SPC BENCHMARK 1™
FULL DISCLOSURE REPORT

NETAPP, INC.
NETAPP® FAS8080 EX *(ALL-FLASH FAS)*

SPC-1 V1.14

Submitted for Review: April 22, 2015
Submission Identifier: A00154

First Edition – April 2015

Trademarks

Table of Contents

AUDIT CERTIFICATION

Chad Morgenstern
NetApp, Inc.
7301 Kit Creek
Morrisville, NC 27513

April 21, 2015

The SPC Benchmark 1™ Reported Data listed below for the NetApp® FAS8080 EX *(All-Flash FAS)*was produced in compliance with the SPC Benchmark 1™ v1.14 Onsite Audit requirements.

SPC Benchmark 1™ v1.14 Reported Data	
Tested Storage Product (TSP) Name: NetApp® FAS8080 EX *(All-Flash FAS)*	
Metric	**Reported Result**
SPC-1 IOPS™	685,281.71
SPC-1 Price-Performance	$2.77/SPC-1 IOPS™
Total ASU Capacity	47,563.542 GB
Data Protection Level	Protected 2 *(RAID DP®)*
Total Price (including three-year maintenance)	$1,897,999.00
Currency Used	U.S. Dollars
Target Country for availability, sales and support	USA

The following SPC Benchmark 1™ Onsite Audit requirements were reviewed and found compliant with 1.14 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.
- The total Application Storage Unit (ASU) Capacity was filled with random data, using an auditor approved tool, prior to execution of the SPC-1 Tests.

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

AUDIT CERTIFICATION *(CONT.)*

- 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.
- 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 System including the number of processors and main memory.
  - ✓ The presence and version number of the SPC-1 Workload Generator on the Host System.
  - ✓ The TSC boundary within the 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 was no difference between the Tested Storage Configuration (TSC) 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:

There are no audit notes.

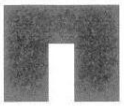
Respectfully,

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



www.netapp.com
+1 408 822 6000 Tel
+1 408 822 4501 Fax
NetApp
495 East Java Drive
Sunnyvale, CA 94089

Date: April 16, 2015

From: Dan Neault, Senior Vice President, Partners, Solutions and Enablement Group
NetApp Inc.
495 East Java Drive
Sunnyvale, CA 94089

To: Mr. Walter E. Baker, SPC Auditor
Gradient Systems
643 Bair Island Road, Suite 103
Redwood City, CA 94063-2755

Subject: SPC-1 Letter of Good Faith for the NetApp FAS8080 EX (All-Flash FAS) 8-Node Cluster

NetApp Inc. is the SPC-1 Test Sponsor for the above listed product. To the best of our knowledge and belief, the required SPC-1 benchmark results and materials we have submitted for that product are complete, accurate, and in full compliance with V1.14 of the SPC-1 benchmark specification.

In addition, we have reported any items in the Benchmark Configuration and execution of the benchmark that affected the reported results even if the items are not explicitly required to be disclosed by the SPC-1 benchmark specification.

Signed: Date:

Dan Neault
Senior Vice President, Partners, Solutions
and Enablement Group

16 April 2015
Date of Signature

EXECUTIVE SUMMARY

Test Sponsor and Contact Information

Test Sponsor and Contact Information	
Test Sponsor Primary Contact	NetApp, Inc. – http://www.netapp.com Chad Morgenstern – chad.morgenstern@netapp.com 7301 Kit Creek Morrisville, NC 27513 Phone: (919) 476-8158 FAX: (919) 476-4272
Test Sponsor Alternate Contact	NetApp, Inc. – http://www.netapp.com Scott Lane – scott.lane@netapp.com 7301 Kit Creek Morrisville, NC 27513 Phone: (919) 476-5484 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	V1.14
SPC-1 Workload Generator revision number	V2.3.0
Date Results were first used publicly	April 22, 2015
Date the FDR was submitted to the SPC	April 22, 2015
Date the Priced Storage Configuration is available for shipment to customers	currently available
Date the TSC completed audit certification	April 21, 2015

Tested Storage Product (TSP) Description

NetApp's most powerful storage array, the NetApp® FAS8080 EX is purpose-built for business-critical workloads requiring massive performance, multi-PB scale, and leading flash integration—including all-flash configurations.

Summary of Results

SPC-1 Reported Data	
Tested Storage Product (TSP) Name: NetApp® FAS8080 EX *(All-Flash FAS)*	
Metric	**Reported Result**
SPC-1 IOPS™	685,281.71
SPC-1 Price-Performance™	$2.77/SPC-1 IOPS™
Total ASU Capacity	47,563.542 GB
Data Protection Level	Protected 2 *(RAID DP®)*
Total Price	$1,897,999.00
Currency Used	U.S. Dollars
Target Country for availability, sales and support	USA

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

SPC-1 Price-Performance™ is the ratio of **Total Price** to **SPC-1 IOPS™**.

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

A **Data Protection Level** of **Protected 2** using ***RAID-DP®***, 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/us/products/platform-os/raid-dp.aspx.

> ***Protected 2:*** *The single point of failure of any* ***component*** *in the configuration will not result in permanent loss of access to or integrity of the SPC-1 Data Repository.*

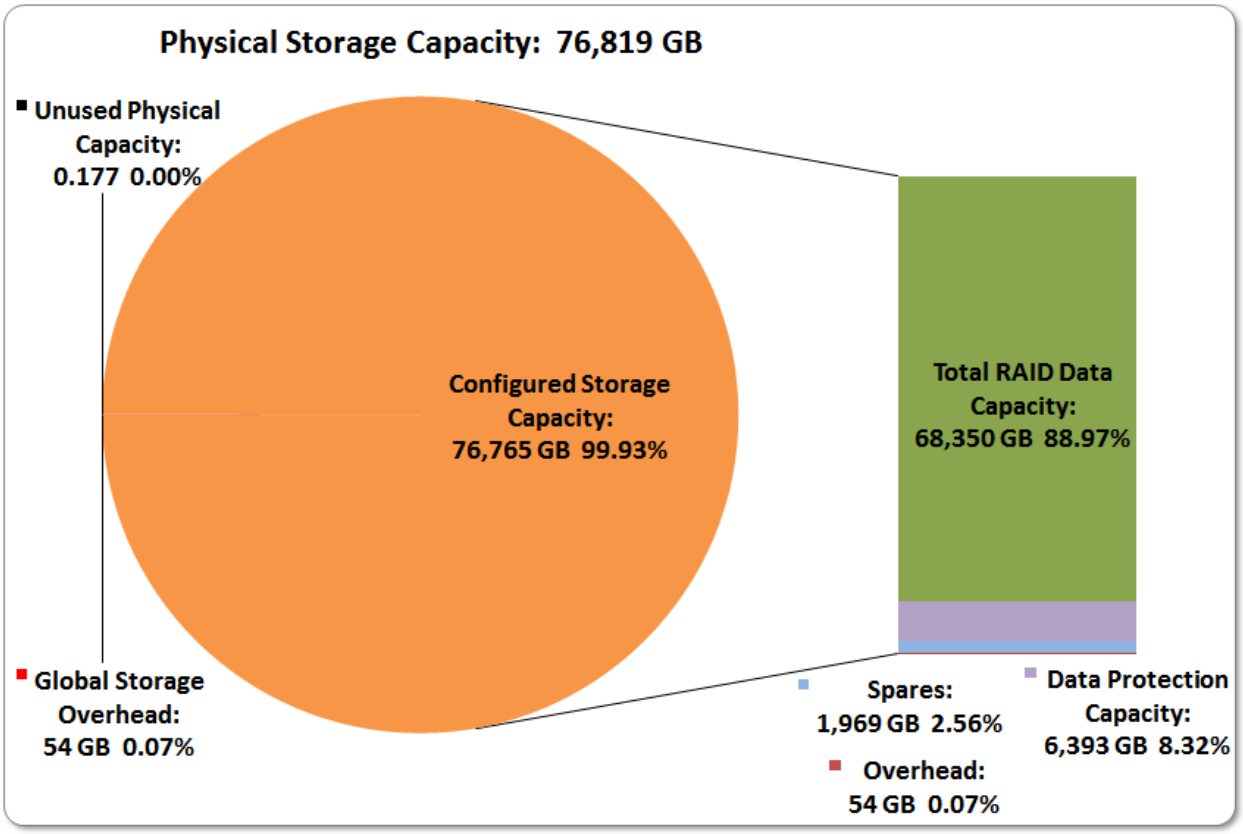
Total Price includes the cost of the Priced Storage Configuration plus three years of hardware maintenance and software support as detailed on page 16.

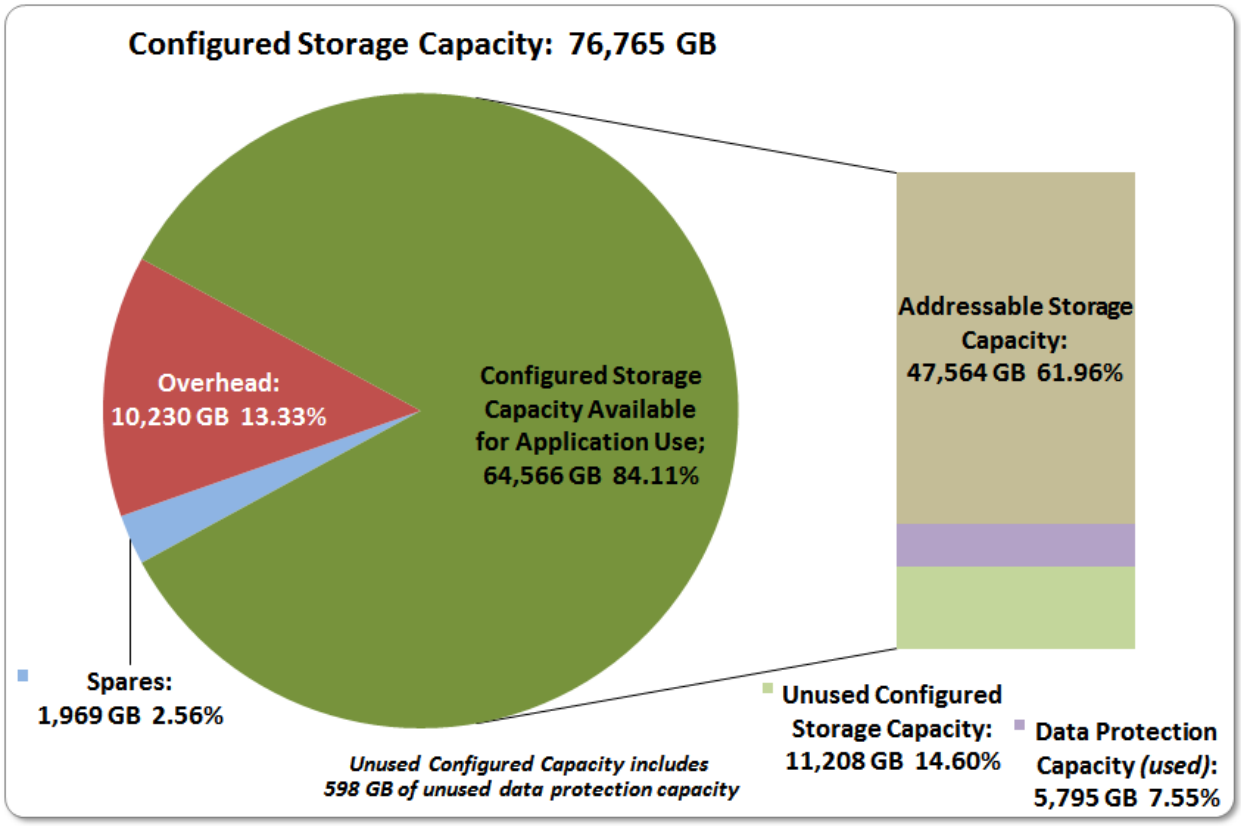
Currency Used is formal name for the currency used in calculating the **Total Price** and **SPC-1 Price-Performance™**. That currency may be the local currency of the **Target Country** or the currency of a difference country *(non-local currency)*.

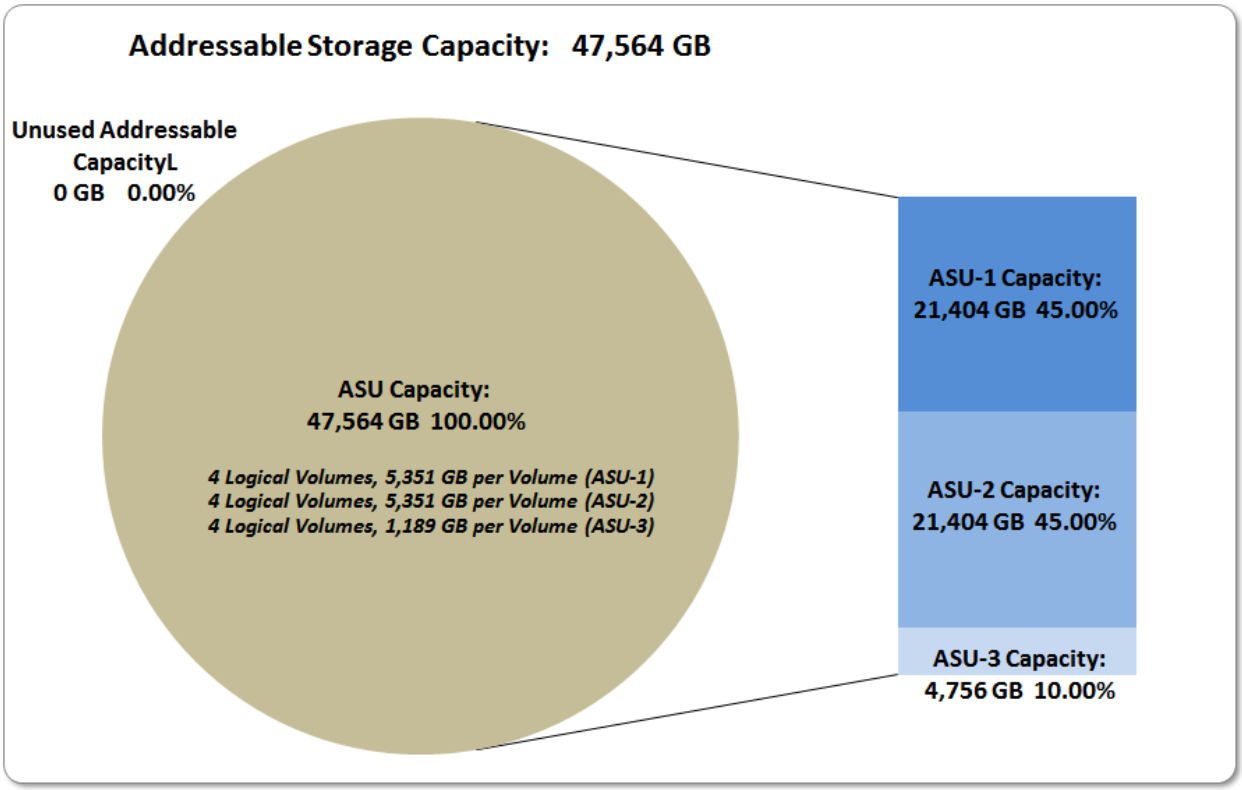
The **Target Country** is the country in which the Priced Storage Configuration is available for sale and in which the required hardware maintenance and software support is provided either directly from the Test Sponsor or indirectly via a third-party supplier.

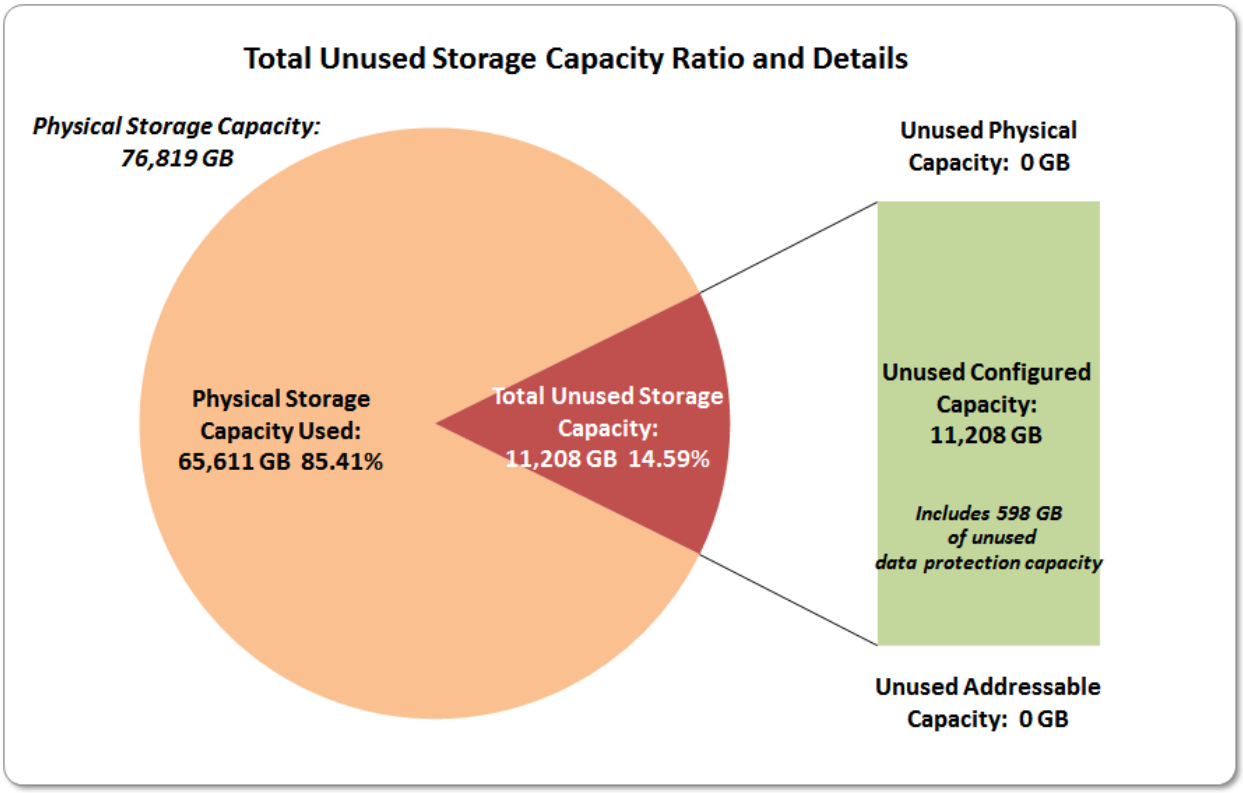
Storage Capacities, Relationships, and Utilization

The following four charts 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.




Configured Storage Capacity: 76,765 GB
Overhead:
10,230 GB 13.33%
Configured Storage Capacity Available for Application Use;
64,566 GB 84.11%
Spares:
1,969 GB 2.56%
Addressable Storage Capacity:
47,564 GB 61.96%
Unused Configured Storage Capacity:
11,208 GB 14.60%
Data Protection Capacity (used):
5,795 GB 7.55%
Unused Configured Capacity includes
598 GB of unused data protection capacity


Addressable Storage Capacity: 47,564 GB
Unused Addressable CapacityL
0 GB 0.00%
ASU Capacity:
47,564 GB 100.00%
4 Logical Volumes, 5,351 GB per Volume (ASU-1)
4 Logical Volumes, 5,351 GB per Volume (ASU-2)
4 Logical Volumes, 1,189 GB per Volume (ASU-3)
ASU-1 Capacity:
21,404 GB 45.00%
ASU-2 Capacity:
21,404 GB 45.00%
ASU-3 Capacity:
4,756 GB 10.00%



SPC-1 Storage Capacity Utilization	
Application Utilization	61.92%
Protected Application Utilization	69.46%
Unused Storage Ratio	14.59%

Application Utilization: Total ASU Capacity *(46,563.542 GB)* divided by Physical Storage Capacity *(76,819.065 GB)*.

Protected Application Utilization: Total ASU Capacity *(46,563.542 GB)* plus total Data Protection Capacity *(6,392.581 GB)* minus unused Data Protection Capacity *(597.883 GB)* divided by Physical Storage Capacity *(76,819.065 GB)*.

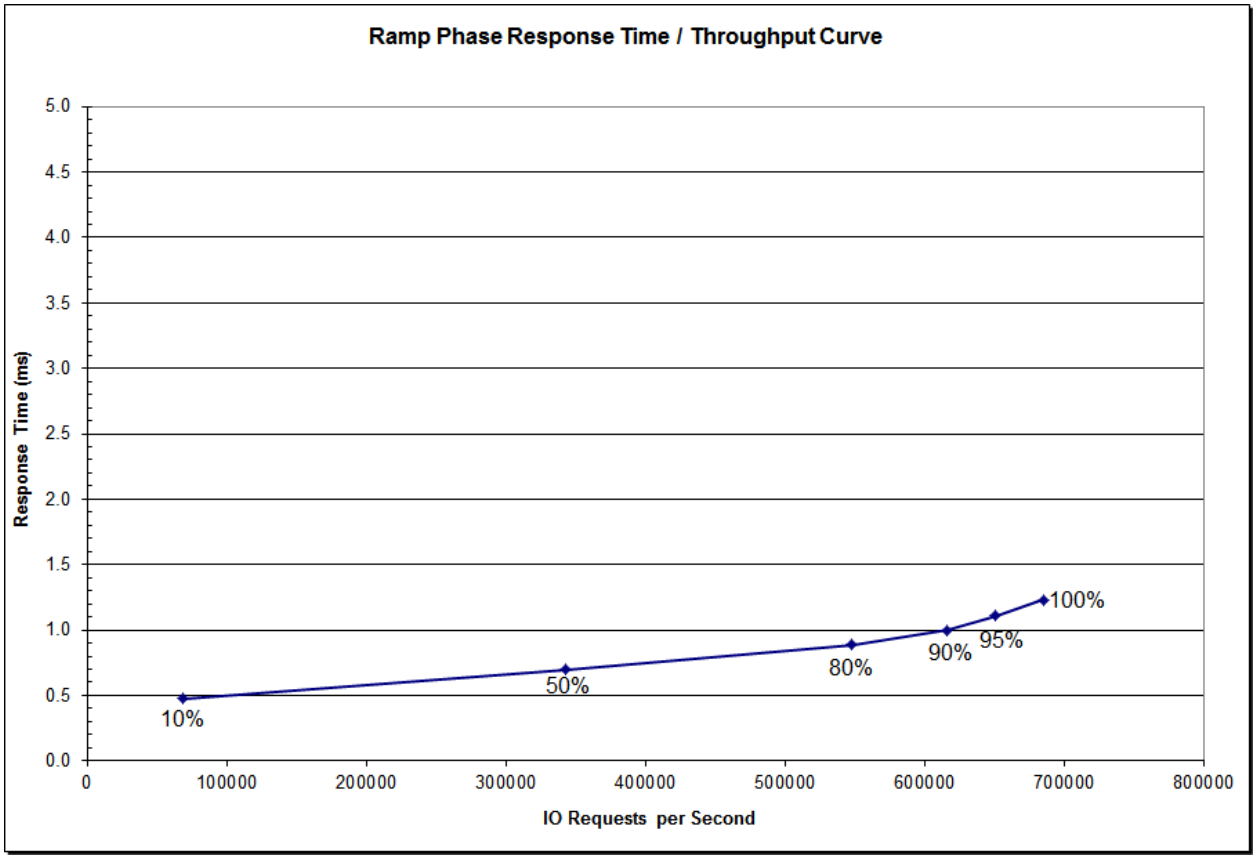
Unused Storage Ratio: Total Unused Capacity *(11,208.123 GB)* divided by Physical Storage Capacity *(76,819.065 GB)* and may not exceed 45%.

Detailed information for the various storage capacities and utilizations is available on pages 25-26.

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	68,497.10	342,546.55	548,211.09	616,717.59	650,962.80	685,281.71
Average Response Time (ms):						
All ASUs	0.48	0.70	0.89	1.00	1.11	1.23
ASU-1	0.43	0.65	0.85	0.97	1.09	1.23
ASU-2	0.51	0.74	0.92	1.02	1.13	1.23
ASU-3	0.56	0.78	0.95	1.05	1.15	1.24
Reads	0.36	0.57	0.76	0.87	0.98	1.14
Writes	0.56	0.78	0.97	1.09	1.20	1.29

Priced Storage Configuration Pricing

	Description	List Price per UNIT	Quantity	Extended List
FAS8080AE-F-002-19.2TB-R6	FAS8080 HA System w/IOXM,19.2TB 48x200GB SSDs,1x4-port 3/6 Gb SAS card	$ 89,000.00	8	$ 712,000.00
SW-2-8080A-FCP	SW-2,FCP,8080A	$ 26,550.00	8	$ 212,400.00
OS-ONTAP-CAP3-0P-C	OS Enable,Per-0.1TB,ONTAP,Ultra-Stor,0P,-C	$ 780.00	768	$ 599,040.00
X-6510-48-16G-R6-C	Switch,Brocade 6510 48-Pt FF w/8G SWL SFPs	$ 48,715.00	2	$ 97,430.00
X-SFP-H10GB-CU3M-R6	Cisco N5020 10GBase Copper SFP+cable, 1m, -C, R6 (Cluster interconnect)	$ 112.00	32	$ 3,584.00
X1967-R6	ClusterNet Interconnect,48Pt,10Gb	$ 34,500.00	2	$ 69,000.00
X6524-R6	Cable,Cntlr-Shelf/Switch,2m,Pair,LC/LC,Op	$ 125.00	50	$ 6,250.00
Third-Party	QLogic QLE2672 16Gig 2port HBA for servers	$ 1,250.00	9	$ 11,250.00
X2065A-EN-R6-C	HBA SAS 4-Port Copper 3/6 Gb QSFP PCIe,EN,-C	$ 1,400.00	8	$ 11,200.00
X6558-EN-R6-C	Cable, SAS Cntlr-Shelf/Shelf-Shelf/HA, 2m, -C, 16 per cluster, 4 x 16	$ 125.00	64	$ 8,000.00
X8712C-EN-R6-C	PDU, 1-Phase, 24 Outlet, 30A, NEMA, -C, R6	$ 550.00	4	$ 2,200.00
X870D-EN-R6-C	Cab,Deep,HeavyDuty,Empty,No PDU,No Rail,EN,-C	$ 3,595.00	2	$ 7,190.00
X8778-R6-C	Mounting Bracket, Tie-Down, 32X0, -C, R6	$ 50.00	8	$ 400.00
CS-A2-4R	SupportEdge Standard Replace 4hr, Hardware Support: 3 years			$ 81,591.00
SW Support	SW Support 3 years			$ 76,464.00
Total ($) LIST				**$ 1,897,999.00**

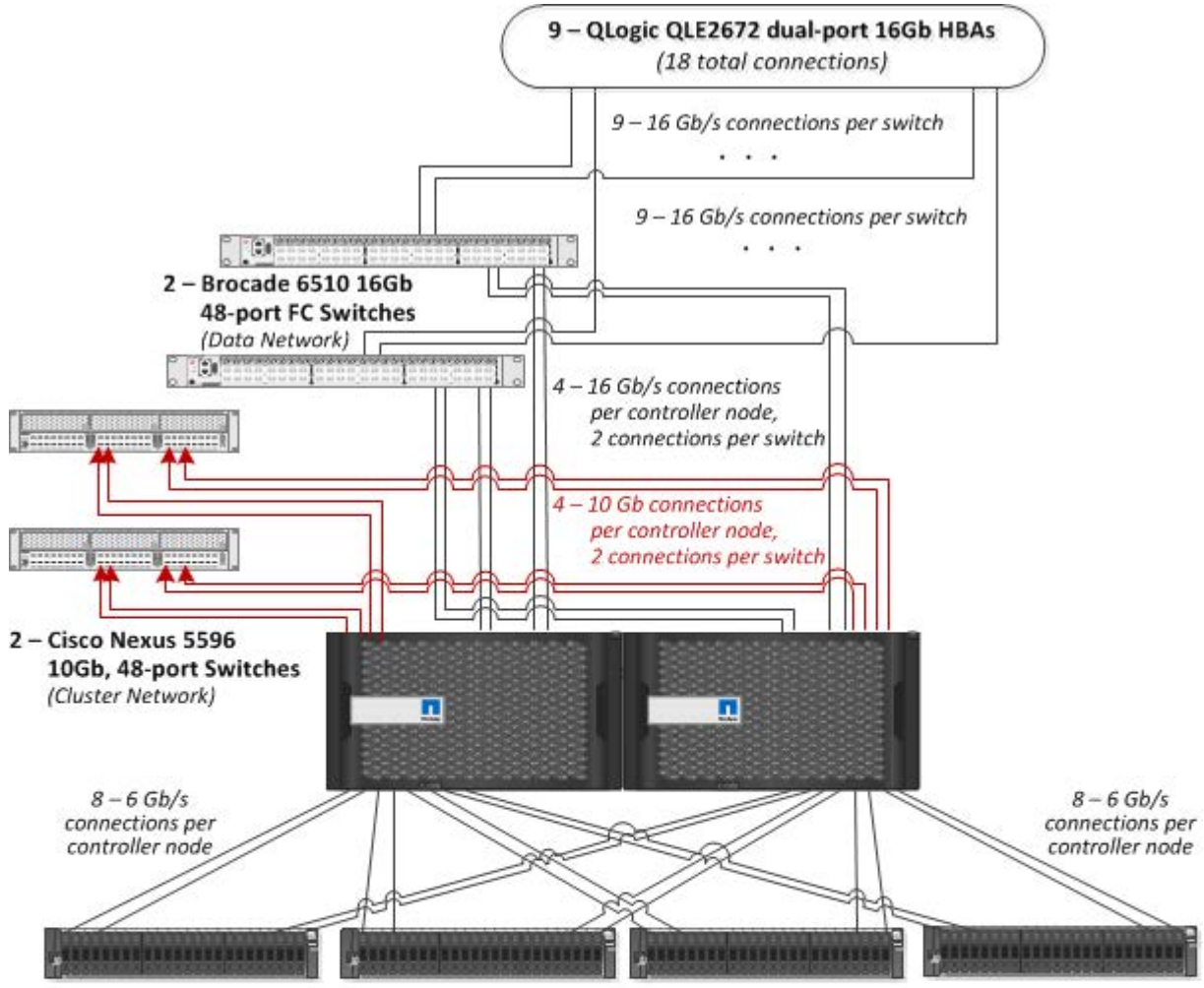
The above pricing includes hardware maintenance and software support for three years, 7 days per week, 24 hours per day. The hardware maintenance and software support provides the following:

- Acknowledgement of new and existing problems within 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 Priced Storage Configuration.

Priced Storage Configuration Diagram



NetApp® FAS8080 EX Storage System *(All Flash FAS)*

4 HA controller pairs, 2 controller nodes per pair
(1 HA controller pair, 2 controller nodes illustrated above)
8 controller nodes total
each controller node includes:
128 GB memory/cache *(1024 GB total)*
4 – 16Gb FC front-end connections *(32 total and used)*
2 – 4-port 6Gb SAS cards *(16 cards total)*
8 – 6Gb SAS backend connections *(64 total and used)*
16 – Disk Shelves *(2 per controller node)*
384 – 200GB eMLC Solid State Drives (SSDs)
(24 SSDs per disk shelf)

Priced Storage Configuration Components

Priced Storage Configuration
9 – QLogic QLE2672 dual-port 16Gb HBAs
2 – Brocade 6510 16Gb 48-port FC switches *(data network)*
NetApp® FAS8080 EX *(All-Flash FAS)*
4 HA controller pairs
2 controller nodes per pair
8 controller nodes total
each controller node includes:
128 GB memory/cache *(1024 GB total)*
4 – 16Gb FC front-end connections *(32 total and used)*
8 – 6Gb SAS backend connections *(64 total and used)* *(in a Multipath, High Availability (HA) configuration)*
2 – 4-port SAS cards *(16 cards total)*
8 – SAS backend ports (*64 ports total)*
2 – Cisco Nexus 5596 10Gb 48-port switches *(cluster network)*
16 – DS2426 Disk Shelves
384 – 200GB eMLC Solid State Drives (SSDs) *(24 SSDs per disk shelf)*
2 – Cabinets – deep, heavy, no PDU, no rail
4 – PDUs – 1-phase, 24 outlet, 30A NEMA *(2 PDUs per cabinet)*

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 21 *(Benchmark Configuration/Tested Storage Configuration Diagram)*.

Host System(s) 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).

The Host System(s) and TSC table of components may be found on page 21 *(Host System(s) and Tested Storage Configuration (TSC) Table of Components)*.

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.

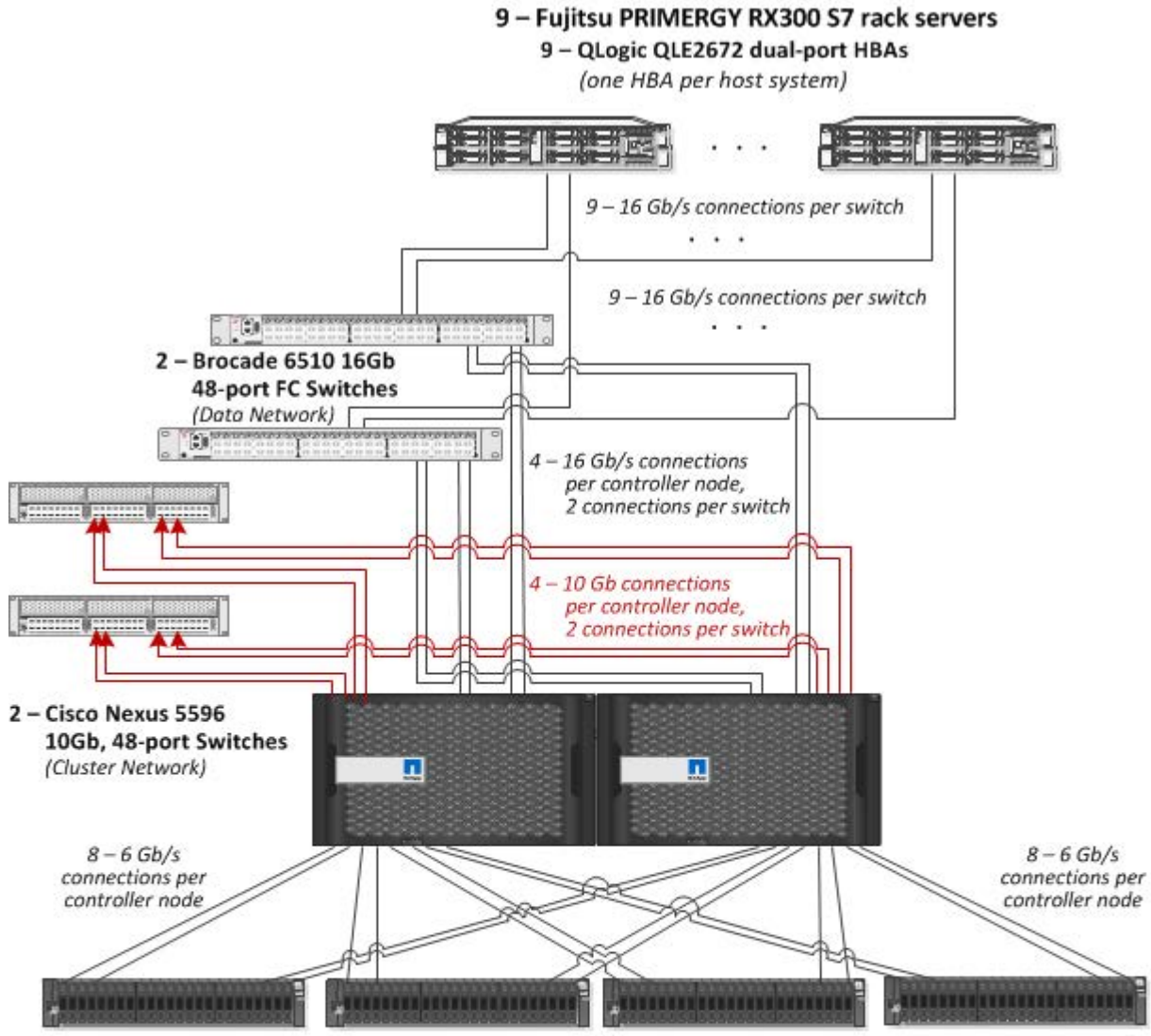
The storage network configuration is illustrated on page 21 *(Benchmark Configuration/Tested Storage Configuration Diagram)*.

There were no specific port mappings for any of the switches in the Tested Storage Configuration. But, one of the ports on each HBA was connected to one data network switch and the other port on each HBA is connected to the other data network switch. This

same approach was applied to the the connections from the controller nodes to the data network switches and to the cluster network switches.

This approach was for reliability/redundancy in business critical environment and not for performance reasons.

Benchmark Configuration/Tested Storage Configuration Diagram



NetApp® FAS8080 EX Storage System *(All Flash FAS)*

4 HA controller pairs, 2 controller nodes per pair
(1 HA controller pair, 2 controller nodes illustrated above)
8 controller nodes total
each controller node includes:
128 GB memory/cache *(1024 GB total)*
4 – 16Gb FC front-end connections *(32 total and used)*
2 – 4-port 6Gb SAS cards *(16 cards total)*
8 – 6Gb SAS backend connections *(64 total and used)*
16 – Disk Shelves *(2 per controller node)*
384 – 200GB eMLC Solid State Drives (SSDs)
(24 SSDs per disk shelf)

Host System and Tested Storage Configuration Components

Host Systems
9 – Fujitsu RX300 S7 servers, each with: 2 – Intel® Xeon® 2.30 GHz E5-2630 processors each with 6 cores, 15 MB Intel® Smart Cache 128 GB main memory Red Hat Enterprise Linux Server 6.4 (64-bit) PCIe 3.0
Priced Storage Configuration
9 – QLogic QLE2672 dual-port 16Gb HBAs
2 – Brocade 6510 16Gb, 48-port FC switches *(data network)*
NetApp® FAS8080 EX *(All-Flash FAS)* 4 HA controller pairs 2 controller nodes per pair 8 controller nodes total each controller node includes: 128 GB memory/cache *(1024 GB total)* 4 – 16Gb FC front-end connections *(32 total and used)* 8 – 6Gb SAS backend connections *(64 total and used)* *(in a Multipath, High Availability (HA) configuration)* 2 – 4-port SAS cards *(16 cards total)* 8 – SAS backend ports (*64 ports total)*
2 – Cisco Nexus 5596 10Gb 48-port switches *(cluster network)*
16 – DS2426 Disk Shelves
384 – 200GB eMLC Solid State Drives (SSDs) *(24 SSDs per disk shelf)*
2 – Cabinets – deep, heavy, no PDU, no rail
4 – PDUs – 1-phase, 24 outlet, 30A NEMA *(2 PDUs per cabinet)*

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 66 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 70 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 100.

ASU Pre-Fill

Clause 5.3.3

Each of the three SPC-1 ASUs (ASU-1, ASU-2 and ASU-3) is required to be completely filled with specified content prior to the execution of audited SPC-1 Tests. The content is required to consist of random data pattern such as that produced by an SPC recommended tool.

The configuration file used to complete the required ASU pre-fill appears in Appendix D: SPC-1 Workload Generator Storage Commands and Parameters on page 100.

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 62 contains definitions of terms specific to the SPC-1 Data Repository.

Storage Capacities and Relationships

Clause 9.4.3.6.1

Two tables and four charts documenting the storage capacities and relationships of the SPC-1 Storage Hierarchy (Clause 2.1) shall be included in the FDR. … The capacity value in each chart may be listed as an integer value, for readability, rather than the decimal value listed in the table below.

SPC-1 Storage Capacities

The Physical Storage Capacity consisted 76,819.065 GB distributed over 384 solid state storage devices (SSDs), each with a formatted capacity of 200.05 GB. There was 0.177 GB (0.0002%) of Unused Storage within the Physical Storage Capacity. Global Storage Overhead consisted of 54.324 GB (0.07%) of the Physical Storage Capacity. There was 11,207.945 GB (14.60%) of Unused Storage within the Configured Storage Capacity. The Total ASU Capacity utilized 100.00% of the Addressable Storage Capacity resulting in 0.000 GB (0.00%) of Unused Storage within the Addressable Storage Capacity. The Data Protection *(RAID DP®)* capacity was 6,392.581 GB of which 5,794.697 GB was utilized. The total Unused Storage capacity was 11,208.123 GB.

Note: The configured Storage Devices may include additional storage capacity reserved for system overhead, which is not accessible for application use. That storage capacity may not be included in the value presented for Physical Storage Capacity.

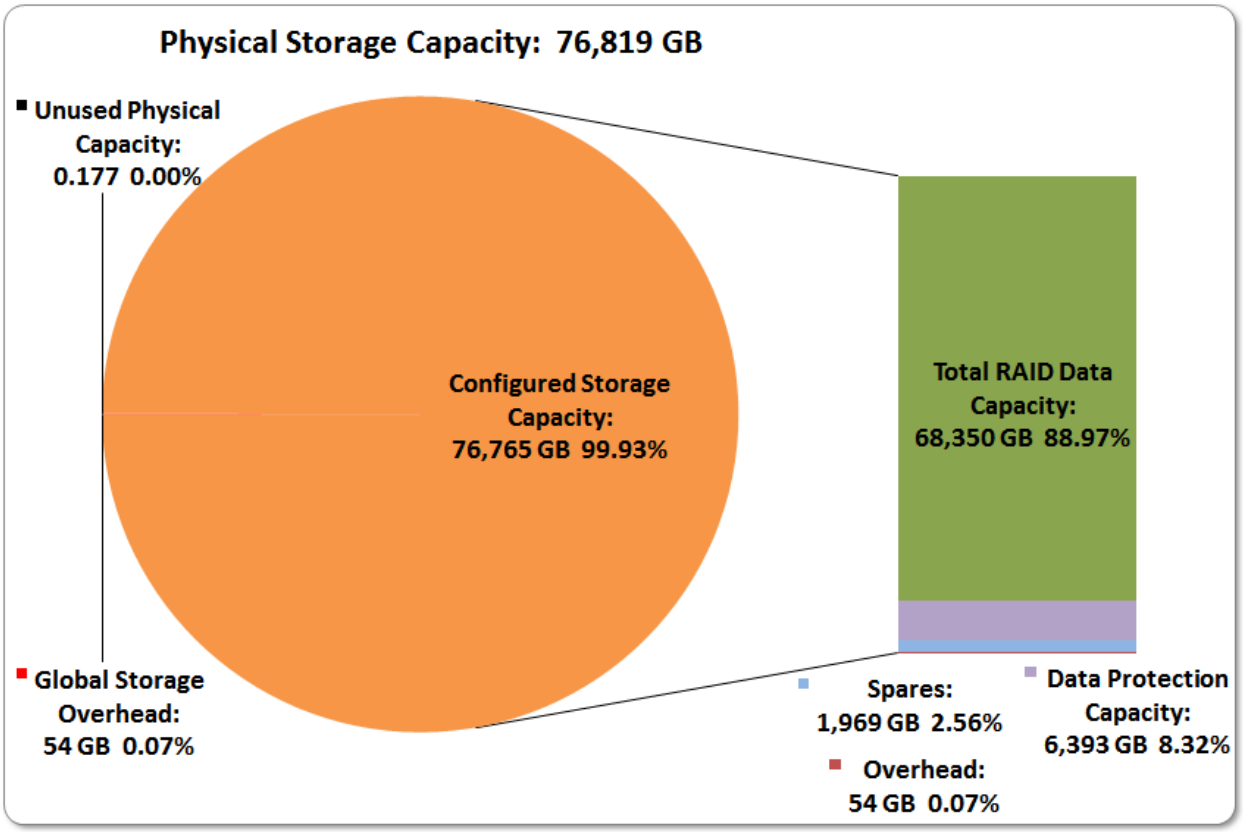
SPC-1 Storage Capacities

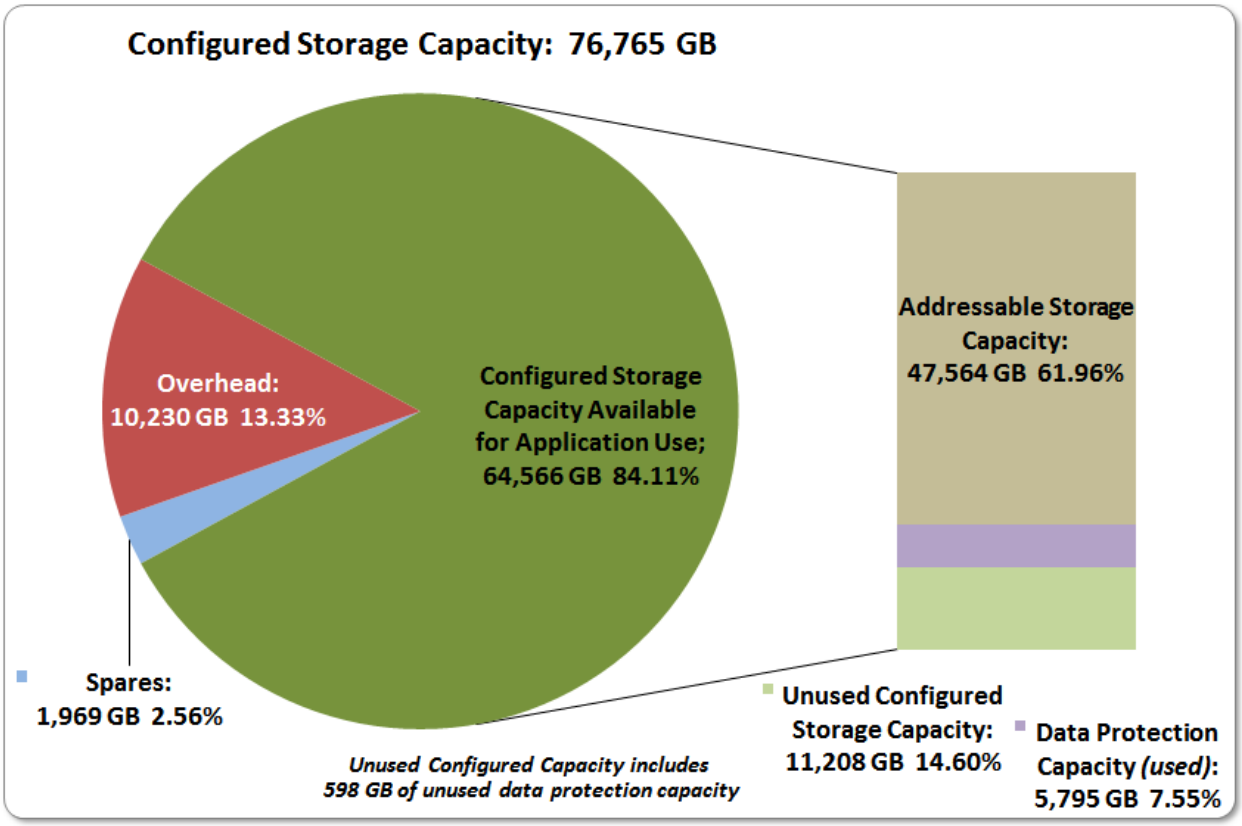
Storage Hierarchy Component	Units	Capacity
Total ASU Capacity	Gigabytes (GB)	47.563.542
Addressable Storage Capacity	Gigabytes (GB)	47,563.542
Configured Storage Capacity	Gigabytes (GB)	76,764.563
Physical Storage Capacity	Gigabytes (GB)	76,819.065
Data Protection *(RAID DP®)*	Gigabytes (GB)	6,392.581
Required Storage *(includes overhead, sparing)*	Gigabytes (GB)	12,198.379
Global Storage Overhead	Gigabytes (GB)	54.324
Total Unused Storage	Gigabytes (GB)	11,208.123

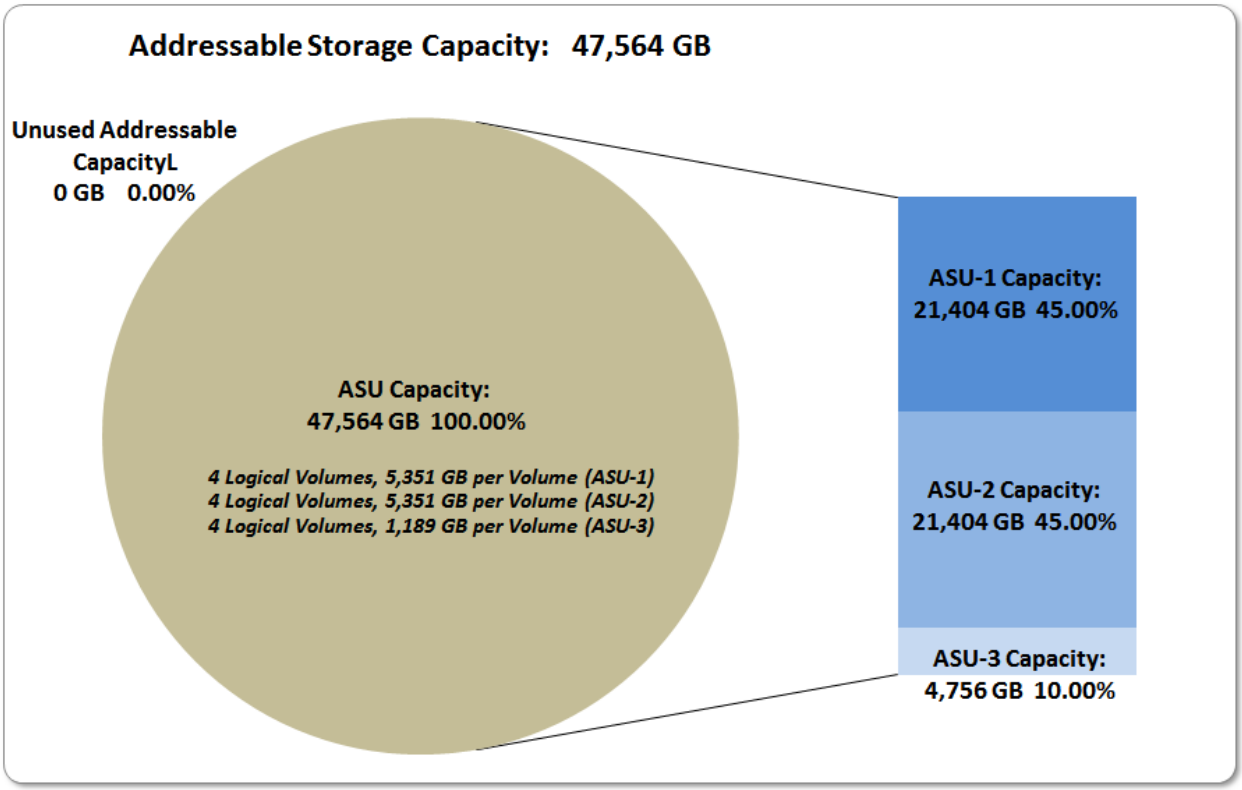
SPC-1 Storage Hierarchy Ratios

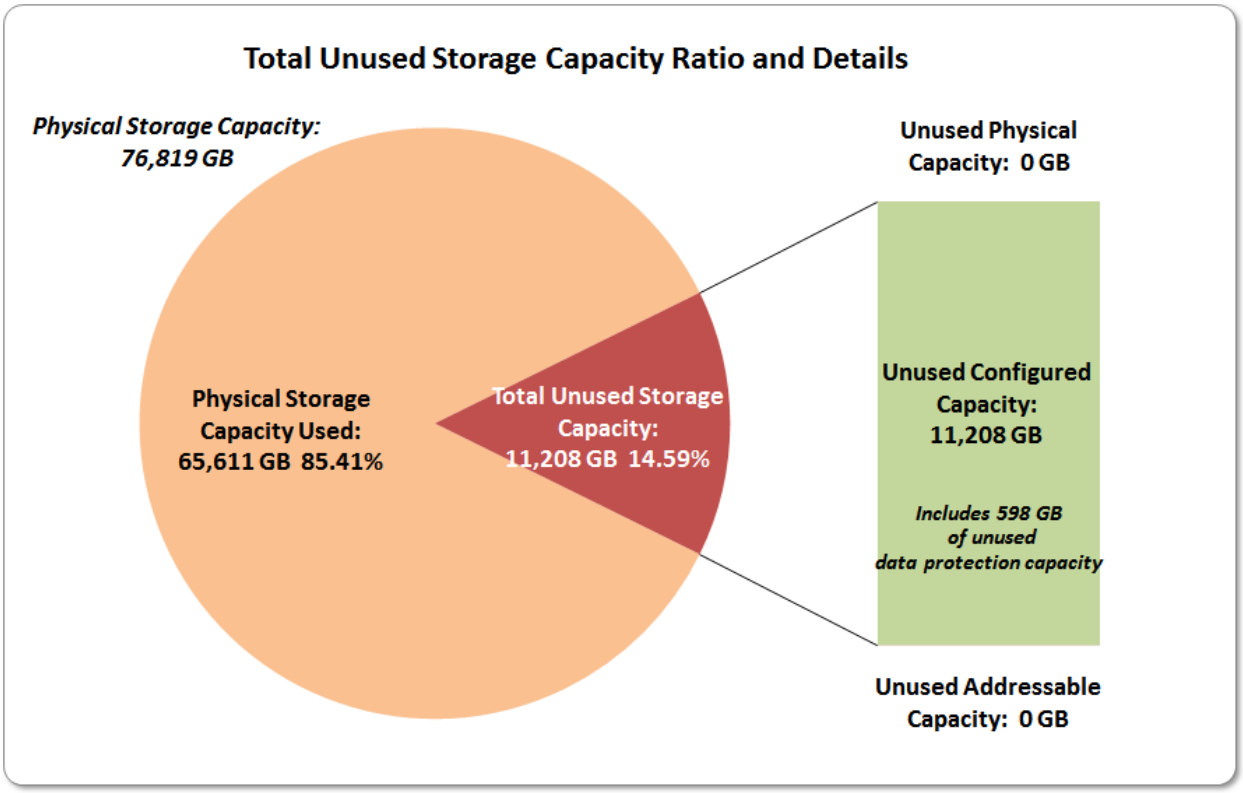
	Addressable Storage Capacity	Configured Storage Capacity	Physical Storage Capacity
Total ASU Capacity	100.00%	61.96%	61.92%
Required for Data Protection *(RAID DP®)*		8.33%	8.32%
Addressable Storage Capacity		61.96%	61.92%
Required Storage *(includes overhead, sparing)*		15.89%	15.88%
Configured Storage Capacity			99.93%
Global Storage Overhead			0.07%
Unused Storage:			
Addressable	0.00%		
Configured		14.60%	
Physical			0.0002%

SPC-1 Storage Capacity Charts




Configured Storage Capacity: 76,765 GB
Overhead:
10,230 GB 13.33%
Configured Storage Capacity Available for Application Use;
64,566 GB 84.11%
Spares:
1,969 GB 2.56%
Addressable Storage Capacity:
47,564 GB 61.96%
Unused Configured Storage Capacity:
11,208 GB 14.60%
Data Protection Capacity (used):
5,795 GB 7.55%
Unused Configured Capacity includes
598 GB of unused data protection capacity


Addressable Storage Capacity: 47,564 GB
Unused Addressable CapacityL
0 GB 0.00%
ASU Capacity:
47,564 GB 100.00%
4 Logical Volumes, 5,351 GB per Volume (ASU-1)
4 Logical Volumes, 5,351 GB per Volume (ASU-2)
4 Logical Volumes, 1,189 GB per Volume (ASU-3)
ASU-1 Capacity:
21,404 GB 45.00%
ASU-2 Capacity:
21,404 GB 45.00%
ASU-3 Capacity:
4,756 GB 10.00%



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	61.92%
Protected Application Utilization	69.46%
Unused Storage Ratio	14.59%

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 (21,403.701 GB)	**ASU-2 (21,403.701 GB)**	**ASU-3 (4,756.139 GB)**
4 Logical Volume 5,350.925 GB per Logical Volume (5,350.925 GB used per Logical Volume)	4 Logical Volume 5,350.925 GB per Logical Volume (5,350.925 GB used per Logical Volume)	4 Logical Volume 1,189.035 GB per Logical Volume (1,189.035 GB used per Logical Volume)

The Data Protection Level used for all Logical Volumes was **Protected 2** using ***Mirroring*** as described on page 11. See "ASU Configuration" in the **IOPS Test Results File** for more detailed configuration information.

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. An SPC-1 glossary on page 62 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.

"Ramp-Up" Test Runs

Clause 5.3.13

In order to warm-up caches or perform the initial ASU data migration in a multi-tier configuration, a Test Sponsor may perform a series of "Ramp-Up" Test Runs as a substitute for an initial, gradual Ramp-Up.

Clause 5.3.13.3

The "Ramp-Up" Test Runs will immediately precede the Primary Metrics Test as part of the uninterrupted SPC-1 measurement sequence.

Clause 9.4.3.7.1

If a series of "Ramp-Up" Test Runs were included in the SPC-1 measurement sequence, the FDR shall report the duration (ramp-up and measurement interval), BSU level, SPC-1 IOPS and average response time for each "Ramp-Up" Test Run in an appropriate table.

There were no "Ramp-Up" Test Runs executed in this set of SPC-1 audited measurements.

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 eight (8) 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 9.4.3.7.2

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 102.

Sustainability Test Results File

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

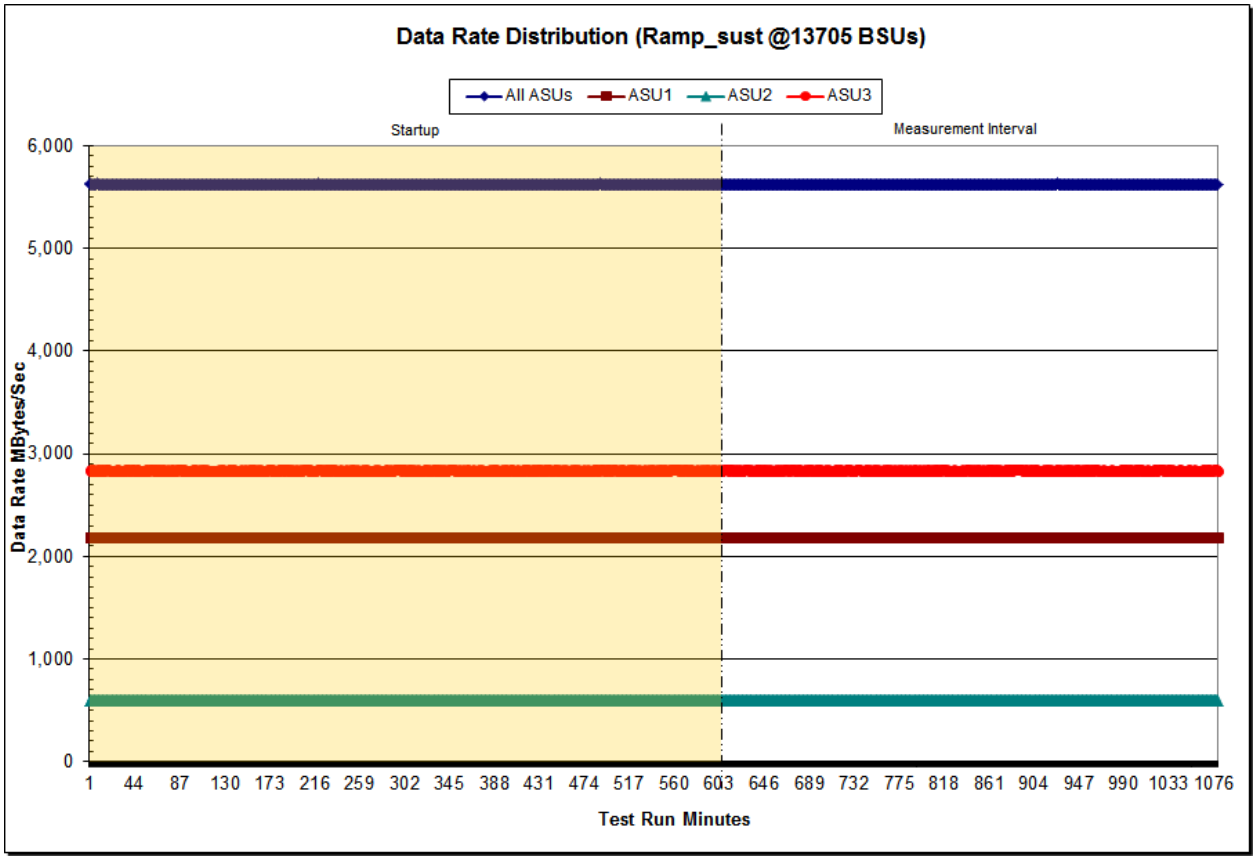
Sustainability Test Results File

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

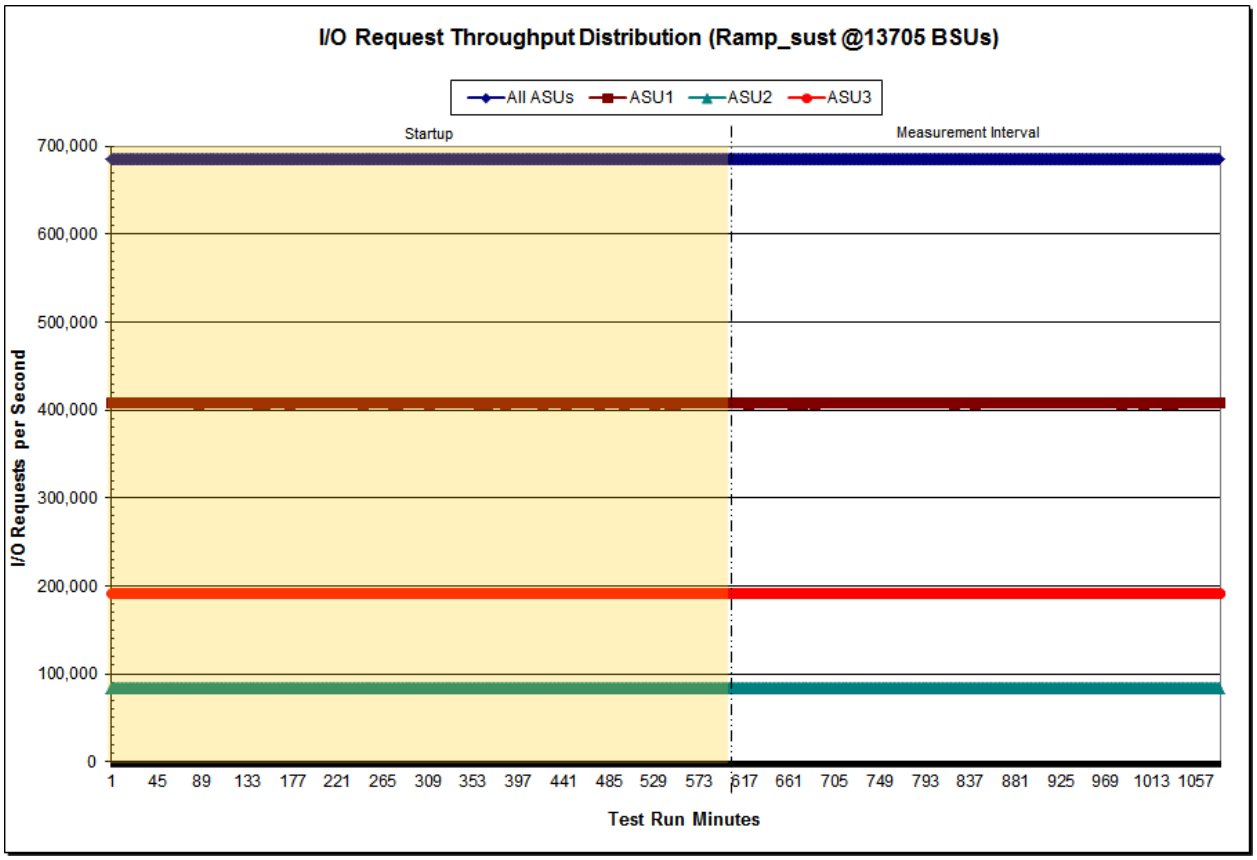


Sustainability – I/O Request Throughput Distribution Data

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

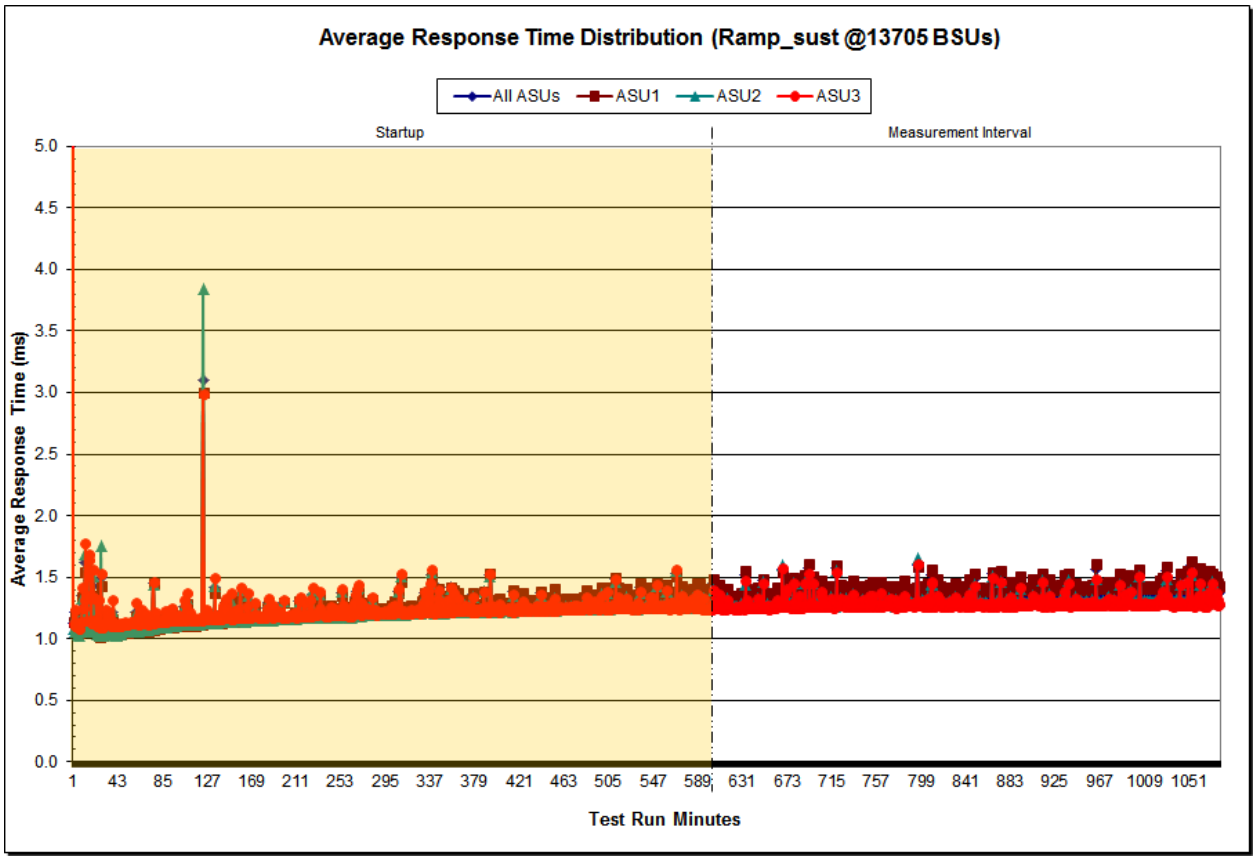


Sustainability – Average Response Time (ms) Distribution Data

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



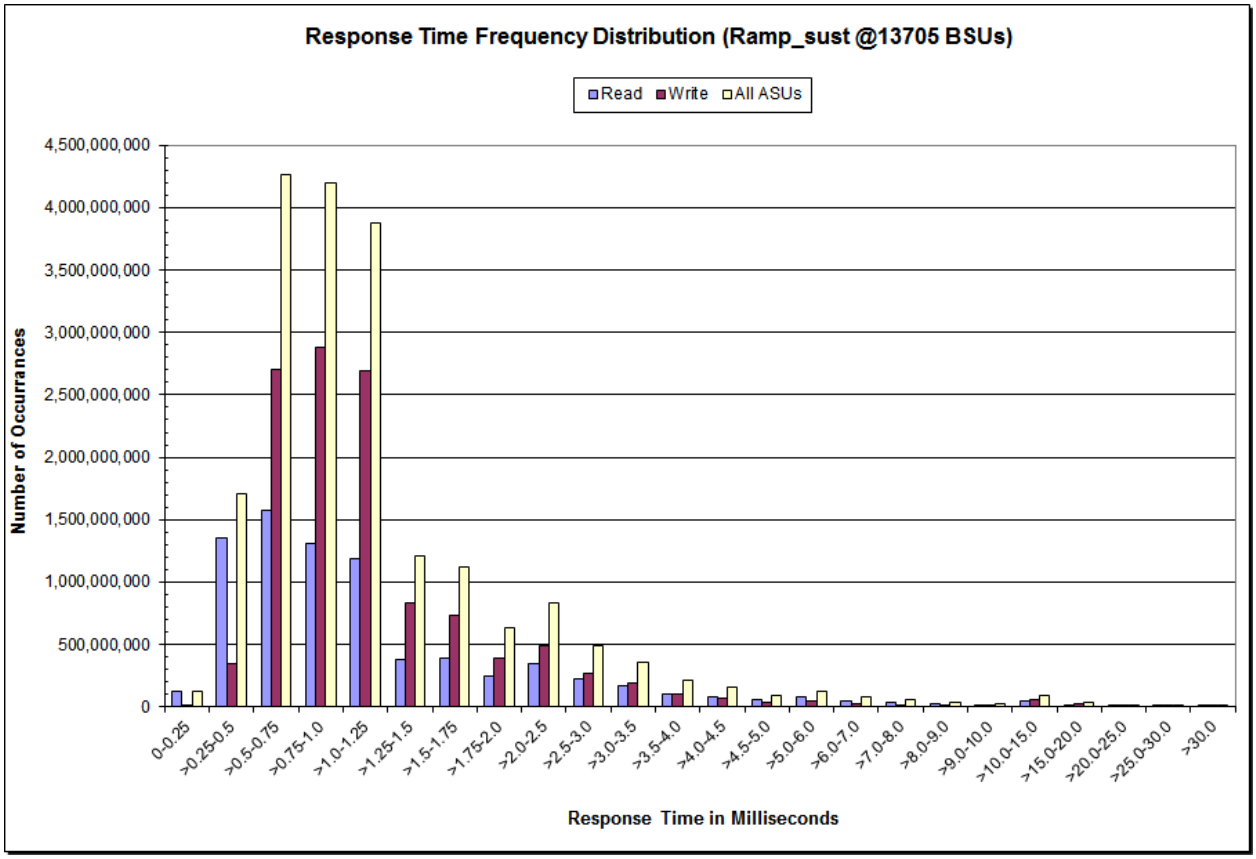
Sustainability – Response Time Frequency Distribution Data

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	119,545,902	1,357,631,101	1,569,274,158	1,309,175,985	1,182,986,071	381,257,949	385,125,591	241,594,220
Write	7,611	347,338,384	2,698,867,971	2,885,897,536	2,694,028,135	830,715,204	737,876,584	392,328,918
All ASUs	119,553,513	1,704,969,485	4,268,142,129	4,195,073,521	3,877,014,206	1,211,973,153	1,123,002,175	633,923,138
ASU1	90,178,724	1,204,981,278	2,340,325,672	2,385,949,164	2,264,043,312	713,917,282	673,397,838	389,525,729
ASU2	29,369,706	303,151,502	565,645,696	461,564,532	409,697,422	135,519,112	130,162,908	74,675,468
ASU3	5,083	196,836,705	1,362,170,761	1,347,559,825	1,203,273,472	362,536,759	319,441,429	169,721,941

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	350,387,481	222,738,198	165,779,840	101,980,649	81,191,195	54,710,911	76,293,438	50,778,204
Write	484,194,136	270,708,861	185,934,759	105,124,193	72,620,323	40,577,988	47,925,564	28,083,246
All ASUs	834,581,617	493,447,059	351,714,599	207,104,842	153,811,518	95,288,899	124,219,002	78,861,450
ASU1	524,882,952	315,905,223	228,458,395	136,424,031	104,090,718	66,390,941	88,018,404	56,202,366
ASU2	99,729,626	59,651,316	43,010,053	25,830,542	19,599,348	12,513,874	16,845,858	10,919,502
ASU3	209,969,039	117,890,520	80,246,151	44,850,269	30,121,452	16,384,084	19,354,740	11,739,582

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	35,497,646	22,798,283	15,122,906	43,086,239	14,107,599	1,909,322	261,075	259,247
Write	17,583,946	13,169,096	11,305,582	52,413,957	22,670,297	9,144,698	1,837,191	1,400,994
All ASUs	53,081,592	35,967,379	26,428,488	95,500,196	36,777,896	11,054,020	2,098,266	1,660,241
ASU1	38,092,569	25,147,708	17,920,510	61,797,475	25,039,634	8,980,315	1,512,407	997,318
ASU2	7,417,787	4,923,958	3,408,366	10,748,248	2,537,747	336,619	86,048	104,452
ASU3	7,571,236	5,895,713	5,099,612	22,954,473	9,200,515	1,737,086	499,811	558,471

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 5.3.15.2

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 5.3.15.3

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	0.001	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 9.4.3.7.3

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 102.

IOPS Test Results File

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

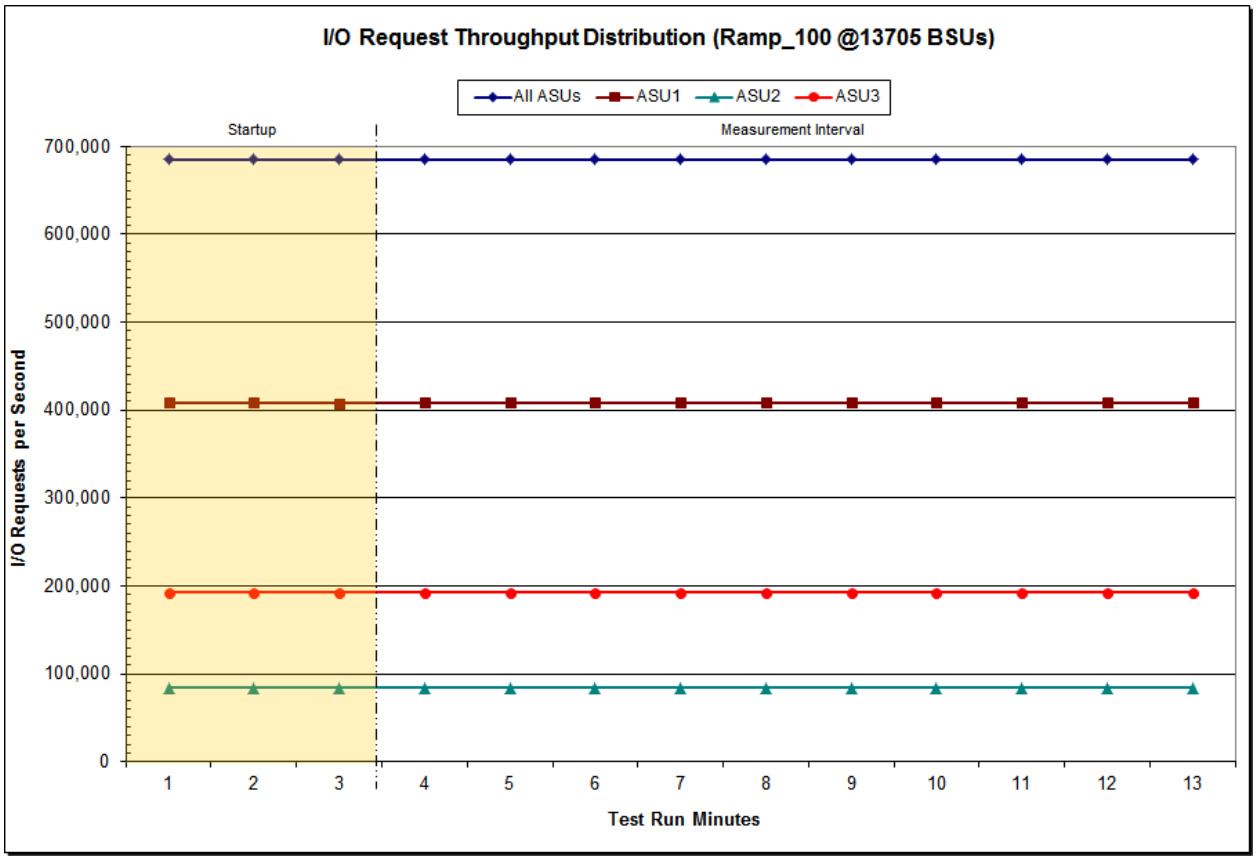
IOPS Test Results File

IOPS Test Run – I/O Request Throughput Distribution Data

13,705 BSUs	Start	Stop	Interval	Duration
Start-Up/Ramp-Up	15:31:48	15:34:49	0-2	0:03:01
Measurement Interval	15:34:49	15:44:49	3-12	0:10:00

60 second intervals	All ASUs	ASU1	ASU2	ASU3
0	685,563.33	408,622.80	84,300.97	192,639.57
1	685,112.65	408,427.17	84,322.25	192,363.23
2	685,116.80	408,249.70	84,257.82	192,609.28
3	685,411.78	408,484.58	84,324.08	192,603.12
4	685,375.53	408,559.78	84,285.67	192,530.08
5	685,143.97	408,467.35	84,170.25	192,506.37
6	685,315.50	408,457.22	84,263.83	192,594.45
7	685,308.73	408,443.95	84,257.25	192,607.53
8	685,262.60	408,305.38	84,285.42	192,671.80
9	685,304.43	408,419.33	84,285.42	192,599.68
10	685,251.20	408,397.00	84,302.07	192,552.13
11	685,181.08	408,284.65	84,270.78	192,625.65
12	685,262.25	408,457.98	84,267.90	192,536.37
Average	***685,281.71***	***408,427.72***	***84,271.27***	***192,582.72***

IOPS Test Run – I/O Request Throughput Distribution Graph

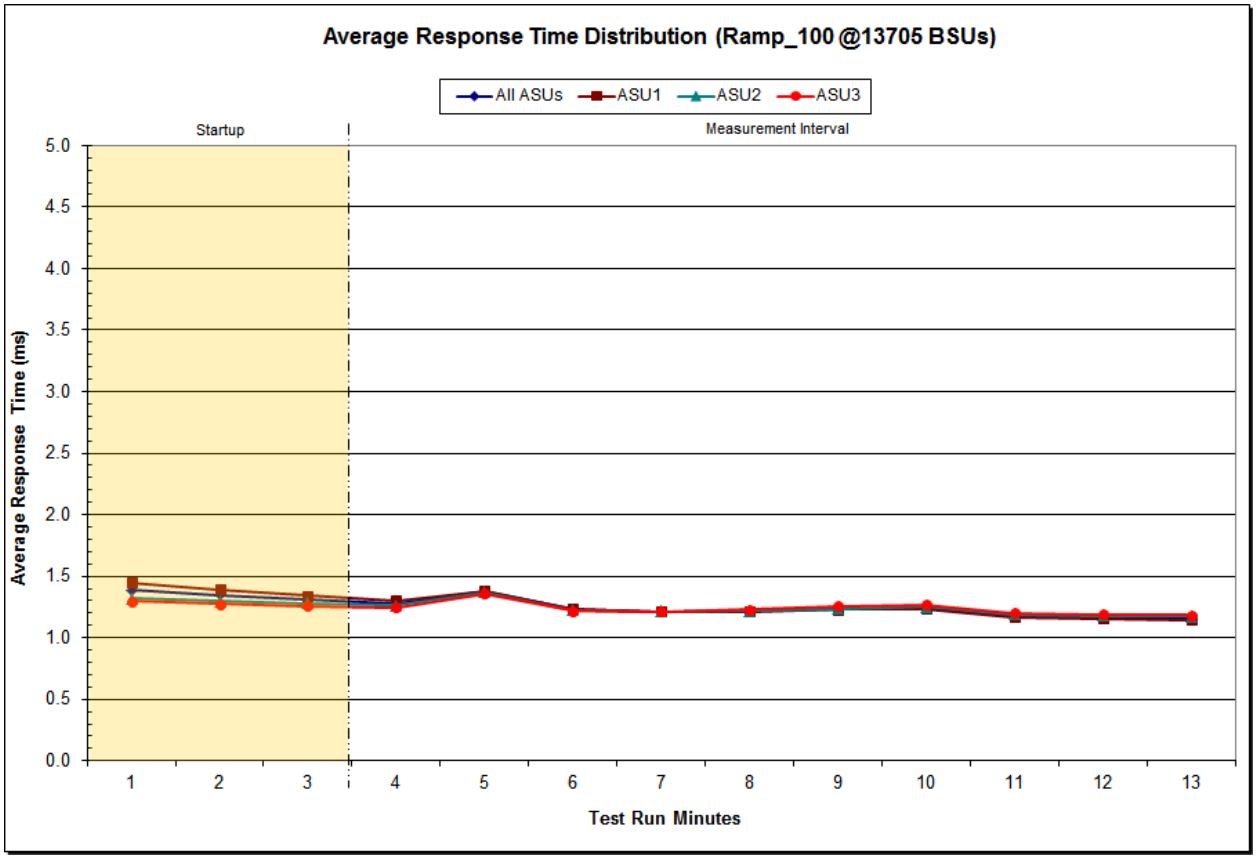


IOPS Test Run – Average Response Time (ms) Distribution Data

13,705 BSUs	Start	Stop	Interval	Duration
Start-Up/Ramp-Up	15:31:48	15:34:49	0-2	0:03:01
Measurement Interval	15:34:49	15:44:49	3-12	0:10:00

60 second intervals	All ASUs	ASU1	ASU2	ASU3
0	1.39	1.45	1.32	1.29
1	1.35	1.39	1.30	1.27
2	1.31	1.34	1.28	1.26
3	1.28	1.30	1.27	1.25
4	1.37	1.38	1.37	1.36
5	1.23	1.24	1.22	1.22
6	1.21	1.21	1.21	1.21
7	1.22	1.21	1.22	1.23
8	1.24	1.23	1.23	1.26
9	1.25	1.24	1.25	1.27
10	1.18	1.17	1.19	1.20
11	1.17	1.15	1.18	1.19
12	1.16	1.15	1.17	1.19
Average	***1.23***	***1.23***	***1.23***	***1.24***

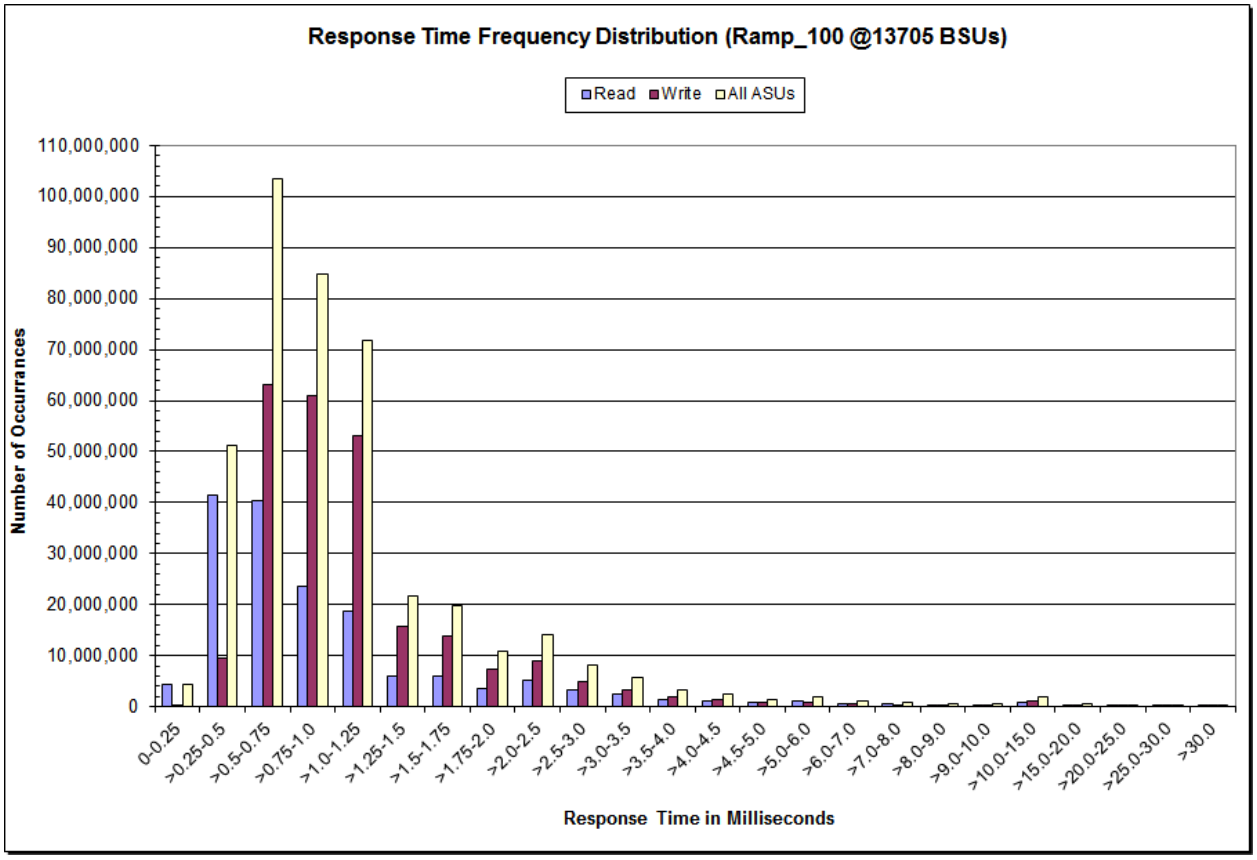
IOPS Test Run – Average Response Time (ms) Distribution Graph



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	4,350,703	41,585,186	40,307,192	23,702,564	18,708,080	5,947,504	5,964,332	3,673,850
Write	350	9,616,836	63,160,866	60,957,595	53,152,714	15,705,916	13,818,402	7,321,087
All ASUs	4,351,053	51,202,022	103,468,058	84,660,159	71,860,794	21,653,420	19,782,734	10,994,937
ASU1	3,479,228	38,555,197	59,322,755	46,965,114	40,257,858	12,243,698	11,350,978	6,430,960
ASU2	871,577	7,280,360	12,679,746	9,444,899	7,969,748	2,566,052	2,443,488	1,394,619
ASU3	248	5,366,465	31,465,557	28,250,146	23,633,188	6,843,670	5,988,268	3,169,358
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	5,225,279	3,254,342	2,371,770	1,429,115	1,101,173	731,855	1,020,120	682,831
Write	8,968,803	4,972,633	3,371,739	1,901,870	1,316,912	748,729	904,922	543,771
All ASUs	14,194,082	8,226,975	5,743,509	3,330,985	2,418,085	1,480,584	1,925,042	1,226,602
ASU1	8,446,110	4,950,447	3,492,500	2,041,497	1,509,189	944,445	1,244,371	795,153
ASU2	1,856,602	1,109,139	793,551	475,521	360,224	231,580	313,024	203,346
ASU3	3,891,370	2,167,389	1,457,458	813,967	548,672	304,559	367,647	228,103
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	482,980	330,340	237,472	778,074	240,335	24,308	2,655	2,286
Write	340,107	259,354	226,485	1,064,634	436,205	171,968	30,240	21,737
All ASUs	823,087	589,694	463,957	1,842,708	676,540	196,276	32,895	24,023
ASU1	538,735	380,677	297,228	1,155,808	460,226	158,126	23,573	12,305
ASU2	138,577	93,766	65,896	214,461	47,080	6,058	1,380	1,955
ASU3	145,775	115,251	100,833	472,439	169,234	32,092	7,942	9,763

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
411,168,221	411,144,198	24,023

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 5.3.15.2

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 5.3.15.3

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	0.001	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 15.

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 9.4.3.7.4

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 102.

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 table and graph for completeness.

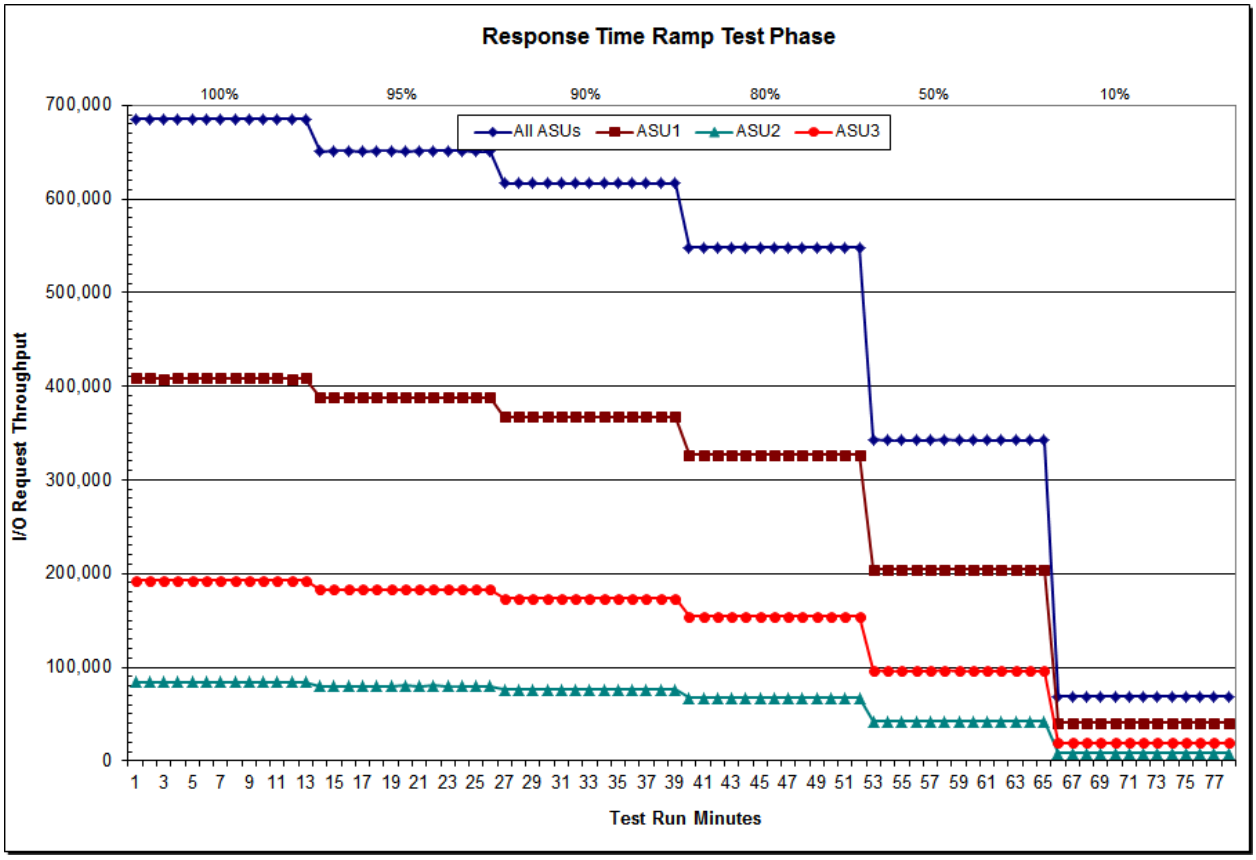
100% Load Level: 13,705 BSUs	Start	Stop	Interval	Duration	95% Load Level: 13,019 BSUs	Start	Stop	Interval	Duration
Start-Up/Ramp-Up	15:31:48	15:34:49	0-3	0:03:01	Start-Up/Ramp-Up	15:48:19	15:51:20	0-3	0:03:01
Measurement Interval	15:34:49	15:44:49	3-12	0:10:00	Measurement Interval	15:51:20	16:01:20	3-12	0:10:00
(60 second intervals)	All ASUs	ASU-1	ASU-2	ASU-3	*(60 second intervals)*	All ASUs	ASU-1	ASU-2	ASU-3
0	685,563.33	408,622.80	84,300.97	192,639.57	0	650,873.43	387,978.38	79,981.92	182,913.13
1	685,112.65	408,427.17	84,322.25	192,363.23	1	650,980.70	387,989.35	80,075.93	182,915.42
2	685,116.80	408,249.70	84,257.82	192,609.28	2	651,031.93	388,021.07	80,067.62	182,943.25
3	685,411.78	408,484.58	84,324.08	192,603.12	3	650,920.62	387,918.17	80,065.57	182,936.88
4	685,375.53	408,559.78	84,285.67	192,530.08	4	650,980.28	387,930.72	80,064.75	182,984.82
5	685,143.97	408,467.35	84,170.25	192,506.37	5	651,059.40	388,073.85	80,056.72	182,928.83
6	685,315.50	408,457.22	84,263.83	192,594.45	6	650,918.73	387,903.20	80,118.93	182,896.60
7	685,308.73	408,443.95	84,257.25	192,607.53	7	650,985.55	388,069.78	80,064.12	182,851.65
8	685,262.60	408,305.38	84,285.42	192,671.80	8	650,968.02	387,971.83	80,122.55	182,873.63
9	685,304.43	408,419.33	84,285.42	192,599.68	9	651,076.43	388,090.72	80,063.07	182,922.65
10	685,251.20	408,397.00	84,302.07	192,552.13	10	651,061.95	388,016.45	80,033.52	183,011.98
11	685,181.08	408,284.65	84,270.78	192,625.65	11	650,845.77	387,975.45	80,007.95	182,862.37
12	685,262.25	408,457.98	84,267.90	192,536.37	12	650,811.23	387,828.65	80,046.15	182,936.43
Average	***685,281.71***	***408,427.72***	***84,271.27***	***192,582.72***	***Average***	***650,962.80***	***387,977.88***	***80,064.33***	***182,920.59***

90% Load Level: 12,334 BSUs	Start	Stop	Interval	Duration	80% Load Level: 10,964 BSUs	Start	Stop	Interval	Duration
Start-Up/Ramp-Up	16:04:45	16:07:46	0-3	0:03:01	Start-Up/Ramp-Up	16:21:01	16:24:02	0-3	0:03:01
Measurement Interval	16:07:46	16:17:46	3-12	0:10:00	Measurement Interval	16:24:02	16:34:02	3-12	0:10:00
(60 second intervals)	All ASUs	ASU-1	ASU-2	ASU-3	*(60 second intervals)*	All ASUs	ASU-1	ASU-2	ASU-3
0	616,865.78	367,684.57	75,847.48	173,333.73	0	548,315.00	326,824.17	67,422.05	154,068.78
1	616,661.27	367,555.63	75,831.67	173,273.97	1	548,239.57	326,713.48	67,457.63	154,068.45
2	616,668.50	367,561.45	75,836.32	173,270.73	2	548,190.05	326,742.75	67,424.17	154,023.13
3	616,682.93	367,523.23	75,877.32	173,282.38	3	548,154.47	326,727.18	67,387.33	154,039.95
4	616,827.07	367,625.90	75,894.60	173,306.57	4	548,170.23	326,706.10	67,432.90	154,031.23
5	616,706.28	367,516.88	75,850.35	173,339.05	5	548,317.25	326,831.55	67,462.17	154,023.53
6	616,689.35	367,531.62	75,841.68	173,316.05	6	548,289.80	326,766.20	67,400.57	154,123.03
7	616,753.72	367,631.63	75,868.35	173,253.73	7	548,278.03	326,801.08	67,427.55	154,049.40
8	616,718.17	367,612.32	75,860.60	173,245.25	8	548,203.78	326,735.60	67,420.17	154,048.02
9	616,851.90	367,597.73	75,894.98	173,359.18	9	548,035.10	326,621.50	67,371.93	154,041.67
10	616,648.18	367,550.02	75,812.33	173,285.83	10	548,272.93	326,781.20	67,437.72	154,054.02
11	616,613.50	367,434.62	75,882.32	173,296.57	11	548,232.52	326,698.12	67,471.72	154,062.68
12	616,684.82	367,546.88	75,888.35	173,249.58	12	548,156.80	326,723.15	67,451.45	153,982.20
Average	***616,717.59***	***367,557.08***	***75,867.09***	***173,293.42***	***Average***	***548,211.09***	***326,739.17***	***67,426.35***	***154,045.57***

Response Time Ramp Distribution (IOPS) Data *(continued)*

50% Load Level: 6,852 BSUs	Start	Stop	Interval	Duration	10% Load Level: 1,370 BSUs	Start	Stop	Interval	Duration
Start-Up/Ramp-Up	16:36:50	16:39:51	0-3	0:03:01	Start-Up/Ramp-Up	16:52:05	16:55:06	0-3	0:03:01
Measurement Interval	16:39:51	16:49:51	3-12	0:10:00	Measurement Interval	16:55:06	17:05:06	3-12	0:10:00
(60 second intervals)	All ASUs	ASU-1	ASU-2	ASU-3	*(60 second intervals)*	All ASUs	ASU-1	ASU-2	ASU-3
0	342,668.90	204,185.35	42,110.92	96,372.63	0	68,487.95	40,829.53	8,408.55	19,249.87
1	342,635.30	204,220.35	42,161.65	96,253.30	1	68,467.70	40,814.67	8,425.72	19,227.32
2	342,554.32	204,206.87	42,102.05	96,245.40	2	68,472.88	40,793.88	8,428.03	19,250.97
3	342,585.25	204,203.48	42,130.55	96,251.22	3	68,477.38	40,816.90	8,434.28	19,226.20
4	342,491.53	204,164.12	42,095.30	96,232.12	4	68,531.80	40,858.48	8,425.35	19,247.97
5	342,693.57	204,213.35	42,139.00	96,341.22	5	68,476.73	40,793.40	8,447.83	19,235.50
6	342,538.68	204,151.90	42,136.63	96,250.15	6	68,465.45	40,786.92	8,438.08	19,240.45
7	342,615.67	204,212.70	42,151.85	96,251.12	7	68,465.53	40,795.18	8,437.53	19,232.82
8	342,548.00	204,119.83	42,107.25	96,320.92	8	68,521.45	40,821.47	8,421.52	19,278.47
9	342,481.95	204,089.58	42,163.78	96,228.58	9	68,468.55	40,806.27	8,409.70	19,252.58
10	342,580.35	204,205.42	42,120.18	96,254.75	10	68,500.98	40,840.68	8,421.28	19,239.02
11	342,500.42	204,080.12	42,098.70	96,321.60	11	68,507.93	40,805.28	8,434.10	19,268.55
12	342,430.03	204,125.78	42,084.05	96,220.20	12	68,555.18	40,859.42	8,430.12	19,265.65
Average	***342,546.55***	***204,156.63***	***42,122.73***	***96,267.19***	*Average*	***68,497.10***	***40,818.40***	***8,429.98***	***19,248.72***

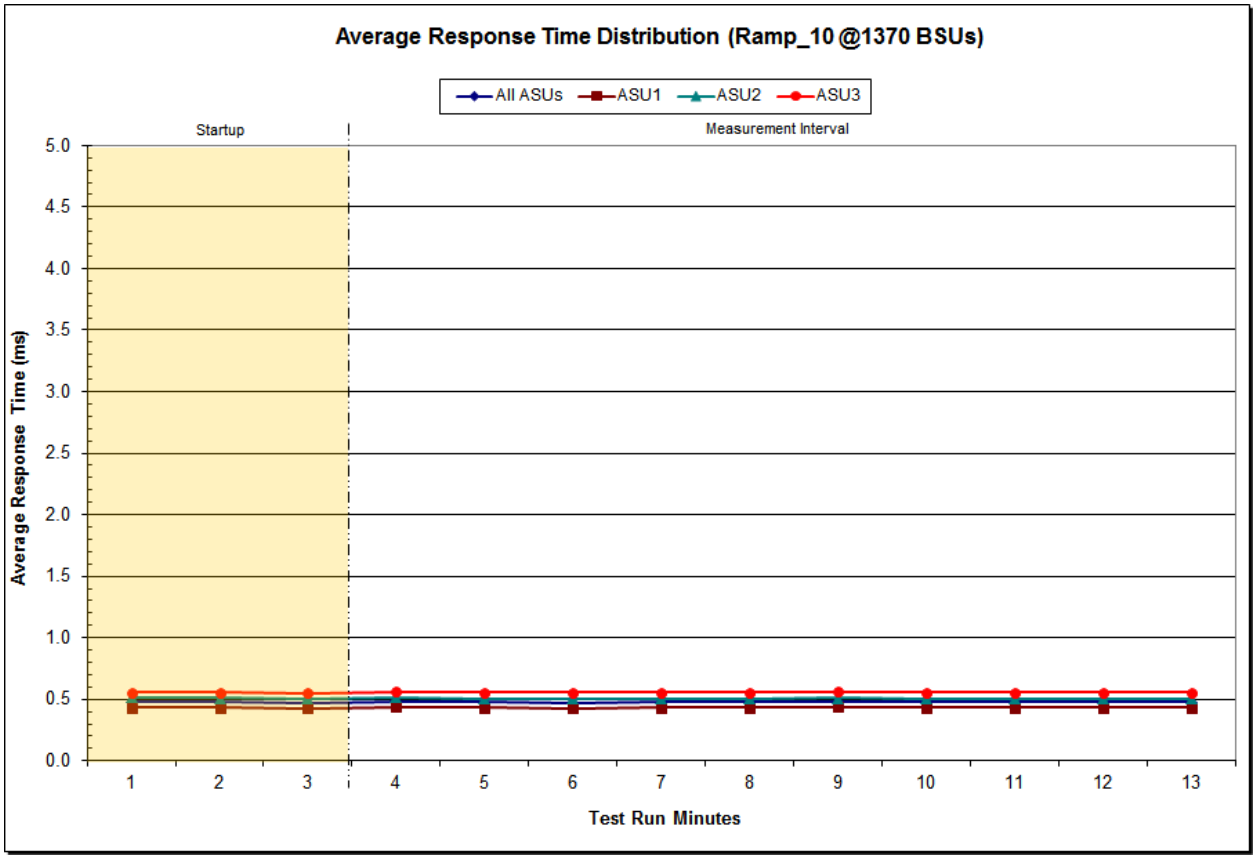
Response Time Ramp Distribution (IOPS) Graph



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

1,370 BSUs	Start	Stop	Interval	Duration
Start-Up/Ramp-Up	16:52:05	16:55:06	0-2	0:03:01
Measurement Interval	16:55:06	17:05:06	3-12	0:10:00
60 second intervals	**All ASUs**	**ASU1**	**ASU2**	**ASU3**
0	0.48	0.43	0.52	0.56
1	0.48	0.43	0.51	0.55
2	0.47	0.43	0.51	0.55
3	0.48	0.44	0.52	0.56
4	0.48	0.43	0.51	0.56
5	0.47	0.43	0.50	0.55
6	0.48	0.43	0.51	0.56
7	0.48	0.43	0.51	0.56
8	0.48	0.44	0.51	0.56
9	0.48	0.43	0.50	0.56
10	0.47	0.43	0.51	0.55
11	0.48	0.43	0.50	0.56
12	0.48	0.43	0.50	0.56
Average	***0.48***	***0.43***	***0.51***	***0.56***

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 5.3.15.2

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 5.3.15.3

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	0.2809	0.0699	0.2100	0.0180	0.0700	0.0350	0.2810
COV	0.001	0.001	0.003	0.001	0.005	0.002	0.003	0.001

Repeatability Test

Clause 5.4.5

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.

Clause 9.4.3.7.5

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 102.

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	***685,281.71***
Repeatability Test Phase 1	685,224.78
Repeatability Test Phase 2	685,274.57

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	***0.48 ms***
Repeatability Test Phase 1	0.47 ms
Repeatability Test Phase 2	0.48 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 plus one (1) millisecond (ms).

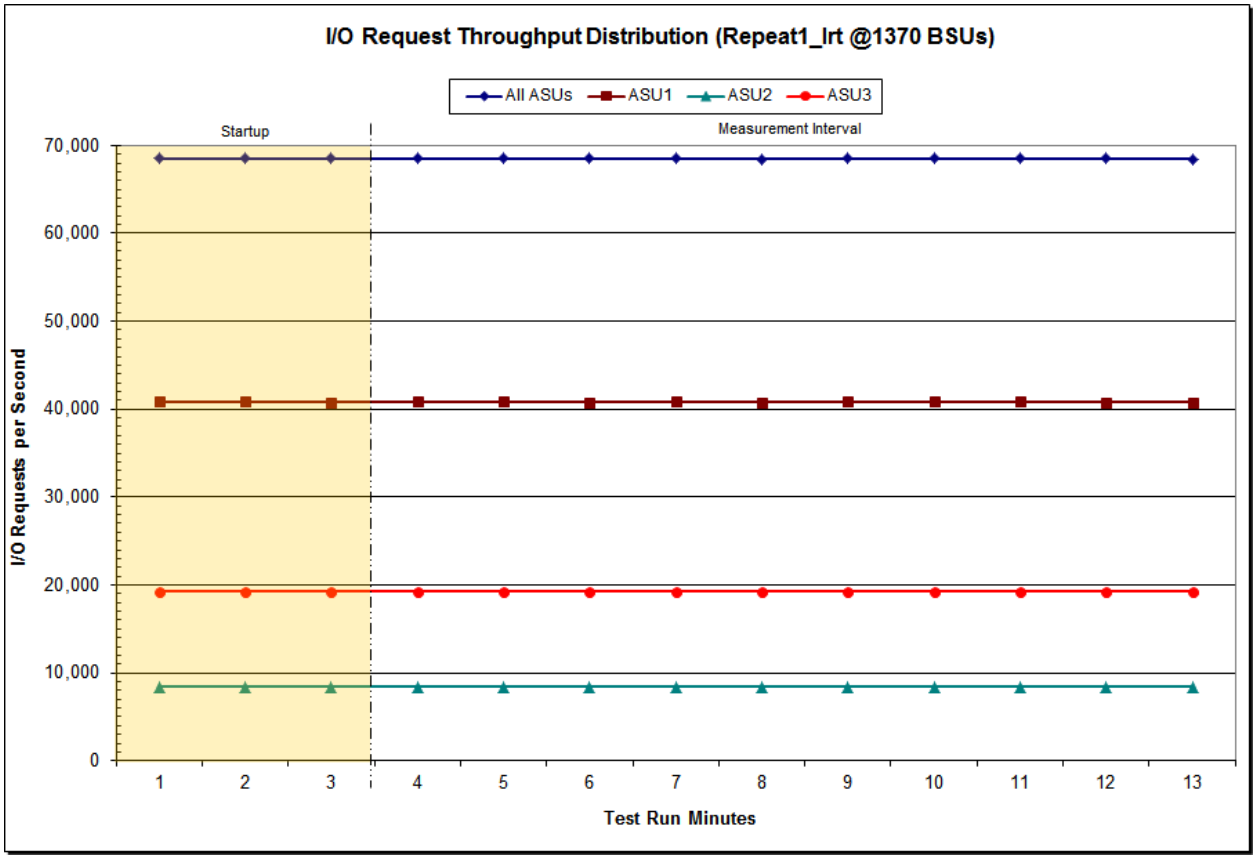
A link to the test result file generated from each Repeatability Test Run is listed below.

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

1,370 BSUs	Start	Stop	Interval	Duration
Start-Up/Ramp-Up	17:07:25	17:10:25	0-2	0:03:00
Measurement Interval	17:10:25	17:20:25	3-12	0:10:00
60 second intervals	**All ASUs**	**ASU1**	**ASU2**	**ASU3**
0	68,535.95	40,859.90	8,451.93	19,224.12
1	68,555.25	40,870.78	8,436.42	19,248.05
2	68,504.05	40,800.83	8,422.07	19,281.15
3	68,553.45	40,860.20	8,435.47	19,257.78
4	68,502.98	40,839.05	8,407.37	19,256.57
5	68,511.07	40,819.12	8,431.28	19,260.67
6	68,495.42	40,825.65	8,435.63	19,234.13
7	68,478.27	40,817.65	8,423.62	19,237.00
8	68,490.32	40,839.70	8,431.03	19,219.58
9	68,496.37	40,830.98	8,402.28	19,263.10
10	68,549.92	40,866.73	8,420.12	19,263.07
11	68,495.18	40,814.68	8,421.87	19,258.63
12	68,463.38	40,799.82	8,431.43	19,232.13
Average	***68,503.64***	***40,831.36***	***8,424.01***	***19,248.27***

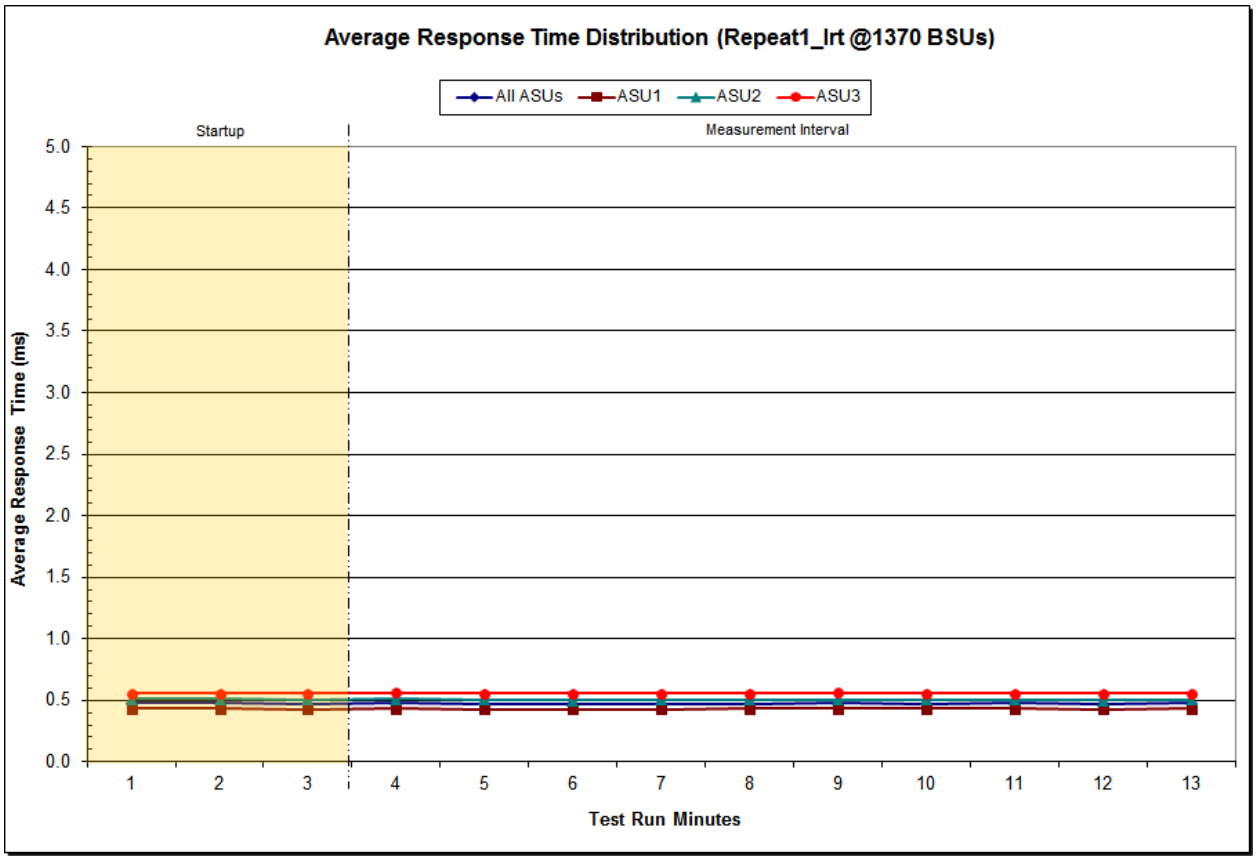
Repeatability 1 LRT – I/O Request Throughput Distribution Graph



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

1,370 BSUs	Start	Stop	Interval	Duration
Start-Up/Ramp-Up	17:07:25	17:10:25	0-2	0:03:00
Measurement Interval	17:10:25	17:20:25	3-12	0:10:00
60 second intervals	**All ASUs**	**ASU1**	**ASU2**	**ASU3**
0	0.48	0.43	0.51	0.56
1	0.48	0.43	0.51	0.56
2	0.47	0.43	0.51	0.56
3	0.48	0.43	0.51	0.56
4	0.47	0.43	0.51	0.56
5	0.47	0.43	0.50	0.55
6	0.47	0.43	0.50	0.55
7	0.47	0.43	0.50	0.56
8	0.48	0.43	0.51	0.56
9	0.47	0.43	0.50	0.56
10	0.48	0.43	0.50	0.56
11	0.47	0.43	0.50	0.56
12	0.47	0.43	0.50	0.56
Average	***0.47***	***0.43***	***0.51***	***0.56***

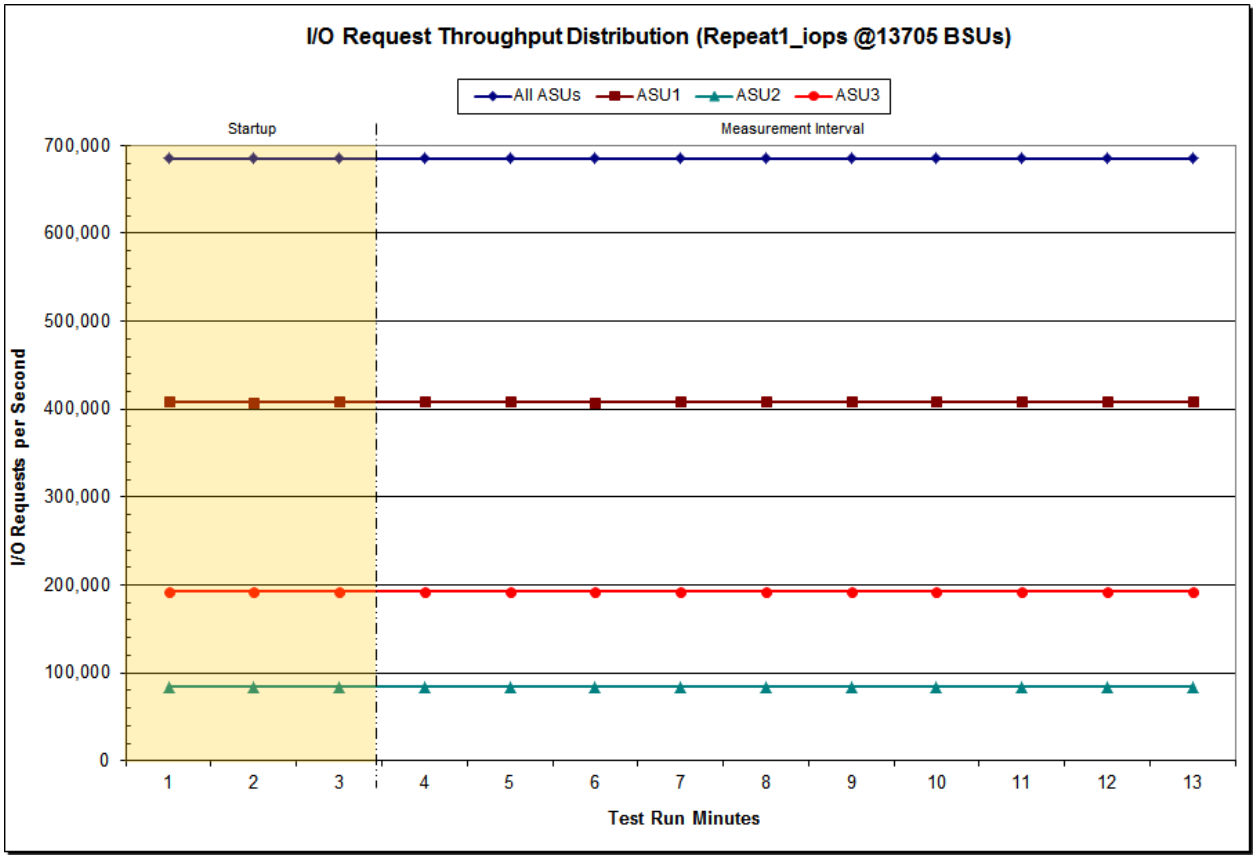
Repeatability 1 LRT –Average Response Time (ms) Distribution Graph



Repeatability 1 IOPS – I/O Request Throughput Distribution Data

13,705 BSUs	Start	Stop	Interval	Duration
Start-Up/Ramp-Up	17:24:03	17:27:04	0-2	0:03:01
Measurement Interval	17:27:04	17:37:04	3-12	0:10:00
60 second intervals	**All ASUs**	**ASU1**	**ASU2**	**ASU3**
0	685,342.42	408,457.30	84,367.42	192,517.70
1	684,888.85	408,220.73	84,228.95	192,439.17
2	685,223.42	408,357.35	84,321.55	192,544.52
3	685,439.07	408,525.87	84,312.97	192,600.23
4	685,271.68	408,405.48	84,274.80	192,591.40
5	684,896.87	408,145.02	84,213.08	192,538.77
6	685,169.98	408,420.75	84,263.90	192,485.33
7	685,315.53	408,415.28	84,332.08	192,568.17
8	685,323.13	408,497.27	84,216.90	192,608.97
9	685,216.33	408,412.98	84,311.22	192,492.13
10	685,257.60	408,446.05	84,269.48	192,542.07
11	685,266.92	408,387.42	84,295.85	192,583.65
12	685,090.70	408,283.68	84,271.27	192,535.75
Average	***685,224.78***	***408,393.98***	***84,276.16***	***192,554.65***

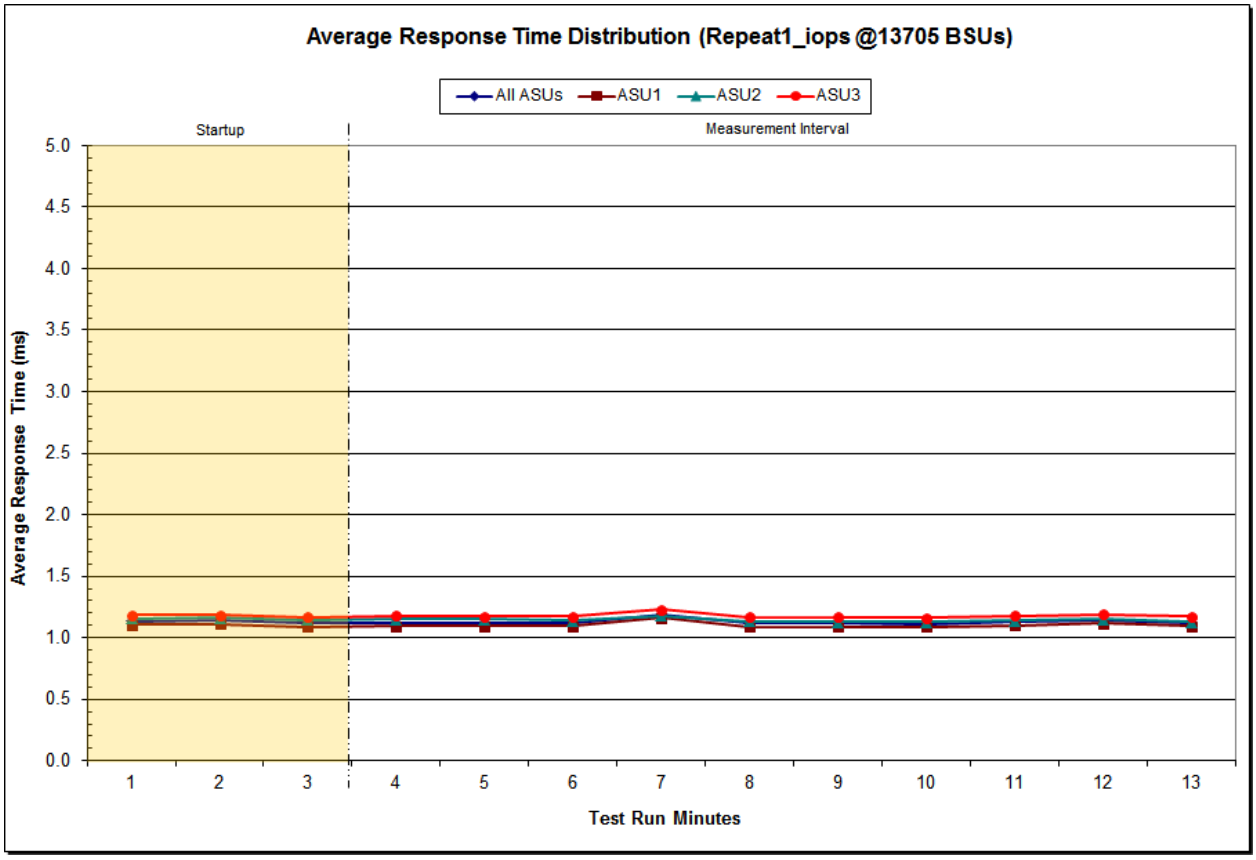
Repeatability 1 IOPS – I/O Request Throughput Distribution Graph



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

13,705 BSUs	Start	Stop	Interval	Duration
Start-Up/Ramp-Up	17:24:03	17:27:04	0-2	0:03:01
Measurement Interval	17:27:04	17:37:04	3-12	0:10:00
60 second intervals	**All ASUs**	**ASU1**	**ASU2**	**ASU3**
0	1.14	1.11	1.16	1.19
1	1.14	1.11	1.16	1.19
2	1.12	1.09	1.15	1.17
3	1.12	1.09	1.15	1.18
4	1.12	1.09	1.15	1.17
5	1.12	1.09	1.14	1.17
6	1.18	1.16	1.18	1.23
7	1.12	1.09	1.13	1.17
8	1.12	1.09	1.13	1.17
9	1.11	1.09	1.13	1.16
10	1.13	1.10	1.14	1.18
11	1.14	1.12	1.15	1.19
12	1.12	1.09	1.13	1.17
Average	***1.13***	***1.10***	***1.14***	***1.18***

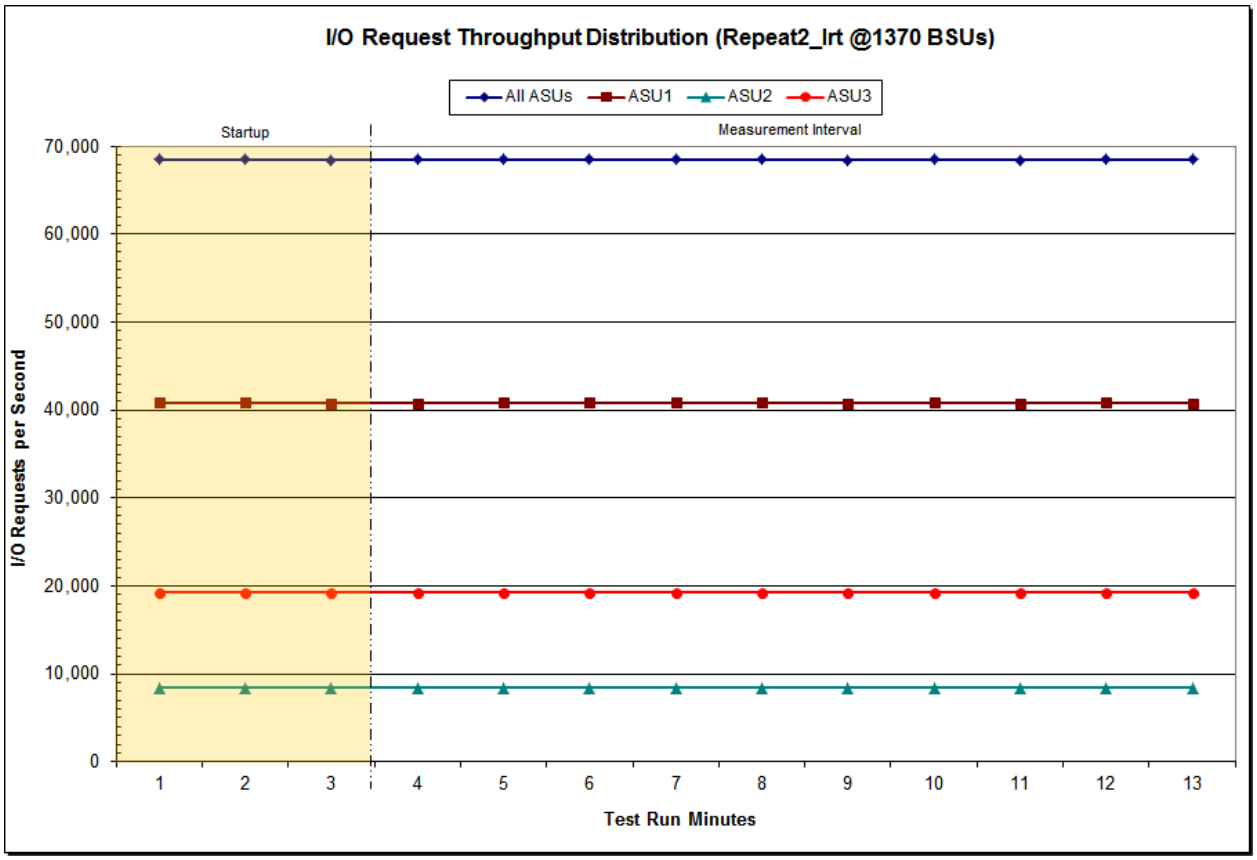
Repeatability 1 IOPS –Average Response Time (ms) Distribution Graph



Repeatability 2 LRT – I/O Request Throughput Distribution Data

1,370 BSUs	Start	Stop	Interval	Duration
Start-Up/Ramp-Up	17:39:23	17:42:23	0-2	0:03:00
Measurement Interval	17:42:23	17:52:23	3-12	0:10:00
60 second intervals	**All ASUs**	**ASU1**	**ASU2**	**ASU3**
0	68,514.98	40,839.55	8,426.95	19,248.48
1	68,503.83	40,830.65	8,405.45	19,267.73
2	68,464.75	40,787.80	8,422.10	19,254.85
3	68,484.08	40,810.42	8,422.98	19,250.68
4	68,495.78	40,852.70	8,418.02	19,225.07
5	68,528.43	40,835.98	8,428.35	19,264.10
6	68,529.23	40,827.20	8,418.40	19,283.63
7	68,541.58	40,864.28	8,431.35	19,245.95
8	68,440.32	40,798.22	8,418.25	19,223.85
9	68,490.53	40,830.22	8,414.85	19,245.47
10	68,478.03	40,780.57	8,438.78	19,258.68
11	68,509.95	40,854.28	8,420.25	19,235.42
12	68,526.52	40,802.12	8,442.98	19,281.42
Average	***68,502.45***	***40,825.60***	***8,425.42***	***19,251.43***

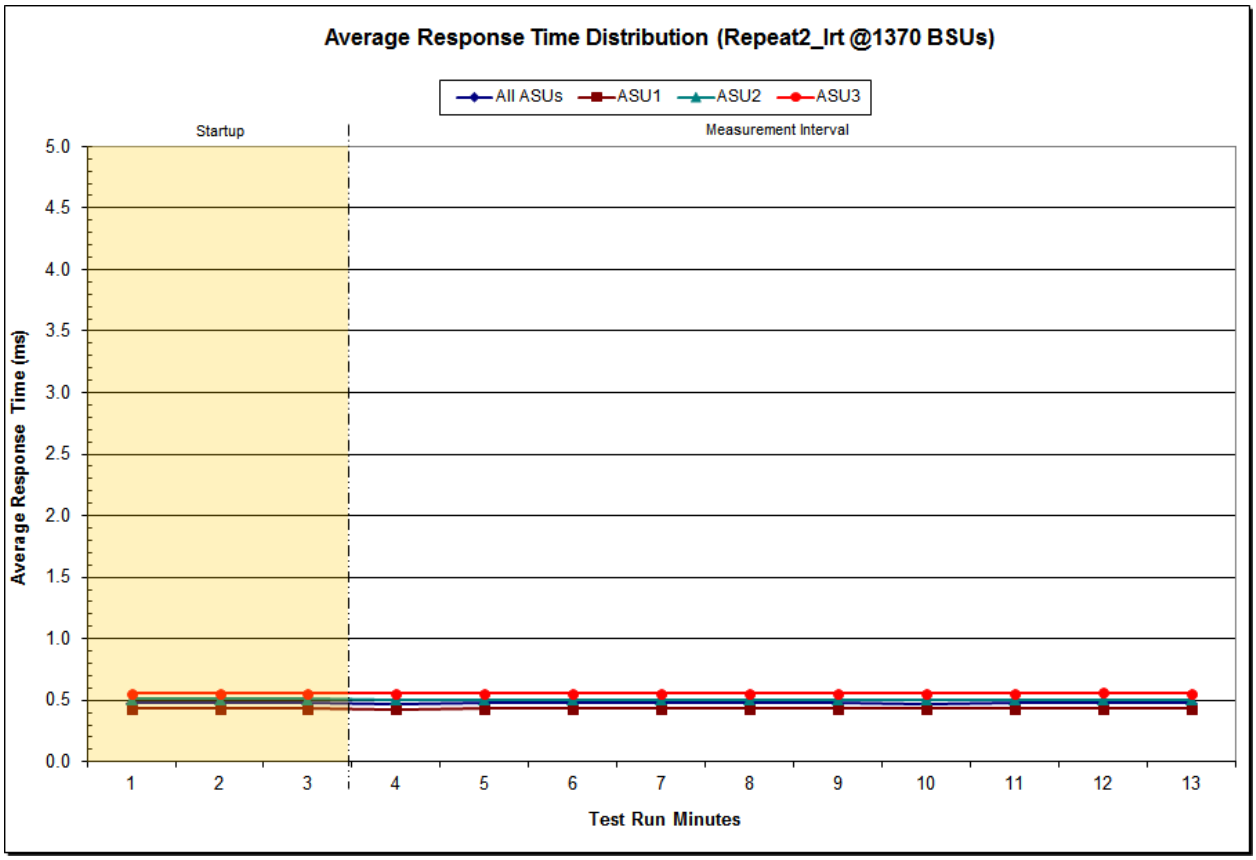
Repeatability 2 LRT – I/O Request Throughput Distribution Graph



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

1,370 BSUs	Start	Stop	Interval	Duration
Start-Up/Ramp-Up	17:39:23	17:42:23	0-2	0:03:00
Measurement Interval	17:42:23	17:52:23	3-12	0:10:00
60 second intervals	**All ASUs**	**ASU1**	**ASU2**	**ASU3**
0	0.48	0.43	0.51	0.56
1	0.48	0.43	0.51	0.56
2	0.48	0.43	0.51	0.56
3	0.47	0.43	0.51	0.55
4	0.48	0.43	0.51	0.56
5	0.48	0.43	0.50	0.56
6	0.48	0.43	0.51	0.56
7	0.47	0.43	0.50	0.56
8	0.48	0.43	0.50	0.56
9	0.47	0.43	0.50	0.56
10	0.48	0.43	0.50	0.56
11	0.48	0.44	0.51	0.56
12	0.48	0.43	0.50	0.56
Average	***0.48***	***0.43***	***0.51***	***0.56***

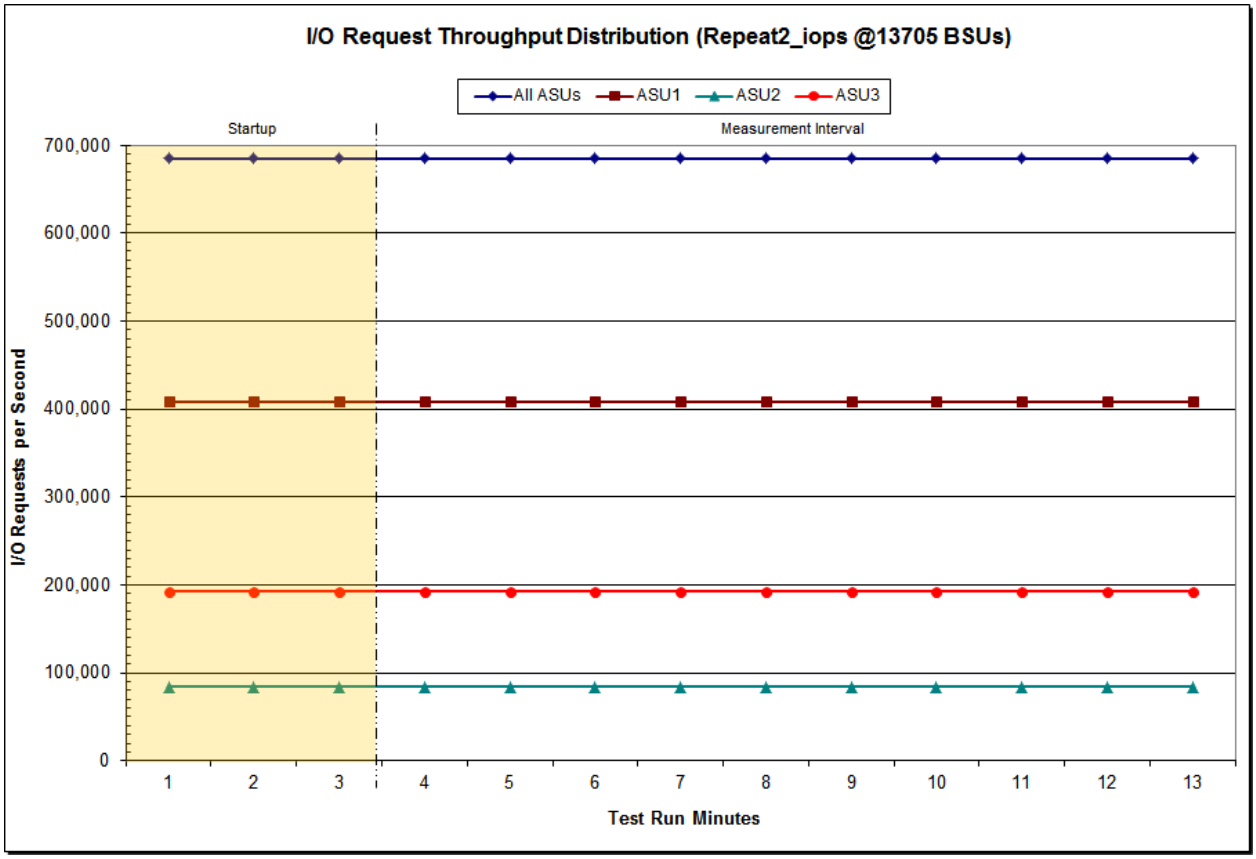
Repeatability 2 LRT –Average Response Time (ms) Distribution Graph



Repeatability 2 IOPS – I/O Request Throughput Distribution Data

1,370 BSUs	Start	Stop	Interval	Duration
Start-Up/Ramp-Up	17:56:03	17:59:04	0-2	0:03:01
Measurement Interval	17:59:04	18:09:04	3-12	0:10:00
60 second intervals	**All ASUs**	**ASU1**	**ASU2**	**ASU3**
0	685,250.22	408,445.18	84,301.90	192,503.13
1	685,169.15	408,369.00	84,286.22	192,513.93
2	685,082.55	408,344.87	84,264.22	192,473.47
3	685,335.90	408,562.75	84,338.83	192,434.32
4	685,369.93	408,445.03	84,308.77	192,616.13
5	685,303.43	408,515.20	84,253.40	192,534.83
6	685,204.50	408,309.63	84,253.27	192,641.60
7	685,345.27	408,392.72	84,275.03	192,677.52
8	685,245.90	408,431.45	84,212.05	192,602.40
9	685,197.92	408,351.40	84,285.97	192,560.55
10	685,307.75	408,454.88	84,269.90	192,582.97
11	685,249.05	408,475.55	84,174.20	192,599.30
12	685,186.05	408,345.50	84,258.17	192,582.38
Average	***685,274.57***	***408,428.41***	***84,262.96***	***192,583.20***

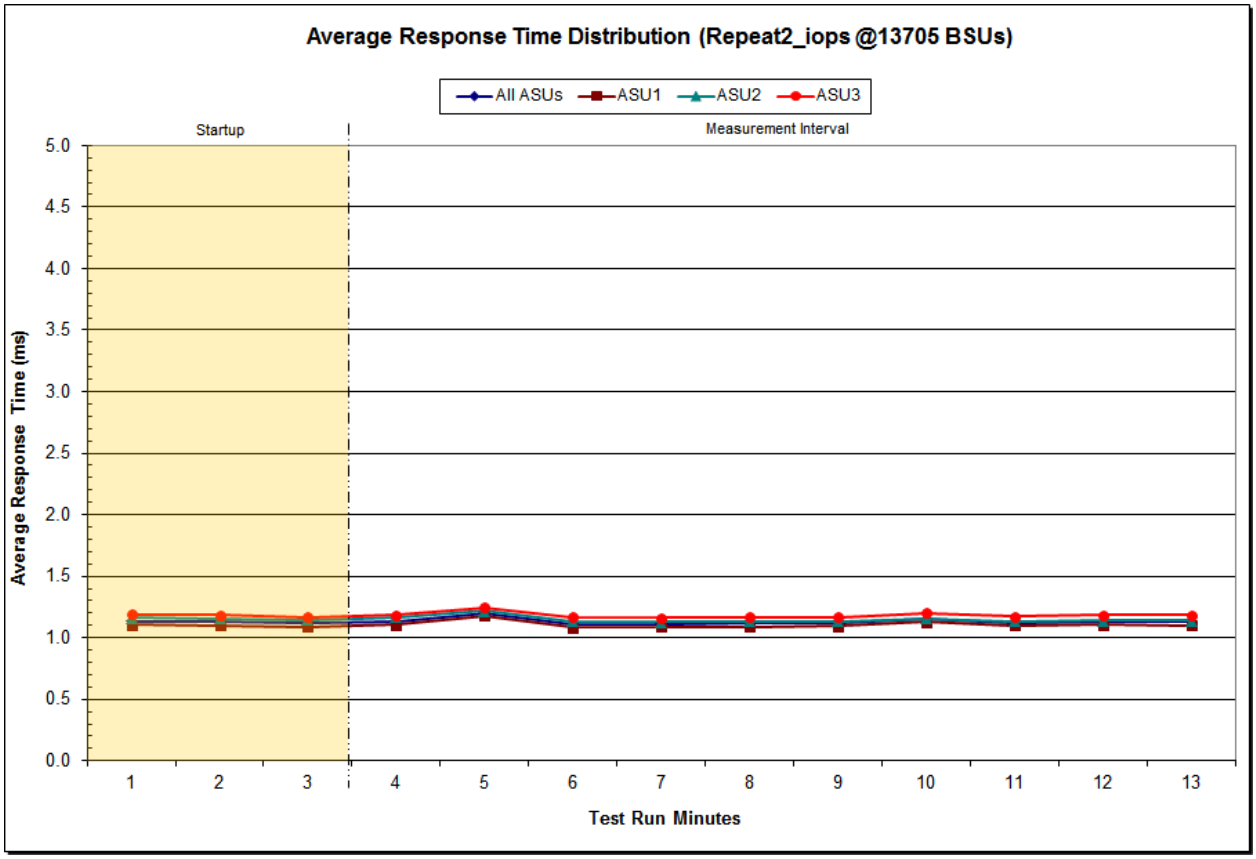
Repeatability 2 IOPS – I/O Request Throughput Distribution Graph



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

1,370 BSUs	Start	Stop	Interval	Duration
Start-Up/Ramp-Up	17:56:03	17:59:04	0-2	0:03:01
Measurement Interval	17:59:04	18:09:04	3-12	0:10:00
60 second intervals	**All ASUs**	**ASU1**	**ASU2**	**ASU3**
0	1.14	1.11	1.16	1.19
1	1.13	1.10	1.16	1.18
2	1.12	1.09	1.15	1.17
3	1.14	1.11	1.17	1.19
4	1.20	1.18	1.22	1.25
5	1.11	1.09	1.13	1.17
6	1.11	1.09	1.13	1.16
7	1.12	1.09	1.13	1.17
8	1.12	1.09	1.13	1.17
9	1.15	1.13	1.16	1.20
10	1.12	1.10	1.14	1.17
11	1.13	1.10	1.14	1.18
12	1.13	1.10	1.14	1.18
Average	***1.13***	***1.11***	***1.15***	***1.18***

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.*

Clauses5.1.10 and 5.3.15.2

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 5.3.15.3

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	0.2101	0.0180	0.0700	0.0350	0.2810
COV	0.002	0.000	0.002	0.001	0.003	0.002	0.001	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	0.2810	0.0700	0.2100	0.0180	0.0700	0.0350	0.2810
COV	0.001	0.000	0.001	0.000	0.001	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	0.2810	0.0700	0.2100	0.0180	0.0700	0.0350	0.2810
COV	0.003	0.001	0.002	0.001	0.003	0.001	0.003	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	0.000	0.000	0.001	0.000	0.001	0.001	0.001	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 (may be contained in an appendix).*

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 102.

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	456,341,120
Total Number of Logical Blocks Verified	204,291.280
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 9.4.3.9

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® FAS8080 EX *(All-Flash FAS)* 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 16.

TESTED STORAGE CONFIGURATION (TSC) AND PRICED STORAGE CONFIGURATION DIFFERENCES

Clause 9.4.3.3.8

The Executive Summary shall contain 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 16.

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® FAS8080 EX.

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 1: The single point of failure of any ***storage device*** in the configuration will not result in permanent loss of access to or integrity of the SPC-1 Data Repository.

Protected 2: The single point of failure of any ***component*** in the configuration will not result in permanent loss of access to or integrity of the SPC-1 Data Repository.

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.

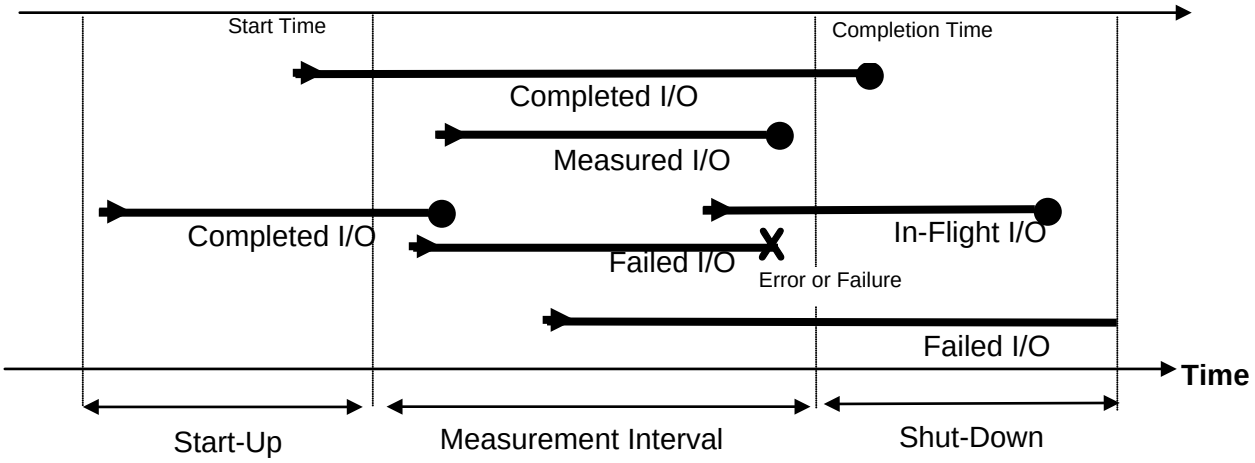
Steady State: The consistent and sustainable throughput of the TSC. During this period the load presented to the TSC by the Workload Generator is constant.

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

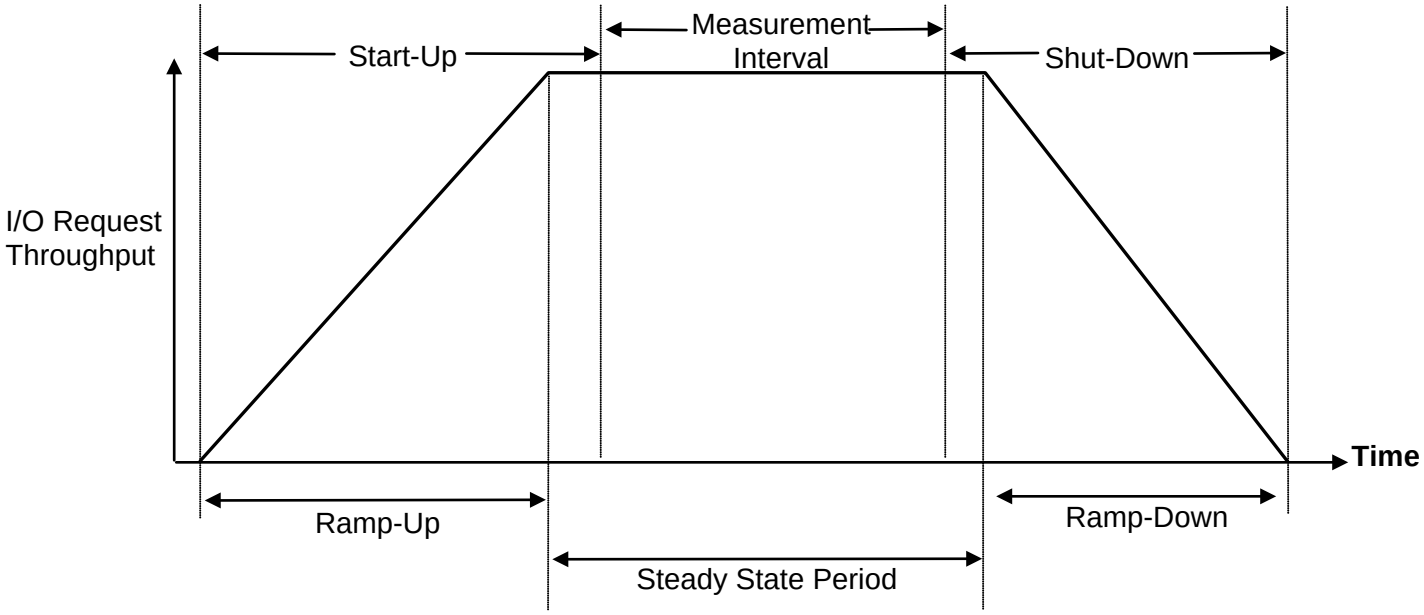
Test Run: The execution of SPC-1 for the purpose of producing or supporting an SPC-1 test result. SPC-1 Test Runs may have a finite and measured Ramp-Up period, Start-Up period, Shut-Down period, and Ramp-Down period as illustrated in the "SPC-1 Test Run Components" below. All SPC-1 Test Runs shall have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

QLogic HBA Driver

- 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 6.4 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)

  This change may be applied at any point in the TSC creation/configuration process documented in *Appendix C: Tested Storage Configuration (TSC) Creation*.

Red Hat Enterprise Linux 6.4 (64-bit) – I/O Scheduler

- 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/grub.conf file** at the end of the kernel line on each Host System.

  elevator=noop

  This change may be applied at any point in the TSC creation/configuration process documented in *Appendix C: Tested Storage Configuration (TSC) Creation*.

Red Hat Enterprise Linux 6.4 (64-bit) – System Services

The following system services were disabled using the listed commands executed on each Host System from the command line interface.

Notes about chkconfig and service

chkconfig provides a simple command-line tool for maintaining the **/etc/rc[0-6].d** directory hierarchy by relieving system administrators of the task of directly manipulating the numerous symbolic links in those directories.

service runs a System V init script in as predictable environment as possible, removing most environment variables and with current working directory set to ***/***. The ***script*** parameter specifies a System V init script, located in **/etc/init.d/SCRIPT**. The invocation ***service <command> <options>*** passes the command and its options unmodified to the **init** script for execution. All scripts should support at least the ***start*** and ***stop*** options.

- **cpuspeed**

  chkconfig cpuspeed off ; service cpuspeed stop

  cpuspeed *monitors the system's idle percentage and reduces or raises the CPUs' clock speeds and voltages accordingly to minimize power consumption when idle and maximize performance when needed*

- **hypervkvpd**

  chkconfig hypervkvpd off; service hypervkvpd stop

  hypervkvpd *is Linux daemon that allows Windows Host environments to collect information about a guest Linux system in the Hyper-V environment.*

- **ip6tables**

  chkconfig ip6tables off; service ip6tables stop

  ip6tables *is used to set up, maintain, and inspect the tables of IPv4 packet filter rules in the Linux kernel.*

- **iptables**

  chkconfig iptables off; service iptables stop

  iptables *is used to set up, maintain, and inspect the tables of IPv4 packet filter rules in the Linux kernel.*

- **iscsi**

  chkconfig iscsi off; service iscsi stop

  iscsi *is a service that allows external clients to connect to a server using the iSCSI transport protocol for I/O.*

- **ksm**

  chkconfig ksm off; service ksm stop

  ksm *enables the kernel to examine two or more already running programs and compare their memory. If any memory regions or pages are identical,* ***ksm*** *reduces multiple identical memory pages to a single page. The* ***ksm*** *service starts and stops the* ***ksm*** *kernel thread.*

- **ksmtuned**

  chkconfig ksmtuned off; service ksmtuned stop

  ksmtuned *controls and tunes the* ***ksm****, dynamically managing same-page merging.*

- **libvirtd**

  chkconfig libvirtd off; service libvirtd stop

libvirtd *is the server side daemon component of the* ***libvirt*** *virtualization management system. This daemon runs on host servers and performs required management tasks for virtualized guests.*

- **ibvirt-guests**

  chkconfig ibvirt-guests off; service ibvirt-guests stop

  ibvirt-guests *is responsible for suspending active* ***ibvirt*** *guests on shutdown and resuming them on next boot.*

- **virt-who**

  chkconfig virt-who off; service virt-who stop

  virt-who *is an agent for reporting virtual guest IDs to Subscription Asset Manager.*

- **cgconfig**

  chkconfig cgconfig off; service cgconfig stop

  The ***cgconfig*** *service installed with the libcgroup package provides a convenient way to create hierarchies, attach subsystems to hierarchies, and manage cgroups within those hierarchies. The* ***cgconfig*** *service is not started by default on Red Hat Enterprise Linux 6.*

- **bluetooth**

  chkconfig bluetooth off; service bluetooth stop

  The Bluetooth daemon

- **cups**

  chkconfig cups off; service cups stop

  cups *manages print jobs and queues and provides network printing.*

Storage Controllers

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

SPC Root and Data Aggregates

- The **nosnap** value was changed from the default ***off*** to a setting of ***on*** for each data aggregate and the root aggregates using the following command:

  aggr options <aggr_name> nosnap on

The following changes are only applied to the data aggregates.

- Change the aggregate Snapshot™ reserve, **snap reserve**, to 0 bytes so that no space is reserved for Snapshot copies.
- **snap reserve** for volume **vol0** is set to ***0*** on each **vol0** using the following command:

  snap reserve vol0 0

- **snap reserve** for root aggregates is left unchanged.
- 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 a space reservation requirement of ***none*** instead of the default volume.

LUN Options

Create the LUNs with space reservations using the **space-reserve enabled** option during LUN creation. When reservations are enabled for one or more LUNs, Data ONTAP® reserves enough space in the volume so that writes to those LUNs do not fail because of a lack of disk space.

Storage System Options

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

The minutes between ***perfstat*** collection value was changed from the default of 5 to 0. This setting disables the onboard ***perfstat*** portion of performance archiver. The default configuration triggers collection of non-counter manager commands every five minutes. Setting the following value to 0 favors consistent performance. The command to make this change is as follows:

set diag ; statistics archive config modify –perfstat-sampling-period 0

The ***raid scrub schedule*** was changed from the system default of daily (1am – 5am) and Sunday (1am to 1pm) to Sunday 1am – 7am. The default schedule is unnecessary for Solid State Drives; as such the schedule is changed to Sundays only.

The following command was executed manually, from any Host System, for each of the 8 nodes, replacing ***node ****** with the appropriate node designation.

storage raid-options modify –node * raid.scrub.schedule –value 6h@sun@1

This change may be applied at any point in the TSC creation/configuration process documented in *Appendix C: Tested Storage Configuration (TSC) Creation*.

APPENDIX C: TESTED STORAGE CONFIGURATION (TSC) CREATION

This documentation begins with the all of benchmark configuration hardware and software installed and configured on 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 on page 66 *(Appendix B: Customer Tunable Parameters and Options)*.

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

Create and Configure RAID Groups, Aggregates, Volumes and LUNs

This first script, **config_storage.py**, is invoked from any of the Host Systems where Python is installed, as follows:

python config_storage.py --config < config file> --log <dir to store output logs> -n

The script does the following on each controller:

- Set the following option: ***wafl.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 copy setting as follows:
  -Set aggregate option **nosnap=on**.
  -Set aggregate snap sched to 0 0 0
  -Delete any existing Snapshot copy.
- Set root volume Snapshot copy setting as follows:
  -Set volume option **nosnap=on**
  -Set volume snap sched to 0 0 0
  -Set snap reserve=0
  -Delete any existing Snapshot copy.
- Create the SPC-1 data aggregates as specified in **config_full**, which will create two RAID groups, one with 23 non-partitioned disks and one with 24 partitioned disk as well as 1 non-partitioned spare.

- Set the SPC-1 data aggregate Snapshot copy 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_alloc-blocks=256***.
- Create a **SVM**, "**vs1**", as specified in **config_full**.
  *(A **SVM** is storage virtual machine 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**:
  - 12 volumes of 783,732,446 KB each in the data aggregate. Those volumes will contain the LUNs for ASU-1.
  - 12 volumes of 783,732,446 KB each in the data aggregate. Those volumes will contain the LUNs for ASU-2.
  - 12 volumes of 174,162,766 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**, which will create the following on each storage controller:
  - One LUN of 653,110,372 KB in each data volume designated for ASU-1.
  - One LUN of 653,110,372 KB in each data volume designated for ASU-2.
  - One LUN of 145,135,638 KB in each data volume designated for ASU-3.
- Set the following fields on each of the LUNs:
  -ostype=linux
  -space-reserve=enabled
- Create an **igroup** and map the LUNs as specified in **config_full**, which will do the following:
  - Create an **igroup**,"**ig1**", with the following settings:
    -protocol=fcp
    -ostype=linux
    -Contains sixteen initiator WWPN *(two from each Host System specified in **config_full**)*
  - Map each LUN to **igroup**,"**ig1**" and assign the LUN to **SVM** "**vs1**".

- Create four data LIFs *(logical interfaces)*, as specified in **config_full**, using physical on board FC target ports **0e, 0f, 0g** and **0h** on each storage controller with the following settings, which will assign each LIF to the **SVM** "**vs1**":

  -vserver=vs1
  -role=data
  -data-protocol=fcp
  -home-node=<*name of the cluster node physical FC target ports reside on*>
  -home-port=<*either* ***0e, 0f, 0g*** *or* ***0h****, as appropriate*>

- Start an FCP server *(fibre channel service)* on the **SVM** "**vs1**",with the following setting:
  -state-admin=up
  -vserver=vs1

Create and Configure SPC-1 Logical Volumes

The **config_hosts.py** script is executed on the Master Host System after successful completion of **config_storage.py** and does the following:

- 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 6.1 command: ***sanlun lun show –p***.
- Create the RHEL 6.4 LVM2 logical volumes, which correspond to the SPC-1 Logical volumes, as specified in **config_full**.
  1. Initialize the first eight logical volumes for ASU-1,.one from each storage controller using **pvcreate**.
  2. Create a volume group comprised of the eight multipath devices that were initialized in the previous step.
  3. Create a logical volume with the following settings:
     -i 8 *(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 SPC-1 Logical Volumes: 4 for ASU-1, 4 for ASU-2 and 4 for ASU-3.
     - One SPC-1 Logical Volume per volume group.
     - Each SPC-1 Logical Volume is striped across eight multipath devices (LUNs); one LUN from each storage controller.
- Reboot all eight Host Systems so that each Host System will scan the LUNs and the SPC-1 Logical Volumes. Logical volume meta data was written on the LUNs when they were created from the Master Host System, therefore, when the remaining Host Systems scan the LUNs, the same meta data will be read by all Host Systems resulting in same SPC-1 ASU Logical Volumes visible on all Host Systems.

- The script will wait until all Host Systems are back up and then build SPC-1 configuration and JVM slave configuration files for Master and Slave Host Systems, and place the appropriate configuration files in the SPC-1 run directory on each Host System defined in **config_full**.

Referenced File and Scripts

config_full

```
[DEFAULT]
timeserver = 10.60.252.15
timezone = EST5EDT
sanlun = /usr/sbin/sanlun

## Benchmark configuration data
[SPC-1]
bsu_rate = 13700
rundir = /opt/SPC1RunDir
Xms = 4096m
Xmx = 4096m
Xss = 512k
size_linux = /SPC1RunDir/scripts/spc_audit/bin/size_linux

## Host configuration data
[hosts]
master = spc1
slaves = spc1 spc2 spc3 spc4 spc5 spc7 spc8 spc9 spc10
login = root
password = Netapp123
post_reboot_init_time = 90

[spc1]
slave_num = 1
ip_addr = 10.63.173.44
initiators = 21:00:00:0e:1e:1a:38:80 21:00:00:0e:1e:1a:38:81

[spc2]
slave_num = 2
ip_addr = 10.63.173.46
initiators = 21:00:00:0e:1e:1b:16:90 21:00:00:0e:1e:1b:16:91

[spc3]
slave_num = 3
ip_addr = 10.63.173.48
initiators = 2100000e1e1b2b40 2100000e1e1b2b41

[spc4]
slave_num = 4
ip_addr = 10.63.173.50
initiators = 2100000e1e1a3660 2100000e1e1a3661

[spc5]
slave_num = 5
ip_addr = 10.63.153.72
initiators = 2100000e1e1663e0 2100000e1e1663e1

[spc7]
slave_num = 7
ip_addr = 10.63.153.76
initiators = 2100000e1e166360 2100000e1e166361
```

```
[spc8]
slave_num = 8
ip_addr = 10.63.153.78
initiators = 2100000e1e1669f0 2100000e1e1669f1

[spc9]
slave_num = 9
ip_addr = 10.63.153.80
initiators = 2100000e1e1662c0 2100000e1e1662c1

[spc10]
slave_num = 10
ip_addr = 10.63.153.82
initiators = 2100000e1e1665d0 2100000e1e1665d1


[lvm]
stripes = 8
stripesize = 1024
extents = 100%%VG
lv_create_host = spc1

## ONTAP Storage configuration data
[cluster]
name = spc1
nodes = spc1-01 spc1-02 spc1-03 spc1-04 spc1-05 spc1-06 spc1-07 spc1-08
ip_addr = 10.63.173.157
login: admin
password: Netapp123
data_aggrs = aggr1
vserver = vs1
host_ostype = linux
igroup_name = ig1
fcp_targets = 0e 0f 0g 0h

[spc1-01]
node_num = 1
root_aggr = aggr0Spc1

[spc1-02]
node_num = 2
root_aggr = aggr0Spc2

[spc1-03]
node_num = 3
root_aggr = aggr0Spc3

[spc1-04]
node_num = 4
root_aggr = aggr0Spc4

[spc1-05]
node_num = 5
root_aggr = aggr0Spc5

[spc1-06]
node_num = 6
root_aggr = aggr0Spc6

[spc1-07]
node_num = 7
root_aggr = aggr0Spc7
```

```
[spc1-08]
node_num = 8
root_aggr = aggr0Spc8

[aggr1]
diskcount = 47
raidtype = raid_dp
maxraidsize = 24
volume-style = flex

[vs1]
rootvolume = vs1_rootvol0
aggregate = aggr1_n1

[asu1]
vol_size = 783732446KB
vols_per_node = 4
luns_per_vol = 1
lun_size = 653110372KB

[asu2]
vol_size = 783732446KB
vols_per_node = 4
luns_per_vol = 1
lun_size = 653110372KB

[asu3]
vol_size = 174162766KB
vols_per_node = 4
luns_per_vol = 1
lun_size = 145135638KB
```

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
        interval=60, retries=5

        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=120):
            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
```

config_storage.py

```
import getopt
import sys
import time

print sys.path
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 = "cluster time-service ntp server create -server %s" %
(timeserver)
        #storage.send_cluster(ntp_create)
    storage.send_cluster("ntp config modify -enabled true")

def disable_onboard_perfstat(storage):
    storage.send_cluster("statistics archive config modify -perfstat-sampling-period
0")

def weekly_raid_scrubs(storage):
    storage.node_run_all("storage raid-options modify -node * -name
raid.scrub.schedule -value 6h@sun@1")

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))
        #              (diskcount,     node,    raidtype,    maxraidsize) +
        #              " -volume-style %s" % (vol_style))
        storage.send_cluster(aggr_create, timeout=180)
        # waits a max of 15*60 seconds for aggr create to complete
        # storage.wait_for_last_job_complete(interval=60, retries=10)
        time.sleep(15)
        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) +
                 "-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 enabled -vserver %s" %
                          (storage.cluster.vserver))
            storage.send_cluster(lun_create)
            lun_map = ("lun map -path %s -igroup ig1 -vserver %s"
                       %(lun_path, storage.cluster.vserver))
            storage.send_cluster(lun_map)
            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(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)


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)
        disable_onboard_perfstat(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_lifs(storage)
        create_igroup(storage)
        create_data_luns(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
```

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

ASU Pre-Fill

The content of command and parameter file, used in this benchmark to execute the required ASU pre-fill, is listed below.

```
compratio=1
hd=localhost,jvms=12
sd=default,threads=12,openflags=o_direct
sd=sd1,lun=/dev/mapper/asu1_vg00-asu1_lv00,size=5350925271040
sd=sd2,lun=/dev/mapper/asu1_vg01-asu1_lv01,size=5350925271040
sd=sd3,lun=/dev/mapper/asu1_vg02-asu1_lv02,size=5350925271040
sd=sd4,lun=/dev/mapper/asu1_vg03-asu1_lv03,size=5350925271040
sd=sd5,lun=/dev/mapper/asu2_vg04-asu2_lv04,size=5350925271040
sd=sd6,lun=/dev/mapper/asu2_vg05-asu2_lv05,size=5350925271040
sd=sd7,lun=/dev/mapper/asu2_vg06-asu2_lv06,size=5350925271040
sd=sd8,lun=/dev/mapper/asu2_vg07-asu2_lv07,size=5350925271040
sd=sd9,lun=/dev/mapper/asu3_vg08-asu3_lv08,size=1189034852352
sd=sd10,lun=/dev/mapper/asu3_vg09-asu3_lv09,size=1189034852352
sd=sd11,lun=/dev/mapper/asu3_vg10-asu3_lv10,size=1189034852352
sd=sd12,lun=/dev/mapper/asu3_vg11-asu3_lv11,size=1189034852352

wd=wd1,sd=sd*,seekpct=eof,rdpct=0,xfersize=1m
rd=rd1,wd=wd*,elapsed=72h,interval=60,iorate=max
```

Primary Metrics and Repeatability Tests

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)* and Repeatability *(Repeatability Test Phase 1 and Repeatability Test Phase 2)* Tests is listed below.

```
host=master
slaves=(h1_slave1,h1_slave2,h1_slave3,h1_slave4,h1_slave5,h1_slave6,h1_slave7,h1_sla
ve8,h1_slave9,h1_slave10,h1_slave11,h1_slave12,h1_slave13,h1_slave14,h1_slave15,h1_s
lave16,h2_slave1,h2_slave2,h2_slave3,h2_slave4,h2_slave5,h2_slave6,h2_slave7,h2_slav
e8,h2_slave9,h2_slave10,h2_slave11,h2_slave12,h2_slave13,h2_slave14,h2_slave15,h2_sl
ave16,h3_slave1,h3_slave2,h3_slave3,h3_slave4,h3_slave5,h3_slave6,h3_slave7,h3_slave
8,h3_slave9,h3_slave10,h3_slave11,h3_slave12,h3_slave13,h3_slave14,h3_slave15,h3_sla
ve16,h4_slave1,h4_slave2,h4_slave3,h4_slave4,h4_slave5,h4_slave6,h4_slave7,h4_slave8
,h4_slave9,h4_slave10,h4_slave11,h4_slave12,h4_slave13,h4_slave14,h4_slave15,h4_slav
e16,h5_slave1,h5_slave2,h5_slave3,h5_slave4,h5_slave5,h5_slave6,h5_slave7,h5_slave8,
h5_slave9,h5_slave10,h5_slave11,h5_slave12,h5_slave13,h5_slave14,h5_slave15,h5_slave
16,h7_slave1,h7_slave2,h7_slave3,h7_slave4,h7_slave5,h7_slave6,h7_slave7,h7_slave8,h
7_slave9,h7_slave10,h7_slave11,h7_slave12,h7_slave13,h7_slave14,h7_slave15,h7_slave1
6,h8_slave1,h8_slave2,h8_slave3,h8_slave4,h8_slave5,h8_slave6,h8_slave7,h8_slave8,h8
_slave9,h8_slave10,h8_slave11,h8_slave12,h8_slave13,h8_slave14,h8_slave15,h8_slave16
,h9_slave1,h9_slave2,h9_slave3,h9_slave4,h9_slave5,h9_slave6,h9_slave7,h9_slave8,h9_
slave9,h9_slave10,h9_slave11,h9_slave12,h9_slave13,h9_slave14,h9_slave15,h9_slave16,
h10_slave1,h10_slave2,h10_slave3,h10_slave4,h10_slave5,h10_slave6,h10_slave7,h10_sla
ve8,h10_slave9,h10_slave10,h10_slave11,h10_slave12,h10_slave13,h10_slave14,h10_slave
15,h10_slave16)

sd=asu1_l00,lun=/dev/mapper/asu1_vg00-asu1_lv00,size=5350925271040
sd=asu1_l01,lun=/dev/mapper/asu1_vg01-asu1_lv01,size=5350925271040
sd=asu1_l02,lun=/dev/mapper/asu1_vg02-asu1_lv02,size=5350925271040
```

```
sd=asu1_l03,lun=/dev/mapper/asu1_vg03-asu1_lv03,size=5350925271040
sd=asu2_l04,lun=/dev/mapper/asu2_vg04-asu2_lv04,size=5350925271040
sd=asu2_l05,lun=/dev/mapper/asu2_vg05-asu2_lv05,size=5350925271040
sd=asu2_l06,lun=/dev/mapper/asu2_vg06-asu2_lv06,size=5350925271040
sd=asu2_l07,lun=/dev/mapper/asu2_vg07-asu2_lv07,size=5350925271040
sd=asu3_l08,lun=/dev/mapper/asu3_vg08-asu3_lv08,size=1189034852352
sd=asu3_l09,lun=/dev/mapper/asu3_vg09-asu3_lv09,size=1189034852352
sd=asu3_l10,lun=/dev/mapper/asu3_vg10-asu3_lv10,size=1189034852352
sd=asu3_l11,lun=/dev/mapper/asu3_vg11-asu3_lv11,size=1189034852352
```

SPC-1 Persistence Test

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

```
sd=asu1_l00,lun=/dev/mapper/asu1_vg00-asu1_lv00,size=5350925271040
sd=asu1_l01,lun=/dev/mapper/asu1_vg01-asu1_lv01,size=5350925271040
sd=asu1_l02,lun=/dev/mapper/asu1_vg02-asu1_lv02,size=5350925271040
sd=asu1_l03,lun=/dev/mapper/asu1_vg03-asu1_lv03,size=5350925271040
sd=asu2_l04,lun=/dev/mapper/asu2_vg04-asu2_lv04,size=5350925271040
sd=asu2_l05,lun=/dev/mapper/asu2_vg05-asu2_lv05,size=5350925271040
sd=asu2_l06,lun=/dev/mapper/asu2_vg06-asu2_lv06,size=5350925271040
sd=asu2_l07,lun=/dev/mapper/asu2_vg07-asu2_lv07,size=5350925271040
sd=asu3_l08,lun=/dev/mapper/asu3_vg08-asu3_lv08,size=1189034852352
sd=asu3_l09,lun=/dev/mapper/asu3_vg09-asu3_lv09,size=1189034852352
sd=asu3_l10,lun=/dev/mapper/asu3_vg10-asu3_lv10,size=1189034852352
sd=asu3_l11,lun=/dev/mapper/asu3_vg11-asu3_lv11,size=1189034852352
```

Slave JVMs

Each Slave JVM was invoked with a command and parameter file similar to the example listed below. The only difference in each file was **host** parameter value, which was unique to each Slave JVM, e.g. **h1_slave1**…**h10_slave16**.

```
master=10.63.173.44
host=h1_slave1
```

APPENDIX E: SPC-1 WORKLOAD GENERATOR INPUT PARAMETERS

The first script, **spc1.start1.sh**, was invoked to collect a set of configuration information, execute the required ASU pre-fill, invoke a script, **start_all_slaves_jvms.sh**, to start all of the Slave JVMs, execute the Primary Metrics Test *(Sustainability Test Phase, IOPS Test Phase, and Response Time Ramp Test Phase)*, the Repeatability Test *(Repeatability Test Phase 1 and Repeatability Test Phase 2)*, invoke a script, **stop_all_slaves_jvms.sh**, to stop all of the Slave JVMs, and execute the SPC-1 Persistence Test Run 1 *(write phase)* in an uninterrupted sequence.

After completing the required Test Storage Configuration power off/power on cycle, the **spc1.start2.sh** script was invoked to execute the SPC-1 Persistence Test Run 2 *(read phase)* and collect a second set of configuration information.

spc1.start1.sh

```
#/bin/bash

export CLASSPATH=/opt/SPC1RunDir
export LD_LIBRARY_PATH=/opt/SPC1RunDir

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

# Vdbench luns pre-fill run
cd /opt/SPC1RunDir
/opt/SPC1RunDir/vdbench/vdbench -f /opt/SPC1RunDir/vdbench/prefill_luns.txt -o
/opt/SPC1RunDir/vdbench/luns.prefill


# SPC-1 Start All Slaves
/opt/SPC1RunDir/start_all_slaves_jvms.sh

# Metrics
java -Xms4096m -Xmx4096m -Xss512k metrics -b 13705 -t 28800 -r 600 -s 36000:180

# SPC-1 benchmark Repeatability1 Test
java -Xms4096m -Xmx4096m -Xss512k repeat1 -b 13705  -s 180

# SPC-1 benchmark Repeatability2 Test
java -Xms4096m -Xmx4096m -Xss512k repeat2 -b 13705 -s 180

# Stop All slaves
/opt/SPC1RunDir/stop_all_slaves_jvms.sh

# Prepare spc1.cfg file for Persistence1 Test
/bin/mv /opt/SPC1RunDir/spc1.cfg /opt/SPC1RunDir/spc1.cfg.multihost
/bin/cp /opt/SPC1RunDir/spc1.cfg.persist /opt/SPC1RunDir/spc1.cfg
#
# SPC-1 benchmark Persistence1 Test
java -Xms8192m -Xmx8192m -Xss512k persist1 -b 13705
/opt/audit.ksh 13705
```

start_all_slaves_jvms.sh

This script used SSH to invoke a script on each of the nine Host Systems to start the Slave JVMs assigned to the Host System.

```
#!/bin/bash
/usr/bin/ssh spc1 'cd /opt/SPC1RunDir; /opt/SPC1RunDir/launch_host1_slaves.sh &' &
sleep 2
/usr/bin/ssh spc2 'cd /opt/SPC1RunDir; /opt/SPC1RunDir/launch_host2_slaves.sh &' &
sleep 2
/usr/bin/ssh spc3 'cd /opt/SPC1RunDir; /opt/SPC1RunDir/launch_host3_slaves.sh &' &
sleep 2
/usr/bin/ssh spc4 'cd /opt/SPC1RunDir; /opt/SPC1RunDir/launch_host4_slaves.sh &' &
sleep 2
/usr/bin/ssh spc5 'cd /opt/SPC1RunDir; /opt/SPC1RunDir/launch_host5_slaves.sh &' &
sleep 2
/usr/bin/ssh spc7 'cd /opt/SPC1RunDir; /opt/SPC1RunDir/launch_host7_slaves.sh &' &
sleep 2
/usr/bin/ssh spc8 'cd /opt/SPC1RunDir; /opt/SPC1RunDir/launch_host8_slaves.sh &' &
sleep 2
/usr/bin/ssh spc9 'cd /opt/SPC1RunDir; /opt/SPC1RunDir/launch_host9_slaves.sh &' &
sleep 2
/usr/bin/ssh spc10 'cd /opt/SPC1RunDir; /opt/SPC1RunDir/launch_host10_slaves.sh &' &
```

launch_host1_slaves.sh

An example of the script invoked to start the Slave JVMs on each Host System. This specific script started the Slave JVMs on the first Host System.

```
#!/bin/bash
export CLASSPATH=/opt/SPC1RunDir
export LD_LIBRARY_PATH=/opt/SPC1RunDir
java -Xms4096m -Xmx4096m -Xss512k spc1 -f /opt/SPC1RunDir/h10_slave1.txt  -o
/opt/SPC1RunDir/h10_slave1 2>&1 &
java -Xms4096m -Xmx4096m -Xss512k spc1 -f /opt/SPC1RunDir/h10_slave2.txt  -o
/opt/SPC1RunDir/h10_slave2 2>&1 &
java -Xms4096m -Xmx4096m -Xss512k spc1 -f /opt/SPC1RunDir/h10_slave3.txt  -o
/opt/SPC1RunDir/h10_slave3 2>&1 &
java -Xms4096m -Xmx4096m -Xss512k spc1 -f /opt/SPC1RunDir/h10_slave4.txt  -o
/opt/SPC1RunDir/h10_slave4 2>&1 &
java -Xms4096m -Xmx4096m -Xss512k spc1 -f /opt/SPC1RunDir/h10_slave5.txt  -o
/opt/SPC1RunDir/h10_slave5 2>&1 &
java -Xms4096m -Xmx4096m -Xss512k spc1 -f /opt/SPC1RunDir/h10_slave6.txt  -o
/opt/SPC1RunDir/h10_slave6 2>&1 &
java -Xms4096m -Xmx4096m -Xss512k spc1 -f /opt/SPC1RunDir/h10_slave7.txt  -o
/opt/SPC1RunDir/h10_slave7 2>&1 &
java -Xms4096m -Xmx4096m -Xss512k spc1 -f /opt/SPC1RunDir/h10_slave8.txt  -o
/opt/SPC1RunDir/h10_slave8 2>&1 &
java -Xms4096m -Xmx4096m -Xss512k spc1 -f /opt/SPC1RunDir/h10_slave9.txt  -o
/opt/SPC1RunDir/h10_slave9 2>&1 &
java -Xms4096m -Xmx4096m -Xss512k spc1 -f /opt/SPC1RunDir/h10_slave10.txt  -o
/opt/SPC1RunDir/h10_slave10 2>&1 &
java -Xms4096m -Xmx4096m -Xss512k spc1 -f /opt/SPC1RunDir/h10_slave11.txt  -o
/opt/SPC1RunDir/h10_slave11 2>&1 &
java -Xms4096m -Xmx4096m -Xss512k spc1 -f /opt/SPC1RunDir/h10_slave12.txt  -o
/opt/SPC1RunDir/h10_slave12 2>&1 &
java -Xms4096m -Xmx4096m -Xss512k spc1 -f /opt/SPC1RunDir/h10_slave13.txt  -o
/opt/SPC1RunDir/h10_slave13 2>&1 &
```

```
java -Xms4096m -Xmx4096m -Xss512k spc1 -f /opt/SPC1RunDir/h10_slave14.txt  -o
/opt/SPC1RunDir/h10_slave14 2>&1 &
java -Xms4096m -Xmx4096m -Xss512k spc1 -f /opt/SPC1RunDir/h10_slave15.txt  -o
/opt/SPC1RunDir/h10_slave15 2>&1 &
java -Xms4096m -Xmx4096m -Xss512k spc1 -f /opt/SPC1RunDir/h10_slave16.txt  -o
/opt/SPC1RunDir/h10_slave16 2>&1 &
```

stop_all_slaves_jvms.sh

This script used SSH to stop the Slave JVMs on each Host System.

```
#!/bin/bash
ssh spc1 'pkill java'
sleep 1
ssh spc2 'pkill java'
sleep 1
ssh spc3 'pkill java'
sleep 1
ssh spc4 'pkill java'
sleep 1
ssh spc5 'pkill java'
sleep 1
ssh spc7 'pkill java'
sleep 1
ssh spc8 'pkill java'
sleep 1
ssh spc9 'pkill java'
sleep 1
ssh spc10 'pkill java'
```

spc1.start2.sh

This script was invoked to execute the SPC-1 Persistence Test Run 2 *(read phase)* and collect a second set of configuration information.

```
#!/bin/bash

export CLASSPATH=/opt/SPC1RunDir
export LD_LIBRARY_PATH=/opt/SPC1RunDir

# SPC-1 benchmark run - Persistence2 after power cycle of TSC
cd /opt/SPC1RunDir
java -Xms8192m -Xmx8192m -Xss512k persist2
sleep 5

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

APPENDIX F: THIRD-PARTY QUOTATION

QLogic QLE2672 HBAs

Digicom Technology LLC

4300 W. Lake Mary Blvd
Unit # 1010-129
Lake Mary, FL 32746

Estimate

Date	Estimate #
4/16/2015	2112

Name / Address
NetApp Attn: Chad

Project

Description	Qty	Rate	Total
QLE2672 QLOGIC SANBLADE 16GB FC 2P PCIE HBA NEW BULK PACK	9	1,250.00	11,250.00

Feel free to contact us if you have any questions
sales@digicomtechnology.net
(407) 416-3252

Total $11,250.00