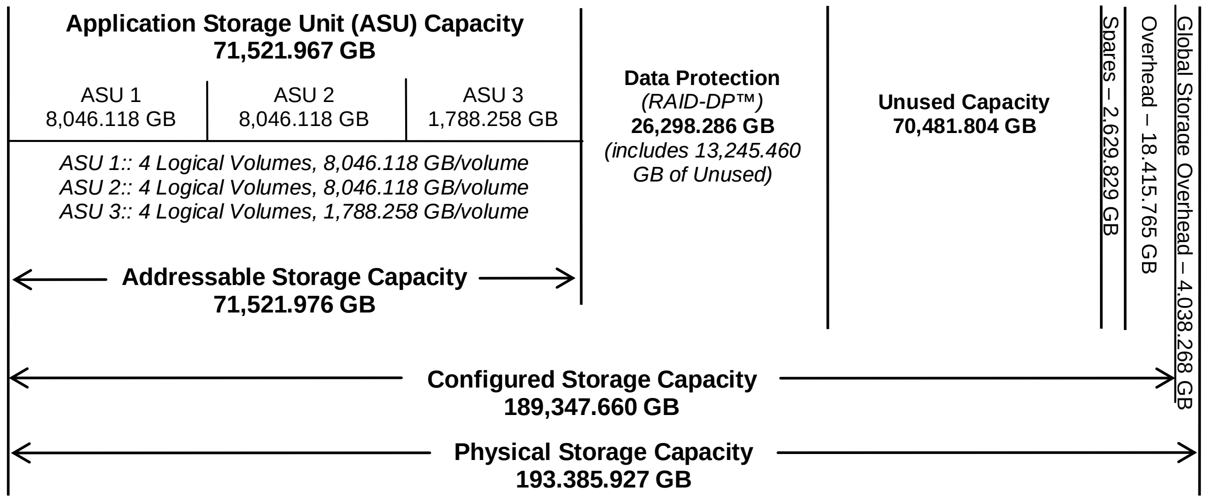
Total ASU (Application Storage Unit) **Capacity** represents the total storage capacity read and written in the course of executing the SPC-1 benchmark.

A **Data Protection Level** of **Protected** using NetApp's RAID-DP™, a RAID-6 implementation, which provides double-parity RAID protection against data loss with negligible performance overhead and no cost penalty compared to single-parity RAID. Additional information is available at the following location:

http://www.netapp.com/products/software/raid-dp.html

Storage Capacities, Relationships, and Utilization

The following diagram and table document the various storage capacities, used in this benchmark, and their relationships, as well as the storage utilization values required to be reported.


Application Storage Unit (ASU) Capacity
71,521.967 GB
ASU 1
8,046.118 GB
ASU 2
8,046.118 GB
ASU 3
1,788.258 GB
ASU 1:: 4 Logical Volumes, 8,046.118 GB/volume
ASU 2:: 4 Logical Volumes, 8,046.118 GB/volume
ASU 3:: 4 Logical Volumes, 1,788.258 GB/volume
Addressable Storage Capacity
71,521.976 GB
Data Protection
(RAID-DP™)
26,298.286 GB
(includes 13,245.460 GB of Unused)
Unused Capacity
70,481.804 GB
Spares – 2,629.829 GB
Overhead – 18,415.765 GB
Global Storage Overhead – 4,038.268 GB
Configured Storage Capacity
189,347.660 GB
Physical Storage Capacity
193.385.927 GB

SPC-1 Storage Capacity Utilization	
Application Utilization	36.98%
Protected Application Utilization	43.83%
Unused Storage Ratio	43.20%

Application Utilization: Total ASU Capacity *(71,521.976 GB)* divided by Physical Storage Capacity *(193,385.927 GB)*

Protected Application Utilization: Total ASU Capacity *(71,521.976 GB)* plus total Data Protection Capacity *(26,298.286 GB)* minus unused Data Protection Capacity *(13,053.826 GB)* divided by Physical Storage Capacity *(193,385.927 GB)*

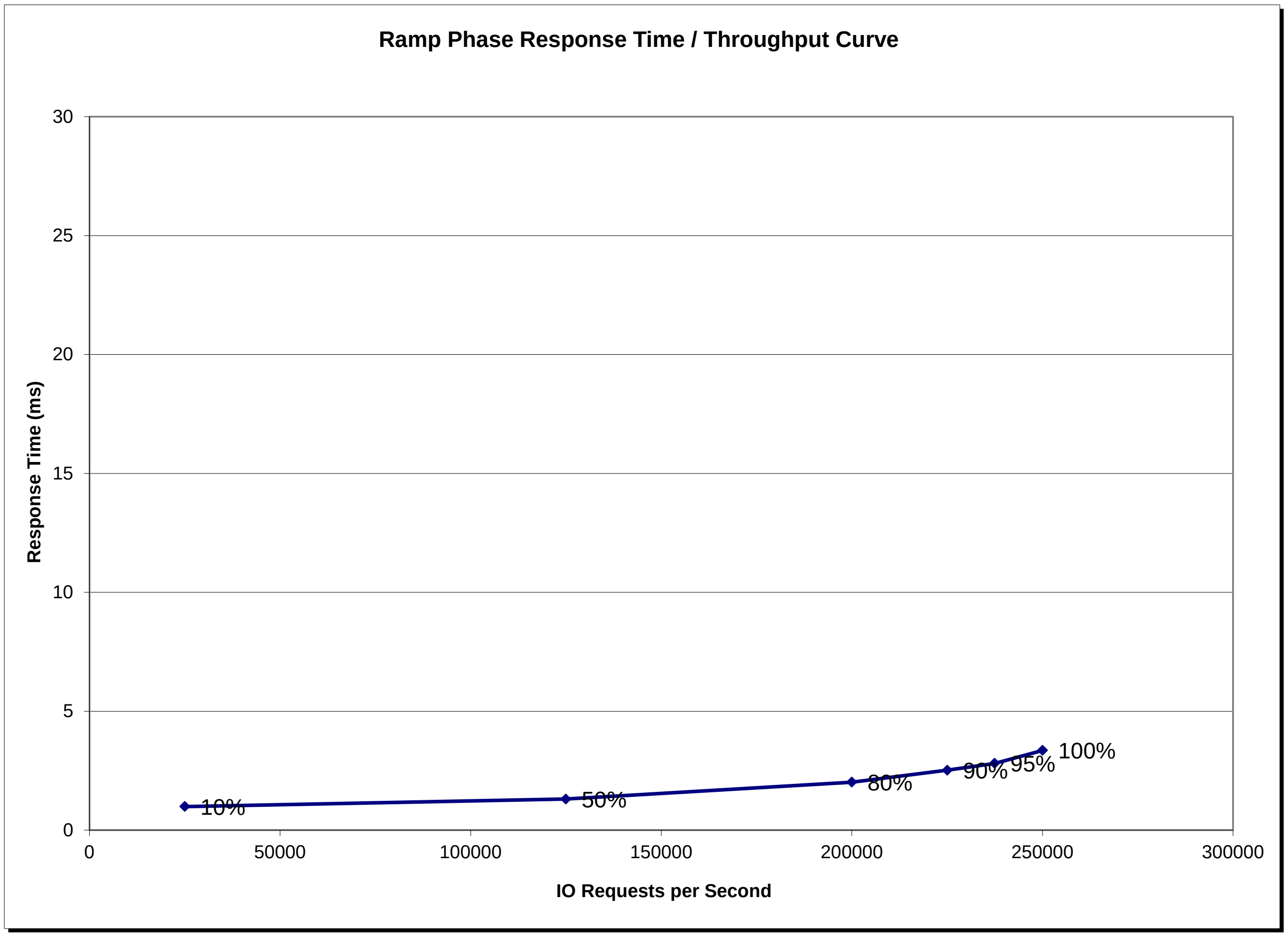
Unused Storage Ratio: Total Unused Capacity *(83,534.630 GB)* divided by Physical Storage Capacity *(193,385.927 GB)* and may not exceed 45%.

Detailed information for the various storage capacities and utilizations is available on pages 21-22 in the Full Disclosure Report.

Response Time – Throughput Curve

The Response Time-Throughput Curve illustrates the Average Response Time (milliseconds) and I/O Request Throughput at 100%, 95%, 90%, 80%, 50%, and 10% of the workload level used to generate the SPC-1 IOPS™ metric.

The Average Response Time measured at any of the above load points cannot exceed 30 milliseconds or the benchmark measurement is invalid.



Response Time – Throughput Data

	10% Load	50% Load	80% Load	90% Load	95% Load	100% Load
I/O Request Throughput	25,009.20	125,001.07	199,989.66	225,030.60	237,487.76	250,039.67
Average Response Time (ms):						
All ASUs	0.99	1.31	2.02	2.52	2.81	3.35
ASU-1	1.18	1.58	2.44	3.06	3.42	4.11
ASU-2	1.27	1.65	2.42	2.91	3.19	3.64
ASU-3	0.48	0.59	0.95	1.21	1.35	1.63
Reads	1.77	2.38	3.52	4.30	4.76	5.58
Writes	0.48	0.61	1.04	1.36	1.54	1.91

Priced Storage Configuration Pricing

	Description	Quantity	List Price	Extended List
FAS6240 (Cluster-Mode)	includes 512 GB FlashCache	6	$ 89,215	$ 535,290
DS4243-1507-24S-R5	DSK SHLF, 24x450GB, 15K, 3Gb SAS, IOM3, -C, R5	18	$ 48,489	$ 872,802
X-320-0008-R5	24-Pt Brocade 300 Full Fab FC 8Gbps, -C, R5 (FC Switch)	2	$ 2,939	$ 5,878
X1968-R5	ClusterNet Interconnect,20Pt,10Gb	2	$ 1	$ 2
X-SFP-H10GB-CU1M-R6	Cisco N5020 10GBase Copper SFP+cable, 1m, -C, R6 (for Cluster interconnect)	20	$ 80	$ 1,600
X1095A-R6	HBA,QLogic QLE2562,2-Port,8Gb,PCIe,R6	6	$ 2,005	$ 12,030
X6524-R6	LC-LC Optical cable connecting Host Qlogic HBA to FC switch (pair)	6	$ 125	$ 750
X6524-R6	LC-LC Optical cable connecting controller onboard FC target port to FC switch (pair)	6	$ 125	$ 750
X2065	4-port SAS cards connecting controller to DS4243 shelves	12	$ 2,000	$ 24,000
X6558-R6	Cable, SAS Cntlr-Shelf/Shelf-Shelf/HA, 2m, -C	48	$ 125	$ 6,000
X6559-R6	Cable, SAS Cntlr-Shelf/Shelf-Shelf/HA, 5m, -C	24	$ 170	$ 4,080
X8712C-R6	PDU, 1-Phase, 24 Outlet, 30A, NEMA, -C, R6	6	$ 550	$ 3,300
X870C-R6	Cab, Deep, Empty, No PDU, No Rails, -C	3	$ 3,550	$ 10,650
X8778-R6	Mounting Bracket, Tie-Down, 32X0, -C, R6	40	$ 50	$ 2,000
CS-A-INST-4R	3-years SupportEdge Standard Replace 4hr, Hardware Support			$ 149,097
PS install				$ 44,374
Total ($)				**$ 1,672,602**

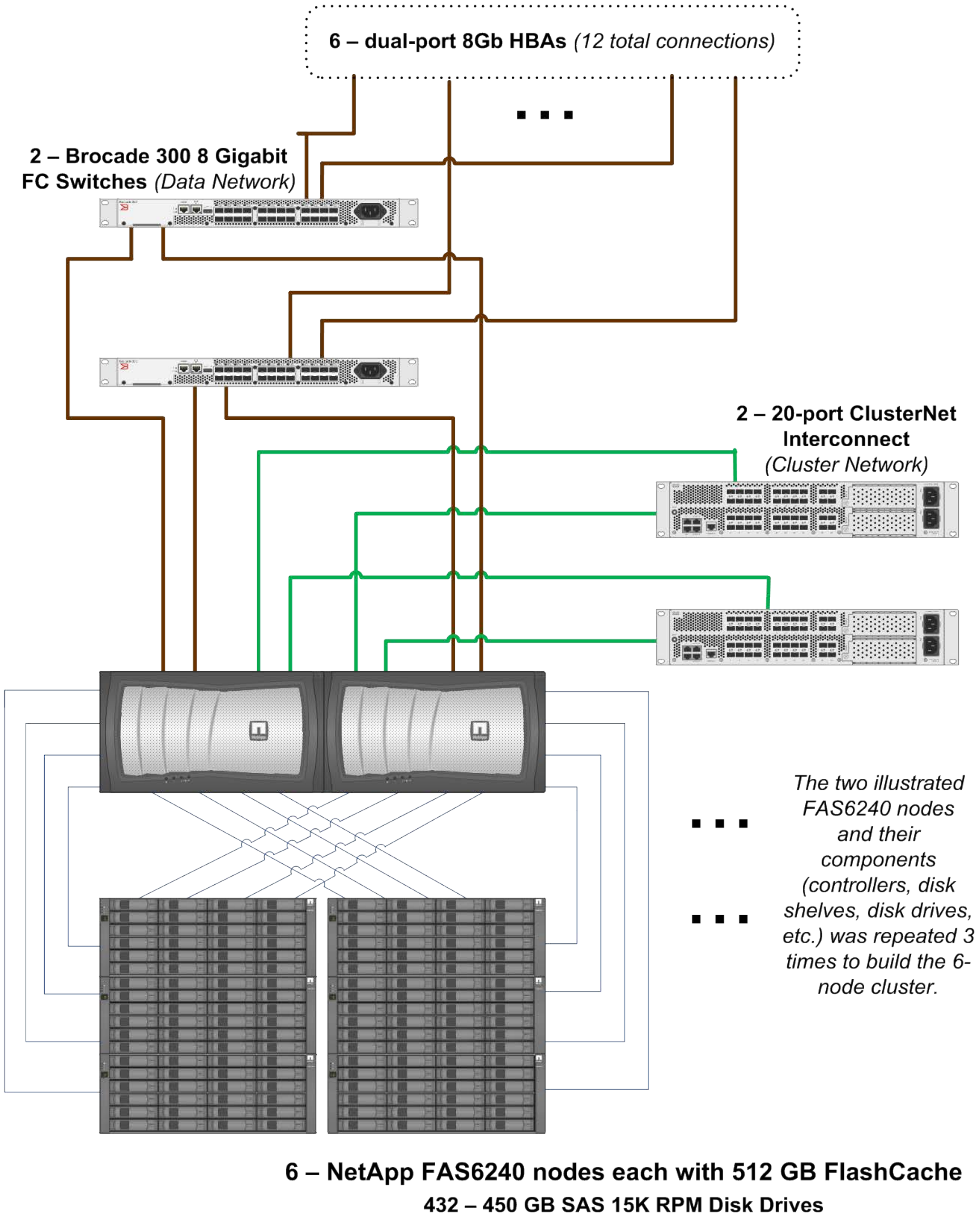
The above pricing includes hardware maintenance and software support for three years, 7 days per week, 24 hours per day. The hardware maintenance and software support provides the following:

- Acknowledgement of new and existing problems with four (4) hours.
- Onsite presence of a qualified maintenance engineer or provision of a customer replaceable part within four (4) hours of the above acknowledgement for any hardware failure that results in an inoperative Price Storage Configuration that can be remedied by the repair or replacement of a Priced Storage Configuration component.

Differences between the Tested Storage Configuration (TSC) and Priced Storage Configuration

There were no differences between the TSC and the Priced Storage Configuration.

Priced Storage Configuration Diagram


6 – dual-port 8Gb HBAs (12 total connections)
2 – Brocade 300 8 Gigabit FC Switches (Data Network)
2 – 20-port ClusterNet Interconnect (Cluster Network)
The two illustrated FAS6240 nodes and their components (controllers, disk shelves, disk drives, etc.) was repeated 3 times to build the 6-node cluster.
6 – NetApp FAS6240 nodes each with 512 GB FlashCache
432 – 450 GB SAS 15K RPM Disk Drives

Priced Storage Configuration Components

Priced Storage Configuration
6 – dual port 8 Gb FC HBAs
NetApp FAS6240 *(cluster)*   6 – controller nodes each with: 48 GB memory/cache   512 GB FlashCache   2 – FC 8Gb FC front-end connections *(12 total)*   8 – SAS backend connections *(48 total)*   *(in a Multipath High Availability (HA) configuration)*
Data ONTAP® 8.1.1
2 – 20-port 10Gb ClusterNet Interconnects *(Cluster Network)*
2 – Brocade 300 8 Gb 24-port switches *(Data Network)*,
12 – 4-port SAS cards *(controller nodes to DS4243 shelves)*
18 – DS4243 Disk Shelves
432 – 450 GB, 15K RPM SAS disk drives   24 disk drives per DS4243 Disk Shelf

In each of the following sections of this document, the appropriate Full Disclosure Report requirement, from the SPC-1 benchmark specification, is stated in italics followed by the information to fulfill the stated requirement.

CONFIGURATION INFORMATION

Benchmark Configuration (BC)/Tested Storage Configuration (TSC) Diagram

Clause 9.4.3.4.1

A one page Benchmark Configuration (BC)/Tested Storage Configuration (TSC) diagram shall be included in the FDR…

The Benchmark Configuration (BC)/Tested Storage Configuration (TSC) is illustrated on page 18 *(Benchmark Configuration/Tested Storage Configuration Diagram).*

Storage Network Configuration

Clause 9.4.3.4.1

…

5. *If the TSC contains network storage, the diagram will include the network configuration. If a single diagram is not sufficient to illustrate both the Benchmark Configuration and network configuration in sufficient detail, the Benchmark Configuration diagram will include a high-level network illustration as shown in Figure 9-8. In that case, a separate, detailed network configuration diagram will also be included as described in Clause 9.4.3.4.2.*

Clause 9.4.3.4.2

If a storage network was configured as a part of the Tested Storage Configuration and the Benchmark Configuration diagram described in Clause 9.4.3.4.1 contains a high-level illustration of the network configuration, the Executive Summary will contain a one page topology diagram of the storage network as illustrated in Figure 9-9.

The storage network portion of Benchmark Configuration (BC)/Tested Storage Configuration (TSC) is illustrated on page 18 *(Benchmark Configuration/Tested Storage Configuration Diagram).*

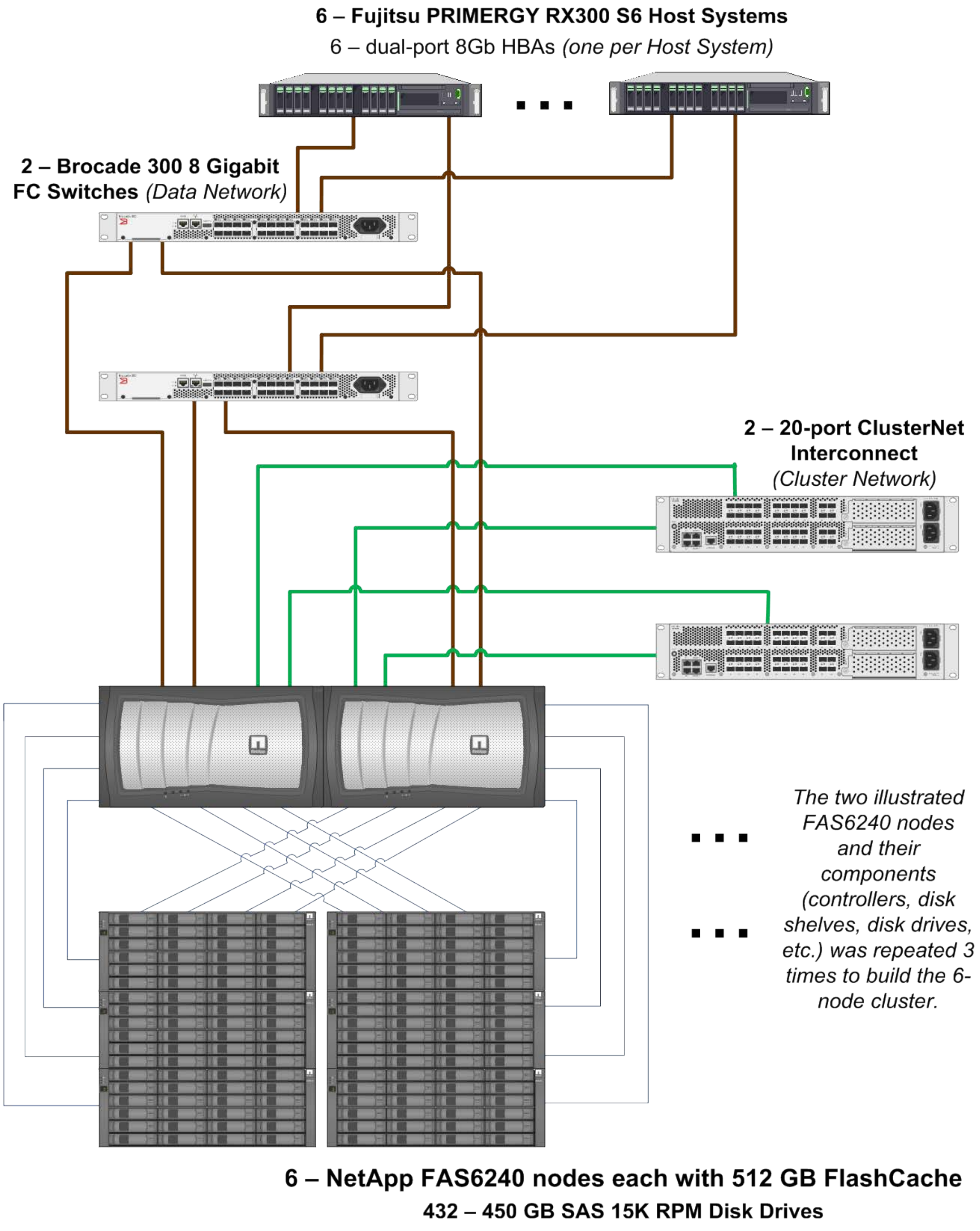
Host System and Tested Storage Configuration (TSC) Table of Components

Clause 9.4.3.4.3

The FDR will contain a table that lists the major components of each Host System and the Tested Storage Configuration (TSC). Table 9-10 specifies the content, format, and appearance of the table.

The Host System and TSC table of components may be found on page 19 *(Host Systems and Tested Storage Configuration Components).*

Benchmark Configuration/Tested Storage Configuration Diagram


6 – Fujitsu PRIMERGY RX300 S6 Host Systems
6 – dual-port 8Gb HBAs *(one per Host System)*
2 – Brocade 300 8 Gigabit FC Switches *(Data Network)*
2 – 20-port ClusterNet Interconnect *(Cluster Network)*
The two illustrated FAS6240 nodes and their components (controllers, disk shelves, disk drives, etc.) was repeated 3 times to build the 6-node cluster.
6 – NetApp FAS6240 nodes each with 512 GB FlashCache
432 – 450 GB SAS 15K RPM Disk Drives

Host Systems and Tested Storage Configuration Components

Host Systems:	**Tested Storage Configuration (TSC)**
6 – Fujitsu Primergy RX300 S6 each with: 2 – six core Intel Xeon E5645 processors 2.40 GHz core speed, 12 MB shared cache	6 – dual port 8 Gb FC HBAs
48 GB main memory per Host System	**NetApp FAS6240 *(cluster)*** 6 – controller nodes each with: 48 GB memory/cache 512 GB FlashCache 2 – FC 8GB FC front-end connections (12 total, 12 used) 8 – SAS backend connections *(48 total, 48 used)* *(in a Multipath High Availability (HA) configuration)*
Red Hat Enterprise Linux 6.2 (64-bit)	Data ONTAP® 8.1.1
NetApp Fibre Channel Linux Host Utilities 6.0	2 – 20-port 10Gb ClusterNet Interconnects *(Cluster Network)*
PCIe	2 – Brocade 300 8 Gb 24-port switches *(Data Network)*
	12 – 4-port SAS cards *(controller nodes to DS4243 shelves)*
	18 – DS4243 Disk Shelves
	432 – 450 GB, 15K RPM SAS disk drives 24 disk drives per DS4243 Disk Shelf

Customer Tunable Parameters and Options

Clause 9.4.3.5.1

All Benchmark Configuration (BC) components with customer tunable parameter and options that have been altered from their default values must be listed in the FDR. The FDR entry for each of those components must include both the name of the component and the altered value of the parameter or option. If the parameter name is not self-explanatory to a knowledgeable practitioner, a brief description of the parameter's use must also be included in the FDR entry.

"Appendix B: Customer Tunable Parameters and Options" on page 60 contains the customer tunable parameters and options that have been altered from their default values for this benchmark.

Tested Storage Configuration (TSC) Description

Clause 9.4.3.5.2

The FDR must include sufficient information to recreate the logical representation of the TSC. In addition to customer tunable parameters and options (Clause 4.2.4.5.3), that information must include, at a minimum:

- *A diagram and/or description of the following:*
  - *All physical components that comprise the TSC. Those components are also illustrated in the BC Configuration Diagram in Clause 9.2.4.4.1 and/or the Storage Network Configuration Diagram in Clause 9.2.4.4.2.*
  - *The logical representation of the TSC, configured from the above components that will be presented to the Workload Generator.*
- *Listings of scripts used to create the logical representation of the TSC.*
- *If scripts were not used, a description of the process used with sufficient detail to recreate the logical representation of the TSC.*

"Appendix C: Tested Storage Configuration (TSC) Creation" on page 62 contains the detailed information that describes how to create and configure the logical TSC.

SPC-1 Workload Generator Storage Configuration

Clause 9.4.3.5.3

The FDR must include all SPC-1 Workload Generator storage configuration commands and parameters.

The SPC-1 Workload Generator storage configuration commands and parameters for this measurement appear in "Appendix D: SPC-1 Workload Generator Storage Commands and Parameters" on page 92.

SPC-1 DATA REPOSITORY

This portion of the Full Disclosure Report presents the detailed information that fully documents the various SPC-1 storage capacities and mappings used in the Tested Storage Configuration. "SPC-1 Data Repository Definitions" on page 56 contains definitions of terms specific to the SPC-1 Data Repository.

Storage Capacities and Relationships

Clause 9.4.3.6.1

Two tables and an illustration documenting the storage capacities and relationships of the SPC-1 Storage Hierarchy (Clause 2.1) shall be included in the FDR.

SPC-1 Storage Capacities

SPC-1 Storage Capacities

Storage Hierarchy Component	Units	Capacity
Total ASU Capacity	Gigabytes (GB)	71,521.976
Addressable Storage Capacity	Gigabytes (GB)	71,521.976
Configured Storage Capacity	Gigabytes (GB)	189,347.660
Physical Storage Capacity	Gigabytes (GB)	193,385.927
Data Protection *(RAID-DP™)*	Gigabytes (GB)	26,298.286
Required Storage	Gigabytes (GB)	21,045.593
Global Storage Overhead	Gigabytes (GB)	4,038.268
Total Unused Storage	Gigabytes (GB)	83,534.630

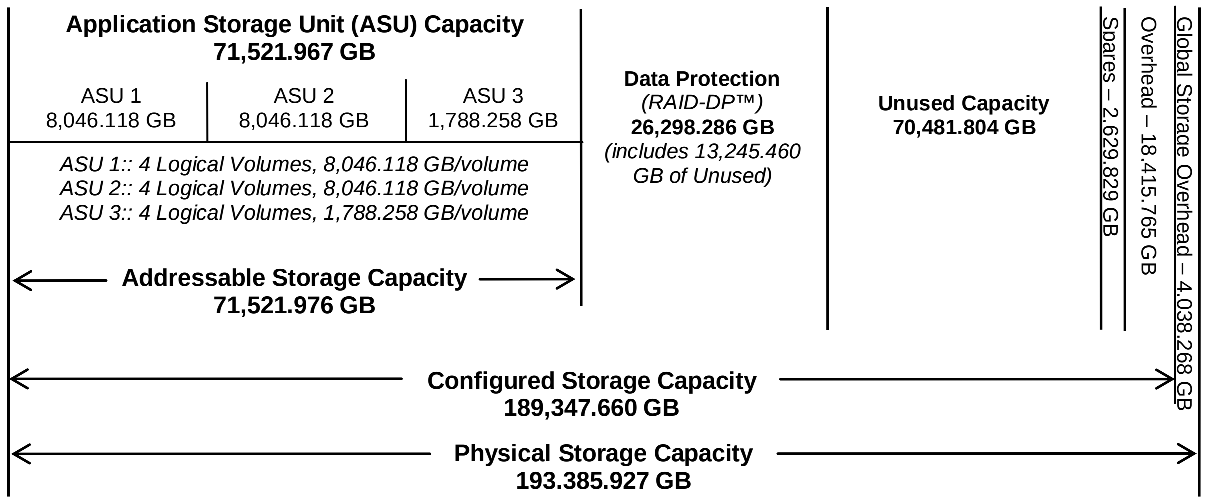
SPC-1 Storage Hierarchy Ratios

	Addressable Storage Capacity	Configured Storage Capacity	Physical Storage Capacity
Total ASU Capacity	100.00%	37.757	36.98%
Required for Data Protection *(RAID-DP™)*		13.85%	13.60%
Addressable Storage Capacity		37.77%	36.98%
Required Storage		11.11%	10.88%
Configured Storage Capacity			97.91%
Global Storage Overhead			2.09%
Unused Storage:			
Addressable	0.00%		
Configured		37.22%	
Physical			0.00%

The Physical Storage Capacity consisted of 193,385.927 GB distributed over 192 disk drives, each with a formatted capacity of 447.906 GB and 240 disk drives, each with a formatted capacity of 447.450 GB. There was 0.000 GB (0%) of Unused Storage within the Physical Storage Capacity. Global Storage Overhead consisted of 4,038.268 GB (2.09%) of the Physical Storage Capacity. There was 70,481.804 GB (37.22%) of Unused Storage within the Configured Storage Capacity. The Total ASU Capacity utilized 100% of the Addressable Storage Capacity resulting in GB (0%) of Unused Storage within the Addressable Storage Capacity. The Data Protection *(RAID-DP™)* capacity was 26,298.286 GB of which 13,245.460 GB was utilized. The total Unused Storage was 70,481.804 GB.

SPC-1 Storage Capacities and Relationships Illustration

The various storage capacities configured in the benchmark result are illustrated below *(not to scale)*.


Application Storage Unit (ASU) Capacity
71,521.967 GB
ASU 1
8,046.118 GB
ASU 2
8,046.118 GB
ASU 3
1,788.258 GB
ASU 1:: 4 Logical Volumes, 8,046.118 GB/volume
ASU 2:: 4 Logical Volumes, 8,046.118 GB/volume
ASU 3:: 4 Logical Volumes, 1,788.258 GB/volume
Addressable Storage Capacity
71,521.976 GB
Data Protection
(RAID-DP™)
26,298.286 GB
(includes 13,245.460 GB of Unused)
Unused Capacity
70,481.804 GB
Spares – 2,629.829 GB
Overhead – 18,415.765 GB
Global Storage Overhead – 4,038.268 GB
Configured Storage Capacity
189,347.660 GB
Physical Storage Capacity
193.385.927 GB

Logical Volume Capacity and ASU Mapping

Clause 9.4.3.6.3

A table illustrating the capacity of each ASU and the mapping of Logical Volumes to ASUs shall be provided in the FDR. … Logical Volumes shall be sequenced in the table from top to bottom per its position in the contiguous address space of each ASU. The capacity of each Logical Volume shall be stated. … In conjunction with this table, the Test Sponsor shall provide a complete description of the type of data protection (see Clause 2.4.5) used on each Logical Volume.

Logical Volume Capacity and Mapping

ASU-1 (32,184.472 GB)	ASU-2 (32,184.472 GB)	ASU-3 (7,153.033 GB)
4 Logical Volumes 8,046.118 GB per Logical Volume (8,046.118 used per Logical Volume)	4 Logical Volumes 8,046.118 GB per Logical Volume (8,046.118 used per Logical Volume)	4 Logical Volumes 1,788.258 GB per Logical Volume (1,788.258 used per Logical Volume)

The Data Protection Level used for all Logical Volumes was **Protected** *(RAID-DP™)* as described on page 11. See "ASU Configuration" in the IOPS Test Results File for more detailed configuration information.

Storage Capacity Utilization

Clause 9.4.3.6.2

The FDR will include a table illustrating the storage capacity utilization values defined for Application Utilization (Clause 2.8.1), Protected Application Utilization (Clause 2.8.2), and Unused Storage Ratio (Clause 2.8.3).

Clause 2.8.1

Application Utilization is defined as Total ASU Capacity divided by Physical Storage Capacity.

Clause 2.8.2

Protected Application Utilization is defined as (Total ASU Capacity plus total Data Protection Capacity minus unused Data Protection Capacity) divided by Physical Storage Capacity.

Clause 2.8.3

Unused Storage Ratio is defined as Total Unused Capacity divided by Physical Storage Capacity and may not exceed 45%.

SPC-1 Storage Capacity Utilization	
Application Utilization	36.98%
Protected Application Utilization	43.83%
Unused Storage Ratio	43.20%

SPC-1 BENCHMARK EXECUTION RESULTS

This portion of the Full Disclosure Report documents the results of the various SPC-1 Tests, Test Phases, and Test Runs. "SPC-1 Test Execution Definitions" on page 57 contains definitions of terms specific to the SPC-1 Tests, Test Phases, and Test Runs.

Clause 5.4.3

The Tests must be executed in the following sequence: Primary Metrics, Repeatability, and Data Persistence. That required sequence must be uninterrupted from the start of Primary Metrics to the completion of Persistence Test Run 1. Uninterrupted means the Benchmark Configuration shall not be power cycled, restarted, disturbed, altered, or adjusted during the above measurement sequence. If the required sequence is interrupted other than for the Host System/TSC power cycle between the two Persistence Test Runs, the measurement is invalid.

SPC-1 Tests, Test Phases, and Test Runs

The SPC-1 benchmark consists of the following Tests, Test Phases, and Test Runs:

- **Primary Metrics Test**
  - Sustainability Test Phase and Test Run
  - IOPS Test Phase and Test Run
  - Response Time Ramp Test Phase
    - 95% of IOPS Test Run
    - 90% of IOPS Test Run
    - 80% of IOPS Test Run
    - 50% of IOPS Test Run
    - 10% of IOPS Test Run (LRT)
- **Repeatability Test**
  - Repeatability Test Phase 1
    - 10% of IOPS Test Run (LRT)
    - IOPS Test Run
  - Repeatability Test Phase 2
    - 10% of IOPS Test Run (LRT)
    - IOPS Test Run
- **Data Persistence Test**
  - Data Persistence Test Run 1
  - Data Persistence Test Run 2

Each Test is an atomic unit that must be executed from start to finish before any other Test, Test Phase, or Test Run may be executed.

The results from each Test, Test Phase, and Test Run are listed below along with a more detailed explanation of each component.

Primary Metrics Test – Sustainability Test Phase

Clause 5.4.4.1.1

The Sustainability Test Phase has exactly one Test Run and shall demonstrate the maximum sustainable I/O Request Throughput within at least a continuous three (3) hour Measurement Interval. This Test Phase also serves to insure that the TSC has reached Steady State prior to reporting the final maximum I/O Request Throughput result (SPC-1 IOPS™).

Clause 5.4.4.1.2

The computed I/O Request Throughput of the Sustainability Test must be within 5% of the reported SPC-1 IOPS™ result.

Clause 5.4.4.1.4

The Average Response Time, as defined in Clause 5.1.1, will be computed and reported for the Sustainability Test Run and cannot exceed 30 milliseconds. If the Average Response time exceeds that 30-milliseconds constraint, the measurement is invalid.

Clause 9.4.3.7.1

For the Sustainability Test Phase the FDR shall contain:

1. *A Data Rate Distribution graph and data table.*
2. *I/O Request Throughput Distribution graph and data table.*
3. *A Response Time Frequency Distribution graph and table.*
4. *An Average Response Time Distribution graph and table.*
5. *The human readable Test Run Results File produced by the Workload Generator (may be included in an appendix).*
6. *A listing or screen image of all input parameters supplied to the Workload Generator (may be included in an appendix).*
7. *The Measured Intensity Multiplier for each I/O stream.*
8. *The variability of the Measured Intensity Multiplier, as defined in Clause 5.3.13.3.*

SPC-1 Workload Generator Input Parameters

The SPC-1 Workload Generator input parameters for the Sustainability, IOPS, Response Time Ramp, Repeatability, and Persistence Test Runs are documented in "Appendix E: SPC-1 Workload Generator Input Parameters" on Page 93.

Sustainability Test Results File

A link to the test results file generated from the Sustainability Test Run is listed below.

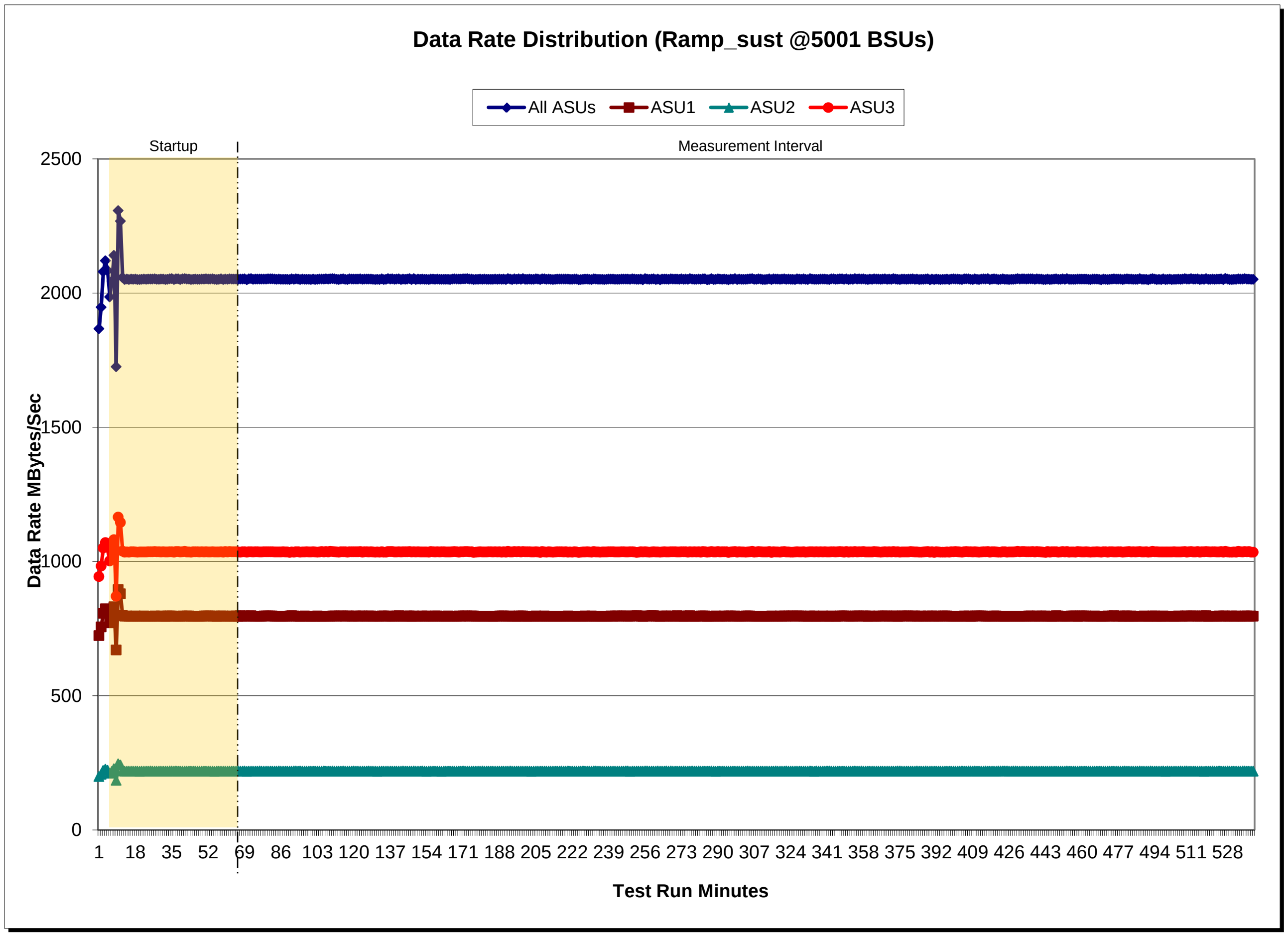
Sustainability Test Results File

Sustainability – Data Rate Distribution Data *(MB/second)*

The Sustainability Data Rate table of data is not embedded in this document due to its size. The table is available via the following URL:

Sustainability Data Rate Table

Sustainability – Data Rate Distribution Graph

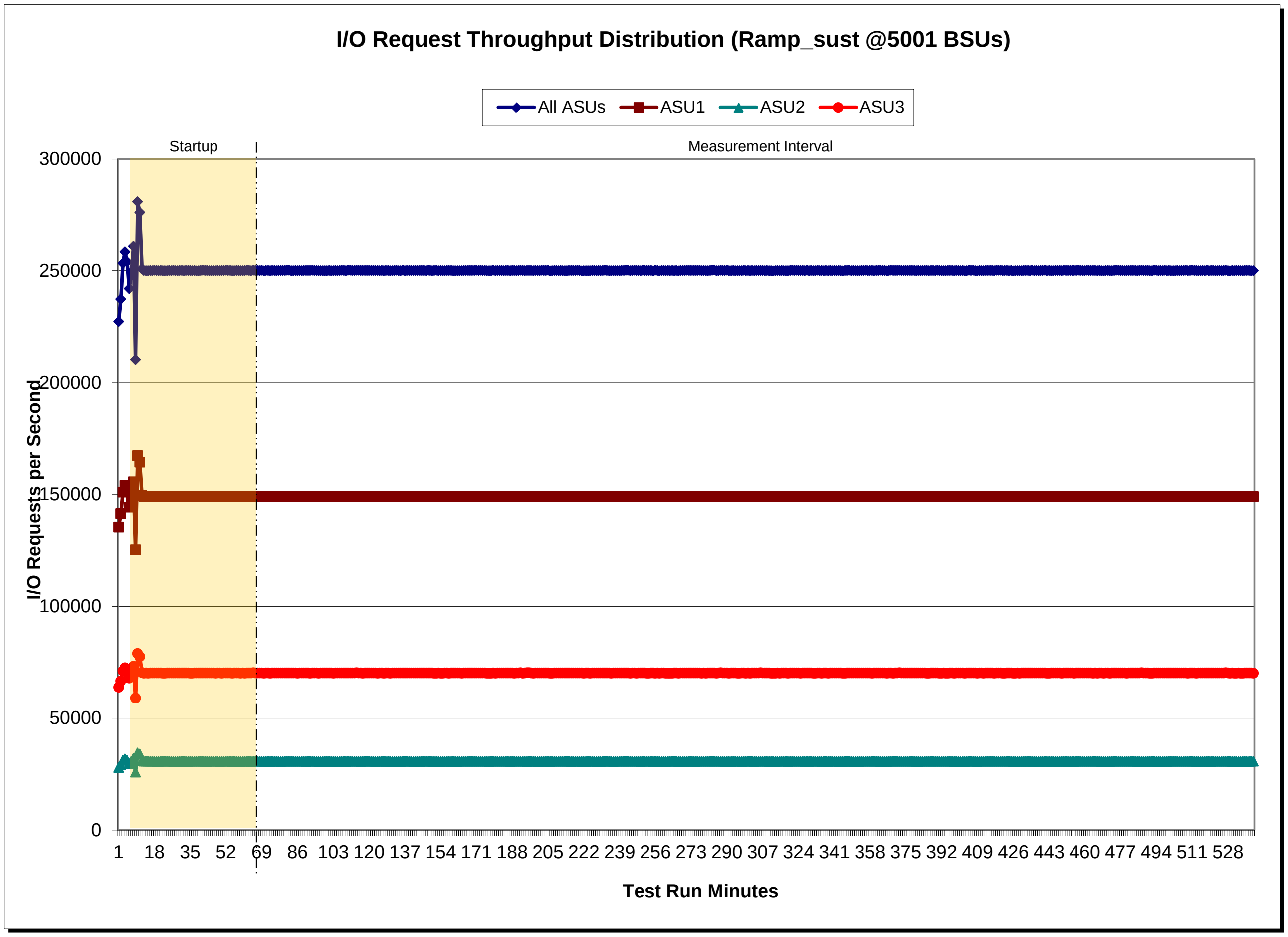


Sustainability – I/O Request Throughput Distribution Data

The Sustainability I/O Request Throughput table of data is not embedded in this document due to its size. The table is available via the following URL:

Sustainability I/O Request Throughput Table

Sustainability – I/O Request Throughput Distribution Graph

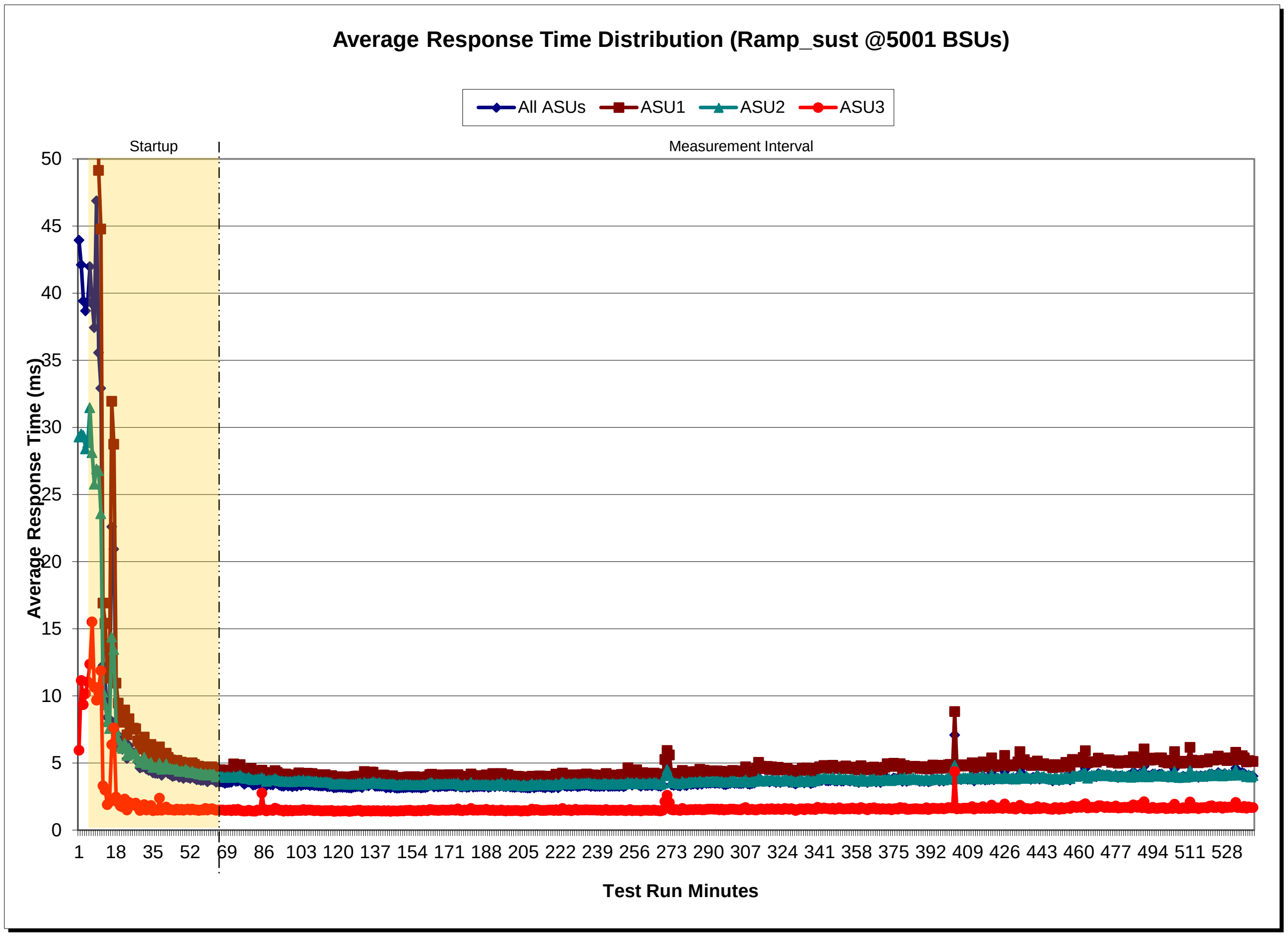


Sustainability – Average Response Time (ms) Distribution Data

The Sustainability Average Response Time table of data is not embedded in this document due to its size. The table is available via the following URL:

Sustainability Average Response Time Table

Sustainability – Average Response Time (ms) Distribution Graph



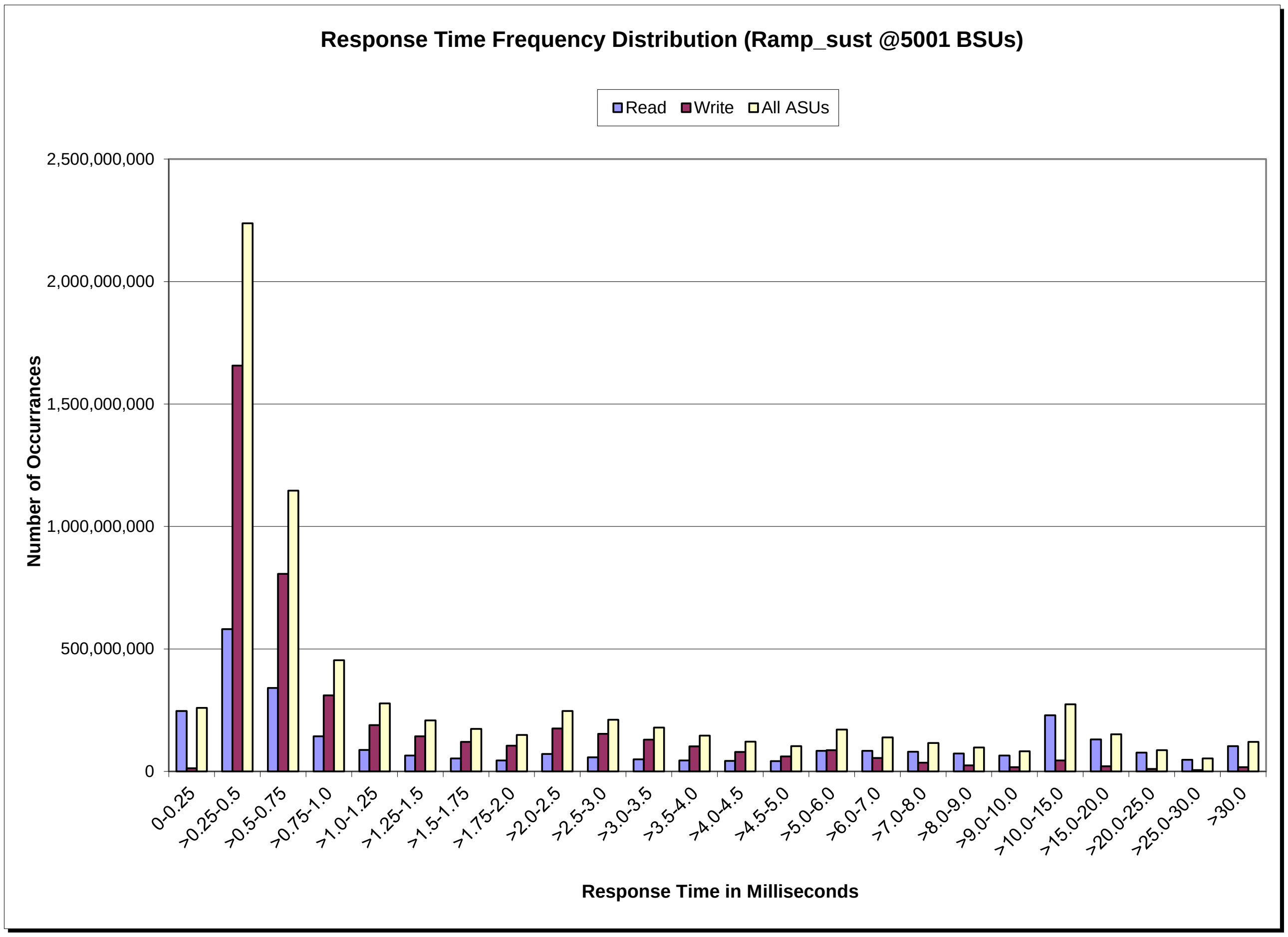
Sustainability – Response Time Frequency Distribution Data

Response Time (ms)	0-0.25	>0.25-0.5	>0.5-0.75	>0.75-1.0	>1.0-1.25	>1.25-1.5	>1.5-1.75	>1.75-2.0
Read	246,756,061	581,288,355	340,375,426	143,270,215	87,948,365	64,379,533	52,656,087	44,334,948
Write	12,278,568	1,656,644,995	806,140,491	310,189,128	189,227,509	143,383,939	120,397,493	104,784,078
All ASUs	259,034,629	2,237,933,350	1,146,515,917	453,459,343	277,175,874	207,763,472	173,053,580	149,119,026
ASU1	189,973,143	1,228,981,885	628,511,500	252,217,435	154,373,964	114,855,363	95,060,023	81,412,944
ASU2	64,384,219	260,330,306	118,683,630	52,766,486	32,789,527	24,740,089	20,688,308	17,808,505
ASU3	4,677,267	748,621,159	399,320,787	148,475,422	90,012,383	68,168,020	57,305,249	49,897,577

Response Time (ms)	>2.0-2.5	>2.5-3.0	>3.0-3.5	>3.5-4.0	>4.0-4.5	>4.5-5.0	>5.0-6.0	>6.0-7.0
Read	71,197,563	57,374,642	48,921,939	44,264,023	42,447,194	41,849,283	84,219,983	84,211,727
Write	175,566,253	153,246,104	129,801,423	102,187,704	79,265,244	61,474,924	86,797,664	54,462,529
All ASUs	246,763,816	210,620,746	178,723,362	146,451,727	121,712,438	103,324,207	171,017,647	138,674,256
ASU1	133,758,871	112,539,851	95,232,674	79,769,458	68,935,444	61,375,120	109,052,465	96,381,207
ASU2	29,319,792	24,857,316	21,159,711	17,581,517	14,896,099	12,883,233	21,802,728	18,052,445
ASU3	83,685,153	73,223,579	62,330,977	49,100,752	37,880,895	29,065,854	40,162,454	24,240,604

Response Time (ms)	>7.0-8.0	>8.0-9.0	>9.0-10.0	>10.0-15.0	>15.0-20.0	>20.0-25.0	>25.0-30.0	>30.0
Read	80,374,087	72,914,347	64,502,132	229,259,301	130,241,708	76,462,946	47,749,827	103,142,462
Write	35,620,172	24,360,218	17,384,531	44,798,226	21,007,142	9,901,505	5,157,563	17,172,837
All ASUs	115,994,259	97,274,565	81,886,663	274,057,527	151,248,850	86,364,451	52,907,390	120,315,299
ASU1	85,454,500	74,348,810	64,094,734	220,082,357	123,500,962	72,330,195	45,081,461	104,689,967
ASU2	15,371,256	13,085,327	11,177,743	38,458,308	21,182,468	11,902,741	7,173,918	14,685,420
ASU3	15,168,503	9,840,428	6,614,186	15,516,862	6,565,420	2,131,515	652,011	939,912

Sustainability – Response Time Frequency Distribution Graph



Sustainability – Measured Intensity Multiplier and Coefficient of Variation

Clause 3.4.3

IM – Intensity Multiplier: *The ratio of I/Os for each I/O stream relative to the total I/Os for all I/O streams (ASU1-1 – ASU3-1) as required by the benchmark specification.*

Clauses 5.1.10 and 5.3.13.2

MIM – Measured Intensity Multiplier: *The Measured Intensity Multiplier represents the ratio of measured I/Os for each I/O stream relative to the total I/Os measured for all I/O streams (ASU1-1 – ASU3-1). This value may differ from the corresponding Expected Intensity Multiplier by no more than 5%.*

Clause 5.3.13.3

COV – Coefficient of Variation: *This measure of variation for the Measured Intensity Multiplier cannot exceed 0.2.*

	ASU1-1	ASU1-2	ASU1-3	ASU1-4	ASU2-1	ASU2-2	ASU2-3	ASU3-1
IM	***0.0350***	***0.2810***	***0.0700***	***0.2100***	***0.0180***	***0.0700***	***0.0350***	***0.2810***
MIM	0.0350	0.2810	0.0700	0.2100	0.0180	0.0700	0.0350	0.2810
COV	0.001	0.000	0.001	0.001	0.002	0.001	0.001	0.000

Primary Metrics Test – IOPS Test Phase

Clause 5.4.4.2

The IOPS Test Phase consists of one Test Run at the 100% load point with a Measurement Interval of ten (10) minutes. The IOPS Test Phase immediately follows the Sustainability Test Phase without any interruption or manual intervention.

The IOPS Test Run generates the SPC-1 IOPS™ primary metric, which is computed as the I/O Request Throughput for the Measurement Interval of the IOPS Test Run.

The Average Response Time is computed for the IOPS Test Run and cannot exceed 30 milliseconds. If the Average Response Time exceeds the 30 millisecond constraint, the measurement is invalid.

Clause 9.4.3.7.2

For the IOPS Test Phase the FDR shall contain:

1. *I/O Request Throughput Distribution (data and graph).*
2. *A Response Time Frequency Distribution.*
3. *An Average Response Time Distribution.*
4. *The human readable Test Run Results File produced by the Workload Generator.*
5. *A listing or screen image of all input parameters supplied to the Workload Generator.*
6. *The total number of I/O Requests completed in the Measurement Interval as well as the number of I/O Requests with a Response Time less than or equal to 30 milliseconds and the number of I/O Requests with a Response Time greater than 30 milliseconds.*

SPC-1 Workload Generator Input Parameters

The SPC-1 Workload Generator input parameters for the Sustainability, IOPS, Response Time Ramp, Repeatability, and Persistence Test Runs are documented in "Appendix E: SPC-1 Workload Generator Input Parameters" on Page 93.

IOPS Test Results File

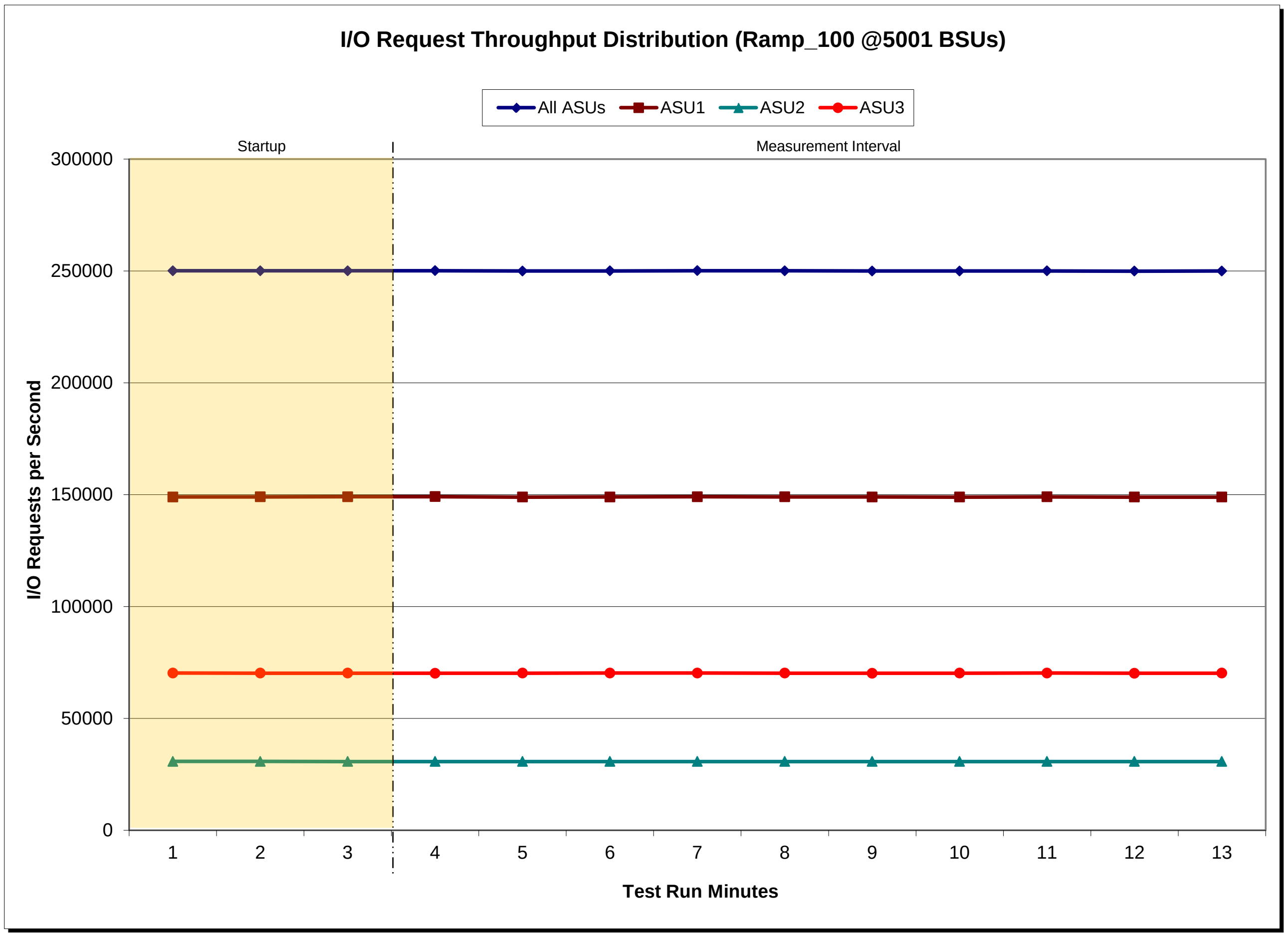
A link to the test results file generated from the IOPS Test Run is listed below.

IOPS Test Results File

IOPS Test Run – I/O Request Throughput Distribution Data

5,001 BSUs	**Start**	**Stop**	**Interval**	**Duration**
Start-Up/Ramp-Up	11:47:28	11:50:29	0-2	0:03:01
Measurement Interval	11:50:29	12:00:29	3-12	0:10:00
60 second intervals	**All ASUs**	**ASU1**	**ASU2**	**ASU3**
0	250,124.05	149,016.60	30,789.40	70,318.05
1	250,092.80	149,036.82	30,801.05	70,254.93
2	250,109.73	149,104.63	30,758.83	70,246.27
3	250,144.40	149,157.47	30,763.68	70,223.25
4	249,995.65	148,969.42	30,772.28	70,253.95
5	250,057.82	148,997.83	30,772.08	70,287.90
6	250,153.72	149,087.33	30,770.58	70,295.80
7	250,084.57	149,075.35	30,748.05	70,261.17
8	249,982.35	148,998.87	30,773.10	70,210.38
9	249,977.03	148,966.12	30,743.43	70,267.48
10	250,072.13	149,032.12	30,745.87	70,294.15
11	249,943.07	148,963.45	30,752.52	70,227.10
12	249,985.98	148,974.77	30,775.58	70,235.63
Average	***250,039.67***	***149,022.27***	***30,761.72***	***70,255.68***

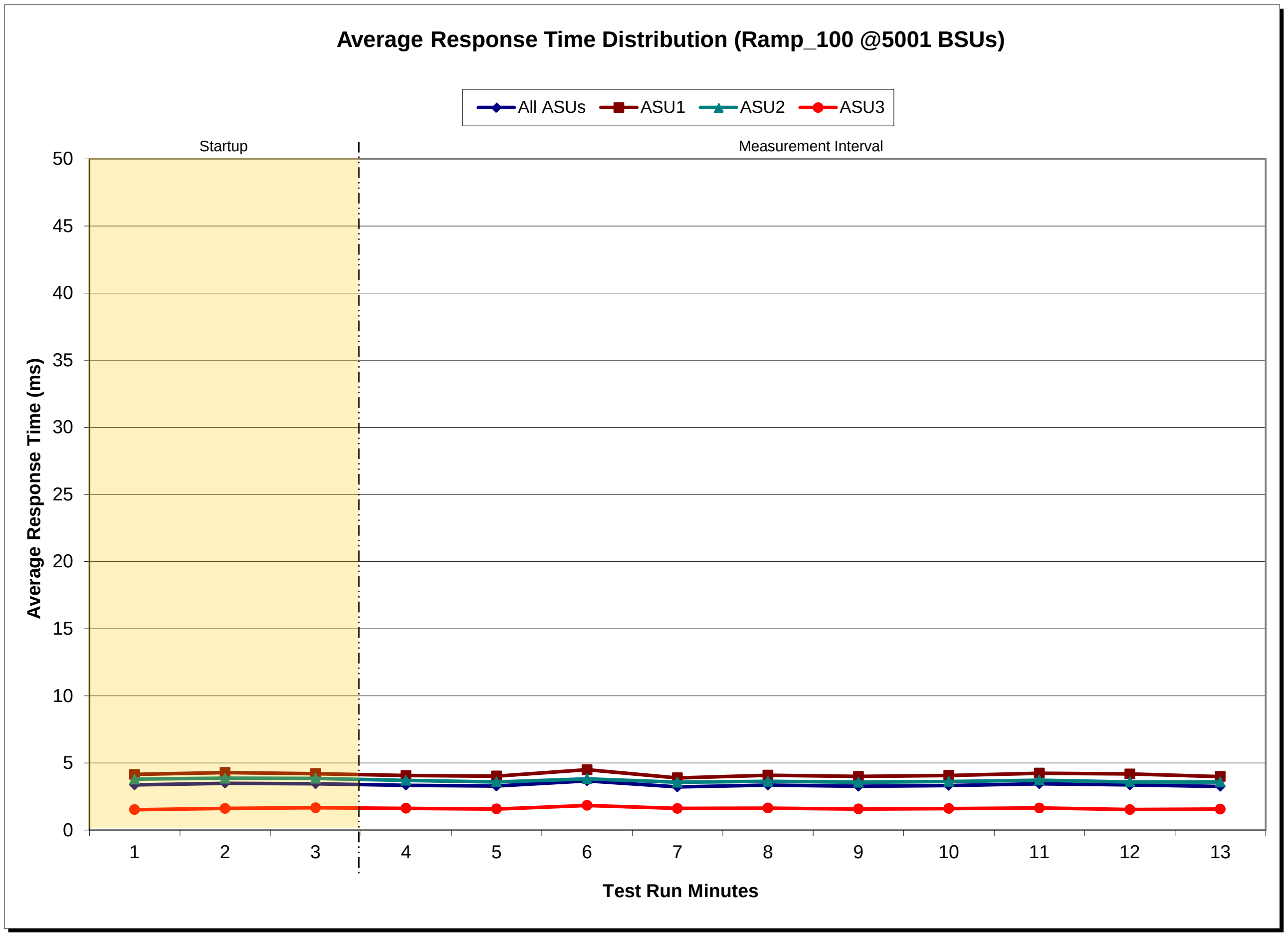
IOPS Test Run – I/O Request Throughput Distribution Graph



IOPS Test Run – Average Response Time (ms) Distribution Data

5,001 BSUs	**Start**	**Stop**	**Interval**	**Duration**
Start-Up/Ramp-Up	11:47:28	11:50:29	0-2	0:03:01
Measurement Interval	11:50:29	12:00:29	3-12	0:10:00
60 second intervals	**All ASUs**	**ASU1**	**ASU2**	**ASU3**
0	3.37	4.15	3.80	1.53
1	3.49	4.29	3.87	1.62
2	3.46	4.21	3.86	1.67
3	3.34	4.07	3.71	1.63
4	3.29	4.03	3.59	1.58
5	3.67	4.50	3.83	1.84
6	3.22	3.90	3.57	1.62
7	3.35	4.09	3.64	1.65
8	3.26	4.00	3.57	1.57
9	3.33	4.08	3.63	1.61
10	3.45	4.24	3.72	1.65
11	3.37	4.18	3.60	1.54
12	3.26	3.99	3.59	1.57
Average	***3.35***	***4.11***	***3.64***	***1.63***

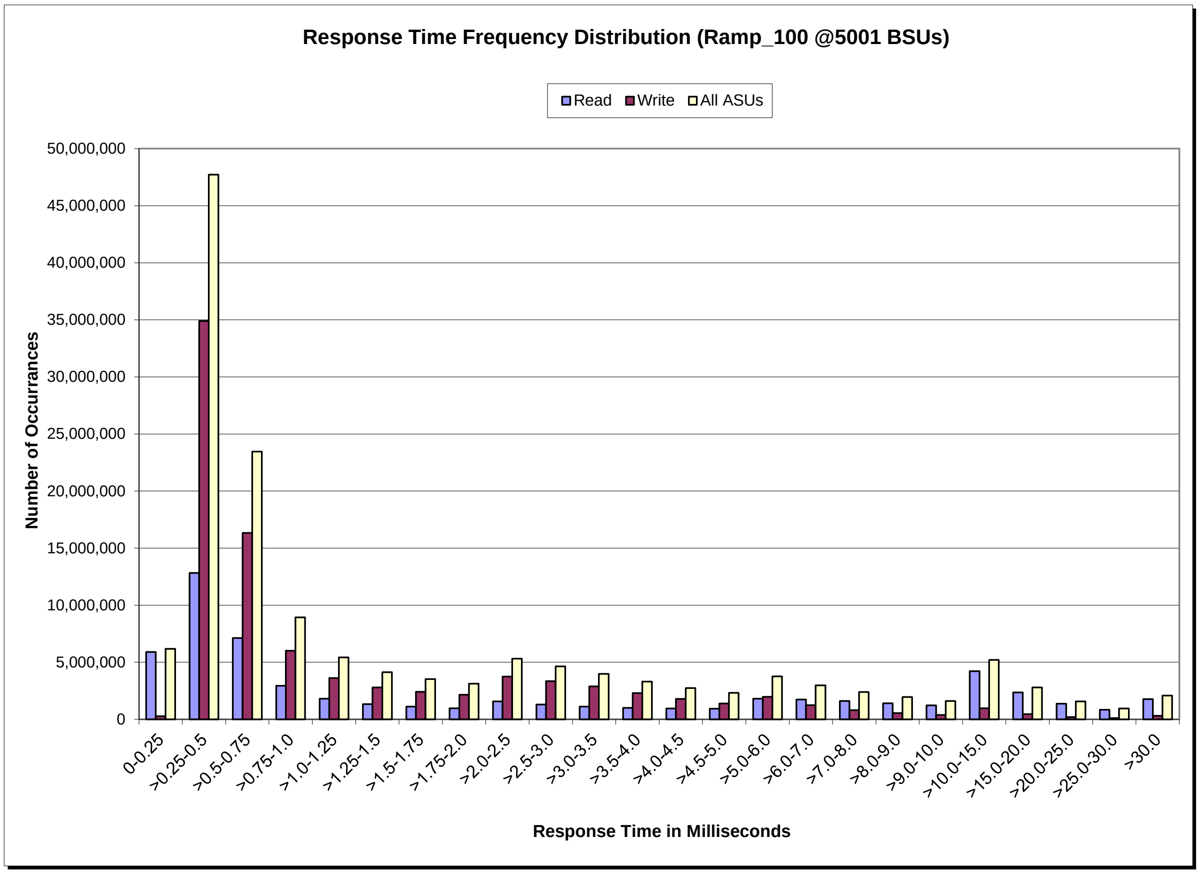
IOPS Test Run – Average Response Time (ms) Distribution Graph



IOPS Test Run – Response Time Frequency Distribution Data

Response Time (ms)	0-0.25	>0.25-0.5	>0.5-0.75	>0.75-1.0	>1.0-1.25	>1.25-1.5	>1.5-1.75	>1.75-2.0
Read	5,897,336	12,831,536	7,119,470	2,935,417	1,799,746	1,334,934	1,115,266	956,778
Write	267,849	34,895,667	16,332,666	6,001,449	3,620,118	2,786,163	2,407,475	2,158,418
All ASUs	6,165,185	47,727,203	23,452,136	8,936,866	5,419,864	4,121,097	3,522,741	3,115,196
ASU1	4,698,311	26,504,670	12,924,554	5,025,453	3,059,979	2,309,562	1,960,289	1,720,334
ASU2	1,364,678	5,471,893	2,425,036	1,041,851	640,723	487,615	417,803	367,922
ASU3	102,196	15,750,640	8,102,546	2,869,562	1,719,162	1,323,920	1,144,649	1,026,940
Response Time (ms)	**>2.0-2.5**	**>2.5-3.0**	**>3.0-3.5**	**>3.5-4.0**	**>4.0-4.5**	**>4.5-5.0**	**>5.0-6.0**	**>6.0-7.0**
Read	1,570,943	1,292,904	1,109,485	1,001,525	950,576	921,524	1,805,651	1,738,603
Write	3,742,758	3,345,087	2,878,352	2,299,947	1,792,355	1,390,200	1,963,945	1,232,803
All ASUs	5,313,701	4,637,991	3,987,837	3,301,472	2,742,931	2,311,724	3,769,596	2,971,406
ASU1	2,909,510	2,499,747	2,141,903	1,803,866	1,551,253	1,365,112	2,369,955	2,018,788
ASU2	621,410	540,313	465,398	391,780	334,734	288,560	490,052	402,735
ASU3	1,782,781	1,597,931	1,380,536	1,105,826	856,944	658,052	909,589	549,883
Response Time (ms)	**>7.0-8.0**	**>8.0-9.0**	**>9.0-10.0**	**>10.0-15.0**	**>15.0-20.0**	**>20.0-25.0**	**>25.0-30.0**	**>30.0**
Read	1,599,557	1,410,841	1,222,069	4,225,282	2,352,311	1,359,991	838,207	1,769,650
Write	794,430	537,132	380,199	973,664	447,258	202,748	101,740	310,774
All ASUs	2,393,987	1,947,973	1,602,268	5,198,946	2,799,569	1,562,739	939,947	2,080,424
ASU1	1,719,650	1,451,162	1,223,619	4,078,583	2,233,663	1,280,823	785,513	1,776,339
ASU2	334,404	278,432	232,941	781,077	422,832	234,267	139,002	281,446
ASU3	339,933	218,379	145,708	339,286	143,074	47,649	15,432	22,639

IOPS Test Run –Response Time Frequency Distribution Graph



IOPS Test Run – I/O Request Information

I/O Requests Completed in the Measurement Interval	I/O Requests Completed with Response Time = or < 30 ms	I/O Requests Completed with Response Time > 30 ms
150,022,799	147,942,375	2,080,424

IOPS Test Run – Measured Intensity Multiplier and Coefficient of Variation

Clause 3.4.3

IM – Intensity Multiplier: *The ratio of I/Os for each I/O stream relative to the total I/Os for all I/O streams (ASU1-1 – ASU3-1) as required by the benchmark specification.*

Clauses 5.1.10 and 5.3.13.2

MIM – Measured Intensity Multiplier: *The Measured Intensity Multiplier represents the ratio of measured I/Os for each I/O stream relative to the total I/Os measured for all I/O streams (ASU1-1 – ASU3-1). This value may differ from the corresponding Expected Intensity Multiplier by no more than 5%.*

Clause 5.3.13.3

COV – Coefficient of Variation: *This measure of variation for the Measured Intensity Multiplier cannot exceed 0.2.*

	ASU1-1	ASU1-2	ASU1-3	ASU1-4	ASU2-1	ASU2-2	ASU2-3	ASU3-1
IM	***0.0350***	***0.2810***	***0.0700***	***0.2100***	***0.0180***	***0.0700***	***0.0350***	***0.2810***
MIM	0.0350	0.2810	0.0700	0.2100	0.0180	0.0700	0.0350	0.2810
COV	0.001	0.000	0.001	0.000	0.002	0.001	0.001	0.000

Primary Metrics Test – Response Time Ramp Test Phase

Clause 5.4.4.3

The Response Time Ramp Test Phase consists of five Test Runs, one each at 95%, 90%, 80%, 50%, and 10% of the load point (100%) used to generate the SPC-1 IOPS™ primary metric. Each of the five Test Runs has a Measurement Interval of ten (10) minutes. The Response Time Ramp Test Phase immediately follows the IOPS Test Phase without any interruption or manual intervention.

The five Response Time Ramp Test Runs, in conjunction with the IOPS Test Run (100%), demonstrate the relationship between Average Response Time and I/O Request Throughput for the Tested Storage Configuration (TSC) as illustrated in the response time/throughput curve on page 13.

In addition, the Average Response Time measured during the 10% Test Run is the value for the SPC-1 LRT™ metric. That value represents the Average Response Time of a lightly loaded TSC.

Clause 9.4.3.7.3

The following content shall appear in the FDR for the Response Time Ramp Phase:

1. *A Response Time Ramp Distribution.*
2. *The human readable Test Run Results File produced by the Workload Generator for each Test Run within the Response Time Ramp Test Phase.*
3. *For the 10% Load Level Test Run (SPC-1 LRT™ metric) an Average Response Time Distribution.*
4. *A listing or screen image of all input parameters supplied to the Workload Generator.*

SPC-1 Workload Generator Input Parameters

The SPC-1 Workload Generator input parameters for the Sustainability, IOPS, Response Time Ramp, Repeatability, and Persistence Test Runs are documented in "Appendix E: SPC-1 Workload Generator Input Parameters" on Page 93.

Response Time Ramp Test Results File

A link to each test result file generated from each Response Time Ramp Test Run list listed below.

95% Load Level

90% Load Level

80% Load Level

50% Load Level

10% Load Level

Response Time Ramp Distribution (IOPS) Data

The five Test Runs that comprise the Response Time Ramp Phase are executed at 95%, 90%, 80%, 50%, and 10% of the Business Scaling Unit (BSU) load level used to produce the SPC-1 IOPS™ primary metric. The 100% BSU load level is included in the following Response Time Ramp data tables and graphs for completeness.

100% Load Level - 5,001 BSUs	**Start**	**Stop**	**Interval**	**Duration**
Start-Up/Ramp-Up	11:47:28	11:50:29	0-2	0:03:01
Measurement Interval	11:50:29	12:00:29	3-12	0:10:00
(60 second intervals)	**All ASUs**	**ASU-1**	**ASU-2**	**ASU-3**
0	250,124.05	149,016.60	30,789.40	70,318.05
1	250,092.80	149,036.82	30,801.05	70,254.93
2	250,109.73	149,104.63	30,758.83	70,246.27
3	250,144.40	149,157.47	30,763.68	70,223.25
4	249,995.65	148,969.42	30,772.28	70,253.95
5	250,057.82	148,997.83	30,772.08	70,287.90
6	250,153.72	149,087.33	30,770.58	70,295.80
7	250,084.57	149,075.35	30,748.05	70,261.17
8	249,982.35	148,998.87	30,773.10	70,210.38
9	249,977.03	148,966.12	30,743.43	70,267.48
10	250,072.13	149,032.12	30,745.87	70,294.15
11	249,943.07	148,963.45	30,752.52	70,227.10
12	249,985.98	148,974.77	30,775.58	70,235.63
Average	***250,039.67***	***149,022.27***	***30,761.72***	***70,255.68***

95% Load Level - 4,750 BSUs	**Start**	**Stop**	**Interval**	**Duration**
Start-Up/Ramp-Up	12:01:57	12:04:58	0-2	0:03:01
Measurement Interval	12:04:58	12:14:58	3-12	0:10:00
(60 second intervals)	**All ASUs**	**ASU-1**	**ASU-2**	**ASU-3**
0	237,669.30	141,637.78	29,217.03	66,814.48
1	237,626.77	141,725.05	29,197.60	66,704.12
2	237,617.98	141,618.62	29,240.17	66,759.20
3	237,548.98	141,567.05	29,249.13	66,732.80
4	237,529.20	141,523.57	29,215.77	66,789.87
5	237,531.17	141,555.08	29,192.30	66,783.78
6	237,335.93	141,422.32	29,179.92	66,733.70
7	237,377.72	141,509.38	29,155.47	66,712.87
8	237,489.52	141,545.10	29,218.38	66,726.03
9	237,584.48	141,588.30	29,212.38	66,783.80
10	237,528.22	141,648.95	29,205.27	66,674.00
11	237,503.05	141,556.78	29,219.97	66,726.30
12	237,449.28	141,519.53	29,195.10	66,734.65
Average	***237,487.76***	***141,543.61***	***29,204.37***	***66,739.78***

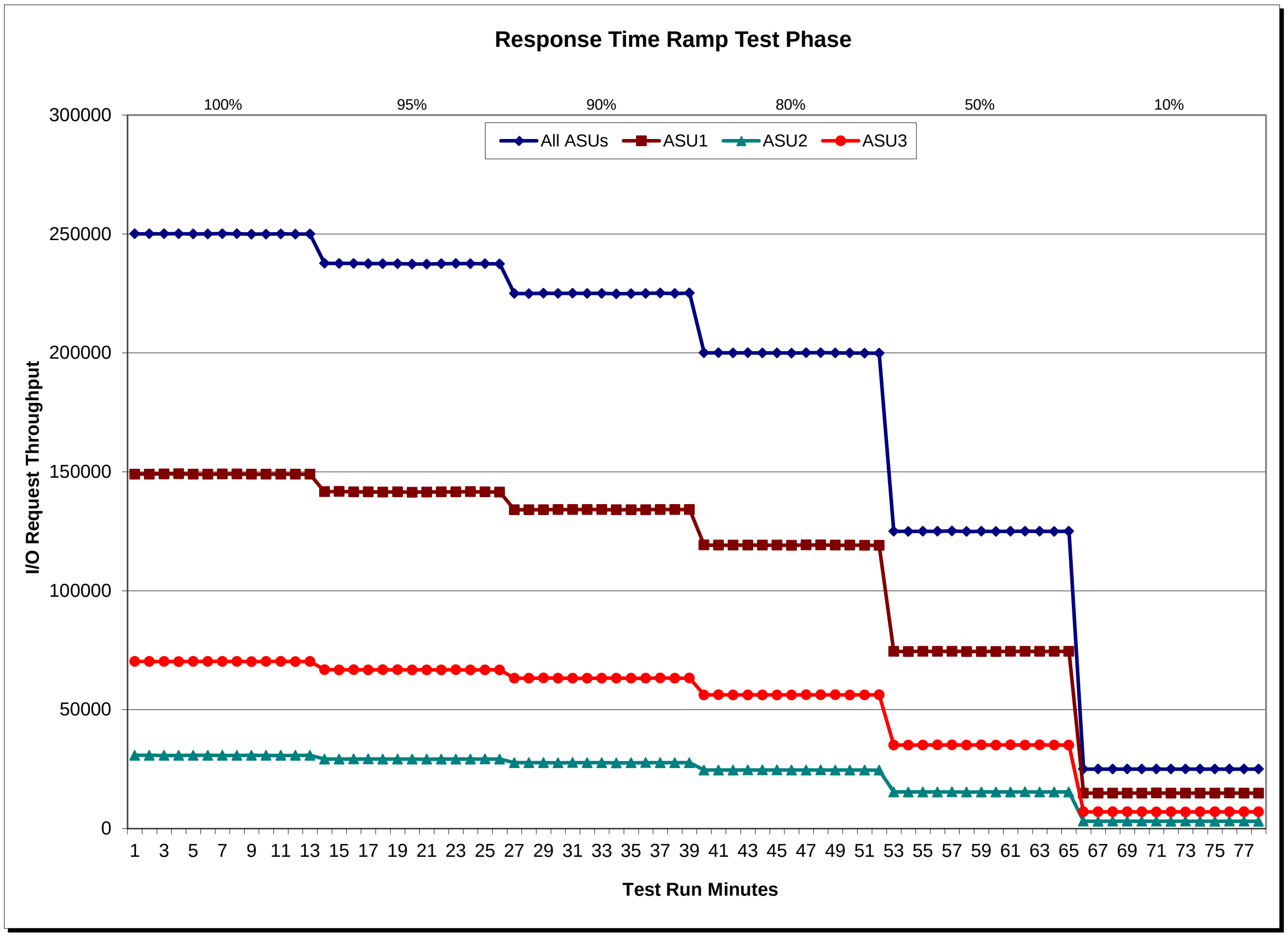
90% Load Level - 4,500 BSUs	**Start**	**Stop**	**Interval**	**Duration**
Start-Up/Ramp-Up	12:16:19	12:19:20	0-2	0:03:01
Measurement Interval	12:19:20	12:29:20	3-12	0:10:00
(60 second intervals)	**All ASUs**	**ASU-1**	**ASU-2**	**ASU-3**
0	224,986.27	134,073.00	27,675.72	63,237.55
1	224,952.22	134,042.78	27,690.50	63,218.93
2	225,060.38	134,090.55	27,694.07	63,275.77
3	225,022.93	134,126.70	27,663.73	63,232.50
4	225,062.27	134,116.67	27,727.97	63,217.63
5	225,052.02	134,160.78	27,703.50	63,187.73
6	225,057.12	134,143.92	27,668.27	63,244.93
7	224,869.32	134,014.00	27,614.15	63,241.17
8	224,946.63	134,092.78	27,656.05	63,197.80
9	225,020.57	134,093.12	27,736.05	63,191.40
10	225,155.00	134,147.90	27,688.52	63,318.58
11	224,964.30	134,107.60	27,680.73	63,175.97
12	225,155.88	134,169.40	27,684.68	63,301.80
Average	***225,030.60***	***134,117.29***	***27,682.37***	***63,230.95***

80% Load Level - 4,000 BSUs	**Start**	**Stop**	**Interval**	**Duration**
Start-Up/Ramp-Up	12:30:42	12:33:43	0-2	0:03:01
Measurement Interval	12:33:43	12:43:43	3-12	0:10:00
(60 second intervals)	**All ASUs**	**ASU-1**	**ASU-2**	**ASU-3**
0	200,060.92	119,255.45	24,606.08	56,199.38
1	200,040.62	119,187.48	24,598.62	56,254.52
2	200,000.22	119,184.78	24,604.63	56,210.80
3	200,041.28	119,218.93	24,623.18	56,199.17
4	199,982.88	119,148.52	24,632.32	56,202.05
5	199,997.75	119,181.68	24,621.28	56,194.78
6	199,923.77	119,140.68	24,595.60	56,187.48
7	200,078.87	119,248.10	24,581.85	56,248.92
8	200,120.02	119,250.98	24,634.55	56,234.48
9	200,016.45	119,189.38	24,579.38	56,247.68
10	199,969.68	119,173.45	24,593.50	56,202.73
11	199,912.53	119,112.65	24,603.80	56,196.08
12	199,853.32	119,092.10	24,544.87	56,216.35
Average	***199,989.66***	***119,175.65***	***24,601.03***	***56,212.97***

50% Load Level - 2,500 BSUs	**Start**	**Stop**	**Interval**	**Duration**
Start-Up/Ramp-Up	12:44:58	12:47:59	0-2	0:03:01
Measurement Interval	12:47:59	12:57:59	3-12	0:10:00
(60 second intervals)	**All ASUs**	**ASU-1**	**ASU-2**	**ASU-3**
0	125,017.45	74,490.40	15,382.98	35,144.07
1	124,948.75	74,460.15	15,390.77	35,097.83
2	124,990.45	74,522.37	15,368.13	35,099.95
3	124,998.82	74,486.70	15,357.68	35,154.43
4	125,120.68	74,539.97	15,399.07	35,181.65
5	124,917.23	74,449.65	15,332.72	35,134.87
6	124,975.47	74,445.28	15,376.55	35,153.63
7	124,957.52	74,453.30	15,387.82	35,116.40
8	125,012.78	74,499.93	15,357.08	35,155.77
9	125,062.32	74,527.23	15,394.10	35,140.98
10	125,017.27	74,509.05	15,356.03	35,152.18
11	124,956.43	74,472.20	15,373.15	35,111.08
12	124,992.15	74,510.42	15,384.33	35,097.40
Average	***125,001.07***	***74,489.37***	***15,371.85***	***35,139.84***

10% Load Level - 500 BSUs	**Start**	**Stop**	**Interval**	**Duration**
Start-Up/Ramp-Up	12:59:09	13:02:10	0-2	0:03:01
Measurement Interval	13:02:10	13:12:10	3-12	0:10:00
(60 second intervals)	**All ASUs**	**ASU-1**	**ASU-2**	**ASU-3**
0	25,000.53	14,894.40	3,078.95	7,027.18
1	25,005.03	14,911.63	3,064.90	7,028.50
2	25,019.95	14,915.58	3,078.48	7,025.88
3	25,005.42	14,903.70	3,081.10	7,020.62
4	25,019.08	14,897.18	3,080.18	7,041.72
5	25,030.78	14,937.28	3,073.42	7,020.08
6	24,997.08	14,901.52	3,064.65	7,030.92
7	25,006.98	14,912.52	3,080.68	7,013.78
8	25,007.80	14,904.82	3,064.97	7,038.02
9	24,987.80	14,898.33	3,064.72	7,024.75
10	25,029.18	14,922.35	3,075.25	7,031.58
11	25,007.20	14,903.28	3,074.25	7,029.67
12	25,000.65	14,905.52	3,066.90	7,028.23
Average	***25,009.20***	***14,908.65***	***3,072.61***	***7,027.94***

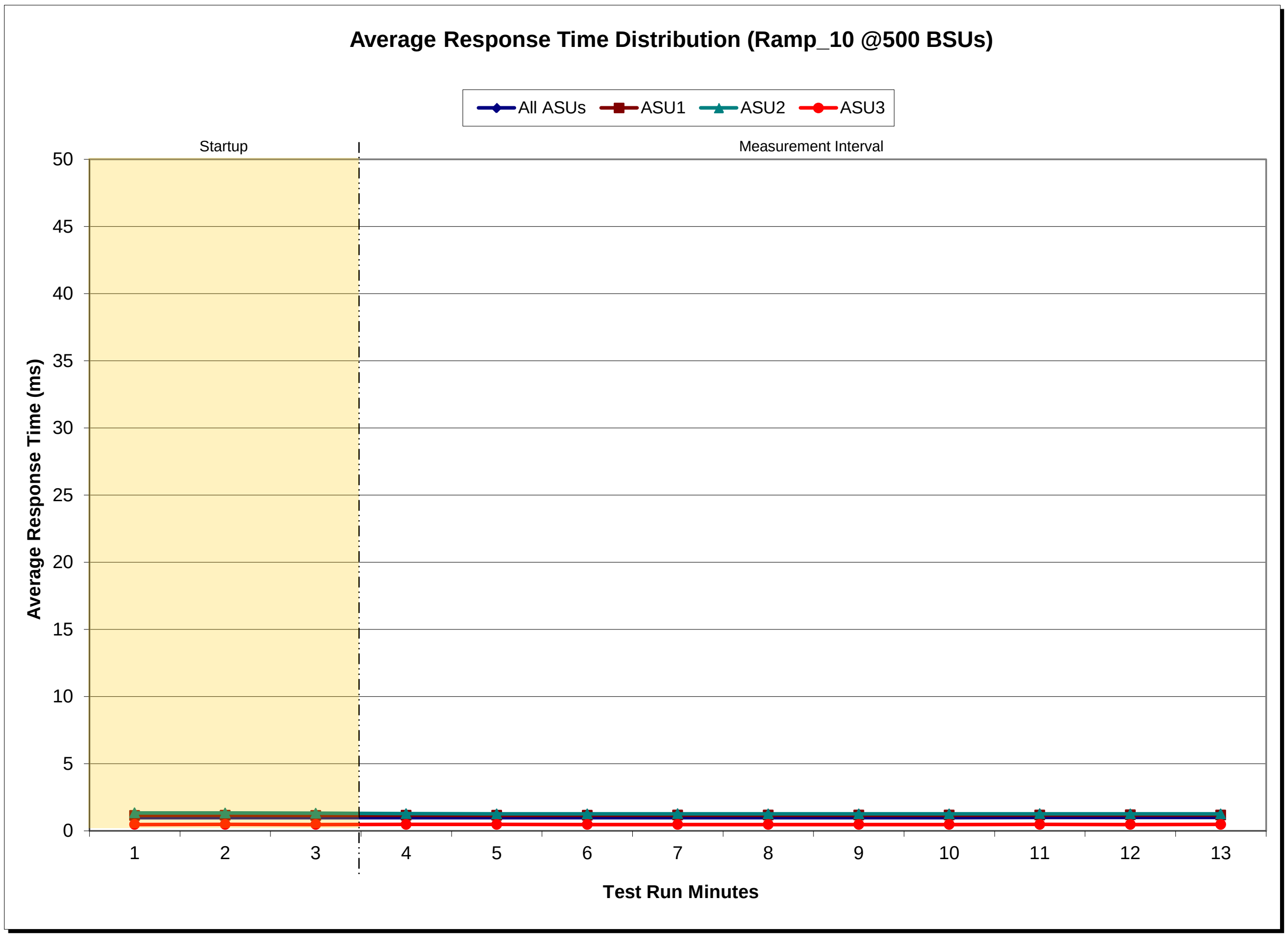
Response Time Ramp Distribution (IOPS) Graph



SPC-1 LRT™ Average Response Time (ms) Distribution Data

500 BSUs	**Start**	**Stop**	**Interval**	**Duration**
Start-Up/Ramp-Up	12:59:09	13:02:10	0-2	0:03:01
Measurement Interval	13:02:10	13:12:10	3-12	0:10:00
60 second intervals	**All ASUs**	**ASU1**	**ASU2**	**ASU3**
0	0.99	1.15	1.34	0.48
1	0.99	1.17	1.33	0.49
2	0.98	1.15	1.32	0.48
3	0.99	1.17	1.28	0.48
4	0.98	1.16	1.27	0.48
5	0.99	1.17	1.27	0.48
6	0.99	1.17	1.28	0.48
7	0.99	1.18	1.27	0.48
8	0.99	1.18	1.27	0.48
9	0.99	1.18	1.27	0.48
10	1.00	1.19	1.28	0.48
11	1.00	1.19	1.27	0.48
12	1.00	1.19	1.27	0.48
Average	***0.99***	***1.18***	***1.27***	***0.48***

SPC-1 LRT™ Average Response Time (ms) Distribution Graph



SPC-1 LRT™ (10%) – Measured Intensity Multiplier and Coefficient of Variation

Clause 3.4.3

IM – Intensity Multiplier: *The ratio of I/Os for each I/O stream relative to the total I/Os for all I/O streams (ASU1-1 – ASU3-1) as required by the benchmark specification.*

Clauses 5.1.10 and 5.3.13.2

MIM – Measured Intensity Multiplier: *The Measured Intensity Multiplier represents the ratio of measured I/Os for each I/O stream relative to the total I/Os measured for all I/O streams (ASU1-1 – ASU3-1). This value may differ from the corresponding Expected Intensity Multiplier by no more than 5%.*

Clause 5.3.13.3

COV – Coefficient of Variation: *This measure of variation for the Measured Intensity Multiplier cannot exceed 0.2.*

	ASU1-1	ASU1-2	ASU1-3	ASU1-4	ASU2-1	ASU2-2	ASU2-3	ASU3-1
IM	***0.0350***	***0.2810***	***0.0700***	***0.2100***	***0.0180***	***0.0700***	***0.0350***	***0.2810***
MIM	0.0351	0.2811	0.0700	0.2099	0.0180	0.0700	0.0349	0.2810
COV	0.003	0.001	0.004	0.002	0.006	0.003	0.004	0.001

Repeatability Test

<u>*Clause 5.4.5*</u>

The Repeatability Test demonstrates the repeatability and reproducibility of the SPC-1 IOPS™ primary metric and the SPC-1 LRT™ metric generated in earlier Test Runs.

There are two identical Repeatability Test Phases. Each Test Phase contains two Test Runs. Each of the Test Runs will have a Measurement Interval of no less than ten (10) minutes. The two Test Runs in each Test Phase will be executed without interruption or any type of manual intervention.

The first Test Run in each Test Phase is executed at the 10% load point. The Average Response Time from each of the Test Runs is compared to the SPC-1 LRT™ metric. Each Average Response Time value must be less than the SPC-1 LRT™ metric plus 5% or less than the SPC-1 LRT™ metric plus one (1) millisecond (ms).

The second Test Run in each Test Phase is executed at the 100% load point. The I/O Request Throughput from the Test Runs is compared to the SPC-1 IOPS™ primary metric. Each I/O Request Throughput value must be greater than the SPC-1 IOPS™ primary metric minus 5%. In addition, the Average Response Time for each Test Run cannot exceed 30 milliseconds.

If any of the above constraints are not met, the benchmark measurement is invalid.

<u>*Clause 9.4.3.7.4*</u>

The following content shall appear in the FDR for each Test Run in the two Repeatability Test Phases:

1. *A table containing the results of the Repeatability Test.*
2. *An I/O Request Throughput Distribution graph and table.*
3. *An Average Response Time Distribution graph and table.*
4. *The human readable Test Run Results File produced by the Workload Generator.*
5. *A listing or screen image of all input parameters supplied to the Workload Generator.*

SPC-1 Workload Generator Input Parameters

The SPC-1 Workload Generator input parameters for the Sustainability, IOPS, Response Time Ramp, Repeatability, and Persistence Test Runs are documented in "Appendix E: SPC-1 Workload Generator Input Parameters" on Page 93.

Repeatability Test Results File

The values for the SPC-1 IOPS™, SPC-1 LRT™, and the Repeatability Test measurements are listed in the tables below.

	SPC-1 IOPS™
Primary Metrics	***250,039.67***
Repeatability Test Phase 1	250,034.98
Repeatability Test Phase 2	250,035.12

The SPC-1 IOPS™ values in the above table were generated using 100% of the specified Business Scaling Unit (BSU) load level. Each of the Repeatability Test Phase values for SPC-1 IOPS™ must greater than 95% of the reported SPC-1 IOPS™ Primary Metric.

	SPC-1 LRT™
Primary Metrics	***0.99 ms***
Repeatability Test Phase 1	1.00 ms
Repeatability Test Phase 2	0.99 ms

The average response time values in the SPC-1 LRT™ column were generated using 10% of the specified Business Scaling Unit (BSU) load level. Each of the Repeatability Test Phase values for SPC-1 LRT™ must be less than 105% of the reported SPC-1 LRT™ Primary Metric or less than the reported SPC-1 LRT™ Primary Metric minus one (1) millisecond (ms)...

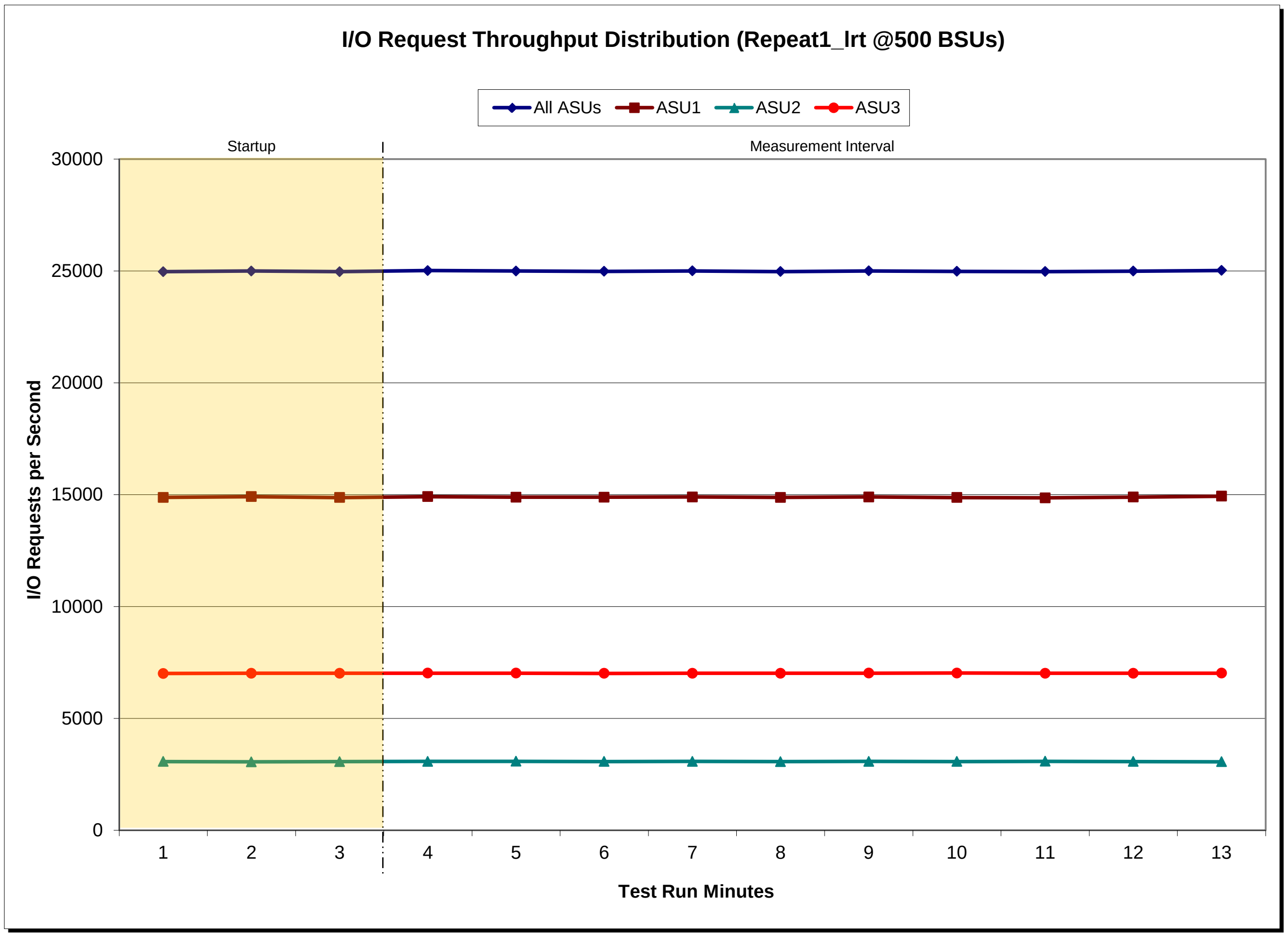
A link to the test result file generated from each Repeatability Test Run is listed below.

Repeatability Test Phase 1, Test Run 1 (LRT)
Repeatability Test Phase 1, Test Run 2 (IOPS)
Repeatability Test Phase 2, Test Run 1 (LRT)
Repeatability Test Phase 2, Test Run 2 (IOPS)

Repeatability 1 LRT – I/O Request Throughput Distribution Data

500 BSUs	Start	Stop	Interval	Duration
Start-Up/Ramp-Up	13:15:31	13:18:31	0-2	0:03:00
Measurement Interval	13:18:31	13:28:31	3-12	0:10:00
60 second intervals	**All ASUs**	**ASU1**	**ASU2**	**ASU3**
0	24,969.22	14,880.20	3,078.65	7,010.37
1	25,000.38	14,916.50	3,061.28	7,022.60
2	24,971.88	14,876.95	3,073.17	7,021.77
3	25,020.77	14,913.47	3,079.30	7,028.00
4	25,001.78	14,891.68	3,085.03	7,025.07
5	24,983.35	14,890.02	3,075.52	7,017.82
6	25,003.10	14,901.88	3,079.83	7,021.38
7	24,974.65	14,880.82	3,071.98	7,021.85
8	25,005.17	14,899.32	3,080.13	7,025.72
9	24,983.32	14,876.17	3,076.45	7,030.70
10	24,972.72	14,862.18	3,088.40	7,022.13
11	24,994.70	14,894.90	3,076.35	7,023.45
12	25,027.30	14,936.12	3,066.67	7,024.52
Average	***24,996.69***	***14,894.66***	***3,077.97***	***7,024.06***

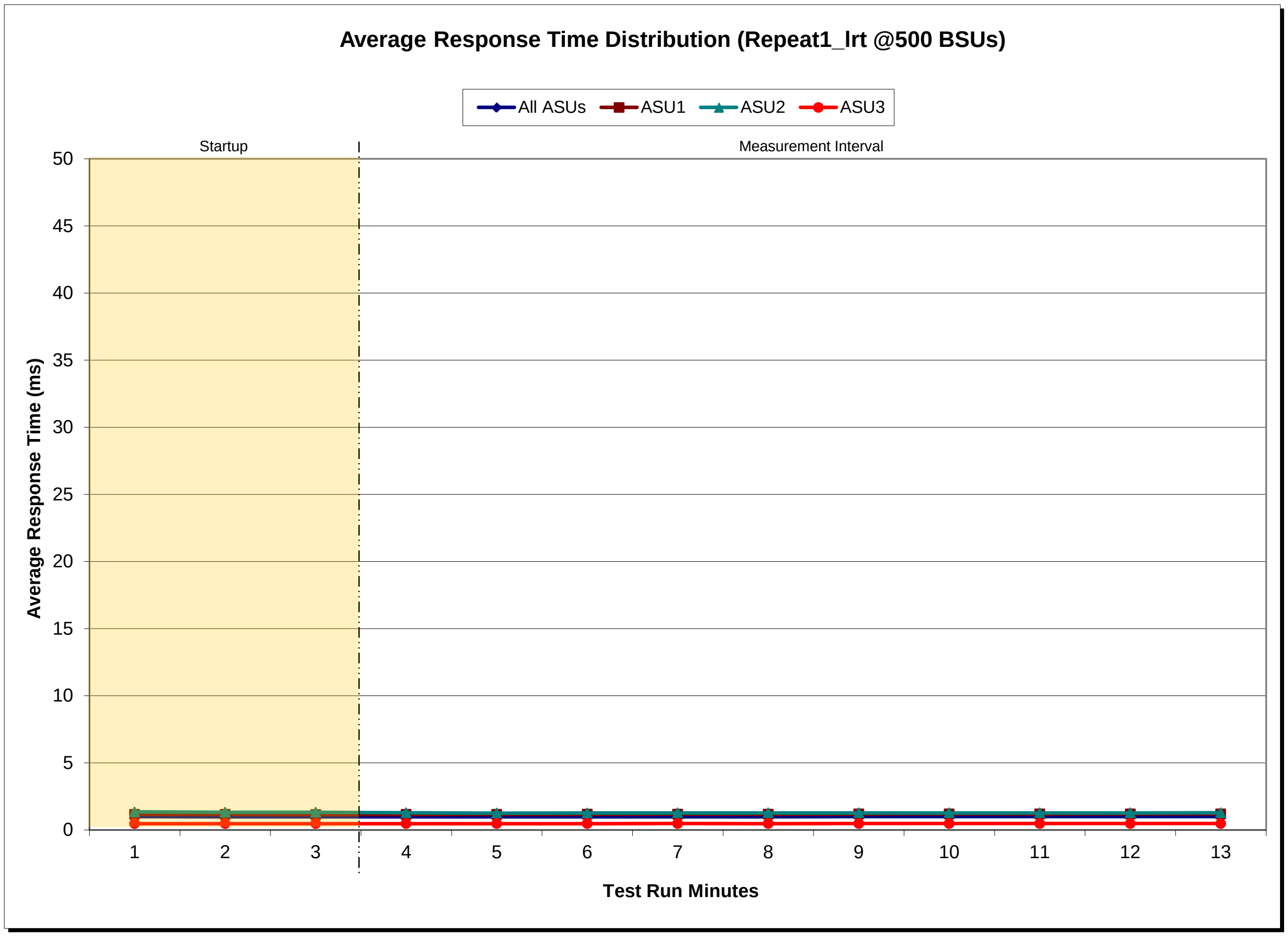
Repeatability 1 LRT – I/O Request Throughput Distribution Graph



Repeatability 1 LRT –Average Response Time (ms) Distribution Data

500 BSUs	Start	Stop	Interval	Duration
Start-Up/Ramp-Up	13:15:31	13:18:31	0-2	0:03:00
Measurement Interval	13:18:31	13:28:31	3-12	0:10:00
60 second intervals	**All ASUs**	**ASU1**	**ASU2**	**ASU3**
0	1.00	1.17	1.36	0.48
1	1.00	1.17	1.32	0.48
2	0.98	1.15	1.33	0.48
3	0.98	1.16	1.28	0.48
4	0.99	1.17	1.26	0.48
5	0.99	1.18	1.27	0.48
6	1.00	1.18	1.28	0.48
7	0.99	1.18	1.28	0.48
8	1.00	1.19	1.28	0.48
9	1.01	1.20	1.27	0.48
10	1.01	1.20	1.28	0.48
11	1.01	1.20	1.28	0.48
12	1.01	1.20	1.29	0.48
Average	***1.00***	***1.19***	***1.28***	***0.48***

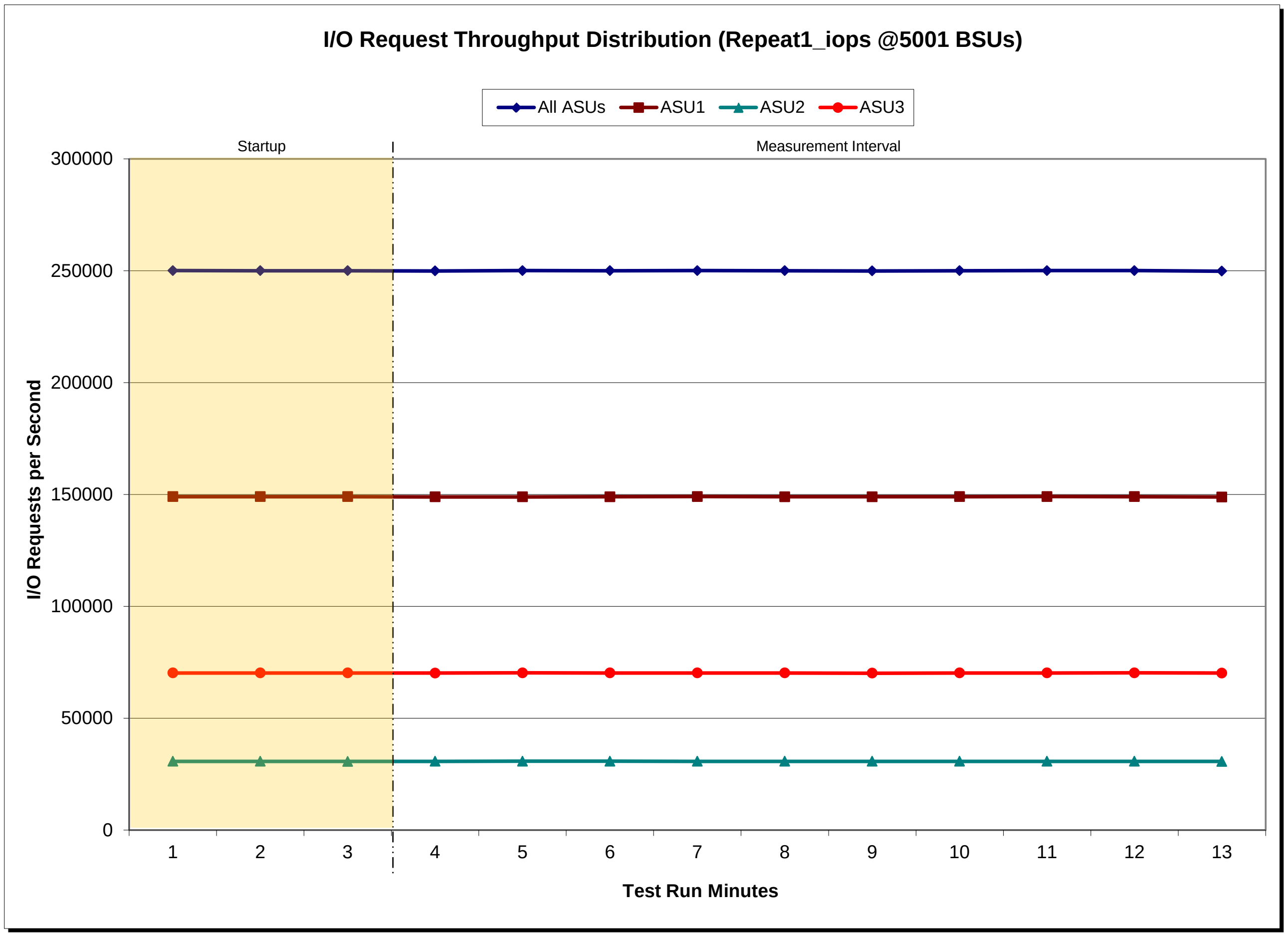
Repeatability 1 LRT –Average Response Time (ms) Distribution Graph



Repeatability 1 IOPS – I/O Request Throughput Distribution Data

5,001 BSUs	**Start**	**Stop**	**Interval**	**Duration**
Start-Up/Ramp-Up	13:29:59	13:33:00	0-2	0:03:01
Measurement Interval	13:33:00	13:43:00	3-12	0:10:00
60 second intervals	**All ASUs**	**ASU1**	**ASU2**	**ASU3**
0	250,112.23	149,079.47	30,750.65	70,282.12
1	250,047.68	149,067.58	30,741.83	70,238.27
2	250,055.42	149,069.80	30,735.92	70,249.70
3	249,971.70	148,978.73	30,765.52	70,227.45
4	250,083.20	148,972.32	30,792.58	70,318.30
5	250,045.78	149,003.15	30,805.30	70,237.33
6	250,110.72	149,088.58	30,765.87	70,256.27
7	250,040.82	148,993.45	30,762.88	70,284.48
8	249,959.32	149,030.23	30,748.53	70,180.55
9	250,046.20	149,059.90	30,749.42	70,236.88
10	250,103.95	149,096.25	30,741.82	70,265.88
11	250,115.98	149,053.37	30,774.95	70,287.67
12	249,872.17	148,920.72	30,720.62	70,230.83
Average	***250,034.98***	***149,019.67***	***30,762.75***	***70,252.57***

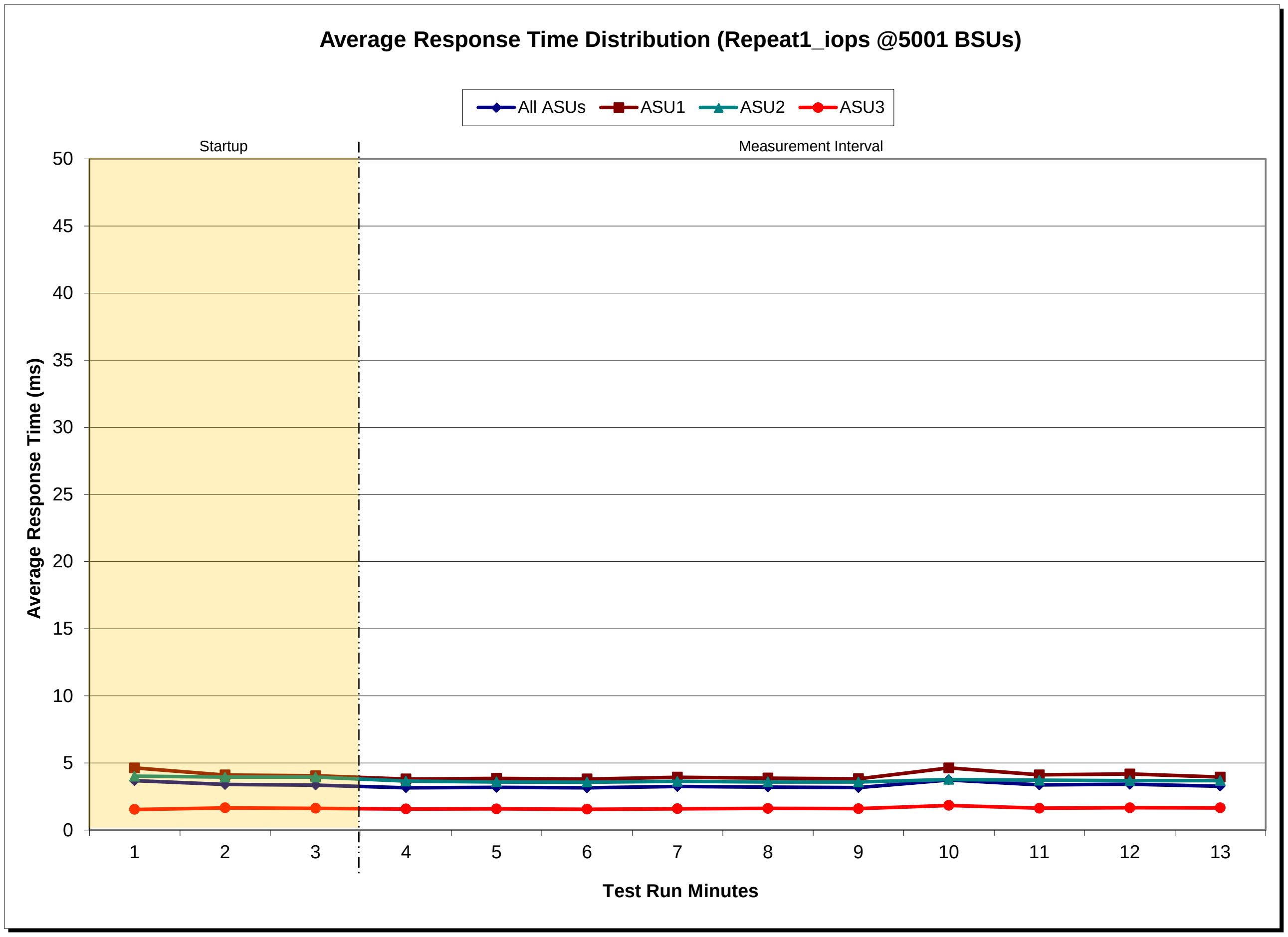
Repeatability 1 IOPS – I/O Request Throughput Distribution Graph



Repeatability 1 IOPS –Average Response Time (ms) Distribution Data

5,001 BSUs	**Start**	**Stop**	**Interval**	**Duration**
Start-Up/Ramp-Up	13:29:59	13:33:00	0-2	0:03:01
Measurement Interval	13:33:00	13:43:00	3-12	0:10:00
60 second intervals	**All ASUs**	**ASU1**	**ASU2**	**ASU3**
0	3.69	4.63	4.02	1.54
1	3.40	4.11	3.95	1.66
2	3.36	4.06	3.95	1.63
3	3.16	3.81	3.65	1.58
4	3.19	3.86	3.59	1.58
5	3.15	3.81	3.58	1.56
6	3.25	3.95	3.63	1.60
7	3.21	3.88	3.60	1.62
8	3.17	3.83	3.59	1.60
9	3.74	4.63	3.77	1.84
10	3.37	4.12	3.72	1.63
11	3.42	4.19	3.69	1.67
12	3.28	3.95	3.69	1.66
Average	***3.29***	***4.00***	***3.65***	***1.63***

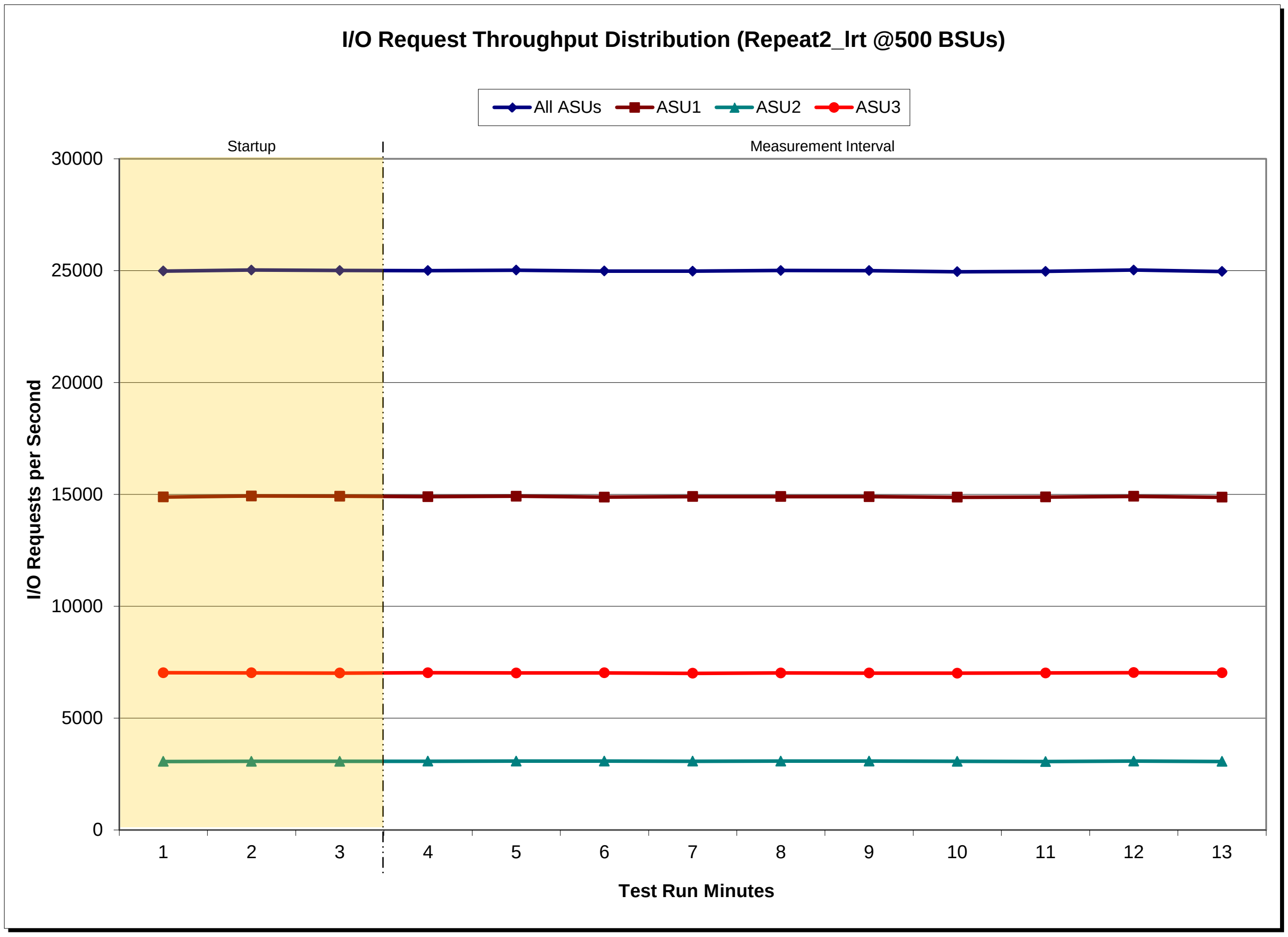
Repeatability 1 IOPS –Average Response Time (ms) Distribution Graph



Repeatability 2 LRT – I/O Request Throughput Distribution Data

500 BSUs	**Start**	**Stop**	**Interval**	**Duration**
Start-Up/Ramp-Up	13:46:21	13:49:21	0-2	0:03:00
Measurement Interval	13:49:21	13:59:21	3-12	0:10:00
60 second intervals	**All ASUs**	**ASU1**	**ASU2**	**ASU3**
0	24,986.10	14,886.93	3,067.83	7,031.33
1	25,027.80	14,929.73	3,071.00	7,027.07
2	25,011.52	14,920.17	3,073.68	7,017.67
3	25,007.52	14,902.05	3,076.05	7,029.42
4	25,022.73	14,921.62	3,079.60	7,021.52
5	24,982.70	14,878.83	3,079.47	7,024.40
6	24,982.47	14,904.72	3,074.07	7,003.68
7	25,008.35	14,907.92	3,081.77	7,018.67
8	25,003.13	14,901.50	3,083.20	7,018.43
9	24,955.25	14,873.63	3,072.00	7,009.62
10	24,968.83	14,886.53	3,061.80	7,020.50
11	25,029.10	14,913.65	3,080.65	7,034.80
12	24,967.45	14,877.40	3,065.22	7,024.83
Average	***24,992.75***	***14,896.79***	***3,075.38***	***7,020.59***

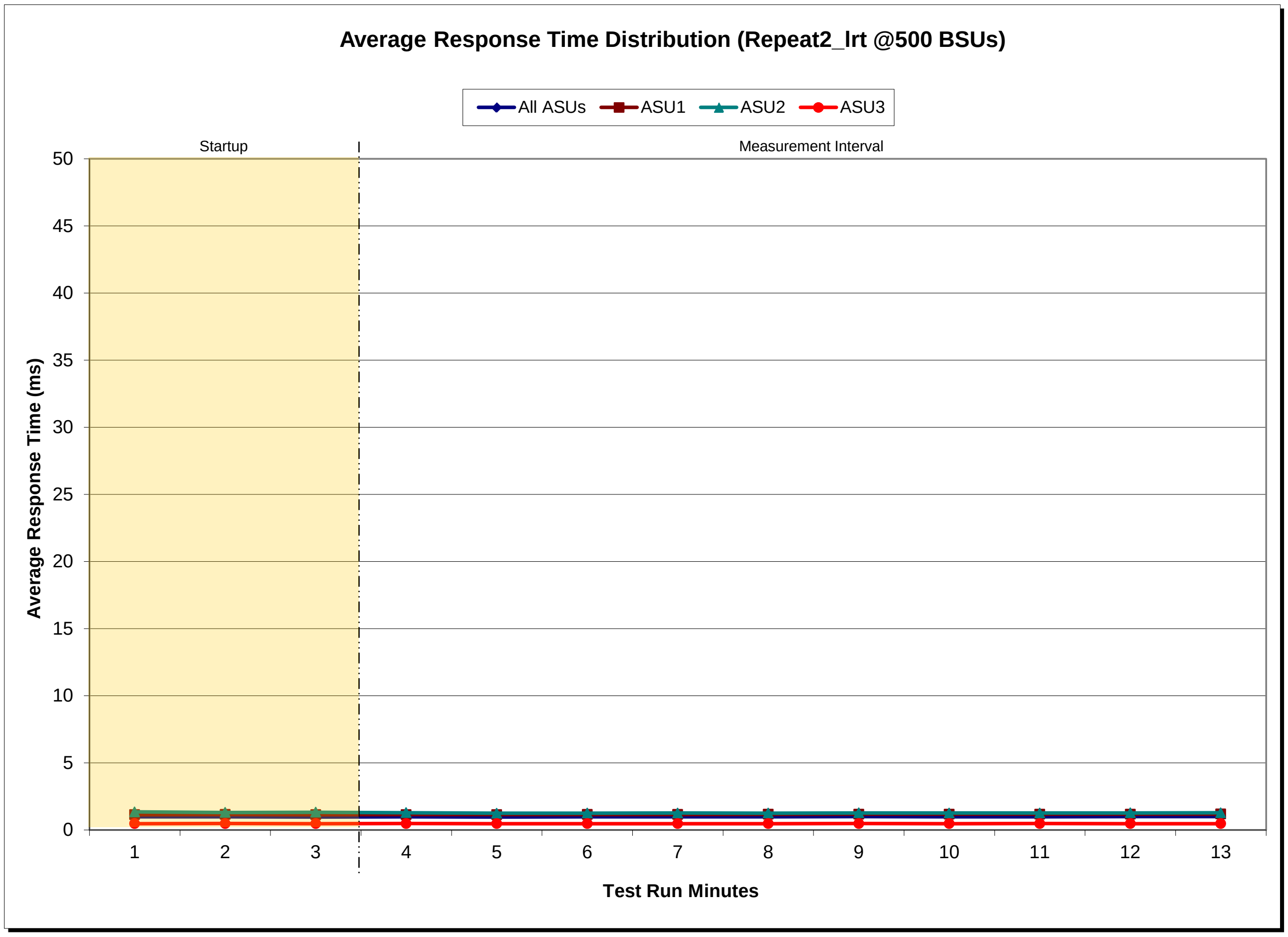
Repeatability 2 LRT – I/O Request Throughput Distribution Graph



Repeatability 2 LRT –Average Response Time (ms) Distribution Data

500 BSUs	Start	Stop	Interval	Duration
Start-Up/Ramp-Up	13:46:21	13:49:21	0-2	0:03:00
Measurement Interval	13:49:21	13:59:21	3-12	0:10:00
60 second intervals	**All ASUs**	**ASU1**	**ASU2**	**ASU3**
0	0.98	1.15	1.35	0.48
1	0.99	1.16	1.31	0.48
2	0.98	1.14	1.33	0.48
3	0.98	1.15	1.29	0.48
4	0.98	1.15	1.26	0.48
5	0.98	1.16	1.26	0.48
6	0.98	1.16	1.27	0.48
7	0.99	1.18	1.26	0.48
8	1.00	1.19	1.27	0.49
9	1.00	1.18	1.27	0.48
10	1.00	1.18	1.27	0.48
11	1.00	1.19	1.27	0.48
12	1.01	1.20	1.29	0.48
Average	***0.99***	***1.17***	***1.27***	***0.48***

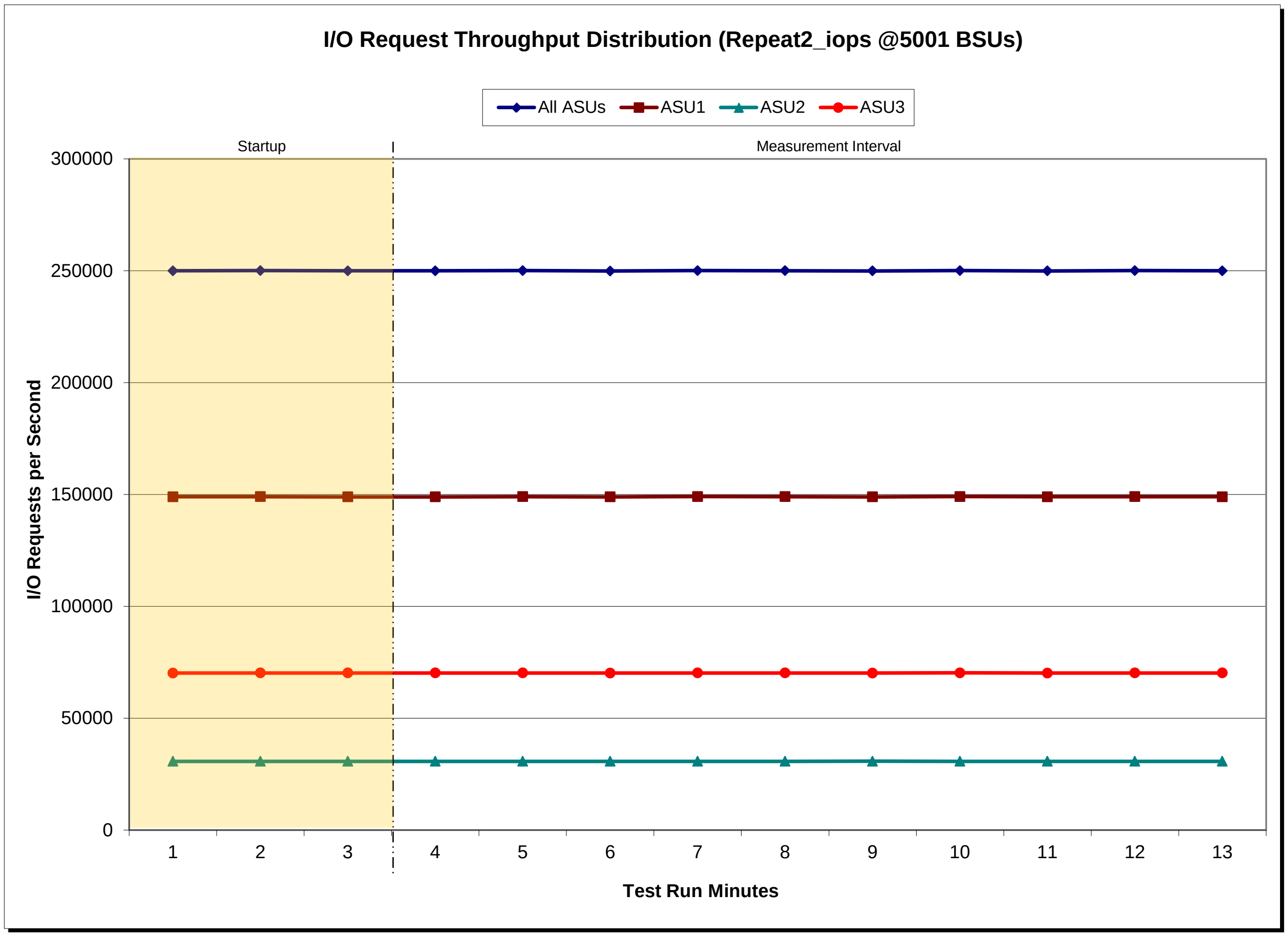
Repeatability 2 LRT –Average Response Time (ms) Distribution Graph



Repeatability 2 IOPS – I/O Request Throughput Distribution Data

5,001 BSUs	**Start**	**Stop**	**Interval**	**Duration**
Start-Up/Ramp-Up	14:00:50	14:03:51	0-2	0:03:01
Measurement Interval	14:03:51	14:13:51	3-12	0:10:00
60 second intervals	**All ASUs**	**ASU1**	**ASU2**	**ASU3**
0	249,986.43	149,008.92	30,743.48	70,234.03
1	250,109.02	149,074.12	30,772.23	70,262.67
2	249,979.80	148,960.18	30,778.40	70,241.22
3	250,002.37	148,940.18	30,778.10	70,284.08
4	250,107.85	149,062.55	30,765.58	70,279.72
5	249,917.53	148,949.57	30,768.60	70,199.37
6	250,112.37	149,094.40	30,743.90	70,274.07
7	250,062.55	149,046.20	30,760.37	70,255.98
8	249,959.37	148,938.67	30,797.18	70,223.52
9	250,121.73	149,084.15	30,747.92	70,289.67
10	249,959.25	149,008.38	30,738.32	70,212.55
11	250,102.67	149,072.78	30,771.77	70,258.12
12	250,005.50	149,006.25	30,751.95	70,247.30
Average	***250,035.12***	***149,020.31***	***30,762.37***	***70,252.44***

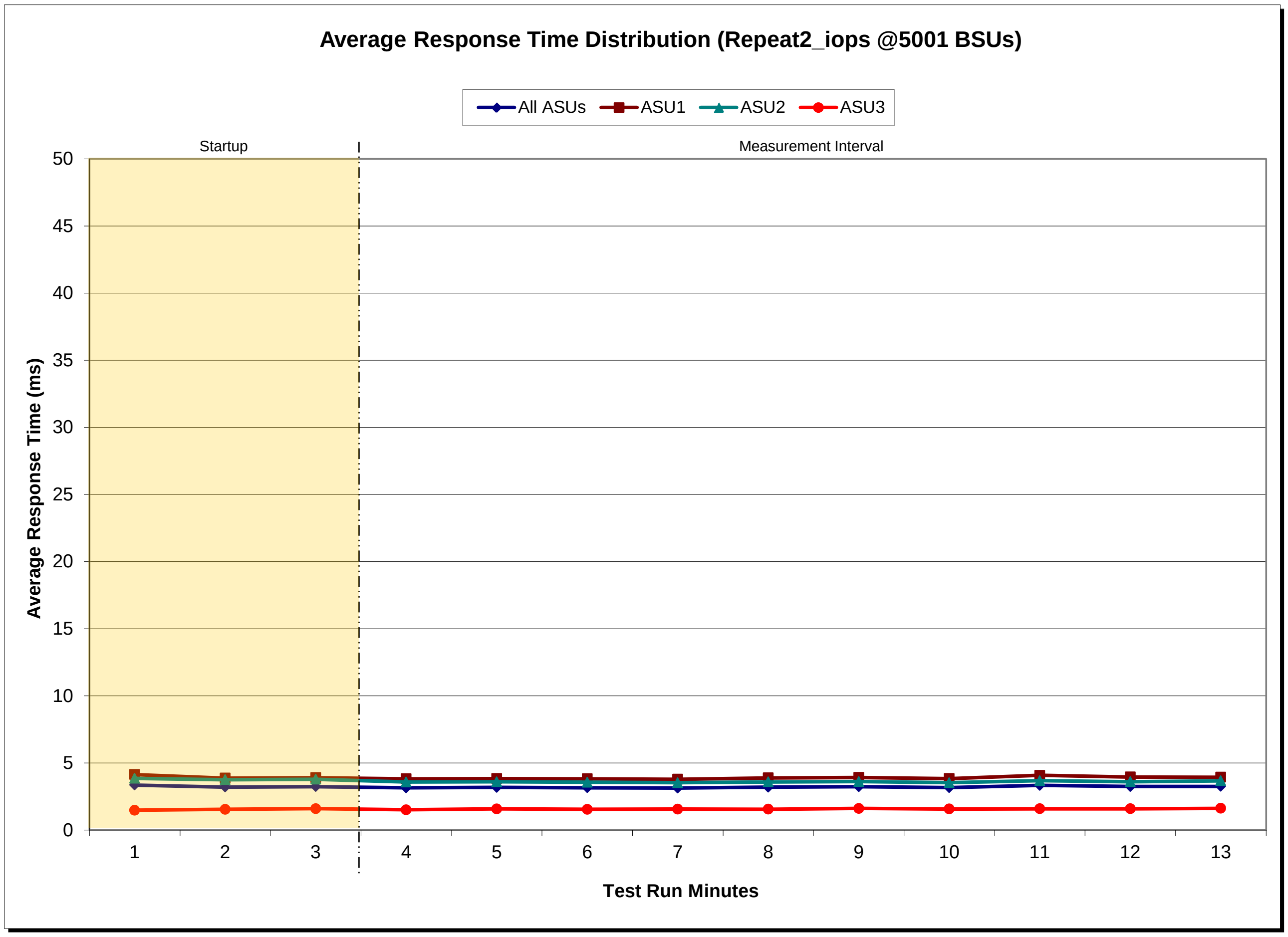
Repeatability 2 IOPS – I/O Request Throughput Distribution Graph



Repeatability 2 IOPS –Average Response Time (ms) Distribution Data

5,001 BSUs	**Start**	**Stop**	**Interval**	**Duration**
Start-Up/Ramp-Up	14:00:50	14:03:51	0-2	0:03:01
Measurement Interval	14:03:51	14:13:51	3-12	0:10:00
60 second intervals	**All ASUs**	**ASU1**	**ASU2**	**ASU3**
0	3.36	4.14	3.86	1.49
1	3.21	3.88	3.76	1.55
2	3.25	3.91	3.80	1.60
3	3.15	3.83	3.58	1.52
4	3.18	3.85	3.60	1.59
5	3.16	3.83	3.57	1.55
6	3.14	3.80	3.54	1.57
7	3.20	3.89	3.59	1.56
8	3.24	3.92	3.62	1.62
9	3.17	3.84	3.53	1.58
10	3.33	4.08	3.68	1.59
11	3.25	3.96	3.61	1.59
12	3.26	3.94	3.68	1.63
Average	***3.21***	***3.89***	***3.60***	***1.58***

Repeatability 2 IOPS –Average Response Time (ms) Distribution Graph



Repeatability 1 (LRT)
Measured Intensity Multiplier and Coefficient of Variation

Clause 3.4.3

IM – Intensity Multiplier: *The ratio of I/Os for each I/O stream relative to the total I/Os for all I/O streams (ASU1-1 – ASU3-1) as required by the benchmark specification.*

Clauses 5.1.10 and 5.3.13.2

MIM – Measured Intensity Multiplier: *The Measured Intensity Multiplier represents the ratio of measured I/Os for each I/O stream relative to the total I/Os measured for all I/O streams (ASU1-1 – ASU3-1). This value may differ from the corresponding Expected Intensity Multiplier by no more than 5%.*

Clause 5.3.13.3

COV – Coefficient of Variation: *This measure of variation for the Measured Intensity Multiplier cannot exceed 0.2.*

	ASU1-1	ASU1-2	ASU1-3	ASU1-4	ASU2-1	ASU2-2	ASU2-3	ASU3-1
IM	***0.0350***	***0.2810***	***0.0700***	***0.2100***	***0.0180***	***0.0700***	***0.0350***	***0.2810***
MIM	0.0351	0.2810	0.0699	0.0180	0.0701	0.0701	0.0350	0.2810
COV	0.005	0.001	0.002	0.p002	0.005	0.003	0.005	0.001

Repeatability 1 (IOPS)
Measured Intensity Multiplier and Coefficient of Variation

	ASU1-1	ASU1-2	ASU1-3	ASU1-4	ASU2-1	ASU2-2	ASU2-3	ASU3-1
IM	***0.0350***	***0.2810***	***0.0700***	***0.2100***	***0.0180***	***0.0700***	***0.0350***	***0.2810***
MIM	0.0350	0.2811	0.0700	0.2099	0.0180	0.0700	0.0350	0.2810
COV	0.001	0.001	0.001	0.001	0.002	0.001	0.001	0.000

Repeatability 2 (LRT)
Measured Intensity Multiplier and Coefficient of Variation

	ASU1-1	ASU1-2	ASU1-3	ASU1-4	ASU2-1	ASU2-2	ASU2-3	ASU3-1
IM	***0.0350***	***0.2810***	***0.0700***	***0.2100***	***0.0180***	***0.0700***	***0.0350***	***0.2810***
MIM	0.0350	0.2811	0.0700	0.2100	0.0180	0.0700	0.0351	0.2809
COV	0.003	0.002	0.002	0.001	0.007	0.002	0.004	0.001

Repeatability 2 (IOPS)
Measured Intensity Multiplier and Coefficient of Variation

	ASU1-1	ASU1-2	ASU1-3	ASU1-4	ASU2-1	ASU2-2	ASU2-3	ASU3-1
IM	***0.0350***	***0.2810***	***0.0700***	***0.2100***	***0.0180***	***0.0700***	***0.0350***	***0.2810***
MIM	0.0350	0.2810	0.0700	0.2100	0.0180	0.0700	0.0350	0.2810
COV	0.001	0.000	0.001	0.000	0.002	0.001	0.002	0.000

Data Persistence Test

Clause 6

The Data Persistence Test demonstrates the Tested Storage Configuration (TSC):

- *Is capable of maintain data integrity across a power cycle.*
- *Ensures the transfer of data between Logical Volumes and host systems occurs without corruption or loss.*

The SPC-1 Workload Generator will write 16 block I/O requests at random over the total Addressable Storage Capacity of the TSC for ten (10) minutes at a minimum of 25% of the load used to generate the SPC-1 IOPS™ primary metric. The bit pattern selected to be written to each block as well as the address of the block will be retained in a log file.

The Tested Storage Configuration (TSC) will be shutdown and restarted using a power off/power on cycle at the end of the above sequence of write operations. In addition, any caches employing battery backup must be flushed/emptied.

The SPC-1 Workload Generator will then use the above log file to verify each block written contains the correct bit pattern.

Clause 9.4.3.8

The following content shall appear in this section of the FDR:

1. *A listing or screen image of all input parameters supplied to the Workload Generator.*
2. *For the successful Data Persistence Test Run, a table illustrating key results. The content, appearance, and format of this table are specified in Table 9-12. Information displayed in this table shall be obtained from the Test Run Results File referenced below in #3.*
3. *For the successful Data Persistence Test Run, the human readable Test Run Results file produced by the Workload Generator.*

SPC-1 Workload Generator Input Parameters

The SPC-1 Workload Generator input parameters for the Sustainability, IOPS, Response Time Ramp, Repeatability, and Persistence Test Runs are documented in "Appendix E: SPC-1 Workload Generator Input Parameters" on Page 93.

Data Persistence Test Results File

A link to each test result file generated from each Data Persistence Test is listed below.

Persistence 1 Test Results File

Persistence 2 Test Results File

Data Persistence Test Results

Data Persistence Test Results	
Data Persistence Test Run Number: 1	
Total Number of Logical Blocks Written	479,936,432
Total Number of Logical Blocks Verified	123,105,728
Total Number of Logical Blocks that Failed Verification	0
Time Duration for Writing Test Logical Blocks	10 minutes
Size in Bytes of each Logical Block	512
Number of Failed I/O Requests in the process of the Test	0

In some cases the same address was the target of multiple writes, which resulted in more Logical Blocks Written than Logical Blocks Verified. In the case of multiple writes to the same address, the pattern written and verified must be associated with the last write to that address.

PRICED STORAGE CONFIGURATION AVAILABILITY DATE

Clause 9.2.4.9

The committed delivery data for general availability (Availability Date) of all products that comprise the Priced Storage Configuration must be reported. When the Priced Storage Configuration includes products or components with different availability dates, the reported Availability Date for the Priced Storage Configuration must be the date at which all components are committed to be available.

The NetApp FAS6240 as documented in this Full Disclosure Report is currently available for customer purchase and shipment.

PRICING INFORMATION

Clause 9.4.3.3.6

The Executive Summary shall contain a pricing spreadsheet as documented in Clause 8.3.1.

Pricing information may be found in the Priced Storage Configuration Pricing section on page 14.

TESTED STORAGE CONFIGURATION (TSC) AND PRICED STORAGE CONFIGURATION DIFFERENCES

Clause 9.4.3.3.7

The Executive Summary shall contain a pricing a list of all differences between the Tested Storage Configuration (TSC) and the Priced Storage Configuration.

A list of all differences between the Tested Storage Configuration (TSC) and Priced Storage Configuration may be found in the Executive Summary portion of this document on page 14.

ANOMALIES OR IRREGULARITIES

Clause 9.4.3.10

The FDR shall include a clear and complete description of any anomalies or irregularities encountered in the course of executing the SPC-1 benchmark that may in any way call into question the accuracy, verifiability, or authenticity of information published in this FDR.

There were no anomalies or irregularities encountered during the SPC-1 Onsite Audit of the NetApp FAS6240 *(cluster)*.

APPENDIX A: SPC-1 GLOSSARY

"Decimal" *(powers of ten)* Measurement Units

In the storage industry, the terms "kilo", "mega", "giga", "tera", "peta", and "exa" are commonly used prefixes for computing performance and capacity. For the purposes of the SPC workload definitions, all of the following terms are defined in "powers of ten" measurement units.

A kilobyte (KB) is equal to 1,000 (10^3) bytes.

A megabyte (MB) is equal to 1,000,000 (10^6) bytes.

A gigabyte (GB) is equal to 1,000,000,000 (10^9) bytes.

A terabyte (TB) is equal to 1,000,000,000,000 (10^{12}) bytes.

A petabyte (PB) is equal to 1,000,000,000,000,000 (10^{15}) bytes

An exabyte (EB) is equal to 1,000,000,000,000,000,000 (10^{18}) bytes

"Binary" *(powers of two)* Measurement Units

The sizes reported by many operating system components use "powers of two" measurement units rather than "power of ten" units. The following standardized definitions and terms are also valid and may be used in this document.

A kibibyte (KiB) is equal to 1,024 (2^{10}) bytes.

A mebibyte (MiB) is equal to 1,048,576 (2^{20}) bytes.

A gigibyte (GiB) is equal to 1,073,741,824 (2^{30}) bytes.

A tebibyte (TiB) is equal to 1,099,511,627,776 (2^{40}) bytes.

A pebibyte (PiB) is equal to 1,125,899,906,842,624 (2^{50}) bytes.

An exbibyte (EiB) is equal to 1,152,921,504,606,846,967 (2^{60}) bytes.

SPC-1 Data Repository Definitions

Total ASU Capacity: The total storage capacity read and written in the course of executing the SPC-1 benchmark.

Application Storage Unit (ASU): The logical interface between the storage and SPC-1 Workload Generator. The three ASUs (Data, User, and Log) are typically implemented on one or more Logical Volume.

Logical Volume: The division of Addressable Storage Capacity into individually addressable logical units of storage used in the SPC-1 benchmark. Each Logical Volume is implemented as a single, contiguous address space.

Addressable Storage Capacity: The total storage (sum of Logical Volumes) that can be read and written by application programs such as the SPC-1 Workload Generator.

Configured Storage Capacity: This capacity includes the Addressable Storage Capacity and any other storage (parity disks, hot spares, etc.) necessary to implement the Addressable Storage Capacity.

Physical Storage Capacity: The formatted capacity of all storage devices physically present in the Tested Storage Configuration (TSC).

Data Protection Overhead: The storage capacity required to implement the selected level of data protection.

Required Storage: The amount of Configured Storage Capacity required to implement the Addressable Storage Configuration, excluding the storage required for the three ASUs.

Global Storage Overhead: The amount of Physical Storage Capacity that is required for storage subsystem use and unavailable for use by application programs.

Total Unused Storage: The amount of storage capacity available for use by application programs but not included in the Total ASU Capacity.

SPC-1 Data Protection Levels

Protected: This level will ensure data protection in the event of a single point of failure of any configured storage device. A brief description of the data protection utilized is included in the Executive Summary.

Unprotected: No claim of data protection is asserted in the event of a single point of failure.

SPC-1 Test Execution Definitions

Average Response Time: The sum of the Response Times for all Measured I/O Requests divided by the total number of Measured I/O Requests.

Completed I/O Request: An I/O Request with a Start Time and a Completion Time (see "I/O Completion Types" below).

Completion Time: The time recorded by the Workload Generator when an I/O Request is satisfied by the TSC as signaled by System Software.

Data Rate: The data transferred in all Measured I/O Requests in an SPC-1 Test Run divided by the length of the Test Run in seconds.

Expected I/O Count: For any given I/O Stream and Test Phase, the product of 50 times the BSU level, the duration of the Test Phase in seconds, and the Intensity Multiplier for that I/O Stream.

Failed I/O Request: Any I/O Request issued by the Workload Generator that could not be completed or was signaled as failed by System Software. A Failed I/O Request has no Completion Time (see "I/O Completion Types" below).

I/O Request Throughput: The total number of Measured I/O requests in an SPC-1 Test Run divided by the duration of the Measurement Interval in seconds.

In-Flight I/O Request: An I/O Request issued by the I/O Command Generator to the TSC that has a recorded Start Time, but does not complete within the Measurement Interval (see "I/O Completion Types" below).

Measured I/O Request: A Completed I/O Request with a Completion Time occurring within the Measurement Interval (see "I/O Completion Types" below).

Measured Intensity Multiplier: The percentage of all Measured I/O Requests that were issued by a given I/O Stream.

Measurement Interval: The finite and contiguous time period, after the TSC has reached Steady State, when data is collected by a Test Sponsor to generate an SPC-1 test result or support an SPC-1 test result.

Ramp-Up: The time required for the Benchmark Configuration (BC) to produce Steady State throughput after the Workload Generator begins submitting I/O Requests to the TSC for execution.

Ramp-Down: The time required for the BC to complete all I/O Requests issued by the Workload Generator. The Ramp-Down period begins when the Workload Generator ceases to issue new I/O Requests to the TSC.

Response Time: The Response Time of a Measured I/O Request is its Completion Time minus its Start Time.

Start Time: The time recorded by the Workload Generator when an I/O Request is submitted, by the Workload Generator, to the System Software for execution on the Tested Storage Configuration (TSC).

Start-Up: The period that begins after the Workload Generator starts to submit I/O requests to the TSC and ends at the beginning of the Measurement Interval.

Shut-Down: The period between the end of the Measurement Interval and the time when all I/O Requests issued by the Workload Generator have completed or failed.

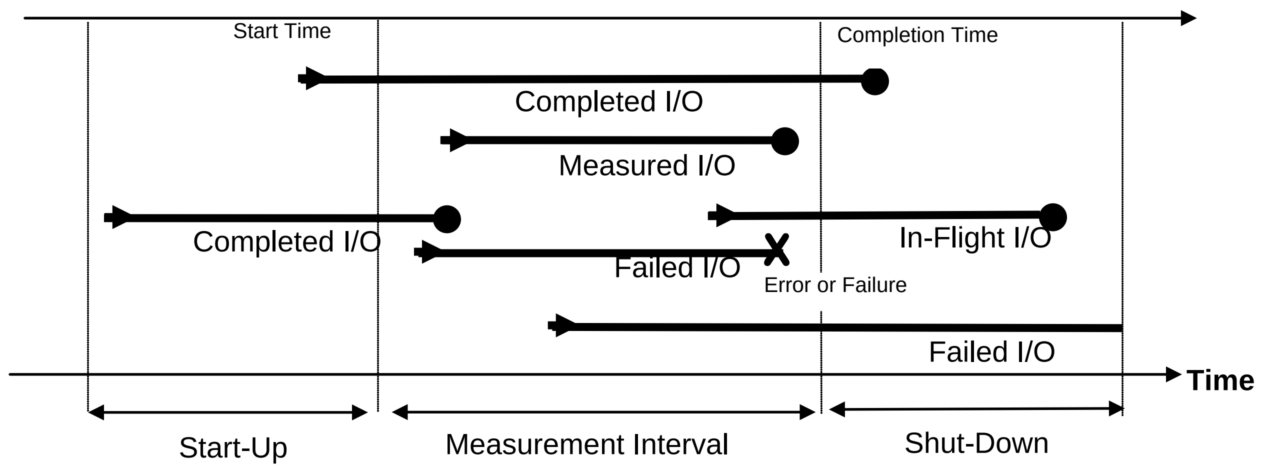
Steady State: The consistent and sustainable throughput of the TSC. During this period the load presented to the TSC by the Workload Generator is constant.

Test: A collection of Test Phases and or Test Runs sharing a common objective.

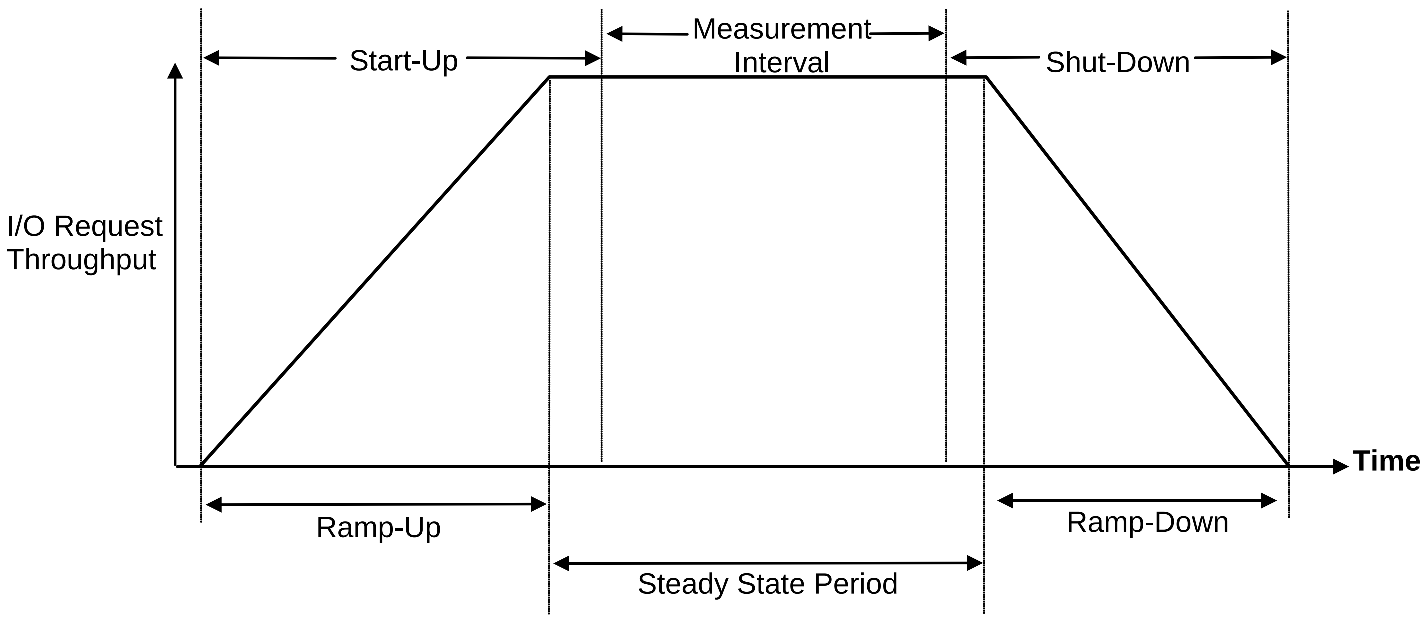
Test Run: The execution of SPC-1 for the purpose of producing or supporting an SPC-1 test result. SPC-1 Test Runs may have a finite and measured Ramp-Up period, Start-Up period, Shut-Down period, and Ramp-Down period as illustrated in the "SPC-1 Test Run Components" below. All SPC-1 Test Runs shall have a Steady State period and a Measurement Interval.

Test Phase: A collection of one or more SPC-1 Test Runs sharing a common objective and intended to be run in a specific sequence.

I/O Completion Types



SPC-1 Test Run Components



APPENDIX B: CUSTOMER TUNABLE PARAMETERS AND OPTIONS

Red Hat Enterprise Linux 6.2 (64-bit)

- Change the I/O scheduler from the default of ***cfq*** to ***noop*** on each Host System, which will result in all incoming I/O requests inserted into a simple, unordered FIFO queue. This was done by adding the following parameter in **/boot/grub.conf file** at the end of the kernel line on each Host System.

  elevator=noop

- Change the QLogic driver queue depth from a default of 64 to 255 on each Host System by manually creating the file, **/etc/modprobe.d/qla2xxx.conf**, which contains the following line:

  options qla2xxx ql2maxqdepth=255

  Once the file is created, rebuild the RHEL 6.2 ramdisk by issuing the CLI commands listed below on each Host System, then reboot each Host System so that the QLogic drive automatically loads with a queue depth of 255. This change is persistent across reboots.

  ```
  cd /boot
  dracut --force "initramfs-$(uname –r).img" $(uname –r)
  ```

Storage Controllers

The following parameters/options were changed on each controller with the Python scripts documented in *Appendix C: Tested Storage Configuration (TSC) Creation*.

Root Aggregate

Change the aggregate parameter, **nosnap**, from its default value of ***off*** to ***on***. This disables the automatic creation of snapshots.

Data Aggregates

- Change the aggregate parameter, **nosnap**, from the default value of ***off*** to ***on***. This disables the automatic creation of snapshots.
- Change the aggregate parameter, **max-write-alloc-blocks**, from the default value of ***64*** to ***256***. This changes the number of blocks written to a spindle before the write allocator changes to a new spindle, providing a better on disk layout for long sequential reads.
- Create the volumes with the space reservation parameter, **space-guarantee**, set to a value of ***none*** instead of the default value of ***default***.

LUN Options

Create the LUNs with the space reservation parameter, **space-reserve**, set to the value of ***disabled*** rather that the default value of ***enabled***.

System Options

The **walf.optimize_write_once** option was changed from the default value of ***on*** to ***off***. This option affects the initial layout of day within a new created aggregate. The default data layout favors applications that do not overwrite data.

APPENDIX C: TESTED STORAGE CONFIGURATION (TSC) CREATION

This documentation begins with the all of benchmark configuration hardware and software installed and configured on both the Host Systems and TSC, but the storage devices have yet to be configured for execution of the SPC-1 Test Runs.

Configure the Storage

Two main Python scripts, **config_storage.py** and **config_hosts.py**, are used to create the RAID groups, aggregates, volumes, LUNs and SPC-1 Logical Volumes that comprise the SPC-1 data repository. In addition, the scripts will change the appropriate customer tunable parameter/options, documented in Appendix B.

The two main Python scripts require three basic Python modules: **harness.py**, **storage.py** and **host.py**. In addition, the two main scripts require a configuration file, **config_full**, which was manually created with the appropriate configuration variable values.

The two main scripts, three subordinate scripts and configuration file appear at the end of this section after the detailed description of each of the two main scripts.

config_storage.py

This script is invoked from any of the Host Systems where Python is installed, as follows:

config_storage.py –c config_full –l *<directory to store output logs>* ***-n –r***

The script does the following on each controller:

- Set the following option: ***walf.optimize_write_once option off***
- Set the timezone and create NTP server as defined in **config_full**.
- Rename the root aggregates as specified in **config_full**.
- Set root aggregate snapshot setting as follows:
  -Set aggregate option **nosnap=on**.
  -Set aggregate snap sched to 0 0 0
  -Delete any existing snapshot.
- Set root volume snapshot setting as follows:
  -Set volume option **nosnap=on**
  -Set volume snap sched to 0 0 0
  -Delete any existing snapshot.
- Create the data aggregates as specified in **config_full**, which will create four RAID groups with a RAID group size of 17 disks and 1 spare.
- Set the data aggregate snapshot setting as follows:
  -Set aggregate option **nosnap=on**.
  -Set aggregate snap sched to 0 0 0
- Set the following field on the data aggregates: ***max-write_allock-blocks=256***.

- Create a **Vserver**, "**vs1**", as specified in **config_full**.
  *(A **Vserver** is virtual storage server in a NetApp clustered storage configuration that manages an individual workload.)*
- Set the following fields for the **Vserver**, "**vs1**":
  -nsswitch=file
  -rootvolume-security-style=unix
  -snapshot-policy none
  -allowed-protocols=fcp
  -percent-snapshot-space=0
- Create the following data volumes on each storage controller as specified in **config_full**:
  - 4 volumes of 2,595,583,163 KB each in the data aggregate. Those volumes will contain the LUNs for ASU-1.
  - 4 volumes of 2,595,583,163 KB each in the data aggregate. Those volumes will contain the LUNs for ASU-2.
  - 4 volumes of 576,796,258 KB each in the data aggregate. Those volumes will contain the LUNs for ASU-3.
- Set the following fields on each of the data volumes:
  -security-style=unix
  -space-guarantee=none
  -percent-snapshot-space=0
  -snapshot-policy none
- Create the LUNs on each storage controller as specified in **config_full**:
  - One LUN of 1,309,577,781 KB in each data volume designated for ASU-1.
  - One LUN of 1,309,577,781 KB in each data volume designated for ASU-2.
  - One LUN of 291,017,285 KB in each data volume designated for ASU-3.
- Set the following fields on each of the LUNs:
  -ostype=linux
  -space-reserve=disabled

- Create an **igroup** and map the LUNs as specified in **config_full**. Specifically:
  - Create an **igroup**,"**ig1**", with the following settings:
    -protocol=fcp
    -ostype=linux
    Contains twelve initiator WWPN *(two from each Host System specified in **config_full**)*
  - Map each LUNs to **igroup**,"**ig1**" and assign the logical volume to **Vserver** "**vs1**".

- Create two data LIFs *(logical interfaces)*, as specified in **config_full**, using physical on board FC target ports **0a** and **0c** on each storage controller with the following settings, which will assign each LIF to the **Vserver** "**vs1**". :
  -vserver=vs1
  -role=data
  -data-protocol=fcp
  -home-node=<*name of the cluster node physical FC target ports reside on*>
  -home-port=<*either **0a** or **0c**, as appropriate*>
- Start a fibre channel service on the **Vserver** "**vs1**",with the following setting:
  -state-admin=up
  -vserver=vs

config_hosts.py

This script is invoked as follows on a system other than the six SPC-1 Host Systems because the script will reboot each of the Host System:

config_hosts.py –c config_full –l *<directory to store output logs>* ***-x***

The script does the following on Master Host System:

- Reboot the Master Host System to scan all LUNs.
- Build a mapping of the LUNs to the associated Red Hat native multipath device on the Master Host System using the output of the NetApp Linux FC Utilities 6.0 command: ***sanlun lun show –p***.
- Create the RHEL 6.2 LVM2 logical volumes, which correspond to the SPC-1 Logical volumes, as specified in **config_full**.
  1. Initialize the first six logical volumes for ASU-1 from each storage controller using **pvcreate**.
  2. Create a volume group comprised of the six multipath devices that were initialized in the previous step.
  3. Create a logical volume with the following settings:
     -i 6 *(number of stripes)*
     -I 1024 *(stripe size in KiB)*
     -l 100%VG *(use 100% of space in the volume group)*
     -n *<name of the lv>*
  4. Repeat the previous steps #1-#3 11 additional times, which will result in the following:
     - A total of 12 logical volumes: 4 for ASU-1, 4 for ASU-2 and 4 for ASU-3.
     - One logical volume per volume group.
     - Each logical volume is striped across six multipath device (LUNs); one LUN from each storage controller.

- Reboot all six Host Systems so that each Host System scans the LUNs and logical volumes. Logical volume metadata was written on the LUNs when created from the Master Host System. As a result, when the remaining Host Systems scan the LUNs, the same metadata will be read, resulting in the same logical volumes visible on all Host Systems.
- The script will wait until all Host Systems are online and then build the SPC-1 configuration files for both the Master and Slave JVMs and place the appropriate configuration files in the appropriate directories of each Host System, as specified in **config_full**.

Referenced Configuration File and Scripts

config_full

```
[DEFAULT]
timeserver = 10.61.175.130
timezone = EST5EDT
sanlun = /usr/sbin/sanlun

## Benchmark configuration data
[SPC-1]
bsu_rate = 5001
rundir = /SPC1RunDir
Xms = 1536m
Xmx = 2048m
Xss = 256k
## implies size_linux is in $PATH
size_linux = /mnt/soltech/spc1/cmode/scripts/spc_audit/bin/size_linux

## Host configuration data
[hosts]
master = stlrx300s6-110
slaves =  stlrx300s6-110 stlrx300s6-111 stlrx300s6-112 stlrx300s6-113 stlrx300s6-114
stlrx300s6-115
login = root
password = Netapp123
post_reboot_init_time = 30

[stlrx300s6-110]
slave_num = 1
ip_addr = 10.61.183.210
initiators = 2100001b32841898 2101001b32a41898

[stlrx300s6-111]
slave_num = 2
ip_addr = 10.61.183.211
initiators = 21000024ff32bdae 21000024ff32bdaf

[stlrx300s6-112]
slave_num = 3
ip_addr = 10.61.183.212
initiators = 21000024ff32c176 21000024ff32c177

[stlrx300s6-113]
slave_num = 4
ip_addr = 10.61.183.213
initiators = 21000024ff32c192 21000024ff32c193

[stlrx300s6-114]
slave_num = 5
ip_addr = 10.61.183.214
initiators = 2100001b328b75c4 2101001b32ab75c4

[stlrx300s6-115]
slave_num = 6
ip_addr = 10.61.183.215
initiators = 2100001b328b93c3 2101001b32ab93c3

[lvm]
stripes = 6
stripesize = 1024
```

```
extents = 100%VG
lv_create_host = stlrx300s6-110

## ONTAP Storage configuration data
[cluster]
name = spc1cluster
nodes = spc1cluster-01 spc1cluster-02 spc1cluster-03 spc1cluster-04 spc1cluster-05
spc1cluster-06
ip_addr = 10.61.183.42
login: admin
password: Netapp123
data_aggrs = aggr1
vserver = vs1
host_ostype = linux
igroup_name = ig1
fcp_targets = 0a 0c

[spc1cluster-01]
node_num = 1
root_aggr = aggr0_spc1cluster_01_0

[spc1cluster-02]
node_num = 2
root_aggr = aggr0_spc1cluster_02_0

[spc1cluster-03]
node_num = 3
root_aggr = aggr0_spc1cluster_03_0

[spc1cluster-04]
node_num = 4
root_aggr = aggr0_spc1cluster_04_0

[spc1cluster-05]
node_num = 5
root_aggr = aggr0_spc1cluster_05_0

[spc1cluster-06]
node_num = 6
root_aggr = aggr0_spc1cluster_06_0

[aggr1]
diskcount = 68
raidtype = raid_dp
maxraidsize = 17
volume-style = flex

[vs1]
rootvolume = vs1_rootvol0
aggregate = aggr1_n1

[asu1]
vol_size = 2595583163KB
vols_per_node = 4
luns_per_vol = 1
lun_size = 1309577781KB

[asu2]
vol_size = 2595583163KB
vols_per_node = 4
luns_per_vol = 1
lun_size = 1309577781KB
```

```
[asu3]
vol_size = 576796258KB
vols_per_node = 4
luns_per_vol = 1
lun_size = 291017285KB
```

config_storage.py

```
import getopt
import sys

from lib.storage import StorageHarness


def usage(err=None):
    if err:
        sys.stderr.write("ERROR: %s\n" % (err))
    sys.stderr.write("Usage:\n")
    sys.stderr.write("> config_storage.py --config <config_file> --log <dir> [--
print] [--ntp_setup] [--rename_root_aggrs]\n")
    sys.stderr.write("  --config|-c <config_file>  use configuration file in \"ini\"
format\n")
    sys.stderr.write("  --log|-l <dir>             directory to store output logs
(default=\"./logs\")\n")
    sys.stderr.write("  --print|-p                 generate a report of commands to
be executed,\n")
    sys.stderr.write("                             without actually executing
anything\n")
    sys.stderr.write("  --ntp_setup|-n             configure timezone and ntp
settings\n")
    sys.stderr.write("  --rename_root_aggrs|-r     rename default root aggr names to
aggr0_n[0-9]+\n")
    sys.stderr.write("  --help|-h\n\n")
    sys.stderr.write("Examples:\n")
    sys.stderr.write("  config_storage.py --config ./spc_script_config\n")
    sys.stderr.write("    - performs storage configuration on NetApp cluster
specified in the\n")
    sys.stderr.write("      file ./spc_script_config\n")
    sys.stderr.write("  config_storage.py --config ./spc_script_config --print\n")
    sys.stderr.write("    - report all of the commands that will be executed in
order to configure\n")
    sys.stderr.write("      the NetApp cluster specified in the config file
./spc_script_config\n\n")
    sys.stderr.write("Notes:\n")
    sys.stderr.write(" - config file format:
U{http://effbot.org/librarybook/configparser.htm}\n")


def command_line_args(argv):
    """Processes command line arguments into user_data dict"""
    user_data = {"print":False, "logdir":"./logs", "ntp_setup":False,
                 "rename_root_aggrs":False}
    if not argv[1:]:
        usage("missing arguments")
        sys.exit(1)
    opts, args = getopt.getopt(argv[1:], "c:l:nphr", ["config=", "print", "help",
                                                      "log=", "ntp_setup",
                                                      "rename_root_aggrs"])
    for a, o in opts:
        if a in ("-h", "--help"):
            usage()
            sys.exit(0)
        elif a in ("-c", "--config"):
            user_data["config_file"] = o
        elif a in ("-p", "--print"):
            user_data["print"] = True
        elif a in ("-l", "--log"):
            user_data["logdir"] = o
        elif a in ("-n", "--ntp_setup"):
            user_data["ntp_setup"] = True
        elif a in ("-r", "--rename_root_aggrs"):
            user_data["rename_root_aggrs"] = True
        else:
            usage("unrecognized argument: %s" % (a))
            sys.exit(1)
    if not user_data.has_key("config_file"):
        usage("missing required argument '--config_file'")
        sys.exit(1)
    return user_data


def configure_cluster_shell(storage):
    """Some shell settings to help with automation"""
    storage.send_cluster("rows 0")
    storage.send_cluster("set -confirmations off")
    storage.send_cluster("set -privilege diagnostic")


def apply_wafl_options(storage):
    storage.node_run_all("options wafl.optimize_write_once off")


def apply_time_settings(storage):
    timezone = storage.config_data("timezone")
    timeserver = storage.config_data("timeserver")
    storage.send_cluster("timezone -timezone %s" % (timezone))
    for node in storage.nodes():
        ntp_create = "ntp server create -node %s -server %s" % (node,
                                                                timeserver)
        storage.send_cluster(ntp_create)
    storage.send_cluster("ntp config modify -enabled true")


def rename_root_aggrs(storage):
    for node in storage.nodes():
        old_aggr_name = storage.config_data("root_aggr", node.name)
        new_aggr_name = "aggr0_n%s" % (node.number)
        rename_cmd = ("storage aggregate rename -aggregate %s -newname %s" %
                      (old_aggr_name, new_aggr_name))
        storage.send_cluster(rename_cmd)
        # save the new name for later use
        node.root_aggr = new_aggr_name


def eliminate_root_aggr_snapshots(storage):
    for node in storage.nodes():
        storage.node_run(node, "aggr options %s nosnap on" % (node.root_aggr))
        storage.node_run(node, "snap sched -A %s 0 0 0" % (node.root_aggr))
        storage.node_run(node, "snap delete -A -a -f %s" % (node.root_aggr))


def eliminate_root_vol_snapshots(storage):
    storage.node_run_all("vol options vol0 nosnap on")
    storage.node_run_all("snap sched vol0 0 0 0")
    storage.node_run_all("snap reserve vol0 0")
    storage.node_run_all("snap delete -a -f vol0")


def create_data_aggrs(storage):
    for node in storage.nodes():
        data_aggr = storage.config_data("data_aggrs", "cluster")
        aggr_name = "%s_n%s" % (data_aggr, node.number)
        diskcount = storage.config_data("diskcount", data_aggr)
        raidtype = storage.config_data("raidtype", data_aggr)
        maxraidsize = storage.config_data("maxraidsize", data_aggr)
        vol_style = storage.config_data("volume-style", data_aggr)
        aggr_create = ("aggregate create -aggregate %s " % (aggr_name) +
                       "-diskcount %s -nodes %s -raidtype %s -maxraidsize %s" %
                       (diskcount,     node,    raidtype,    maxraidsize) +
                       " -volume-style %s" % (vol_style))
        storage.send_cluster(aggr_create, timeout=180)
        # waits a max of 10*60 seconds for aggr create to complete
        storage.wait_for_last_job_complete(interval=60, retries=10)
        storage.node_run(node, "aggr options %s nosnap on" % (aggr_name))
        storage.node_run(node, "snap sched -A %s 0 0 0" % (aggr_name))
        # save this for later use
        node.data_aggr = aggr_name


def set_max_walloc_blocks_on_data_aggrs(storage, max_walloc_blocks="256"):
    aggr_modify = ("aggr modify -aggregate aggr1_n* " +
                   "-max-write-alloc-blocks %s" % (max_walloc_blocks))
    storage.send_cluster(aggr_modify, timeout=120)


def create_vserver(storage):
    vserver = storage.config_data("vserver", "cluster")
    vs_root = storage.config_data("rootvolume", vserver)
    aggr = storage.config_data("aggregate", vserver)
    vs_create = ("vserver create -vserver %s -rootvolume %s -aggregate %s " %
                 (               vserver,    vs_root,       aggr) +
                 "-ns-switch file -rootvolume-security-style unix " +
                 "-snapshot-policy none")
    storage.send_cluster(vs_create)
    storage.send_cluster("vserver modify -vserver %s -allowed-protocols fcp" %
                         (vserver))
    storage.send_cluster("volume modify -volume %s -vserver %s " %
                         (               vs_root,   vserver) +
                         "-percent-snapshot-space 0")
    # save the vserver for later use
    storage.cluster.vserver = vserver


def create_data_vols(storage):
    for node in storage.nodes():
        node.asu_vols = {}
        vol_num = 0
        vol_num += create_node_asu_vols(storage, node, "asu1", vol_num)
        vol_num += create_node_asu_vols(storage, node, "asu2", vol_num)
        vol_num += create_node_asu_vols(storage, node, "asu3", vol_num)


def create_node_asu_vols(storage, node, asu, vol_num):
    num_vols = int(storage.config_data("vols_per_node", asu))
    # save vols per asu in a list fo later use
    node.asu_vols[asu] = []
    for i in range(num_vols):
        vol_name = "%s_n%s_v%02d" % (asu, node.number, vol_num)
        vol_size = storage.config_data("vol_size", asu)
        vol_create = ("volume create -volume %s -vserver %s -aggregate %s " %
                      (vol_name, storage.cluster.vserver, node.data_aggr) +
                      "-size %s -state online -security-style unix " %
                      (vol_size) + "-space-guarantee none " +
                      "-percent-snapshot-space 0 -snapshot-policy none")
        storage.send_cluster(vol_create, timeout=60)
        node.asu_vols[asu].append(vol_name)
        vol_num += 1
    return num_vols


def create_data_luns(storage):
    for node in storage.nodes():
        lun_num = 0
        lun_num += create_node_asu_luns(storage, node, "asu1", lun_num)
        lun_num += create_node_asu_luns(storage, node, "asu2", lun_num)
        lun_num += create_node_asu_luns(storage, node, "asu3", lun_num)


def create_node_asu_luns(storage, node, asu, lun_num):
    lun_count = 0
    luns_per_vol = int(storage.config_data("luns_per_vol", asu))
    lun_size = storage.config_data("lun_size", asu)
    lun_ostype = storage.config_data("host_ostype", "cluster")
    for data_vol in node.asu_vols[asu]:
        vol_path = "/vol/%s" % (data_vol)
        for i in range(luns_per_vol):
            lun_name = "%s_n%s_l%02d" % (asu, node.number, lun_num)
            lun_path ="%s/%s" % (vol_path, lun_name)
            lun_create = ("lun create %s -size %s -ostype %s " %
                          (lun_path, lun_size, lun_ostype) +
                          "-space-reserve disabled -vserver %s" %
                          (storage.cluster.vserver))
            storage.send_cluster(lun_create)
            lun_num += 1
            lun_count += 1
    return lun_count


def get_initiator_wwpns(storage):
    """Returns a list of wwpns from each host definition"""
    wwpns = set()
    for host in (storage.config_data("master", "hosts").split() +
                 storage.config_data("slaves", "hosts").split()):
        for wwpn in storage.config_data("initiators", host).split():
            wwpns.add(wwpn)
    return sorted(wwpns)


def create_igroup_and_map_luns(storage):
    igroup_name = storage.config_data("igroup_name", "cluster")
    igroup_ostype = storage.config_data("host_ostype", "cluster")
    initiator_wwpns = ",".join(get_initiator_wwpns(storage))
    igroup_create = ("igroup create -igroup %s -protocol fcp -ostype %s " %
                     (igroup_name, igroup_ostype) +
                     "-vserver %s -initiator %s" % (storage.cluster.vserver,
                                                    initiator_wwpns))
    storage.send_cluster(igroup_create)
    storage.send_cluster("lun map -path /vol/asu*/asu* -igroup %s -vserver %s"
                         %(igroup_name, storage.cluster.vserver))
```

```
def create_data_lifs(storage):
    vserver = storage.cluster.vserver
    for node in storage.nodes():
        for fcp_port in storage.config_data("fcp_targets", "cluster").split():
            lif_name = "n%s%s" % (node.number, fcp_port)
            lif_create = ("network interface create -vserver %s -lif %s " %
                          (vserver, lif_name) + "-role data " +
                          "-data-protocol fcp -home-node %s -home-port %s" %
                          (node, fcp_port))
            storage.send_cluster(lif_create)


def create_fcp_server(storage):
    fcp_create = ("vserver fcp create -status-admin up -vserver %s" %
                  (storage.cluster.vserver))
    storage.send_cluster(fcp_create)


if __name__ == "__main__":
    # Begin main
    logger = None
    user_args = command_line_args(sys.argv)
    storage = StorageHarness(user_args["config_file"],
                             logdir=user_args["logdir"],
                             diag_mode=user_args["print"])
    try:
        storage.initialize()
        logger = storage.getLogger()
        storage.console_message("Starting storage configuration...")
        logger.info("Starting storage configuration...")
        configure_cluster_shell(storage)
        apply_wafl_options(storage)
        if user_args["ntp_setup"]:
            apply_time_settings(storage)
        if user_args["rename_root_aggrs"]:
            rename_root_aggrs(storage)
            eliminate_root_aggr_snapshots(storage)
            eliminate_root_vol_snapshots(storage)
        create_data_aggrs(storage)
        set_max_walloc_blocks_on_data_aggrs(storage, max_walloc_blocks="256")
        create_vserver(storage)
        create_data_vols(storage)
        create_data_luns(storage)
        create_igroup_and_map_luns(storage)
        create_data_lifs(storage)
        create_fcp_server(storage)
    finally:
        storage.cleanup()
        logger.info("...storage configuration complete!")
        storage.console_message("...storage configuration complete!")
```

config_hosts.py

```
import os
import re
import getopt
import sys
import math

from lib.host import SPCHarness, SanlunDevMap, LinuxVolumeGroup, LinuxVolume


def usage(err=None):
    if err:
        sys.stderr.write("ERROR: %s\n" % (err))
    sys.stderr.write("Usage:\n")
    sys.stderr.write("> config_hosts.py --config <config_file> --log <dir> [--print]
[--sanlun_out <sanlun_output_file>] [--spc1_config]\n")
    sys.stderr.write("  --config|-c <config_file>  use configuration file in \"ini\"
format\n")
    sys.stderr.write("  --log|-l <dir>             directory to store output logs
(default=\"./logs\")\n")
    sys.stderr.write("  --print|-p                 generate a report of commands to
be executed,\n")
    sys.stderr.write("                             without actually executing
anything\n")
    sys.stderr.write("  --sanlun_out|-s            provide a sanlun output file to
instead of running sanlun bin\n")
    sys.stderr.write("                             NOTE: this is useful in
conjunction with the --print option\n")
    sys.stderr.write("  --spc1_config|-x           build SPC-1 input configuration
files and update SPC-1 RunDir\n")
    sys.stderr.write("  --help|-h\n\n")
    sys.stderr.write("Examples:\n")
    sys.stderr.write("  config_hosts.py --config ./spc_script_config\n")
    sys.stderr.write("    - performs host configuration on NetApp cluster specified
in the\n")
    sys.stderr.write("      file ./spc_script_config\n")
    sys.stderr.write("  config_hosts.py --config ./spc_script_config --print --
sanlun_out=./sanlun.out\n")
    sys.stderr.write("    - report all of the commands that will be executed in
order to configure\n")
    sys.stderr.write("      SPC hosts specified in the config file
./spc_script_config using the output\n")
    sys.stderr.write("      on sanlun in the file ./sanlun.out\n\n")
    sys.stderr.write("Notes:\n")
    sys.stderr.write(" - config file format:
U{http://effbot.org/librarybook/configparser.htm}\n")
    sys.stderr.write(" - --print without --sanlun_out will cause 'sanlun lun show -
p' to be\n")
    sys.stderr.write("   executed on the SPC master host\n")
    sys.stderr.write(" - --spc1_config builds the master, SPC1.cfg file and a
<jvm>.txt file\n")
    sys.stderr.write("   for each JVM to be created and will update the rundir with
these\n")
    sys.stderr.write("   files if 'rundir' is specified in the config file\n")


def command_line_args(argv):
    """Processes command line arguments into user_data dict"""
    user_data = {"print": False, "logdir": "./logs", "sanlun_out": None,
                 "spc1_config": False}
    if not argv[1:]:
        usage("missing arguments")
        sys.exit(1)
    opts, args = getopt.getopt(argv[1:], "c:l:ps:hx", ["config=", "print",
                                                      "help", "log=",
                                                      "sanlun_out=",
                                                      "spc1_config"])
    for a, o in opts:
        if a in ("-h", "--help"):
            usage()
            sys.exit(0)
        elif a in ("-c", "--config"):
            user_data["config_file"] = o
        elif a in ("-p", "--print"):
            user_data["print"] = True
        elif a in ("-l", "--log"):
            user_data["logdir"] = o
        elif a in ("-s", "--sanlun_out"):
            user_data["sanlun_out"] = o
        elif a in ("-x", "--spc1_config"):
            user_data["spc1_config"] = True
        else:
            usage("unrecognized argument: %s" % (a))
            sys.exit(1)
    if not user_data.has_key("config_file"):
        usage("missing required argument '--config_file'")
        sys.exit(1)
    return user_data


def reboot_lv_host(harness):
    lv_host = harness.get_lv_host()
    harness.reboot_host(lv_host)


def build_sanlun_map(harness, sanlun_outfile):
    re_no_sanlun = "No such file or directory"
    sanlun_bin = harness.config_data("sanlun")
    sanlun_output = harness.send_lv_host("%s lun show -p"  % (sanlun_bin))
    # Use the sanlun output file if one was provided
    if sanlun_outfile != None:
        sanlun_output = open(sanlun_outfile, "r")
    elif re.search(re_no_sanlun, sanlun_output):
        raise RuntimeError("%s. Please make sure the 'sanlun' field is " %
                           (sanlun_output) + "set correctly in your config " +
                           "file: %s" % (harness.config_file))
    return SanlunDevMap(sanlun_output)


def build_asu_map(dev_map, asu_map):
    """Runs through devices found from sanlun and updates their ASU and LUN#"""
    for dev in dev_map:
        lun_path = dev.lun_path
        (asu_num, lun_num) = extract_info_from_lun_path(lun_path)
        dev.asu = asu_num
        dev.lun_num = lun_num
        update_asu_map(dev.asu, dev.lun_num, dev, asu_map)
    return asu_map


def extract_info_from_lun_path(lun_path):
        """Retrieves asu# and lun# from storage lun path
        @attention: assumes that Volumes and LUNs were named according to the
        config_storage.py naming conventions.

        """
        re_lun_path = "/vol/[^/]+/(asu[1-3])_n[\d]+_(l[\d]+)"
        match = re.search(re_lun_path, lun_path)
        key_info = None
        if match:
            asu_num = match.group(1)
            lun_num = match.group(2)
            key_info = (asu_num, lun_num)
        else:
            raise RuntimeError("Could not determine ASU and LUN num from %s " %
                               (lun_path) + "using regexp: %s" %
                               (re_lun_path))
        return key_info


def update_asu_map(asu, lun_num, cur_dev, asu_map):
    """Divide devices into lists that can be refernced via the key (asu, lun#)

    """
    asu_key = (asu, lun_num)
    if not asu_map.has_key(asu_key):
        asu_map[asu_key] = []
    asu_map[asu_key].append(cur_dev)


def define_logical_volumes(harness, asu_map, lv_devs, spc1_devs):
    cur = 0
    lv_stripes = harness.config_data("stripes", "lvm")
    lv_stripesize = harness.config_data("stripesize", "lvm")
    lv_extents = harness.config_data("extents", "lvm")
    sorted_keys = sorted(asu_map.keys())
    for (asu, lun_num) in sorted_keys:
        asu_key = (asu, lun_num)
        dev_list = asu_map[asu_key]
        vg_name = "%s_vg%02d" % (asu, cur)
        lv_name = "%s_lv%02d" % (asu, cur)
        lv_dev = LinuxVolume(lv_name, LinuxVolumeGroup(vg_name, dev_list),
                             lv_stripes, lv_stripesize, lv_extents)
        lv_devs.append(lv_dev)
        spc1_devs[asu_key] = lv_dev
        cur += 1


def create_logical_volumes(harness, lv_list):
    for lv in lv_list:
        create_volume_group(harness, lv.volume_group)
        harness.send_lv_host("lvcreate -i %s -I %s -l %s -n %s %s" %
                             (lv.stripes, lv.stripe_size, lv.extents, lv.name,
                              lv.volume_group.name))


def create_volume_group(harness, volume_group):
    initialize_partition(harness, volume_group)
    vg_create = "vgcreate %s " % (volume_group.name)
    for dev in volume_group.dev_list:
        vg_create += "%s " % (dev.mpath_dev)
    harness.send_lv_host(vg_create)


def initialize_partition(harness, volume_group):
    for dev in volume_group.dev_list:
        harness.send_lv_host("pvcreate %s " % (dev.mpath_dev))


def set_spc1_dev_sizes(harness, spc1_devs, size_out=None):
    for lv_dev in spc1_devs.values():
        set_dev_size(harness, lv_dev, size_out)


def set_dev_size(harness, device, size_out=None):
    re_size_bytes = "file size is ([0-9]+)"
    re_no_size_linux = "No such file or directory"
    size_linux = harness.config_data("size_linux", "SPC-1")
    cmd = "%s %s" % (size_linux, device.path())
    size_output = harness.send_lv_host(cmd)
    if size_output is None and size_out != None:
        size_output = size_out
    match = re.search(re_size_bytes, size_output)
    if match:
        device.size = match.group(1)
    elif re.search(re_no_size_linux, size_output):
        raise RuntimeError("%s. Please check your 'size_linux' field in " %
                           (size_output) + "your config file: %s" %
                           (harness.config_file))
    else:
        raise Exception("unable to determine size of device %s from '%s' " %
                        (device, size_linux) + "output: '%s'\n" % (size_output)
                        + "...please check logs in %s " % (harness.logdir) +
                        "for errors.")


def jvm_per_host(harness):
    """Returns the number of JVMs per host needed to meet the BSU rate"""
    logger = harness.getLogger()
    num_hosts = len(harness.slaves)
    bsu_rate = float(harness.config_data("bsu_rate", "SPC-1"))
    bsu_per_host = math.ceil(bsu_rate/num_hosts)
    jvms_per_host = math.ceil(bsu_per_host/100.0)
    logger.debug("Building SPC-1 config for %s hosts, total BSUs=%s, " %
                 (num_hosts, bsu_rate) + "BSUs per host=%s, JVMs per host=%s" %
                 (bsu_per_host, jvms_per_host))
    return int(jvms_per_host)


def build_spc1_config_files(harness, spc1_devs):
    slave_jvms = build_slave_config(harness, spc1_devs)
    build_master_config(harness, spc1_devs, slave_jvms)


def build_master_config(harness, spc1_devs, slave_jvm_list):
    master_config = []
    master_config.append(jvm_settings(harness))
    master_config.append("host=master")
    master_config.append("slaves=(%s)" % (",".join(slave_jvm_list)))
    master_config.extend(spc1_config_body(harness, spc1_devs))
    write_spc1_config(harness, harness.master, master_config, "spc1.cfg")


def build_slave_config(harness, spc1_devs):
    """Builds slave configurations and returns slave jvm names for master"""
    slave_jvms = []
    for host in harness.slaves:
        slave_num = harness.config_data("slave_num", host.name)
        jvm_num = 1
        for jvm in range(jvm_per_host(harness)):
            config_lines = []
            jvm_name = "h%s_slave%s" % (slave_num, jvm_num)
            slave_jvms.append(jvm_name)
            config_lines.extend(spc1_config_slave_header(harness, jvm_name))
            config_lines.extend(spc1_config_body(harness, spc1_devs))
            write_spc1_config(harness, host, config_lines,
                              ("%s.txt" % (jvm_name)))
            jvm_num += 1
    return slave_jvms


def write_spc1_config(harness, host, config_lines, filename):
    rundir = harness.config_data("rundir", "SPC-1")
    config_file = os.path.join(rundir, filename)
    harness.send_host(host, "rm -f %s" % (config_file))
    for line in config_lines:
        harness.send_host(host, "echo '%s' >> %s" % (line, config_file))


def spc1_config_slave_header(harness, jvm_name):
    config_lines = []
    config_lines.append(jvm_settings(harness))
    config_lines.append("master=%s" % (harness.master.ip_addr))
    config_lines.append("host=%s" % (jvm_name))
    return config_lines


def jvm_settings(harness):
    Xms = harness.config_data("Xms", "SPC-1")
    Xmx = harness.config_data("Xmx", "SPC-1")
    Xss = harness.config_data("Xss", "SPC-1")
    return 'javaparms="-Xms%s -Xmx%s -Xss%s"' % (Xms, Xmx, Xss)


def spc1_config_body(harness, spc1_devs):
    config_lines = []
    sorted_keys = sorted(spc1_devs.keys())
    for (asu, lun_num) in sorted_keys:
        sd_name = "%s_%s" % (asu, lun_num)
        lv_dev = spc1_devs[(asu, lun_num)]
        lun_name = lv_dev.path()
        size = lv_dev.size
        config_line = "sd=%s,lun=%s,size=%s" % (sd_name, lun_name, size)
        config_lines.append(config_line)
    return config_lines


if __name__ == "__main__":
    # Begin main
    logger = None
    dev_map = None
    asu_map = {}
    lv_devs = []
    spc1_devs = {}
    user_args = command_line_args(sys.argv)
    spc_harness = SPCHarness(user_args["config_file"],
                             logdir=user_args["logdir"],
                             diag_mode=user_args["print"],
                             name="config_hosts")
    try:
        spc_harness.initialize()
        logger = spc_harness.getLogger()
        logger.info("Starting SPC-1 host configuration...")
        spc_harness.console_message("Starting SPC-1 host configuration...")
        reboot_lv_host(spc_harness)
        dev_map = build_sanlun_map(spc_harness, user_args["sanlun_out"])
        build_asu_map(dev_map, asu_map)
        define_logical_volumes(spc_harness, asu_map, lv_devs, spc1_devs)
        create_logical_volumes(spc_harness, lv_devs)
        spc_harness.reboot_hosts()
        if user_args["spc1_config"]:
            size_out = None
            if user_args["print"]:
                # to help with diag mode
                size_out = "file size is 000 (00.00 megabytes)"
            set_spc1_dev_sizes(spc_harness, spc1_devs, size_out)
            build_spc1_config_files(spc_harness, spc1_devs)
    finally:
        try:
            spc_harness.cleanup()
            logger.info("...SPC-1 host configuration complete!")
            spc_harness.console_message("...SPC-1 host configuration complete!")
        except:
            pass
```

harness.py

```
import os
import logging
import socket
import time
import sys

import ConfigParser
import pexpect

# Some regular expressions to help with prompt matching
re_unix_prompt = '(?m)([\r\n][^%#$\r\n]*[%#$][\s])'
# The regexp needs self.name applied with format(name) operation
re_cluster_prompt = '(?m)({0}(#|%|>|\*>|::)|::[a-zA-Z0-9 ]*[*]*> )'


def get_logger(category='main', filename=None):
    if len(logging.getLogger(category).handlers) <= 0:
        if not filename:
            raise IOError("no filename provided for logger('%s')" % (category))
        return file_logger(filename, open_mode="a", category=category)
    else:
        return logging.getLogger(category)


def log_formatter(category='main', date_format = '%a %b %d %X %Z %Y'):
    hostname = socket.gethostname()
    fmt = ('%(asctime)s (' + hostname + ' %(process)d-%(threadName)s' + ') ' +
           '[%(lineno)d %(module)s.%(funcName)s]: ' +
           '%(levelname)s: %(message)s')
    logging.Formatter.converter = time.gmtime
    return logging.Formatter(fmt=fmt, datefmt=date_format)


def file_logger(filename, open_mode="a", category="main", level=None):
    if level is None:
        level = logging.DEBUG
    hand = logging.FileHandler(filename, open_mode)
    log = logging.getLogger(category)
    log.setLevel(level)
    hand.setFormatter(log_formatter(category))
    log.addHandler(hand)
    return log


class DiagReport(object):

    def __init__(self):
        self.commands = []

    def __len__(self):
        return len(self.commands)

    def __getitem__(self, index):
        return self.commands[index]

    def __contains__(self, key):
        return key in self.commands

    def __str__(self):
        return self.generate()

    def append(self, item):
        self.commands.append(item)

    def generate(self):
        report = "---Diagnostic Report---\n"
        for i in range(len(self)):
            report += "%s. %s\n" % (i+1, self.commands[i])
        return report


class ScriptHarness(object):

    def __init__(self, config_file, name=None, logdir=None, diag_mode=False):
        self.config_file = config_file
        self.name = name
        self.logdir = logdir
        self.config = None
        self.logger = None
        self.diag_report = None
        self.logfile = None
        if diag_mode:
            self.console_message("toggling diagnostic report mode", "DEBUG")
            self.diag_report = DiagReport()
        if self.name is None:
            self.name = "unknown_script_harness"

    def __str__(self):
        return "script harness %s" % (self.name)

    def _time(self):
        return time.asctime(time.localtime(time.time()))

    def console_message(self, msg, level="INFO"):
        sys.stderr.write("%s: %s:(%s): %s\n" % (self._time(), level, self.name,
                                                msg))

    def create_logdir(self):
        """Creates the directory where logging output will be contained
        @postcondition: self.logdir is not None
        @postcondition: os.isdri(self.logdir)

        """
        if not self.logdir:
            self.logdir = self.default_logdir()
        if not os.path.isdir(self.logdir):
            self.console_message("logging in directory: %s" % (self.logdir))
            os.mkdir(self.logdir)

    def default_logdir(self):
        """Returns a path to the default logging directory"""
        return os.path.join(os.getcwd(), "logs")

    def main_logfile(self):
        """Returns the full path to the main script harness logfile

        @precondition: self.logdir is not None

        """
        self.logfile = os.path.join(self.logdir, "%s.log" % (self.name))
        return self.logfile

    def setup_logging(self):
        """Sets up the main script logger

        @postcondition: self.logger is not None

        """
        log_file = self.main_logfile()
        self.create_logdir()
        self.console_message("main script log is: %s" % (log_file))
        self.logger = get_logger(category=self.name, filename=log_file)

    def getLogger(self):
        """Returns the main logger for this harness

        @note: this will initialize and create the logger if it hasn't already
        been created. Client code should call harness.getLogger().
        @return: logger obj

        """
        if self.logger is None:
            self.setup_logging()
        return self.logger

    def initialize(self):
        self.check_config_file()
        self.init_config()
        self.getLogger()

    def check_config_file(self):
        """Checks to see that self.config_file is readable

        @raise IOError: if config_file not readable

        """
        if not os.access(self.config_file, os.R_OK):
            raise IOError("%s cannot open config file %s for reading" %
                          (self, self.config_file))

    def init_config(self):
        """Initialize self.config from config_file using ConfigParser

        @see: http://docs.python.org/library/configparser.html#module-ConfigParser
        @postcondition: self.config is not None

        """
        self.config = ConfigParser.ConfigParser()
        self.config.read(self.config_file)

    def config_data(self, field, section="DEFAULT"):
        """Get configuration data that was initialized from the config_file

        @return: entry for '[section] field: entry'
        @rtype: str

        """
        if self.config is None:
            raise LookupError("%s was not yet initialized" % (self))
        return self.config.get(section, field)

    def cleanup(self):
        if self.diag_report is not None:
            print self.diag_report.generate()


class InteractiveSSH(object):

    def __init__(self, login, passwd, hostname, logdir, re_shell_prompt):
        self.login = login
        self.passwd = passwd
        self.hostname = hostname
        self.logdir = logdir
        self.re_shell_prompt = re_shell_prompt
        self.session = None
        self.logfile = None
        self.logfile_fh = None
        self.cmd = None

    def __str__(self):
        return "InteractiveSSH session: %s" % (self.command_string())

    def command_string(self):
        return ('ssh %s %s %s@%s' % ('-o StrictHostKeyChecking=no',
                                     '-o UserKnownHostsFile=/dev/null',
                                     self.login, self.hostname))

    def setup_logfile(self):
        if self.logfile_fh is None or self.logfile_fh.closed:
            self.logfile = os.path.join(self.logdir, "%s_ssh.log" %
                                        (self.hostname))
            self.logfile_fh = open(self.logfile, "a")

    def open(self, timeout=600):
        self.setup_logfile()
        self.session = pexpect.spawn(self.command_string(),
                                     logfile=self.logfile_fh,
                                     timeout=timeout)
        self.session.setecho(False)
        self._match_passwd()
        return self.session

    def _match_passwd(self):
        i = self.session.expect(['ssword:', self.re_shell_prompt])
        if i == 0:
            self.session.sendline(self.passwd)
            self.session.expect(self.re_shell_prompt)
        elif i == 1:
            # @todo: log message here saying we got prompt without password
            pass

    def close(self):
        if self.session:
            if not self.session.eof() and self.session.isalive():
                self.session.close()
            self.session = None
        if self.logfile_fh and not self.logfile_fh.closed:
            self.logfile_fh.close()


class CommandSender(object):
    """Interface for sending commands/retrieving output through a remote shell

    Client code will typically do something like:

      ssh = InteractiveSSH(login, passwd, host, logdir)
      command_sender = CommandSender(ssh.open(), shell_regexp)
      msg_output = command_sender.send("cat /var/log/messages", timeout=5)

    """

    def __init__(self, pexpect_session, re_shell_prompt):
        self.session = pexpect_session
        self.re_shell_prompt = re_shell_prompt

    def send(self, cmd, timeout=60):
        cmd_output = None
        self.clear_session_buffer()
        self.session.sendline(cmd)
        self.expect_prompt(timeout=timeout)
        cmd_output = self.remove_command_echo(cmd, self.session.before)
        return cmd_output.strip()

    def clear_session_buffer(self):
        """Clear out any non-whitespace chars in buffer before next sendline"""
        self.session.timeout = 0.5
        self.session.expect(["^[\s+]$", pexpect.TIMEOUT])

    def remove_command_echo(self, command, output):
        new_output = output
        str_index = output.find(command)
        if str_index > -1:
            new_output = output[str_index + len(command):]
        return new_output

    def expect_prompt(self, timeout):
        self.session.timeout = timeout
        self.session.expect([self.re_shell_prompt])
        self.decontaminate_output()

    def decontaminate_output(self):
        """Checks session to see if output was polluted by esc chars

        esc[A shows up when the prompt + command add up to exactly the
        column width of the terminal and a second prompt is output.

        """
        if (self.session.before.find("\x1b[A") > -1 or
            self.session.after.find("\x1b[A") > -1):
            self.session.expect(["\x1b\[Ka\r\n"])
            self.expect_prompt(5)


class HardwareInterface(object):
    """Class to help communicate with Test Hardware objs"""

    def __init__(self, name, ip_addr, login, password, re_shell_prompt):
        self.name = name
        self.ip_addr = ip_addr
        self.login = login
        self.password = password
        self.re_shell_prompt = re_shell_prompt
        self.ssh = None
        self.sender = None

    def __str__(self):
        return "Host: %s ip addr: %s" % (self.name, self.ip_addr)

    def connect(self, logdir, diag_report):
        self.ssh = InteractiveSSH(self.login, self.password, self.ip_addr,
                                  logdir, self.re_shell_prompt)
        if diag_report is not None:
            diag_report.append("%s" % (self.ssh))
        else:
            self.sender = CommandSender(self.ssh.open(), self.re_shell_prompt)

    def send(self, command, timeout=60, diag_report=None):
        output = None
        if not self.sender and diag_report is None:
            raise RuntimeError("No connection to %s! Please call connect()" %
                               (self) + " method before attempt to send " +
                               "commands.")
        elif diag_report != None:
            diag_report.append("Sent to: %s " % (self) + command)
        else:
            cmd_output = self.sender.send(command, timeout=timeout)
            output = self._check_command_errors(command, cmd_output)
        return output

    def check_command_errors(self, command, output):
        """Sub-classes provide a custom impl that raises exc if error"""
        pass

    def _check_command_errors(self, command, output):
        """Class sub-class command error checker

        @return: output
        @raise exception: if check fails

        """
        if output:
            self.check_command_errors(command, output)
        return output
```

storage.py

```
import re
import time

from lib.harness import ScriptHarness, re_cluster_prompt, HardwareInterface
from lib.harness import InteractiveSSH, CommandSender


class ONTAPCommandError(Exception):
    pass


class StorageHarness(ScriptHarness):

    def __init__(self, config_file, logdir=None, diag_mode=False):
       ScriptHarness.__init__(self, config_file, name="config_storage",
                              logdir=logdir, diag_mode=diag_mode)
       self.cluster = None

    def initialize(self):
        """Initializes and connects to storage

        The Cluster data is initialized and we attempt to make a(n ssh)
        connection with the cluster.

        """
        ScriptHarness.initialize(self)
        self.init_cluster()
        self.init_nodes()
        self.cluster.connect(self.logdir, self.diag_report)

    def init_cluster(self):
        """Initialize the cluster from the configuration data

        @postcondition: self.cluster != None

        """
        self.cluster = Cluster(self.cluster_data("name"),
                               self.cluster_data("ip_addr"),
                               self.cluster_data("login"),
                               self.cluster_data("password"))
        self.logger.debug("initialized: %s" % (self.cluster))

    def init_nodes(self):
        """Initialize the nodes in the cluster from configuration data

        @precondition: self.cluster != None
        @postcondition: len(self.cluster.nodes) > 0

        """
        for node_name in self.cluster_data("nodes").split():
            node_num = self.config_data("node_num", node_name)
            node = Node(node_name, node_num, self.cluster)
            self.cluster.nodes.append(node)
            self.logger.debug("initialized: %s" % (node))

    def nodes(self):
        return self.cluster.nodes

    def cluster_data(self, field):
        """Get cluster configuration data from the config_file

        """
        return self.config_data(field, "cluster")

    def node_run(self, node, command, timeout=60):
        node_run_cmd = "node run -node %s -command %s" % (node, command)
        return self.send_cluster(node_run_cmd, timeout)

    def node_run_all(self, command, timeout=60):
        node_run_cmd = "node run -node * -command %s" % (command)
        return self.send_cluster(node_run_cmd, timeout)

    def send_cluster(self, command, timeout=60):
        output = None
        if self.diag_report is not None:
            self.diag_report.append(command)
        else:
            output = self.cluster.send(command, timeout)
        return output

    def cleanup(self):
        self.cluster.ssh.close()
        ScriptHarness.cleanup(self)

    def wait_for_last_job_complete(self, interval=60, retries=5):
        """Waits for the last job sent to the cluster shell to complete

        This is useful for commands like aggregate create that run in the
        background

        @raise RuntimeError: if job doesn't complete after interval*retries
                             seconds

        """
        for cur_try in range(retries):
            cur_state = self.cluster.last_job_state()
            if cur_state in (None, "Success"):
                self.logger.info("Job %s has completed with state of %s" %
                                 (self.cluster.job_id, cur_state))
                break
            else:
                self.logger.info("Attempt: %s. Job %s is in state %s..." %
                                 (cur_try+1, self.cluster.job_id, cur_state) +
                                 "checking state again in %s seconds." %
                                 (interval))
                time.sleep(interval)
        if cur_try == retries-1 and not cur_state in (None, "Success"):
            raise RuntimeError("It appears that Job %s never completed " %
                               (self.cluster.job_id) + "after %s retries " %
                               (retries) + "of %s second intervals." %
                               (interval))


class Cluster(HardwareInterface):

    job_id_pattern = re.compile("\[Job (\d+)\]", re.M)

    def __init__(self, name, ip_addr, login, password):
        HardwareInterface.__init__(self, name, ip_addr, login, password,
                                   re_cluster_prompt)
        # Applies cluster name to regexp
        self.re_shell_prompt = self.re_shell_prompt.format(self.name)
        self.nodes = []
        self.job_id = None

    def __str__(self):
        return "cluster: %s management ip: %s" % (self.name, self.ip_addr)

    def send(self, command, timeout=60):
        cmd_output = HardwareInterface.send(self, command, timeout)
        self.set_job_id(cmd_output)
        return cmd_output

    def check_command_errors(self, cmd, cmd_output):
        """Checks command output for indication of ONTAP error

        @raise ONTAPCommandError: if errors are detected in command output
        @return: command output

        """
        re_smf_error = "(?mi)Error: (.*)"
        match = re.search(re_smf_error, cmd_output, re.DOTALL)
        if match is not None:
            err_msg = match.group(1)
            raise ONTAPCommandError("ONTAP command '%s' failed with:\n%s" %
                                    (cmd, err_msg))

    def set_job_id(self, output):
        """Sets self.job_id to the last [Job $id] from cluster shell output"""
        match = Cluster.job_id_pattern.search(output)
        if match:
            self.job_id = match.group(1)
        else:
            self.job_id = None
        return output

    def last_job_state(self):
        state = None
        if self.job_id is not None:
            re_state = "(?m)%s %s (\w+)" % (self.job_id, self.name)
            re_no_entries = "There are no entries matching your query"
            job_show_output = self.send("job show -id %s -fields state" %
                                        (self.job_id))
            if re.search(re_state, job_show_output):
                state = re.search(re_state, job_show_output).group(1)
            elif re.search(re_no_entries, job_show_output):
                pass
            else:
                raise RuntimeError("Unable to determine state of Job %s from" %
                                   (self.job_id) + " 'job show' output:\n%s" %
                                   (job_show_output))
        return state


class Node(object):

    def __init__(self, name, number, cluster):
        self.name = name
        self.number = str(number)
        self.cluster = cluster

    def __str__(self):
        return self.name
```

host.py

```
import re
import time
import pexpect
import subprocess
from StringIO import StringIO

from lib.harness import ScriptHarness, InteractiveSSH, re_unix_prompt
from lib.harness import CommandSender, HardwareInterface


class HostRebootFailure(Exception):
    pass


class HostHarness(ScriptHarness):

    def __init__(self, config_file, logdir=None, diag_mode=False,
                 name="host_harness"):
        ScriptHarness.__init__(self, config_file, name=name, logdir=logdir,
                               diag_mode=diag_mode)
        self.hosts = []

    def initialize(self):
        ScriptHarness.initialize(self)
        self.init_hosts()
        self.connect_hosts()

    def connect_hosts(self):
        for host in self.hosts:
            host.connect(self.logdir, self.diag_report)

    def init_hosts(self):
        for hostname in self.config_data("hostnames", "hosts").split():
            self.init_host(hostname)

    def init_host(self, hostname):
        new_host = LinuxHost(hostname, self.config_data("ip_addr", hostname),
                             self.config_data("login", "hosts"),
                             self.config_data("password", "hosts"))
        self.hosts.append(new_host)
        return new_host

    def reboot_hosts(self, reboot_init=None):
        for host in self.hosts:
            self.reboot_host(host, reboot_init)

    def reboot_host(self, host, reboot_init_time=None):
        host.reboot(self.diag_report)
        if reboot_init_time is None:
            reboot_init_time = float(self.config_data("post_reboot_init_time",
                                                       "hosts"))
        if self.diag_report is None:
            self.wait_for_host_to_reboot(host)
            self.console_message("%s came up after reboot successfully"  %
                                 (host))
            self.logger.info("waiting %s seconds for %s to initialize" %
                             (reboot_init_time, host))
            time.sleep(reboot_init_time)
        host.connect(self.logdir, self.diag_report)

    def wait_for_host_to_reboot(self, host, ping_attempts=20, retry_sleep=30):
        """Waits for a host to come up after a reboot

        @raise HostRebootFailure: if ping_attempts have been exceeded

        """
        cur_try = 1
        start_time = time.time()
        while host.ping() != 0:
            if cur_try == ping_attempts:
                raise HostRebootFailure("%s failed to come up after reboot " %
                                        (host) + "after %d seconds." %
                                        (time.time() - start_time))
            cur_try += 1
            self.logger.info("%s not rebooted after %d seconds..." %
                             (host, time.time() - start_time) +
                             "retry again in %s seconds" % (retry_sleep))
            time.sleep(retry_sleep)
        self.logger.info("%s came up after %d seconds...reboot successful!" %
                         (host, time.time() - start_time))

    def send_host(self, host, command, timeout=60):
        output = None
        if self.diag_report is not None:
            self.diag_report.append("Sent to %s: " % (host) + command)
        else:
            output = host.send(command, timeout)
        return output

    def cleanup(self):
        for host in self.hosts:
            host.ssh.close()
        ScriptHarness.cleanup(self)


class SPCHarness(HostHarness):

    def __init__(self, config_file, logdir=None, diag_mode=False,
                 name="spc_harness"):
        HostHarness.__init__(self, config_file, name=name, logdir=logdir,
                             diag_mode=diag_mode)
        self.master = None
        self.slaves = []

    def initialize(self):
        ScriptHarness.initialize(self)
        self.init_master()
        self.init_slaves()
        self.connect_hosts()

    def init_master(self):
        master_host = self.config_data("master", "hosts")
        self.master = self.init_host(master_host)

    def init_slaves(self):
        for slave in self.config_data("slaves", "hosts").split():
            # The master can also be a slave
            if slave == self.master.name:
                self.slaves.append(self.master)
            else:
                self.slaves.append(self.init_host(slave))

    def send_lv_host(self, command, timeout=60):
        lv_host = self.get_lv_host()
        return self.send_host(lv_host, command, timeout)

    def get_lv_host(self):
        name = self.config_data("lv_create_host", "lvm")
        lv_host = None
        for host in self.hosts:
            if host.name == name:
                lv_host = host
                return lv_host
        if lv_host is None:
            raise Exception("No host obj was created with hostname %s" %
                            (name))

    def send_master(self, command, timeout=60):
        return self.send_host(self.master, command, timeout)


class LinuxHost(HardwareInterface):

    def __init__(self, name, ip_addr, login, password, sanlun=None):
        HardwareInterface.__init__(self, name, ip_addr, login, password,
                                   re_unix_prompt)
        self.dev_map = None

    def __str__(self):
        return "Linux host: %s ip addr: %s" % (self.name, self.ip_addr)

    def connect(self, logdir, diag_report):
        HardwareInterface.connect(self, logdir, diag_report)
        # start a sane shell -- should not be required
        self.send("exec sh", timeout=5, diag_report=diag_report)

    def reboot(self, diag_report):
        try:
            self.send("reboot -f -n", diag_report=diag_report)
        except (pexpect.EOF, pexpect.TIMEOUT):
            pass
        finally:
            self.ssh.close()
            self.sender = None

    def ping(self):
        """Returns the result of 1 count, 1 second ping

        @return: 0 on success, <0 means ping failed

        """
        ping_command = '/bin/ping -c 1 -w 1 %s' % (self.ip_addr)
        return subprocess.call(ping_command, shell=True,
                               stdout=open('/dev/null', 'w'),
                               stderr=subprocess.STDOUT)


class SanlunDevMap(object):

    class SanDev(object):

        def __init__(self, vserver, lun_path, asu=None, lun_num=None,
                     mpath_dev=None):
            self.vserver = vserver
            self.lun_path = lun_path
            self.asu = asu
            self.lun_num = lun_num
            self.mpath_dev = None
            if mpath_dev != None:
                self.set_mpath(mpath_dev)

        def set_mpath(self, dev):
            self.mpath_dev = "/dev/mapper/%s" % (dev)

        def __str__(self):
            return "SAN Device: %s:%s Mapped to %s, %s" % (self.vserver,
                                                           self.lun_path,
                                                           self.asu,
                                                           self.mpath_dev)

        def __repr__(self):
            return str(self)

    re_ontap_lun = "ONTAP Path: ([^:]+):([\w/]+)"
    re_host_device = "Host Device: [a-z]*[\(]*([0-9a-f]+)[\)]*"

    def __init__(self, sanlun_output=None):
        self.devs = []
        if sanlun_output != None:
            self.process_sanlun_output(sanlun_output)

    def __str__(self):
        str_map = 'SanlunDevMap containing SAN devices:\n'
        for dev in self:
            str_map += str(dev)
        return str_map

    def __len__(self):
        return len(self.devs)

    def __iter__(self):
        for dev in self.devs:
            yield dev

    def process_sanlun_output(self, output):
        """Processes the output of sanlun lun show -p to build self.devs"""
        output_fh = self._make_file_io(output)
        new_ontap_dev = None
        cur_dev = None
        for line in output_fh:
            if re.search(self.re_ontap_lun, line):
                match = re.search(self.re_ontap_lun, line)
                dev_path = match.group(2)
                cur_dev = SanlunDevMap.SanDev(match.group(1), dev_path)
            elif re.search(self.re_host_device, line):
                match = re.search(self.re_host_device, line)
                cur_dev.set_mpath(match.group(1))
                self.devs.append(cur_dev)
                cur_dev = None
        self.sort_devs()

    def sort_devs(self):
        """Sorts devs by lun_path so that LUNs are striped in correct order

        """
        self.devs.sort(key = lambda x: x.lun_path)

    def _make_file_io(self, output):
        ret_val = None
        if isinstance(output, str):
            ret_val = StringIO(output)
        elif isinstance(output, file):
            ret_val = output
        else:
            raise TypeError("cannot parse sanlun output of type: %s" %
                            type(output))
        return ret_val


class LinuxVolumeGroup(object):

    def __init__(self, name, dev_list):
        self.name = name
        self.dev_list = dev_list
        self.loc = "/dev/mapper"

    def __str__(self):
        return ("VolumeGroup %s/%s consisting of devices: %s" % (self.loc,
                                                                  self.name,
                                                                  self.dev_list))


class LinuxVolume(object):

    def __init__(self, name, volume_group, stripes, stripe_size="1024",
                 extents="100%VG"):
        self.name = name
        self.volume_group = volume_group
        self.stripes = stripes
        self.stripe_size = stripe_size
        self.extents = extents
        self.size = None

    def __str__(self):
        return ("LinuxVolume %s, with stripes=%s, stripe_size=%s, extents=%s"
                % (self.name, self.stripes, self.stripe_size, self.extents) +
                " and the following Volume Group:\n\t%s" % (self.volume_group))

    def path(self):
        return "/dev/mapper/%s-%s" % (self.vg_name(), self.name)

    def vg_name(self):
        return self.volume_group.name
```

APPENDIX D: SPC-1 WORKLOAD GENERATOR STORAGE COMMANDS AND PARAMETERS

The content of SPC-1 Workload Generator command and parameter file used in this benchmark to execute the Primary Metrics *(Sustainability Test Phase, IOPS Test Phase, and Response Time Ramp Test Phase)* Test and Repeatability Test *(Repeatability Test Phase 1 and Repeatability Test Phase 2)*, which used multiple Host Systems, is listed below.

```
javaparms="-Xms1536m -Xmx2048m -Xss256k"
host=master
slaves=(h1_slave1,h1_slave2,h1_slave3,h1_slave4,h1_slave5,h1_slave6,h1_slave7,h1_sla
ve8,h1_slave9,h2_slave1,h2_slave2,h2_slave3,h2_slave4,h2_slave5,h2_slave6,h2_slave7,
h2_slave8,h2_slave9,h3_slave1,h3_slave2,h3_slave3,h3_slave4,h3_slave5,h3_slave6,h3_s
lave7,h3_slave8,h3_slave9,h4_slave1,h4_slave2,h4_slave3,h4_slave4,h4_slave5,h4_slave
6,h4_slave7,h4_slave8,h4_slave9,h5_slave1,h5_slave2,h5_slave3,h5_slave4,h5_slave5,h5
_slave6,h5_slave7,h5_slave8,h5_slave9,h6_slave1,h6_slave2,h6_slave3,h6_slave4,h6_sla
ve5,h6_slave6,h6_slave7,h6_slave8,h6_slave9)
sd=asu1_l00,lun=/dev/mapper/asu1_vg00-asu1_lv00,size=8046117912576
sd=asu1_l01,lun=/dev/mapper/asu1_vg01-asu1_lv01,size=8046117912576
sd=asu1_l02,lun=/dev/mapper/asu1_vg02-asu1_lv02,size=8046117912576
sd=asu1_l03,lun=/dev/mapper/asu1_vg03-asu1_lv03,size=8046117912576
sd=asu2_l04,lun=/dev/mapper/asu2_vg04-asu2_lv04,size=8046117912576
sd=asu2_l05,lun=/dev/mapper/asu2_vg05-asu2_lv05,size=8046117912576
sd=asu2_l06,lun=/dev/mapper/asu2_vg06-asu2_lv06,size=8046117912576
sd=asu2_l07,lun=/dev/mapper/asu2_vg07-asu2_lv07,size=8046117912576
sd=asu3_l08,lun=/dev/mapper/asu3_vg08-asu3_lv08,size=1788258287616
sd=asu3_l09,lun=/dev/mapper/asu3_vg09-asu3_lv09,size=1788258287616
sd=asu3_l10,lun=/dev/mapper/asu3_vg10-asu3_lv10,size=1788258287616
sd=asu3_l11,lun=/dev/mapper/asu3_vg11-asu3_lv11,size=1788258287616
```

The content of SPC-1 Workload Generator command and parameter file used in this benchmark to execute the Persistence Test, which used a single Host System, is listed below.

```
javaparms="-Xms1536m -Xmx2048m -Xss256k"
sd=asu1_00,lun=/dev/mapper/asu1_vg00-asu1_lv00,size=8046117912576
sd=asu1_01,lun=/dev/mapper/asu1_vg01-asu1_lv01,size=8046117912576
sd=asu1_02,lun=/dev/mapper/asu1_vg02-asu1_lv02,size=8046117912576
sd=asu1_03,lun=/dev/mapper/asu1_vg03-asu1_lv03,size=8046117912576
sd=asu2_04,lun=/dev/mapper/asu2_vg04-asu2_lv04,size=8046117912576
sd=asu2_05,lun=/dev/mapper/asu2_vg05-asu2_lv05,size=8046117912576
sd=asu2_06,lun=/dev/mapper/asu2_vg06-asu2_lv06,size=8046117912576
sd=asu2_07,lun=/dev/mapper/asu2_vg07-asu2_lv07,size=8046117912576
sd=asu3_08,lun=/dev/mapper/asu3_vg08-asu3_lv08,size=1788258287616
sd=asu3_09,lun=/dev/mapper/asu3_vg09-asu3_lv09,size=1788258287616
sd=asu3_10,lun=/dev/mapper/asu3_vg10-asu3_lv10,size=1788258287616
sd=asu3_11,lun=/dev/mapper/asu3_vg11-asu3_lv11,size=1788258287616
```

APPENDIX E: SPC-1 WORKLOAD GENERATOR INPUT PARAMETERS

The following script was used to execute the required ASU pre-fill, Primary Metrics Test *(Sustainability Test Phase, IOPS Test Phase, and Response Time Ramp Test Phase)*, Repeatability Test *(Repeatability Test Phase 1 and Repeatability Test Phase 2)*, and Persistence Test Run 1 in an uninterrupted sequence. The script also included the appropriate commands to capture the detailed TSC profile listings required for a Remote Audit.

```
#!/bin/bash

export CLASSPATH=/SPC1InstallDir
export LD_LIBRARY_PATH=/SPC1InstallDir

# Collect Audit Inventory Data before SPC-1 Audit runs
/SPC1RunDir/collect.before.audit.inventory.sh

# Vdbench luns pre-fill run
cd /vdbench503rc11
/vdbench503rc11/vdbench -f /vdbench503rc11/prefill_luns.txt -o
/vdbench503rc11/luns.prefill

# SPC-1 benchmark Metrics Test
cd /SPC1RunDir
/SPC1RunDir/start.all.slave.jvms.sh
java -Xms1536m -Xmx2048m -Xss256k metrics -b 5001 -t 28800 -r 600 -s 3600:180
/SPC1RunDir/stop.all.slave.jvms.sh

# SPC-1 benchmark Repeatability1 Test
/SPC1RunDir/start.all.slave.jvms.sh
java -Xms1536m -Xmx2048m -Xss256k repeat1 -b 5001
/SPC1RunDir/stop.all.slave.jvms.sh

# SPC-1 benchmark Repeatability2 Test
/SPC1RunDir/start.all.slave.jvms.sh
java -Xms1536m -Xmx2048m -Xss256k repeat2 -b 5001
/SPC1RunDir/stop.all.slave.jvms.sh

# Prepare spc1.cfg file for Persistence1 Test
/bin/cp /SPC1RunDir/spc1.cfg /SPC1RunDir/spc1.cfg.multihost
/bin/cp /SPC1RunDir/spc1.cfg.persist /SPC1RunDir/spc1.cfg

# SPC-1 benchmark Persistence1 Test
java -Xms1536m -Xmx2048m -Xss256k persist1 -b 4000
```

Persistence Test Run 2

The following script was used to execute Persistence Test Run 2 *(read phase)* after the required TSC power shutdown and restart.

```
#!/bin/bash

export CLASSPATH=/SPC1InstallDir
export LD_LIBRARY_PATH=/SPC1InstallDir

# SPC-1 benchmark run - Persistence2 after power cycle of TSC
cd /SPC1RunDir
java -Xms1536m -Xmx2048m -Xss256k persist2
sleep 5

# Collect Audit Inventory Data after SPC-1 Audit runs
/SPC1RunDir/collect.after.audit.inventory.sh

# Move audit results directories and files and audit inveroty data to a central
location
/SPC1RunDir/move.audit.results.sh
```

Slave JVMs

All of the Slave JVMs were started prior to each Test Run and terminated at the end of each Test Run using the following scripts, **start.all.slave.jvms.sh** and **stop.all.slave.jvms.sh**.

start.all.slave.jvms.sh

```
#!/bin/bash

ssh stlrx300s6-110 'cd /SPC1RunDir; /SPC1RunDir/launch_host1_slaves.sh' &
sleep 10
ssh stlrx300s6-111 'cd /SPC1RunDir; /SPC1RunDir/launch_host2_slaves.sh' &
sleep 10
ssh stlrx300s6-112 'cd /SPC1RunDir; /SPC1RunDir/launch_host3_slaves.sh' &
sleep 10
ssh stlrx300s6-113 'cd /SPC1RunDir; /SPC1RunDir/launch_host4_slaves.sh' &
sleep 10
ssh stlrx300s6-114 'cd /SPC1RunDir; /SPC1RunDir/launch_host5_slaves.sh' &
sleep 10
ssh stlrx300s6-115 'cd /SPC1RunDir; /SPC1RunDir/launch_host6_slaves.sh' &
sleep 10
```

The **start.all.slave.jvms.sh** script invoked a script on each of the six Host Systems to start the Slave JVMs on each Host System. The script for the first Host System is listed below.

launch_host1_slaves.sh

```
#!/bin/bash

export CLASSPATH=/SPC1InstallDir
export LD_LIBRARY_PATH=/SPC1InstallDir

java -Xms1536m -Xmx2048m -Xss256k spc1 -f /SPC1RunDir/h1_slave1.txt 2>&1 &
sleep 1
java -Xms1536m -Xmx2048m -Xss256k spc1 -f /SPC1RunDir/h1_slave2.txt 2>&1 &
sleep 1
java -Xms1536m -Xmx2048m -Xss256k spc1 -f /SPC1RunDir/h1_slave3.txt 2>&1 &
sleep 1
java -Xms1536m -Xmx2048m -Xss256k spc1 -f /SPC1RunDir/h1_slave4.txt 2>&1 &
sleep 1
java -Xms1536m -Xmx2048m -Xss256k spc1 -f /SPC1RunDir/h1_slave5.txt 2>&1 &
sleep 1
java -Xms1536m -Xmx2048m -Xss256k spc1 -f /SPC1RunDir/h1_slave6.txt 2>&1 &
sleep 1
java -Xms1536m -Xmx2048m -Xss256k spc1 -f /SPC1RunDir/h1_slave7.txt 2>&1 &
sleep 1
java -Xms1536m -Xmx2048m -Xss256k spc1 -f /SPC1RunDir/h1_slave8.txt 2>&1 &
sleep 1
java -Xms1536m -Xmx2048m -Xss256k spc1 -f /SPC1RunDir/h1_slave9.txt 2>&1 &
```

stop.all.slave.jvms.sh

```
#!/bin/bash

ssh stlrx300s6-110 'pkill java'
sleep 10
ssh stlrx300s6-111 'pkill java'
sleep 10
ssh stlrx300s6-112 'pkill java'
sleep 10
ssh stlrx300s6-113 'pkill java'
sleep 10
ssh stlrx300s6-114 'pkill java'
sleep 10
ssh stlrx300s6-115 'pkill java'
sleep 10
```