

NEW ARCHITECTURES FOR APACHE SPARK™ AND BIG DATA

Contents

Key Trends in Big Data	4
Goal of the Study	4
Traditional Big Data Infrastructure in VMware Virtualized Environments	4
The Apache Spark Platform for Big Data	6
Infrastructure Requirements for the Apache Spark Platform	6
CPU	6
Memory	6
Disk	7
Network	7
Leveraging a Distributed File System Other Than HDFS for the Apache Spark Platform	7
Proof of Concept for Apache Spark Software on a vSphere Platform	8
GlusterFS for Apache Spark Software on a vSphere Platform	9
Basics of GlusterFS Architecture	9
Testing with TPC-DS for Apache Spark Software	10
Apache Spark Software with GlusterFS Proof of Concept Infrastructure	10
Physical Infrastructure	10
Servers and Networking	11
Storage	11
Virtual Architecture	11
Apache Spark Virtual Machine Components	11
Block Storage Based on Fibre Channel	11
vSAN Configuration	11
VMware Cloud on AWS	13

Apache Spark Architecture	13
Use Case 1: Distributed Storage Backed by Fibre Channel and Pure Storage FlashArray//M50	13
Use Case 2: Distributed Storage Backed by All-Flash vSAN	14
GlusterFS Configuration	14
The Apache Spark Console	15
TPC-DS Query Results	18
TPC-DS with VMware Cloud on AWS	19
Conclusion	20
References	20
About the Authors	21
Acknowledgments	21
Appendix A: GlusterFS Install and Setup Commands	22
Install Commands	22
Enabling the Service	22
Creating and Configuring Gluster Nodes	22
Name Resolution	22

Key Trends in Big Data

Enterprises are deploying new innovations in technology and replacing their legacy data platforms. The traditional data warehouse technology is not serving their needs. Newer data exists in many forms, such as structured, unstructured, streaming, and others; it is no longer just relational. There is a need to integrate systems that are distributed across the enterprise and provide a single source of truth. There is increasing demand to leverage new application architectures to enhance productivity for developers who want to explore data science, machine learning, and deep learning.

Due to the evolution of compute and storage capabilities such as SSD, Flash, and NVMe, the landscape and requirements for big data have been transformed. Decoupling compute and storage clusters is the ideal. Compute nodes must rapidly scale up from dozens to hundreds of nodes, and many of the modern big data platforms store and process most data in memory. Newer platforms such as Apache Spark™ software are primarily memory resident, with I/O taking place only at the beginning and end of the job. This contrasts with Apache Hadoop® MapReduce, with which every processing phase shows significant I/O activity. These contemporary platforms can leverage any regular local, shared, or distributed file system. Due to the evolving critical nature of these big data platforms, high availability and fault tolerance are mandatory for components with single points of failure.

Goal of the Study

The goal of this study is to determine whether we can leverage shared storage in modern big data platforms. We will look at current in-memory big data platforms and their suitability for virtualization. The in-memory nature of these newer platforms makes them less dependent on I/O and storage protocols. In this study, we leverage virtualized shared storage and the core VMware vSphere® features in an effort to build out a highly available and performant infrastructure for contemporary big data platforms.

Traditional Big Data Infrastructure in VMware Virtualized Environments

Due to the special requirements for Hadoop and MapReduce in traditional big data environments, dedicated clusters with many local disks have usually been recommended in a VMware environment. The sequential I/O and large block size requirements of the Hadoop Distributed File System (HDFS) have not been a good fit for the traditional shared storage used in standard virtualized environments. MapReduce on HDFS has its built-in redundancy from a job and data perspective across physical nodes, which obviates the need for high availability and load balancing.

The anatomy of a MapReduce job includes multiple phases, with I/O to read and write to disk at every phase of the process. I/O throughput and performance are critical for these traditional workloads. Some core features of vSphere, such as VMware vSphere vMotion®, VMware vSphere Distributed Resource Scheduler™ (vSphere DRS), and VMware vSphere High Availability (vSphere HA), cannot be leveraged for the worker nodes in these dedicated clusters, because HDFS and the data reside on local disks, constraining virtual machine (VM) mobility.

Solutions such as those of Dell EMC™ Isilon® provide the ability to serve HDFS, from a shared storage array to a virtual Hadoop cluster, but these are vendor-specific solutions. The usual limitations of a physical cluster to independently scale compute and storage also pertain to a dedicated VMware cluster. Because the traditional architecture requires the use of direct attached storage (DAS), managing these drives is a challenge as compared to centralized management for commonly used SAN and NAS storage in virtualized environments.

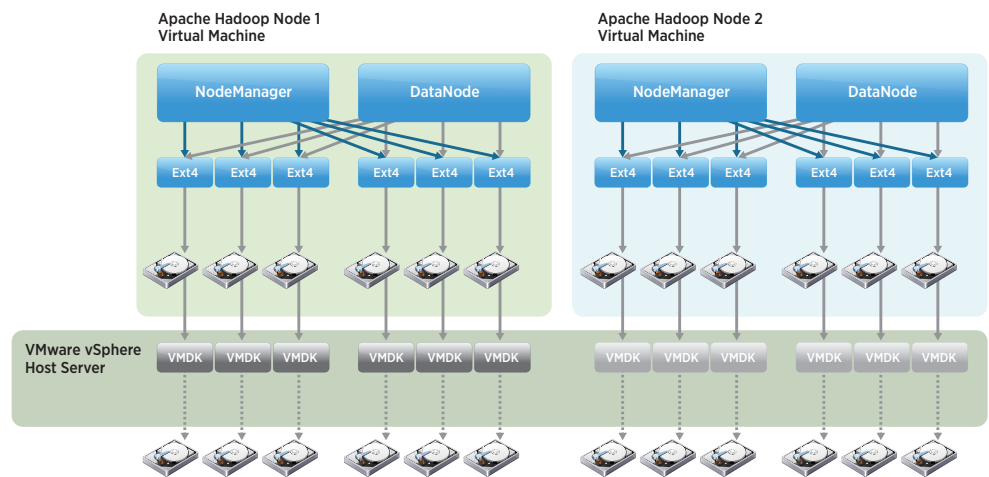


Figure 1. Traditional Architecture for Big Data on vSphere

The Apache Spark Platform for Big Data

The Apache Spark platform is an open-source cluster computing system with an in-memory data processing engine. It has a rich set of APIs for Java, Scala, Python, and R as well as an optimized engine for ETL, analytics, machine learning, and graph processing. Spark is a distributed platform for complex multistage applications. It also supports many higher-level tools, including Spark SQL for SQL and structured data processing, [MLlib](#) for machine learning, [GraphX](#) for graph processing, and [Spark Streaming](#).

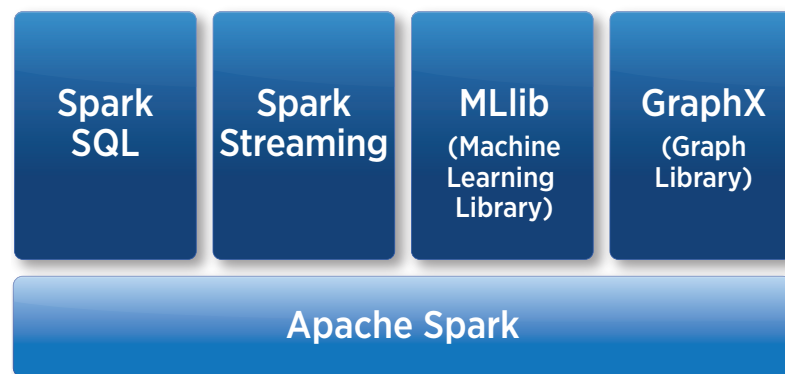


Figure 2. Apache Spark Platform Components

Infrastructure Requirements for the Apache Spark Platform

Before we look at creating a proof of concept (POC) for the Apache Spark platform, we will discuss the requirements for the infrastructure components. See [Apache Spark](#).

CPU

Apache Spark software scales well, to tens of CPU cores per server and to hundreds of servers, due to minimal sharing between threads and because it is truly distributed. We recommend provisioning at least 8-16 cores per machine. The actual number of cores and the number of servers depend on the particular workload.

Memory

In general, Apache Spark software runs well with anywhere from eight to hundreds of gigabytes of memory per machine. In all cases, allocate no more than 75 percent of memory for Spark use; reserve the remainder for the operating system (OS) and buffer cache. To determine how much an application uses for a certain dataset size, load part of the dataset in a Spark resilient distributed dataset (RDD) and click **Storage** on the Spark monitoring UI (<http://<driver-node>:4040>) to see its memory size. The Java VM does not always perform well with more than 200GB of RAM, so size the VM based on these factors.

Disk

Although the Apache Spark platform can perform much of its computation in memory, it uses local disks to store data that doesn't fit in RAM and to preserve intermediate output between stages. HDFS has been the traditional de facto file system for big data, but Spark software can use any available local or distributed file system. The `spark.local.dir` variable can be used to set up the location of the storage to be used for processing.

Network

Due to the in-memory nature of the Apache Spark platform, it can be network bound. We recommend a 10Gbps or higher network for Spark applications. These requirements are primarily due to the distributed nature of Spark computing.

Leveraging a Distributed File System Other Than HDFS for the Apache Spark Platform

Apache Spark software works with any local or distributed file system solution available for the typical Linux platform. As mentioned earlier, HDFS is an older file system and big data storage mechanism that has many limitations. These make it incompatible with the common shared storage, such as SAN and NAS, used in virtualized environments. We opted to use a distributed storage platform that works with Spark software and is compatible with virtualized shared storage. By leveraging virtualized shared storage, the core features of the vSphere platform such as vSphere vMotion, vSphere DRS, vSphere HA, and so on, can be fully utilized for the big data infrastructure.

Many file system candidates present themselves. The most common ones are NFS, Ceph, and GlusterFS. A solution is needed that works well with any shared storage backend for the vSphere platform, including block, NFS, and VMware vSAN™. A solution is required that can be easily set up and scaled across hundreds of nodes, if needed, with minimal complexity. A comprehensive shared storage solution for big data must provide the ability to ingest data from multiple sources using the prevailing mechanisms such as NFS, CIFS, HDFS, Amazon S3 (Object), FTP, and so on.

- NFS, if available, as a data repository for a Spark platform can be leveraged, but it must be highly available. Its innate horizontal scalability is limited.
- Ceph is relatively new and is primarily used in container environments as a shared storage backend.
- GlusterFS is mature and has been commonly used for many years as a distributed file system. It is easy to set up and has its own optimized driver for Linux.

Proof of Concept for Apache Spark Software on a vSphere Platform

Based on the profile and requirements of the newer big data platforms such as Apache Spark, we created a POC to leverage and set up distributed storage not based on HDFS for Spark software on a vSphere platform. The POC exercised SQL query tests, as described in the “Testing with TPC-DS for Apache Spark Software” section, with GlusterFS as the shared file system replacing traditional HDFS. Because past studies have shown GlusterFS to be the most performant file system as compared to HDFS and Ceph, we chose it as the underlying distributed storage for the POC. [Based on a study by IOPscience](#), Table 1 compares the performance of Ceph and GlusterFS to that of HDFS.

IOPscience designed tests to compare reading and writing data across the HDFS, GlusterFS, and Ceph file systems. They primarily used IOzone, a very popular multithreaded file system benchmark tool, for this comparison. Because HDFS is not suited for random writes, some of these tests error out.

	HDFS	HDFS	CEPH	CEPH	CEPH	GLUSTER	GLUSTER
	24 Threads	36 Threads	0.61	0.67.3	0.70	3.3	3.4
Initial Write	239.72	N.A.	52.49	18.93	51.06	234.06	306.34
Rewrite	ERROR	ERROR	54.05	19.31	60.05	311.75	406.9
Random Write	ERROR	ERROR	ERROR	13.96	7	326.89	406.33
Read	155.18	193.65	95.38	53.4	101.58	621.08	688.06
Reread	151.33	207.43	102.04	57.29	133.61	662.92	711.46
Random Read	29.06	39.98	ERROR	5.13	12.05	242.75	284

Table 1. Comparison of Read/Write Performance Across HDFS, Ceph, and GlusterFS Platforms in MBps
(Source: [IOPscience study](#))

GlusterFS for Apache Spark Software on a vSphere Platform

GlusterFS is an open-source scalable network file system that can run on top of any backend storage. It is a proven storage solution for large distributed datastores used in media streaming, data analysis, and bandwidth-intensive tasks.

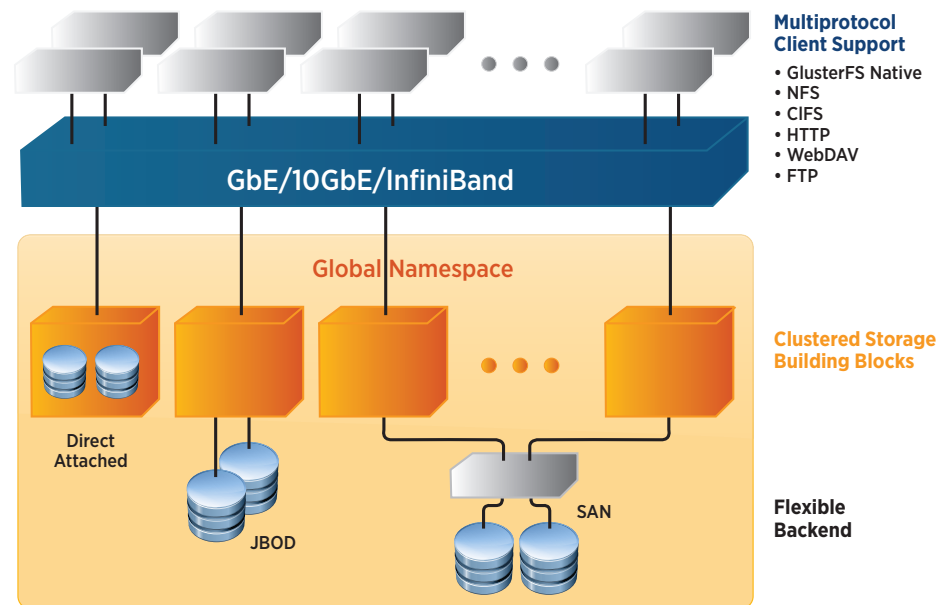


Figure 3. GlusterFS Logical Architecture (Source: [GlusterFS and Hadoop](#))

Basics of GlusterFS Architecture

Figure 4 shows a GlusterFS volume, a collection of bricks. A brick is a logical construct that represents a particular directory on an underlying disk file system in a GlusterFS node. It is used as a building block for the volumes.

Based on requirements, GlusterFS supports various types of volumes. These volumes can be built for scaling, performance, or high availability. The POC uses a replicated GlusterFS volume. In this volume, files are distributed across replicated sets of bricks across servers. This type of volume is used when high availability for the data is required.

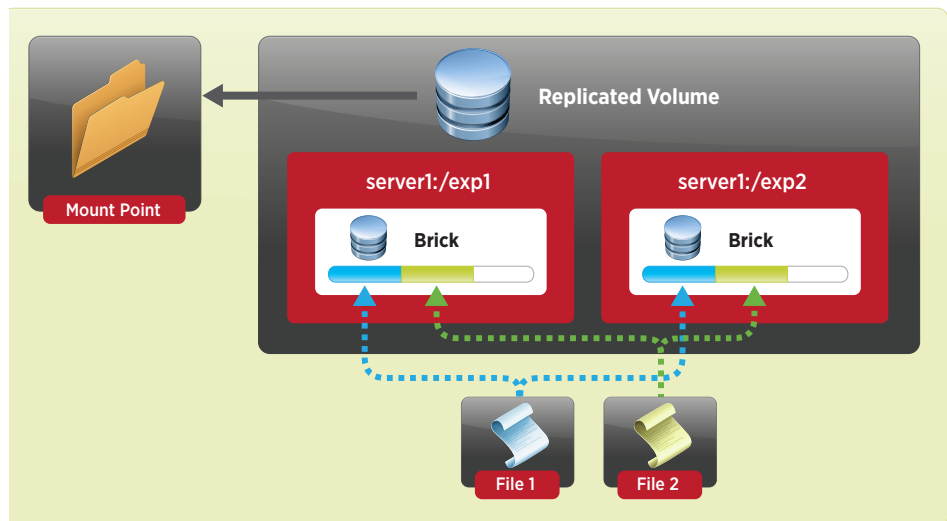


Figure 4. Replicated GlusterFS Volume (Source: [Gluster Quick Start Guide](#))

Testing with TPC-DS for Apache Spark Software

TPC-DS is the common industry-standard tool suite for performance testing of decision support solutions and for validation of the infrastructure. It is one of the tools often used in testing Apache Spark environments. We used subsets of TPC-DS queries to validate the performance of our POC environment. As described on its home page, TPC-DS does the following:

- Examines large volumes of data
- Provides answers to real-world business questions
- Executes queries of various operational requirements and complexities—for example, ad hoc, reporting, iterative OLAP, and data mining
- Characterizes workloads by high CPU and I/O load

Apache Spark Software with GlusterFS Proof of Concept Infrastructure

Physical Infrastructure

The POC was deployed both on premises and on VMware Cloud™ on AWS. We first look at the on-premises deployment, then look at the same deployment on VMware Cloud on AWS, and then compare the results.

The on-premises Apache Spark POC environment was built in the VMware Solutions Lab. The lab leverages VMware and partner resources to build application solutions and POCs. The POC infrastructure had the following components.

Servers and Networking

A four-node vSphere cluster with the following physical server specifications was used to host the Apache Spark cluster components:

- Dell PowerEdge R720 server (44 cores/88 threads)
- 1TB RAM
- 4 x 10Gbps network adapters
- 2 x 16Gbps Fibre Channel ports

Storage

The POC was validated on two different types of backend storage:

- Block storage backed by a Brocade 16Gbps Fibre Channel fabric attached to a Pure Storage FlashArray//M50
- vSAN storage with Western Digital NVMe for cache and Western Digital Optimus MAX SSD 3.5TB drives for the capacity tier, with two of the four 10Gbps ports in each server dedicated for vSAN traffic across two distinct port groups

TPC-DS query times for the two use cases were compared. The TPC-DS dataset was sized at 5TB.

Virtual Architecture

The virtual infrastructure was deployed on a four-node compute cluster. Each server contained 44 physical cores and 1TB of RAM.

Apache Spark Virtual Machine Components

Each VM corresponds with the following Apache Spark node description:

- One Spark master node with 16 vCPUs and 128GB RAM
- Eight Spark worker nodes with 16 vCPUs and 128GB RAM with 2 x 200GB disks used for OS and scratch space respectively (total worker compute capacity: 128 cores and 1,024GB RAM)

The solution used CentOS Linux 7.x VMs with Apache Spark 2.2.0 for master and worker nodes.

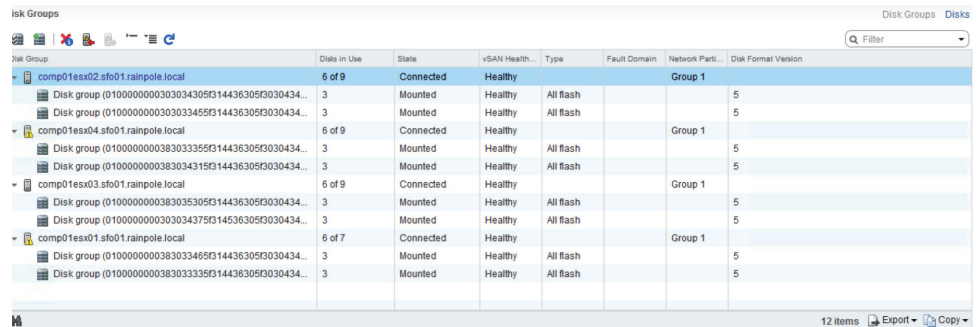
The following two storage methods were used for independent test sequences for purposes of comparison; they were not used together.

Block Storage Based on Fibre Channel

A large, 50TB LUN hosted on a Pure Storage FlashArray//M50 was used as backend storage for all Apache Spark workloads. All vSphere hosts had dual 16Gbps Fibre Channel connections to the storage fabric.

vSAN Configuration

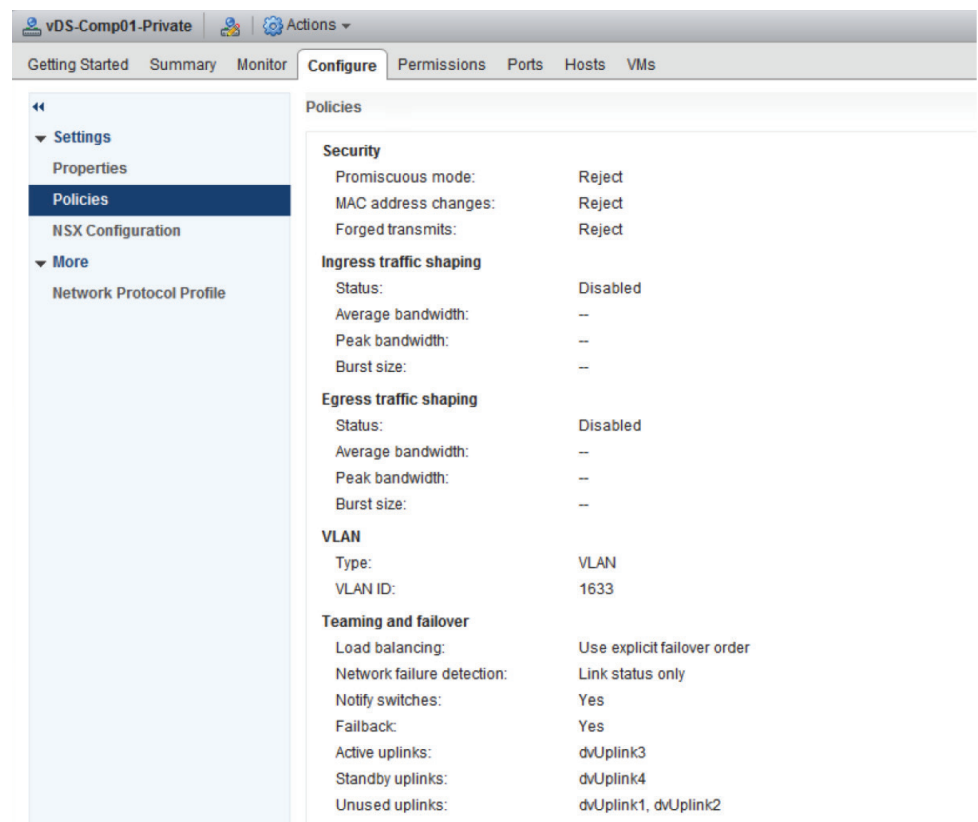
The all-Flash vSAN configuration used the following components. Each node had two disk groups, with one NVMe drive for caching and two Flash drives for capacity.



Disk Group	Disks in Use	State	vSAN Health	Type	Fault Domain	Network Part...	Disk Format Version
comp01esx02.sfo01.rainpole.local	6 of 9	Connected	Healthy			Group 1	
Disk group (0100000000303034305914436305030434...	3	Mounted	Healthy	All flash			5
Disk group (010000000030303435914436305030434...	3	Mounted	Healthy	All flash			5
comp01esx04.sfo01.rainpole.local	6 of 9	Connected	Healthy			Group 1	
Disk group (010000000030303435914436305030434...	3	Mounted	Healthy	All flash			5
Disk group (010000000030303435914436305030434...	3	Mounted	Healthy	All flash			5
comp01esx03.sfo01.rainpole.local	6 of 9	Connected	Healthy			Group 1	
Disk group (010000000030303435914436305030434...	3	Mounted	Healthy	All flash			5
Disk group (0100000000303034375914536305030434...	3	Mounted	Healthy	All flash			5
comp01esx01.sfo01.rainpole.local	6 of 7	Connected	Healthy			Group 1	
Disk group (0100000000303034365914436305030434...	3	Mounted	Healthy	All flash			5
Disk group (010000000030303335914436305030434...	3	Mounted	Healthy	All flash			5

Figure 5. vSAN Disk Group Configuration

Two distinct port groups for use by vSAN traffic were created. Two separate 10Gbps network adapters were dedicated to these port groups. See Figure 6. The two port groups worked across different network boundaries; the vSAN instance load-balanced between them. Under normal circumstances, port group Private1 used VMNIC3 as primary adapter and port group Private2 used VMNIC4 as primary adapter.



vDS-Comp01-Private | Actions

Getting Started | Summary | Monitor | **Configure** | Permissions | Ports | Hosts | VMs

Settings

- Properties
- Policies**
- NSX Configuration
- More
 - Network Protocol Profile

Policies

Security

- Promiscuous mode: Reject
- MAC address changes: Reject
- Forged transmits: Reject

Ingress traffic shaping

- Status: Disabled
- Average bandwidth: --
- Peak bandwidth: --
- Burst size: --

Egress traffic shaping

- Status: Disabled
- Average bandwidth: --
- Peak bandwidth: --
- Burst size: --

VLAN

- Type: VLAN
- VLAN ID: 1633

Teaming and failover

- Load balancing: Use explicit failover order
- Network failure detection: Link status only
- Notify switches: Yes
- Failback: Yes
- Active uplinks: dvUplink3
- Standby uplinks: dvUplink4
- Unused uplinks: dvUplink1, dvUplink2

Figure 6. Dedicated Highly Available Network Configuration for vSAN Traffic

VMware Cloud on AWS

The same POC was deployed on VMware Cloud on AWS, an on-demand service that enables applications to run across cloud environments based on the vSphere platform and having access to a broad range of AWS services. Powered by VMware Cloud Foundation™, this service integrates vSphere, vSAN, and VMware NSX®, along with VMware vCenter Server® management, and is optimized to run on dedicated, elastic, bare-metal AWS infrastructure. With this service, IT teams can manage their cloud-based resources with familiar VMware tools. The following are among the core benefits of VMware Cloud on AWS:

- Simple and consistent operations
- Enterprise-grade capabilities
- Flexible consumption options
- Delivered as a service from VMware

A four-node VMware Cloud on AWS cluster was used for the tests. Details of the VMware Cloud on AWS infrastructure can be found in the [VMware Cloud on AWS Technical Overview](#). VM configuration for all Apache Spark and GlusterFS components was identical to that of the on-premises use case.

Apache Spark Architecture

The Apache Spark compute architecture included one master server and eight worker nodes. See Figure 7. GlusterFS is the distributed file system shared across the entire Spark infrastructure. vSphere DRS automatically load-balances the VMs across the four-cluster server hosts.

Use Case 1: Distributed Storage Backed by Fibre Channel and Pure Storage FlashArray//M50

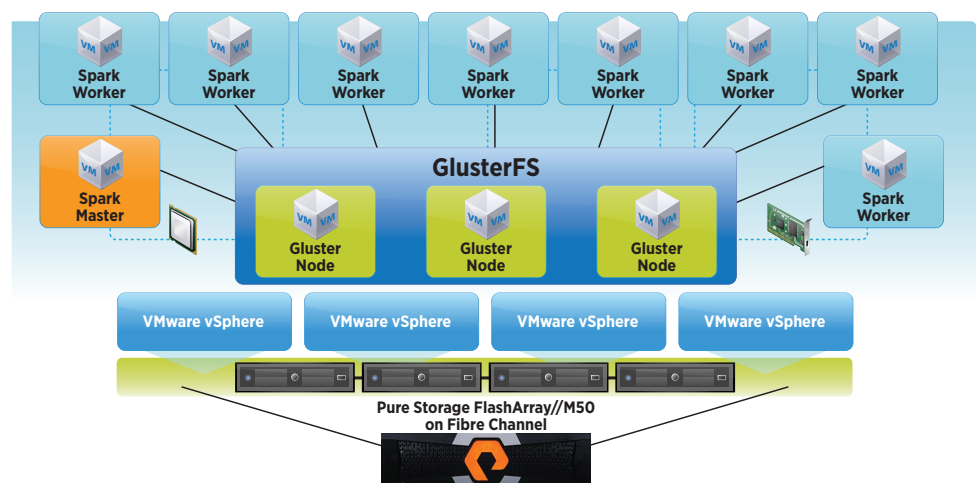


Figure 7. Apache Spark Standalone with GlusterFS on Fibre Channel Storage

The GlusterFS nodes were created with storage from the Pure Storage FlashArray//M50. Each of the three GlusterFS nodes has a 2TB data disk that is peered across the nodes. This 2TB disk is available for the Spark master and worker nodes as the distributed file system. The Gluster nodes were created with storage from the all-Flash vSAN storage. Each of the three GlusterFS nodes has a 2TB data disk that is peered across the nodes.

Use Case 2: Distributed Storage Backed by All-Flash vSAN

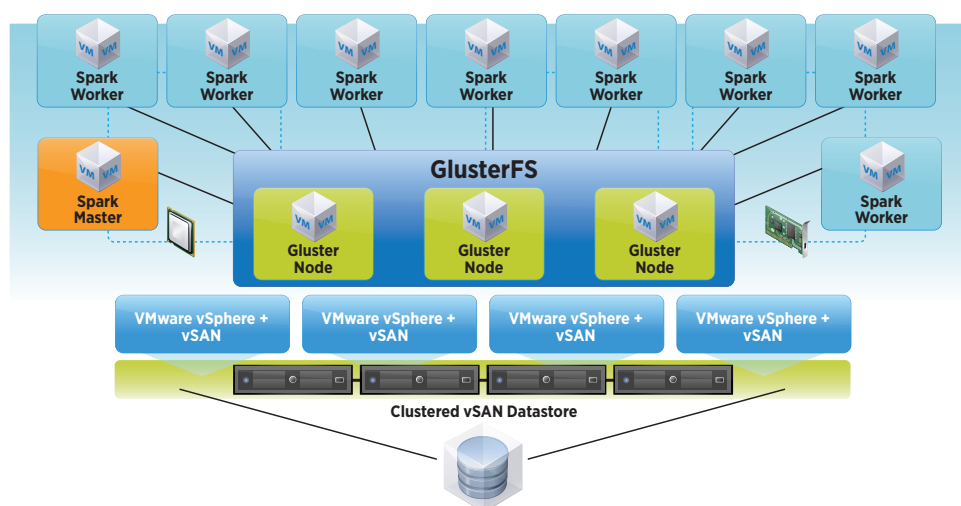


Figure 8. Apache Spark Standalone with GlusterFS on vSAN Storage

GlusterFS Configuration

```

root@gluster01:~# df
Filesystem            1K-blocks    Used  Available Use% Mounted on
/dev/mapper/centos-root 226383916 27678412 198705504 13% /
devtmpfs               61793064      0  61793064  0% /dev
tmpfs                  61803724      0  61803724  0% /dev/shm
tmpfs                  61803724    8708  61795016  1% /run
tmpfs                  61803724      0  61803724  0% /sys/fs/cgroup
/dev/sdb1              508588     311156   197432   62% /boot
/dev/mapper/centos-home 18376704     32940  18343764  1% /home
tmpfs                  12360748      0  12360748  0% /run/user/0
/dev/sdal              2146434048 263546308 1882887740 13% /glusterdata

root@gluster01 ~]# gluster peer status
Number of Peers: 2

Hostname: gluster02
Uuid: a9ad3364-9d36-417e-ba2b-672d2a5248a4
State: Peer in Cluster (Connected)

```

Figure 9. GlusterFS Peering Status

The three nodes acting as servers for GlusterFS were configured identically. They were then peered with each other using proprietary Gluster commands. Appendix A shows the details of the Gluster commands used on a CentOS system to install, peer, and configure nodes. Figure 10 shows the status of the peered nodes.

The data partitions from the peered nodes were then used to create a volume for use as the distributed file system. See Figure 10. The file systems were automatically mounted in the Apache Spark worker and master nodes through `/etc/fstab`, as shown in the example configuration, using the native GlusterFS mount option. The output shown in Figure 10 illustrates Spark nodes that had mounted the distributed file system served by GlusterFS.

```
[root@spark204 ~]# df
filesystem                1K-blocks      Used    Available  Use% Mounted on
/dev/mapper/centos-root  226383916  27729300  198654616   13% /
devtmpfs                  61793064      0    61793064    0% /dev
tmpfs                     61803724      0    61803724    0% /dev/shm
tmpfs                     61803724     8760    61794964    1% /run
tmpfs                     61803724      0    61803724    0% /sys/fs/cgroup
/dev/sdal                 2146434048    954960  2145479088    1% /data
/dev/sdb1                 508588      311156    197432    62% /boot
/dev/mapper/centos-home  18376704     32940    18343764    1% /home
tmpfs                     12360748      0    12360748    0% /run/user/0
gluster01:/sparkvol3     2146434048  316657408  1829776640   15% /sparkdata
[root@spark204 ~]#
```

Figure 10. GlusterFS Mounted Locally on Apache Spark Nodes

The Apache Spark Console

The standalone Apache Spark environment was launched by starting the master server, followed by the slaves that register with the master. Figure 11 shows the master server web console with the registered slave servers. The functions are performed by Java processes representing the workers and the master.

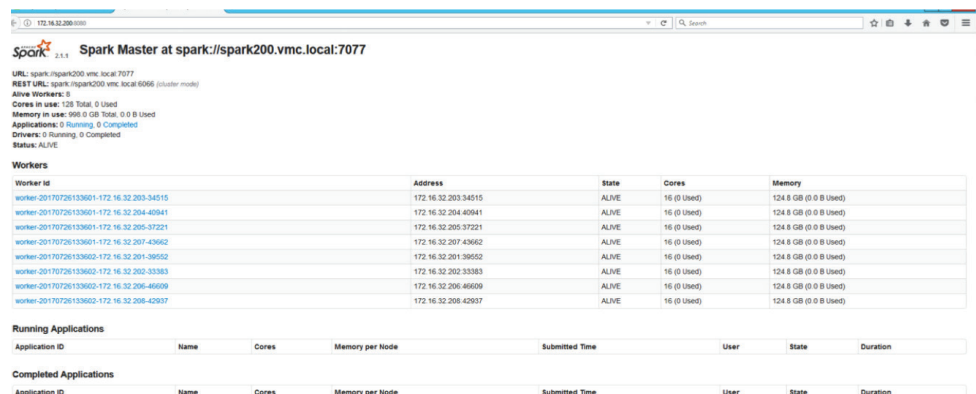
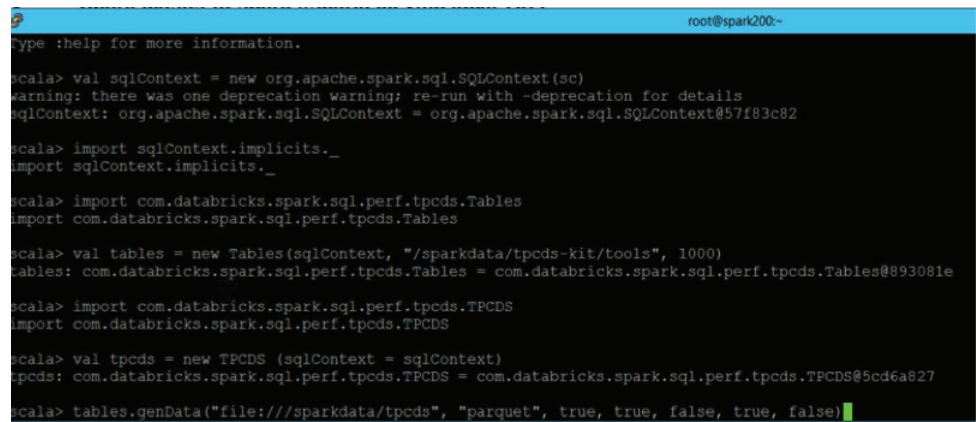


Figure 11. Apache Spark Web Console Dashboard

After the Spark components were up and running, the Spark shell was launched to run interactive jobs across the cluster. Figure 12 shows the use of **spark-shell** to generate a 1TB dataset for TPC-DS testing with Spark SQL.



```

root@spark200:~
Type :help for more information.

scala> val sqlContext = new org.apache.spark.sql.SQLContext(sc)
warning: there was one deprecation warning; re-run with -deprecation for details
sqlContext: org.apache.spark.sql.SQLContext = org.apache.spark.sql.SQLContext@57f83c82

scala> import sqlContext.implicits._
import sqlContext.implicits._

scala> import com.databricks.spark.sql.perf.tpcds.Tables
import com.databricks.spark.sql.perf.tpcds.Tables

scala> val tables = new Tables(sqlContext, "/sparkdata/tpcds-kit/tools", 1000)
tables: com.databricks.spark.sql.perf.tpcds.Tables = com.databricks.spark.sql.perf.tpcds.Tables@893081e

scala> import com.databricks.spark.sql.perf.tpcds.TPCDS
import com.databricks.spark.sql.perf.tpcds.TPCDS

scala> val tpcds = new TPCDS (sqlContext = sqlContext)
tpcds: com.databricks.spark.sql.perf.tpcds.TPCDS = com.databricks.spark.sql.perf.tpcds.TPCDS@5cd6a827

scala> tables.genData("file:///sparkdata/tpcds", "parquet", true, true, false, true, false)

```

Figure 12. Spark Shell Used for TPC-DS Data Load

The Pure Storage FlashArray dashboard tool captured the massive throughput required during the 5TB data-generation process. Figure 13 shows the various performance characteristics. Because this is a “one-off” data-generation process, the increased latency during this period can be tolerated.

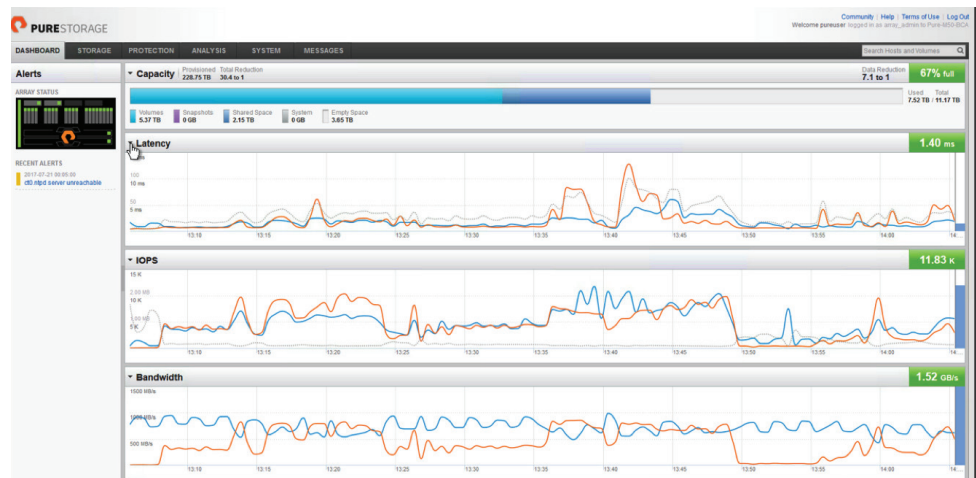


Figure 13. Load on Pure Storage FlashArray During TPC-DS Data Generation

Performance characteristics at the VM level were captured during data generation. Figure 14 shows the metrics for one of the VMs. All eight worker nodes were equally stressed during data generation. The performance screen displays an overview of all critical infrastructure components and their performance during the data load for one of the worker nodes.



Figure 14. Apache Spark Worker Node Performance Metrics During Test

The Apache Spark console enables users to drill down and look at executors and processes for running jobs. See Figure 15. **Summary** displays numbers of active tasks and cores in use. **Executors** shows the characteristics of the executors and the status of completed and active tasks. The standard output and standard error links can be used for troubleshooting.

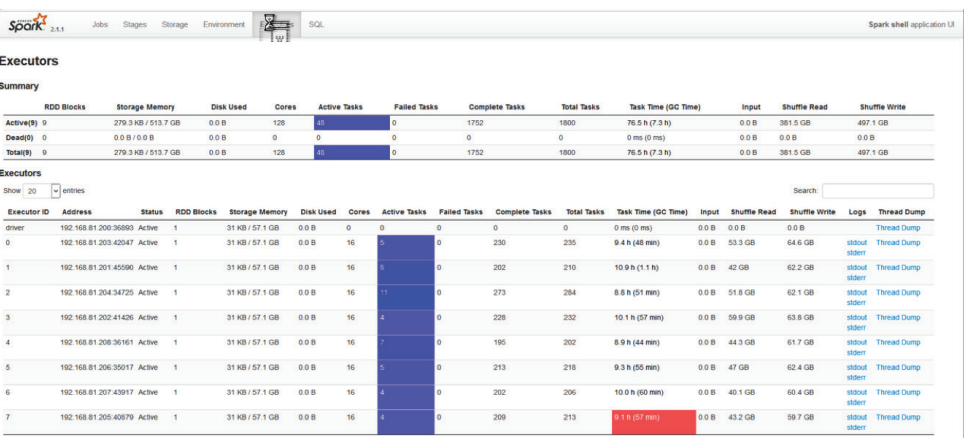


Figure 15. Apache Spark Executors Console During Data Generation

The Spark infrastructure executes the processes across multiple executors and various stages. Details are provided on the number of tasks and the time of completion. See Figure 16. **Completed Jobs** shows the various stages and the duration of each stage.

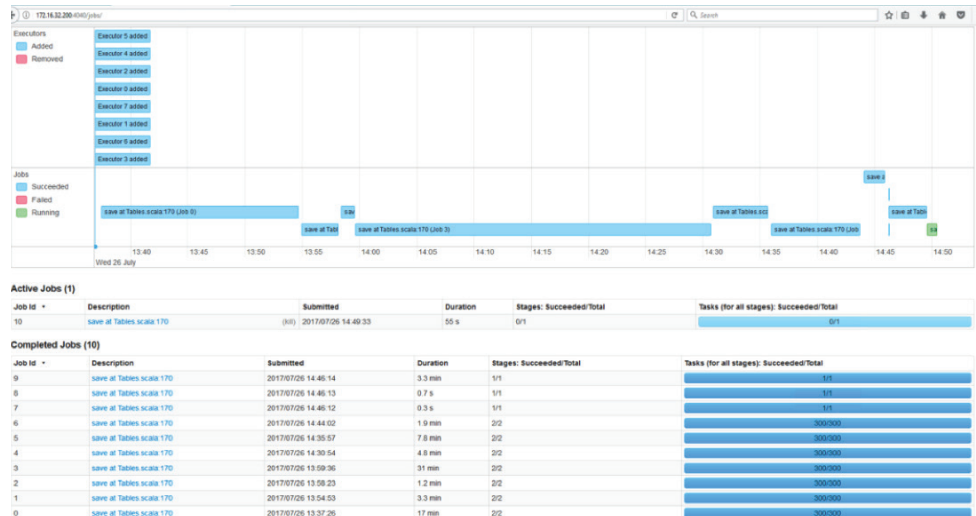


Figure 16. Apache Spark Jobs: Stages and Duration

TPC-DS Query Results

After the data was successfully generated, subsets of TPC-DS queries were run against the data, and the query times were captured. The tests were run multiple times to ensure data warm-up and consistent query results.

First, the query times were observed with the data residing on the Fibre Channel Pure Storage FlashArray//M50. Then the data was migrated via VMware vSphere Storage vMotion® to the all-Flash vSAN datastore. The query times were then observed with the data residing on the vSAN datastore. Table 2 compares the results.

QUERY (SECONDS)	FIBRE CHANNEL BLOCK	vSAN
q19	2.31	2.21
q42	0.32	0.23
q52	0.29	0.26
q55	0.35	0.23
q63	0.41	0.3
q68	2.85	2.5
q73	1.37	0.5
q98	0.54	0.51

Table 2. Interactive Query Time Comparison Between Fibre Channel SAN and vSAN

TPC-DS with VMware Cloud on AWS

A 1TB dataset with one master and eight worker nodes was generated on a VMware Cloud on AWS cluster. Sizing of CPU and memory is similar to that of the on-premises configuration.

The TPC-DS queries were run on VMware Cloud on AWS and were compared with those of the on-premises vSAN instance. Table 3 shows the results.

QUERY TIME (SECONDS)	ON-PREMISES	VMWARE CLOUD ON AWS
q19	2.61	2.86
q42	0.33	0.32
q52	0.27	0.32
q55	0.28	0.33
q63	0.47	0.35
q68	2.75	3.1
q73	2.82	2.49
q98	0.69	0.77

Table 3. Interactive Query Time Comparison Between On-Premises and VMware Cloud on AWS

Conclusion

This proof of concept and associated test results have affirmed that big data platforms such as Apache Spark are excellent candidates for deployment on virtual machines because they are primarily memory resident. GlusterFS provides a high-performance distributed file system for the Spark platform and newer big data workloads. In lieu of HDFS, Spark software can also use Ceph, NFS, and other distributed file systems. Because GlusterFS supports a wide range of protocols and can ingest data from multiple sources, it can potentially serve as the data lake for contemporary big data environments. All major VMware vSphere capabilities—including VMware vSphere vMotion, VMware vSphere Distributed Resource Scheduler, VMware vSphere High Availability, and so on—can be leveraged to provide redundancy and high availability to all big data components, including those that are single points of failure. Dedicated hardware with local storage is no longer the sole technical choice for modern big data applications.

TPC-DS testing showed similar performance for Spark SQL on vSAN and Fibre Channel storage. Our comparison between an on-premises configuration and a similar one on VMware Cloud on AWS also shows consistency, with equivalent results. By uniquely leveraging scalable distributed file systems, modern big data platforms can thrive on a vSphere platform. This solution can be extended to include other memory-resident applications that require a distributed file system for data sharing. The results clearly show that both an on-premises private cloud based on a vSphere platform and the public cloud infrastructure hosted on VMware Cloud on AWS are well suited to run next-generation big data workloads.

References

1. Justin Murray. VMware. "Big Data Video – Benefits of Virtualizing Big Data on vSphere." (2017)
2. Giacinto Donvito et al. IOP Publishing. *Journal of Physics: Conference Series* 513 042014. "Testing of Several Distributed File-Systems (HDFS, Ceph and GlusterFS) for Supporting the HEP Experiments Analysis." (2014)
3. *Spark SQL, DataFrames and Datasets Guide*. (2017)
4. *Getting Started with GlusterFS: Architecture*. (2017)
5. Shubhendu Tripathi. Red Hat. "GlusterFS and Hadoop." (2015)
6. Apache Spark. (2017)
7. TPC-DS. (2017)
8. VMware. *VMware Cloud on AWS Technical Overview*. (2017)

About the Authors

Mohan Potheri is a senior solutions architect at VMware. His focus is on virtualizing business-critical applications.

Justin Murray is a senior technical marketing architect at VMware. His focus is on virtualizing big data.

Acknowledgments

The authors thank Dave Jaffe, staff engineer in performance engineering at VMware; Don Sullivan, product line marketing manager at VMware; and Charu Chaubal, director of technical marketing at VMware, for their help in discussing the solution and for their review of this paper.

Appendix A: GlusterFS Install and Setup Commands

Install Commands

```
yum install centos-release-gluster310
yum install glusterfs gluster-cli glusterfs-libs glusterfs-server
```

Enabling the Service

```
systemctl enable glusterd.service
systemctl start glusterd.service
```

Creating and Configuring Gluster Nodes

Clone the CentOS machine with Gluster components to create three Gluster nodes, gluster01-03.

Name Resolution

Create an `/etc/hosts` file containing the name and IP addresses of all GlusterFS hosts and Spark master and worker nodes. An example follows.

```
cat /etc/hosts
192.168.81.200 spark200.sfo01.rainpole.local spark200 sparkmaster
192.168.81.201 spark201.sfo01.rainpole.local spark201
192.168.81.202 spark202.sfo01.rainpole.local spark202
192.168.81.203 spark203.sfo01.rainpole.local spark203
192.168.81.204 spark204.sfo01.rainpole.local spark204
192.168.81.205 spark205.sfo01.rainpole.local spark205
192.168.81.206 spark206.sfo01.rainpole.local spark206
192.168.81.207 spark207.sfo01.rainpole.local spark207
192.168.81.208 spark208.sfo01.rainpole.local spark208
192.168.81.221 gluster01.sfo01.rainpole.local gluster01
192.168.81.222 gluster02.sfo01.rainpole.local gluster02
192.168.81.223 gluster03.sfo01.rainpole.local gluster03
```

From the shell of the first node, gluster01, peer with the other two nodes via the following:

```
gluster peer probe gluster02
gluster peer probe gluster03
```

Check the peering status. The following is an example of a normally operating peering status:

```
[root@gluster01 ~]# gluster peer status
Number of Peers: 2

Hostname: gluster02
Uuid: 87993141-5f0d-406a-b388-fc6c5412df33
State: Peer in Cluster (Connected)

Hostname: gluster03
Uuid: 62311b71-1905-4363-9979-054fa7339814
State: Peer in Cluster (Connected)
```

Create a directory structure for bricks to be used for Gluster volumes in each of the nodes:

```
for i in 'gluster01 gluster02 gluster03'
do
ssh $i mkdir -p /glusterdata/brick1
done
```

Create a Gluster volume for use as a distributed file system, with three copies of the data across three nodes stored under the brick directory:

```
gluster volume create sparkvol1 replica 3 transport tcp \
gluster01:/glusterdata/brick1 gluster02:/glusterdata/brick1 gluster03:/glusterdata/
brick1
```

Check the status of the volume by typing the following command:

```
[root@gluster01 ~]# gluster volume status
```

Status of volume: sparkvol1

GLUSTER PROCESS	TCP PORT	RDMA PORT	ONLINE	PID
Brick gluster01:/glusterdata/brick1	49152	0	Y	9049
Brick gluster02:/glusterdata/brick1	49152	0	Y	11977
Brick gluster03:/glusterdata/brick1	49152	0	Y	3519
Self-heal Daemon on localhost	N/A	N/A	Y	9069
Self-heal Daemon on gluster03	N/A	N/A	Y	3539
Self-heal Daemon on gluster02	N/A	N/A	Y	11997

Task Status of Volume sparkvol1

The volume is now ready for use. Mount the volume via **fstab** on all Spark nodes, with an entry such as the following one. The mount uses the native GlusterFS protocol, which has the best performance. To make this mount option available, the Spark nodes should have the GlusterFS packages.

```
gluster01:/sparkvol1    /sparkdata    glusterfs    defaults,_netdev 0 0
```

The **df** command shows the mounted file system as in the following:

```
gluster01:/sparkvol1 2146435072 315739392 1830695680 15% /sparkdata
```

