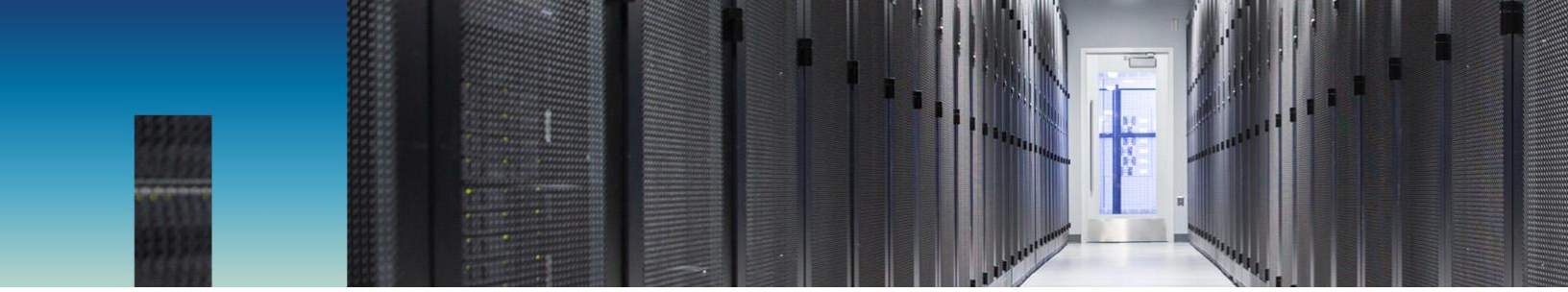


Technical Report and Best Practices Guide

NetApp SolidFire and Cassandra

Stephen Carl, NetApp
October 2017 | TR-4635

Abstract

This technical report offers guidelines with detailed information to help make sure of a stable, high-performance NetApp® SolidFire® SAN and Cassandra NoSQL database. SolidFire storage offers a compelling advantage for a wide range of database application use cases. This paper gives the Cassandra database administrator information about SolidFire storage. It describes important system design paradigms to consider when using SolidFire storage for database applications such as Cassandra. From these design points, the reader learns about application profiles that are best suited to SolidFire storage and how to identify those types of applications. Best practices for configuring SolidFire and Cassandra environments are explained, and the document concludes with descriptions of the test environment and use cases.

NetApp®

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 Introduction

Architecting your next-generation data center (NGDC) requires different tools than those that most businesses use today in their data centers. NGDCs require more flexibility to meet customer and business needs, have different uptime requirements, and have data protection expectations that require tools to match the changing environment.

SolidFire provides the flexibility for an NGDC to meet unplanned demands. Cassandra and NoSQL databases with the scale, availability, and elasticity demands of applications running on NetApp SolidFire provide a data management solution to protect data without affecting application performance.

1.1 NetApp SolidFire

NetApp SolidFire delivers an unstoppable force for data center transformation. Born out of some of the largest cloud infrastructures in the world, SolidFire is built to serve next-generation data center needs such as scaling with multitenancy, set-and-forget management, and guaranteed performance. The SolidFire quality of service (QoS) feature means that all tenants have guaranteed configured performance, which can be dynamically changed as needed. Enterprises are adopting NetApp SolidFire because they demand greater predictability from their shared storage infrastructure. Database administrators benefit from the ability to deploy applications more quickly, guarantee application performance at the volume level, and reduce operational and capital expenses.

1.2 Cassandra

NoSQL databases, including wide column store types such as Apache Cassandra, are providing companies with capabilities for faster data insights in real time, enabling quick response to customer data-related needs and requirements. The scale-out architecture of Apache Cassandra allows customers to easily scale their data size depending on application growth. Organizations are choosing the power and flexibility offered by Cassandra's key-value datastores, and the adoption rate is increasing steadily. For a database deployment, platforms backed by SolidFire offer high-performing block storage for use in the persistent data layer of any application.

2 NetApp SolidFire

2.1 Storage Efficiencies

Thin Provisioning

SolidFire uses 4k granular thin provisioning that does not require any reserve space, increasing effective capacity by immediately consuming less space. This feature increases efficiency and reduces overhead by using the smallest allocation possible while maintaining alignment with the native 4KB allocation format used by modern operating systems.

SolidFire volumes don't use any reserve space. Therefore, it is practical to deploy a volume capacity for the estimated maximum size of the database and to purchase only enough physical hardware to support the actual space consumed by the database. As database space consumption approaches the physical limits of the cluster, additional nodes can be dynamically added to the cluster to increase its physical capacity. This process is completely transparent to applications and imposes no need for downtime or reconfiguration of the operating system or the database.

Furthermore, SolidFire Double Helix replication automatically redistributes existing data over the added nodes to create ideal load balancing of both existing and new data. With this deployment paradigm, logical storage capacity can be configured once for the lifetime of the supported databases rather than using incremental updates to accommodate the needs of the database.

Compression and Deduplication

Each SolidFire node includes a PCIe NVRAM card that serves as a write cache. When a host sends writes, they are divided into 4KB data blocks, which are immediately hashed, compressed, and stored in the NVRAM of the storage nodes before an acknowledgement is returned. The resulting value serves as a block ID that determines block placement, which is randomly distributed across all nodes to create an even load distribution.

The SolidFire deduplication block service identifies blocks that have previously been written based on the block ID. If a block already exists, metadata is updated, and the duplicate is discarded. The process is inline and global to the storage cluster.

The combination of inline compression and deduplication has the following advantages:

- Reduces repetitive writes to media, increasing the life of the drives
- Increases system performance by minimizing system resources
- Evenly distributes capacity and performance loads across the system, eliminating hot spots

One key to SolidFire technology is its ability to minimize writes by doing the compression in memory before the actual writes to disk. The writes from the host are divided into 4KB data blocks, which are immediately hashed and compressed into the node's NVRAM write cache. Each compressed block is synchronously replicated to one or more additional storage nodes for data protection. Any future writes coming to the nodes are compared against the hash value and are discarded if they are already present.

2.2 Multitenancy

Quality of Service (QoS) Control

NetApp SolidFire all-flash arrays present performance and capacity as dynamic independent pools. This feature enables administrators to set the performance requirements for all the databases or tenants hosted on the same cluster. The minimum, maximum, and burst control settings in the QoS policy guarantee the required performance and can be dynamically changed any time. If SolidFire hardware resources are pushed close to their physical limits, IT staff can dynamically and seamlessly add more SolidFire nodes to the existing cluster. The Double Helix data distribution automatically redistributes data for optimal load balancing over all hardware resources. This process is transparent to upstream applications.

VLAN Tagging

Workload consolidation is another major effort going on in many data centers. This means that multiple applications and potentially different customers or divisions operate on the same equipment within the data center. VLAN tagging is a feature that enhances the security in the solution, separates network traffic, and provides consistency in how a network is configured. VLAN tagging is available in all SolidFire all-flash arrays.

2.3 Scale

In the modern data center, a constant struggle exists between applications and storage. Applications are very sensitive and require adequate storage performance and capacity to operate. Any imbalance can negatively affect an application's ability to perform, causing a ripple effect to the business. Flash solves many of the performance problems, but due to the significant performance available in an all-flash array, most systems run out of capacity before performance.

As a new node is introduced to a cluster, its performance and capacity (block and metadata) are added to the collective pool of the SolidFire cluster. When block drives are added into the system, SolidFire evenly distributes the existing block data to the newly added block capacity. This results in the newly added node having an equal share of block data compared to each other node in the system.

In a database environment, you are not likely to need to add an exactly equal amount of performance or capacity, and it's never a guarantee that you need to add the same level of capacity and performance that you have been purchasing. Therefore, it is essential to choose an architecture that allows you to pick a ratio of performance and capacity that best fits your needs. SolidFire is designed to provide what you need when you need it by delivering the following:

- **Nondisruptive scale-out/scale-in.** Add or remove nodes to a SolidFire cluster without disrupting service or compromising volume-level QoS settings. Data is automatically redistributed in the background across all nodes in the cluster, maintaining perfect balance as the system grows.
- **Instant resource availability.** Newly added storage capacity and performance resources are instantly available to each volume within the system, eliminating the need to reallocate volumes over new drives.
- **Simplified capacity planning.** Initial implementations begin as small as a 4-node/4U cluster configuration and scale out easily in 1U node increments, allowing performance and capacity resources to be added as needs dictate. Eliminate multiyear capacity and performance projections and scale on demand.
- **Seamless generational upgrades.** New nodes with more capacity and performance are simply added to the established cluster, while old nodes are removed, retired, or repurposed. No rebalancing, restriping, or volume reallocation is required. And all QoS settings remain enforced.

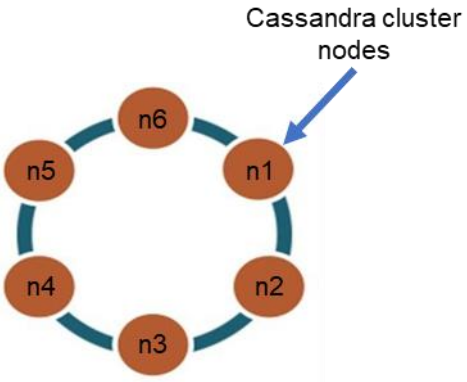
3 Cassandra DataStax Enterprise

3.1 Cassandra Architecture Overview

Cassandra is designed to handle big data workloads across multiple nodes with no single point of failure. Its architecture is based on the understanding that system and hardware failures can and do occur. Cassandra addresses the problem of failures by employing a peer-to-peer distributed system across homogeneous nodes where data is distributed among all nodes in the cluster. Each node frequently exchanges state information about itself and other nodes across the cluster using a peer-to-peer communication protocol.

Cassandra's architecture is responsible for its ability to scale, perform, and offer continuous uptime. Rather than using a legacy master-slave or a manual and difficult-to-maintain sharded design, Cassandra has a masterless "ring" architecture that is elegant, easy to set up, and easy to maintain, as shown in Figure 1. All nodes in the cluster play an identical role and communicate using a distributed, scalable protocol called "gossip." This distributed environment has no single point of failure and therefore can offer true continuous availability and uptime.

Figure 1) Cassandra ring architecture.

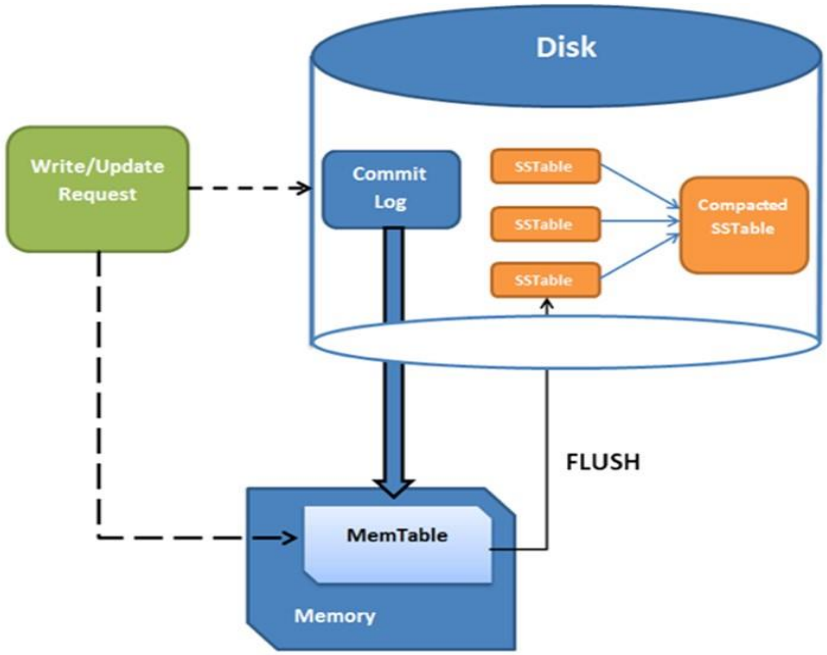


Cassandra replicates data to multiple nodes in a cluster, which enables reliability, continuous availability, and fast I/O operations. The number of data copies that are replicated is referred to as the replication factor. A replication factor of 1 means that there is only one copy of each row in a cluster; a replication factor of 3 means three copies of the data are stored across the cluster.

After a keyspace and its replication have been created, Cassandra automatically maintains that replication even when nodes are removed, are added, or fail.

Each Cassandra cluster node uses a sequentially written commit log to capture write activity to make sure of data availability. Data is distributed and indexed in an in-memory structure called a memtable, which resembles a writeback cache. When the memtable is full, the data is written to an immutable file on disk called an SSTable. Buffering writes in memory allows writes to always be a fully sequential operation of many megabytes of disk I/O happening simultaneously. This approach enables high write performance and durability. The write path diagram is shown in Figure 2.

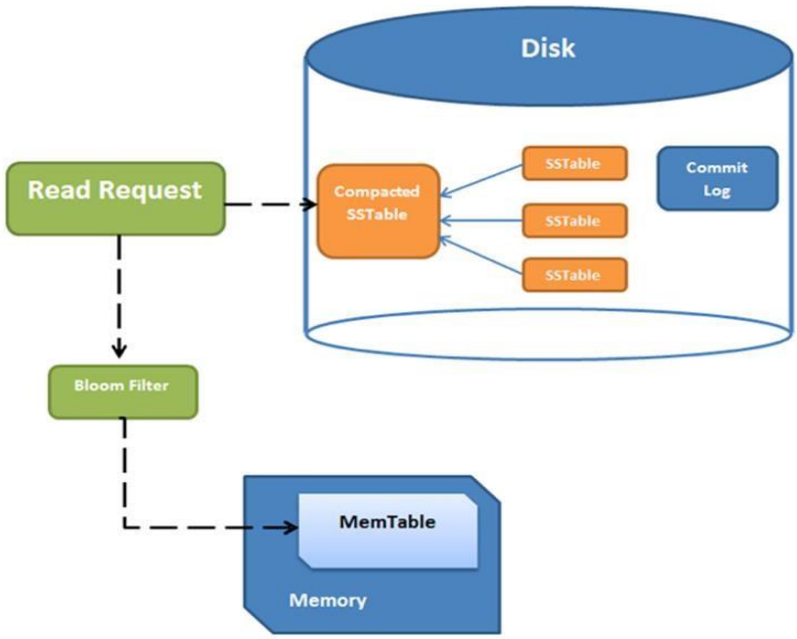
Figure 2) Cassandra write path.



More than one SSTable can exist for a single Cassandra logical data table. A process named compaction occurs periodically, combining multiple SSTables into one for faster read access.

For a read request, Cassandra consults an in-memory data structure called a Bloom filter that checks the probability of an SSTable having the needed data. The Bloom filter can tell very quickly whether the file probably has the needed data or certainly does not have it. If the answer is a tentative yes, Cassandra consults another layer of in-memory caches, then fetches the compressed data on disk. If the answer is no, Cassandra doesn't trouble with reading that SSTable at all and moves on to the next. The read path diagram is shown in Figure 3.

Figure 3) Cassandra read path.



For more information about Cassandra capabilities, seen this brief introduction.

DB Compaction Strategies and Impact to Disk Space and Performance

An operational storage requirement is the need to keep sufficient disk space free and available for Cassandra's compaction. Compaction is a mechanism used to improve data locality and flush stale data from the system. The default compaction strategy is size tiered, which temporarily might require up to double the size of the table it is compacting. Because multiple tables might be compacted simultaneously, it is prudent to keep free space in excess of your live data volume to make sure of optimal performance.

Leveled compaction strategy (LCS) dramatically reduces the temporary disk space requirements, while providing tighter bounds on the latency of read operations. The tradeoff is that LCS generally produces twice as many write operations over time. LCS is generally not recommended for write-heavy workloads for this reason.

Size-tiered compaction strategy (STCS):

- Recommended for write-intense workloads
- Compaction done by merging similar-sized SSTables into one large SSTable
- Requires 50% more disk space than the data size LCS
- Recommended for read-intensive workloads
- Requires about 10% more disk space in addition to the space occupied by the data

LCS is significantly more I/O intensive than other compaction strategies; as a result, it might introduce additional latency for reads.

Date-tiered compaction strategy (DTCS):

- Recommended for time series data
- Compaction done based on SSTable age/time stamp
- Requires 50% more disk space than the data size

Sizing Calculations

```
Database volume size = (total_data_size + replication_overhead + compaction_overhead)
total_data_size = (column_overhead + row_overhead + primary_key_index) column_overhead =
regular_total_column_size + counter_and_expiring_total_column_size
```

Every column in Cassandra incurs 15 bytes of overhead. For counter columns and expiring columns, an additional 8 bytes of storage overhead need to be added (that is, 15 + 8 = 23 bytes of overhead).

Calculate the column overhead from the user-provided column name size and value size input values in the following manner:

```
regular_total_column_size = column_name_size + column_value_size + 15 bytes
counter_and_expiring_total_column_size = column_name_size + column_value_size + 23 bytes
```

Every row in Cassandra incurs 23 bytes of overhead.

```
row_overhead = number of rows x 23 bytes
```

Every row of the primary key index incurs 32 bytes of overhead.

```
primary_key_index = number of rows x (32 bytes + average_key_size)
```

For a replication factor of 1, there is no overhead for replicas because only one copy of data is stored in the cluster. Therefore, if the replication factor is greater than 1:

```
replication_overhead = total_data_size * ( replication_factor - 1 )
```

From the earlier discussion about compaction strategy and the overhead each requires, we have the following options:

If STCS or DTCS is used:

```
compaction overhead = (0.5) * total_data_size
```

If LCS is used:

```
compaction overhead = (0.1) * total_data_size
```

Now that we know the database volume size, we can create a Dynamic Disk Pool such that:

```
the number of database volumes = the number of nodes sharing the array
Capacity = number of database volumes x database volume size
```

We also need to consider the size of the commit log.

- The commit log is used to periodically back up in-memory SSTables to storage volumes. For 32-bit JVM, Java heap size is 32MB. For 64-bit JVM, Java heap size is 8GB.
- The default size for the commit log is governed by `commitlog_total_space_in_mb` in the `Cassandra.yaml` configuration file.
- The storage for the commit log must be on a disk group separate from the database volumes.

Only one commit log volume is needed per DB instance.

```
Commit log size = Cassandra JVM heap size + 25%
```

4 Application Considerations

4.1 Database Consolidation

SolidFire is an optimal storage system well suited for database consolidation. SolidFire per-volume QoS controls mean that individual databases get the I/O throughput they need without being affected by other databases running in parallel on the same storage system. With QoS and data reduction efficiencies, you can get significantly higher database density from a shared storage infrastructure. The use case of deploying hundreds of individual databases is an excellent fit for SolidFire.

With SolidFire, a single LUN can be used to provision all the data files and achieve the required performance, instead of spreading data files across multiple volumes, controllers, and arrays. SolidFire QoS guarantees performance for each database and can eliminate the need to implement complex logical volume management configurations to consolidate multiple LUNs or volumes to meet the performance needs of your business.

4.2 Data Protection and Disaster Recovery

SolidFire Double Helix data protection is a shared-nothing, distributed replication algorithm that spreads two redundant copies of data across all drives within the entire cluster. The architecture means no single point of failure across the solution and, when combined with storage efficiency and QoS, provides a compelling disaster recovery (DR) solution. This solution allows the same storage resources to be shared by DR and dev/test without any risk of a performance penalty.

4.3 Dev/Test

NetApp Storage Snapshot™ copies provide a point-in-time view of the contents of an active file system or storage volume. Snapshot copies are used for rapid recovery of datasets that might be damaged or corrupted, as well as to create space-efficient copies of datasets for development, test, and related uses through the deduplication feature. The cloning process can be coupled with SolidFire QoS control so that database clones can coexist with production copies without performance effects on upstream applications.

The copy volume feature of NetApp SolidFire allows admins to refresh an existing clone copy of a database without any file system remount operations. In this use case, you frequently refresh a copy of the database by only taking changes from the production copy.

5 NetApp SolidFire Configuration

A 4-node SolidFire all-flash storage system was configured as the storage for the Cassandra. For this testing environment, a single SolidFire 4-node cluster was deployed for the Cassandra cluster nodes and database storage volumes. Separate SolidFire volumes were dedicated for the Cassandra commit logs for additional performance benefits as per Cassandra recommendations. In this test, SolidFire SF2405 nodes were used.

SolidFire has features for thin provisioning, always-on deduplication, compression, scalability, database consolidation, and QoS control, allowing flexibility for solutions, including Cassandra databases. One of the most beneficial capabilities of NetApp SolidFire for a Cassandra database deployment is the ability to control IOPS per database volume for each Cassandra cluster node. This allows adjustment of the IOPS to a greater throughput if necessary, avoiding bottlenecks of performance and cluster compaction. SolidFire can also provision additional storage capacity without disruptions. Coupled with the SolidFire point-in-time capabilities, the combined features make a compelling solution.

6 Operating System Configuration

The guidelines in this document apply to Linux distributions of the DataStax Enterprise Cassandra edition. Ubuntu version 16.04 LTS was used in this document's testing. Alternate distributions can be used if they have full compatibility with the DataStax Enterprise software.

6.1 Update Kernel Parameters

Update the kernel parameters for your host operating system to the following values:

```
vm.dirty_ratio = 15
vm.dirty_background_ratio = 5
vm.swappiness = 1
net.core.somaxconn = 4096
net.ipv4.tcp_fin_timeout = 30
net.ipv4.tcp_keepalive_intvl = 30
net.ipv4.tcp_keepalive_time = 120
net.ipv4.tcp_max_syn_backlog = 4096
```

6.2 Optimize Network Performance

Consider the following guidelines for optimal network performance:

- Enable jumbo frames for all host network interfaces.
- To isolate the data traffic, configure the interface that is used for the Cassandra data traffic with a different subnet from the public network.

6.3 Optimize iSCSI Parameters

The Linux iSCSI initiator configuration works with NetApp SolidFire volumes in its default configuration. To maximize system throughput, increase the number of sessions per target (`nr_sessions`) from the default of `1` to `8`.

1. Make the following changes to the iSCSI daemon in the `/etc/iscsi/iscsid.conf` file:

```
iscsid.startup = /etc/rc.d/init.d/iscsid force-start
node.startup = automatic
node.leading_login = No
node.session.timeo.replacement_timeout = 120
node.conn[0].timeo.login_timeout = 15
node.conn[0].timeo.logout_timeout = 15
node.conn[0].timeo.noop_out_interval = 5
node.conn[0].timeo.noop_out_timeout = 5
node.session.err_timeo.abort_timeout = 15
node.session.err_timeo.lu_reset_timeout = 30
node.session.err_timeo.tgt_reset_timeout = 30
node.session.initial_login_retry_max = 8
node.session.cmds_max = 128
node.session.queue_depth = 32
node.session.xmit_thread_priority = -20
node.session.iscsi.InitialR2T = No
node.session.iscsi.ImmediateData = Yes
node.session.iscsi.FirstBurstLength = 262144
node.session.iscsi.MaxBurstLength = 16776192
node.conn[0].iscsi.MaxRecvDataSegmentLength = 262144
node.conn[0].iscsi.MaxXmitDataSegmentLength = 0
discovery.sendtargets.iscsi.MaxRecvDataSegmentLength = 32768
node.conn[0].iscsi.HeaderDigest = None
node.session.iscsi.FastAbort = Yes
node.startup = automatic
node.session.nr_sessions = 8
```

2. Make discovery of iSCSI devices persistent over reboots.

```
chkconfig iscsid
```

3. To rescan the new storage volumes, run the following commands:

```
iscsiadm -m discovery -t sendtargets -p <SolidFire SVIP> --op update –n node.session.nr_sessions
–v 2
iscsiadm -m node -L all
```

6.4 Tune the I/O Scheduler

Run the following commands to tune the Linux operating system to take advantage of the performance characteristics of the SolidFire storage system (`<devpath>` is the device name).

```
echo 0 > /sys/<devpath>/queue/rotational
echo noop > /sys/<devpath>/queue/scheduler
echo 128 > /sys/<devpath>/queue/nr_requests
echo 2 > /sys/<devpath>/queue/rq_affinity
echo 0 > /sys/<devpath>/queue/add_random
```

To persist reboots, create file /lib/udev/rules.d/99-solidfire.rules and these entries:

```
ACTION=="add", \
SUBSYSTEMS=="scsi", \
KERNEL=="sd*", \
ATTRS{vendor}=="SolidFir", \
ATTR{queue/scheduler}="noop", \
ATTR{queue/add_random}="0", \
ATTR{queue/rq_affinity}="2", \
ATTR{queue/nr_requests}="1024", \
ATTR{queue/max_sectors_kb}="2048"
```

6.5 Configure the Multipath Driver

Install and configure the Linux multipath driver (`multipathd`) by making the following changes to the `/etc/multipath.conf` file.

```
defaults {
        user_friendly_names yes
}

devices {
        device {
               vendor "SolidFir"
               product "SSD SAN"
               path_grouping_policy multibus
               path_checker tur
               hardware_handler  "0"
               failback immediate
               rr_weight uniform
               rr_min_io 10
               rr_min_io_rq 10
               features "0"
               no_path_retry 24
               prio const
               }
         }
```

Optionally, you can enable persistent mapping of `/dev/mapper` entries by associating the NetApp SolidFire storage system device's worldwide identifier (WWID) with a specific operating system alias. For this option, make the following additions to the `/etc/multipath.conf` file:

```
multipaths {

multipath {
wwid 36f47acc100000000707a646c000003b1
alias mongo-rs0
}
}
```

6.6 Enable Multipathing

For Ubuntu distributions, install multipath tools:

```
sudo apt-get install multipath-tools
```

6.7 Create SolidFire Volume

1. Create a disk device partition by using the multipath device `mpatha` as shown in the Configuring SolidFire for Linux guide.

```
root@nosql-ubuntu-6:~# ls -al /dev/mapper/mpath*
lrwxrwxrwx 1 root root 7 Aug 25 12:31 /dev/mapper/mpatha -> ../dm-2
lrwxrwxrwx 1 root root 7 Aug 25 12:31 /dev/mapper/mpatha-part1 -> ../dm-3
```

2. Create the file system.

```
mkfs.xfs -K /dev/dm-3
```

3. Update the `/etc/fstab` to mount the file system and persist through host reboots.

```
UUID="686277b8-b711-4fc8-b021-3e6430f4a358" /data/db xfs     noatime,discard,nobarrier,_netdev
0       0
```

7 Cassandra Configuration

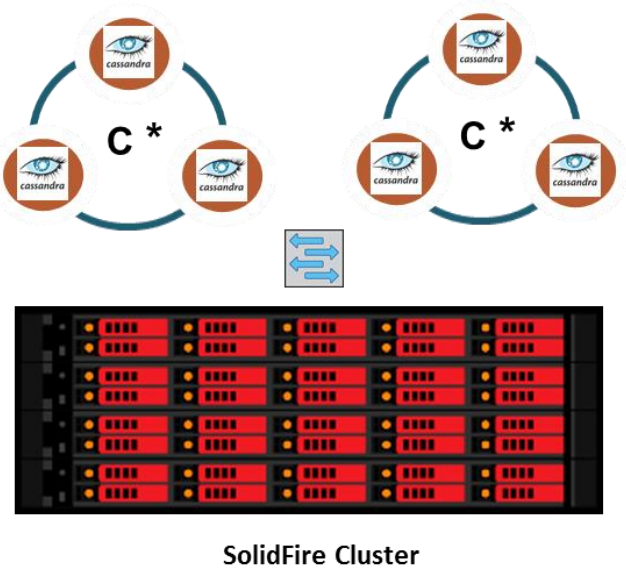
7.1 Installing DataStax Cassandra Enterprise

An Ubuntu Linux distribution was used for the Cassandra nodes. DataStax Cassandra version 5.1.2 was installed on each node per product guide recommendations. Before you begin your installation process, check the DataStax documentation release notes.

For the testing in this document, two Cassandra sharded clusters were deployed. The clusters were configured with a replication factor of 3, which is a common setup for Cassandra cluster deployments.

Each Cassandra server was a VMware virtual machine with the NetApp SolidFire volumes created for the database storage in the 4-node SolidFire cluster. An additional SolidFire volume was created for the Cassandra commit logs. For more information about the Cassandra configuration, see Figure 4). For information about the Cassandra node server specifications, see Table 1.

Figure 4) Cassandra cluster configuration.



In the tested configuration, two Cassandra clusters used a 3-node topology, as shown in Figure 4. Cassandra deployment replicates multiple copies in the cluster using a network topology strategy during keyspace creation. For more information about replication in DataStax Cassandra, see this document.

Installation of the DataStax Enterprise version on an Ubuntu Linux operating system was used from this installation documentation. For each iteration of a new set of database tests, the DataStax Cassandra software was uninstalled, and all data in the NetApp SolidFire volumes was deleted, followed by a new installation and empty volumes for new database data.

DataStax Cassandra settings for the test configuration followed this production notes documentation.

Having Oracle Java version 8 installed on the operating system is recommended for DataStax Enterprise. Instructions are in these Java installation instructions.

8 Test Environment

The configuration depicted in Figure 5 shows the logical design of the test solution deployment.

The Cassandra clusters use SolidFire volumes for the database storage and commit logs. The commit log is a special capped collection that keeps a rolling record of all operations that modify the data stored in the databases. Cassandra applies database operations to the clustered nodes and records them to the commit logs on the nodes and then to a per–column family structure called a memtable. When a memtable is full, it is written to disk as an SSTable to the SolidFire volumes. The commit log is synced per a timed parameter in milliseconds for data durability. This enables all the Cassandra nodes to have a consistent view of all data in the cluster.

Each Cassandra cluster has three VMware virtual machines loaded with the Ubuntu Linux operating system and a recent distribution version of DataStax Enterprise Cassandra.

Figure 5) Logical test configuration.

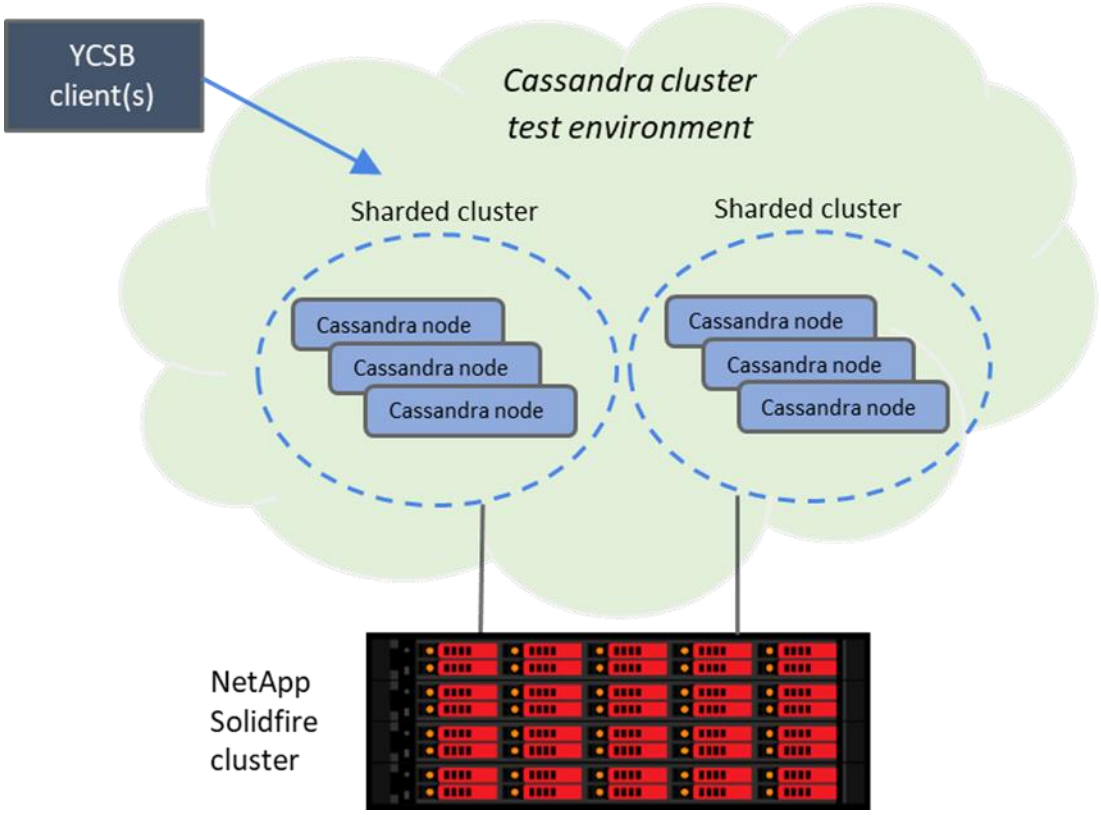


Table 1) Cassandra nodes for three clusters.

Cassandra Cluster	Qty	DataStax Cassandra Version	Type	Cores/ vCPUs	RAM	OS
Cluster 1	3	5.1.2	VMware	8	32GB	Ubuntu 16.04
Cluster 2	3	5.1.2	VMware	8	32GB	Ubuntu 16.04

All Cassandra data nodes were configured on a SolidFire SF2405 4-node cluster running NetApp SolidFire Element® OS version 9.0.0.1549.

Table 2) YCSB client server.

YCSB Client	Qty	YCSB Version	Type	Cores/ vCPUs	RAM	OS
YCSB	2	0.12.0	VMware	8	32GB	Ubuntu 16.04

9 Cassandra Performance Validation

The SolidFire cluster provides the storage required for the Cassandra sharded 3-node cluster. Cassandra keyspaces and tables can be recovered directly back into the same Cassandra database cluster (operational recovery). They can also be recovered to a different Cassandra database cluster (testing and development refresh) with a different topology (the number of nodes on the destination cluster differs from the node count of the source cluster).

Database performance is defined by the speed at which a database computes basic CRUD operations. A basic operation is an action performed by the workload executor that drives multiple client threads. Each thread executes a sequential series of operations by making calls to the database interface layer, both to load the database (the load phase) and to execute the workload (the transaction phase). The threads throttle the rate at which they generate requests, so that we may directly control the offered load against the database. In addition, the threads measure the latency and achieved throughput of their operations and report these measurements to the statistics module. A very common use case for applications deploying Cassandra is in environments with heavy write operations having 100% of inserts (writes only).

In the test environment, Casandra cluster performance was measured using the Yahoo Server Cloud Benchmarking (YCSB) tool. A test is defined by the following choices.

YCSB was configured during all tests with the focus on the four following workloads to documents within the database:

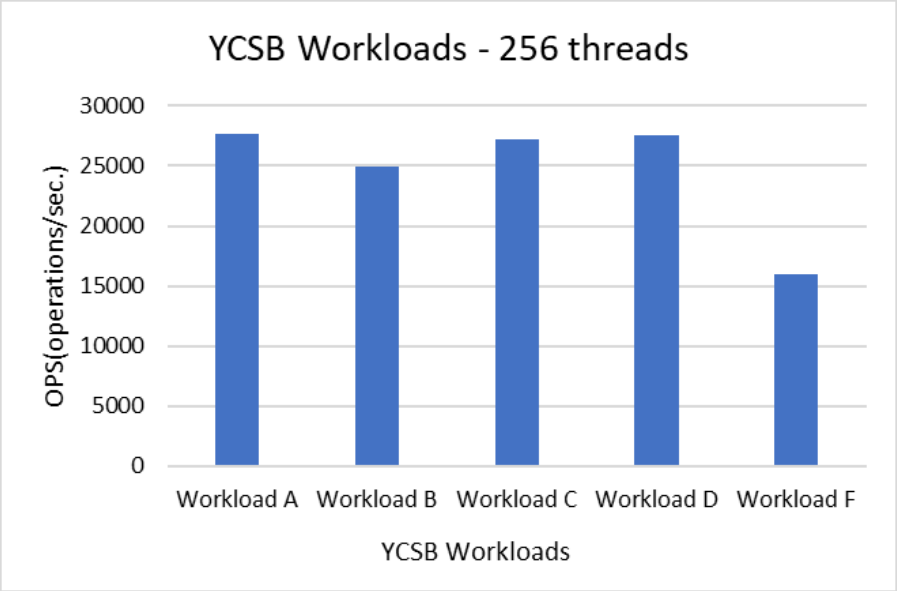
- 100% inserts (workload A; data load of the database)
- 100% reads (workload C)
- 50% read, 50% updates (workload A)
- 95% reads, 5% updates (workload B)

A replication factor of three was configured for each Cassandra cluster to enable three copies of each data record/document within all three Cassandra nodes per cluster.

The YCSB tool run from a single client loaded the data into each Cassandra cluster database. A minimum of 32 and a maximum of 256 client threads were tested, measuring the latency and throughput for the 3-node Cassandra clusters. The following figures show the YCSB performance output for the NetApp SolidFire and Cassandra clustered test environment.

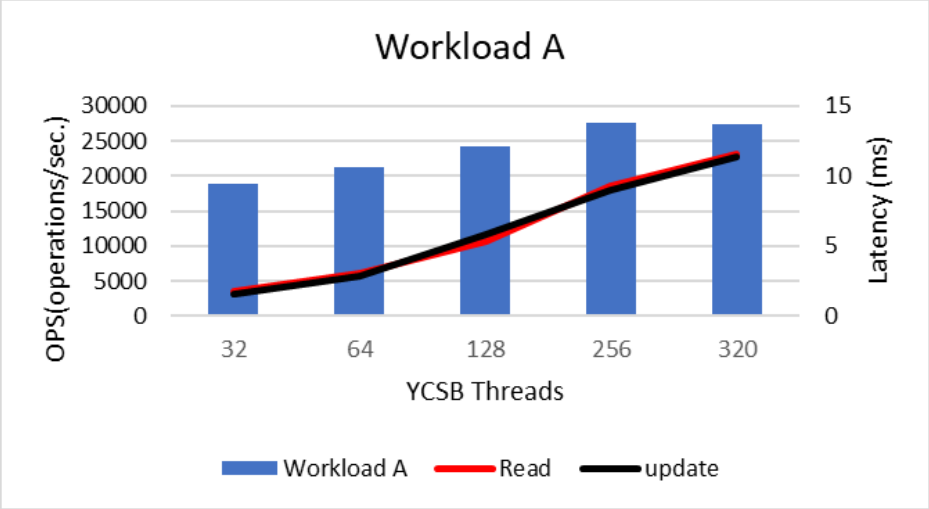
In Figure 6, workloads A, B, C, D, and F show the throughput and latency from the YCSB tool.

Figure 6) YCSB workloads 3-node cluster.



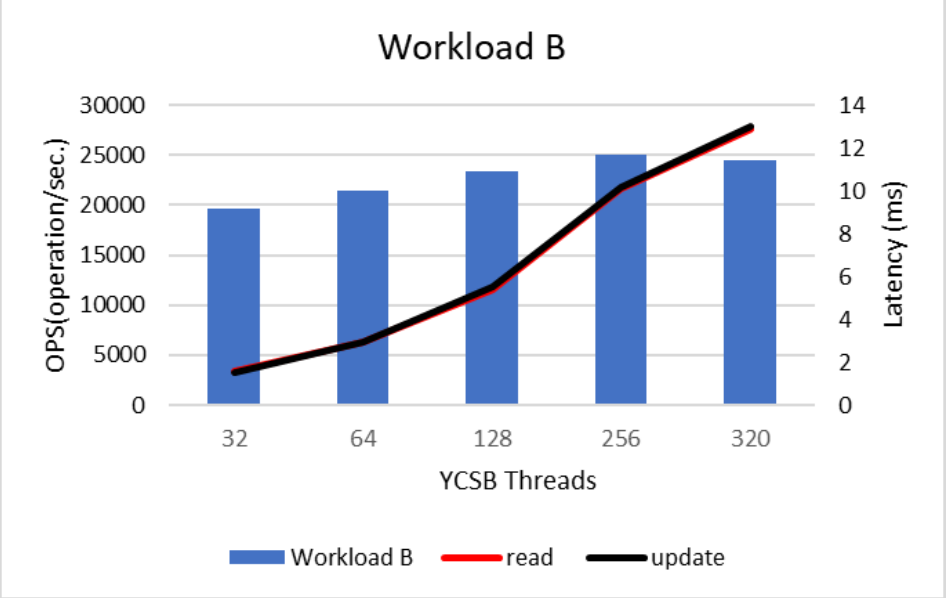
In Figure 7, workload A shows the throughput and latencies for various YCSB client thread counts.

Figure 7) YCSB workload A.



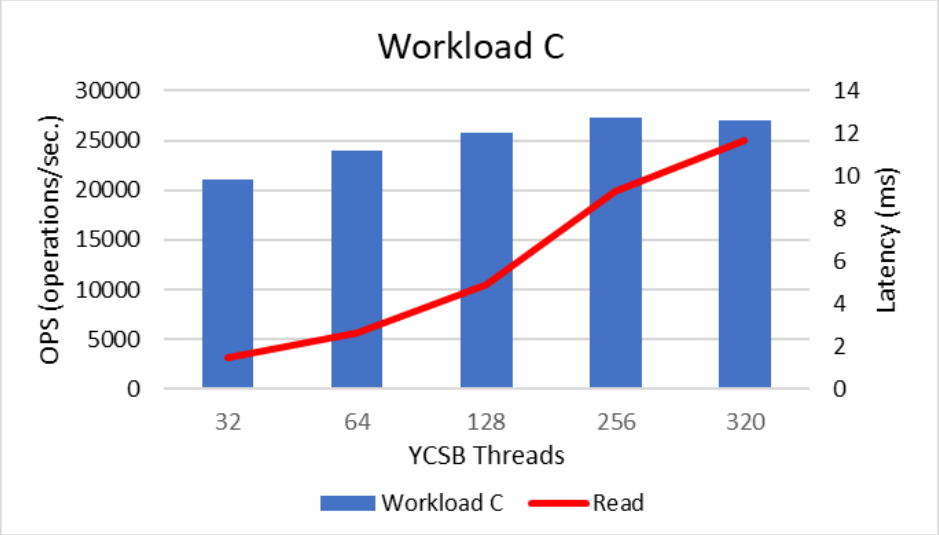
In Figure 8, workload B throughput and latencies are shown.

Figure 8) YCSB workload B.



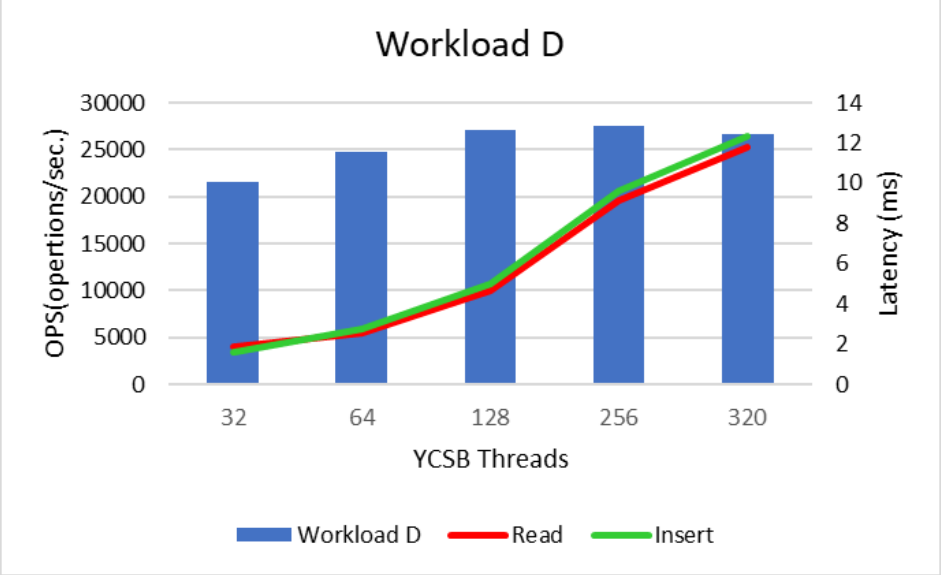
In Figure 9, workload C throughput and latencies are shown.

Figure 9) YCSB workload C.



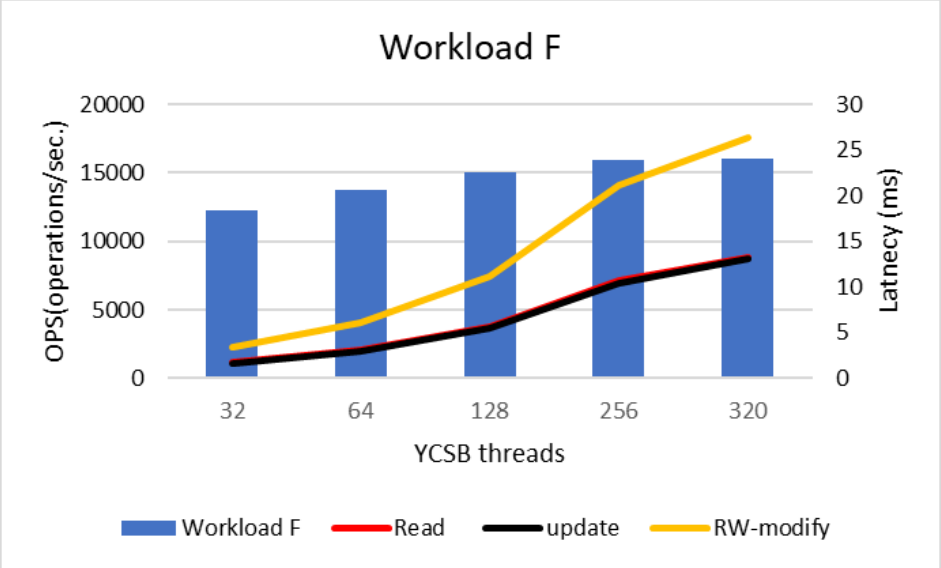
In Figure 10, workload D throughput and latencies are shown.

Figure 10) YCSB workload D.



In Figure 11, workload F throughput and latencies are shown.

Figure 11) YCSB workload F.



Test Results

As shown in the figures, Cassandra sharded clusters running on NetApp SolidFire provide high throughput of database operations with low latency. The performance combined with the volume backup and cloning capabilities provide administrators a compelling method to recover and scale existing clusters or deploy new database instances for multiple use cases. Additional QoS granular control on volumes can enable performance guarantees for database or mixed workloads. This allows high utilization of NetApp SolidFire resources for storage nodes, preventing unnecessary scaling.

Summary

NetApp SolidFire provides high performance and availability of Cassandra data, as seen in the test results using YCSB client testing simulation of real customer workloads.

NetApp SolidFire offers a compelling customer solution for multiple use case workloads in a clustered Cassandra environment. SolidFire scalability is well suited to dynamically provide additional capacity and/or performance as required for the Cassandra cluster environment for both a current deployment and when the cluster needs to scale out to meet additional use case requirements. The QoS feature allows adjustment of the IOPS to greater throughput if necessary, avoiding bottlenecks of performance and cluster problems for replica set and sharded clusters. SolidFire can also provision additional storage capacity without disruptions. Coupled with point-in-time Snapshot and cloning capabilities, the combined features make a compelling solution.

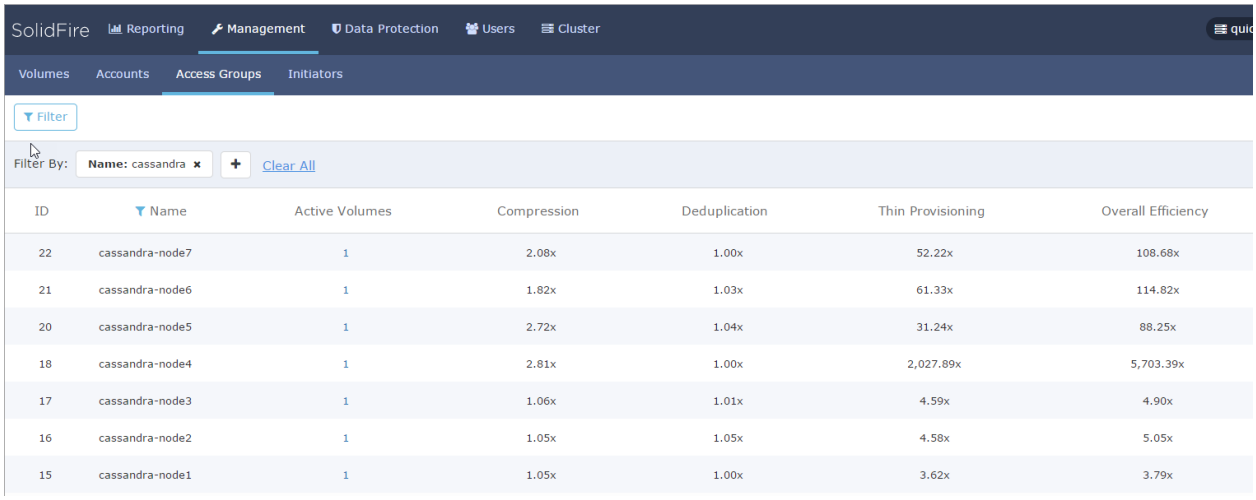
NetApp SolidFire provides simple centralized management with the SolidFire UI or the vCenter plug-in, giving you full control of managing your entire infrastructure through an intuitive user interface. A robust suite of APIs enables additional seamless integration into higher-level management, orchestration, backup, and disaster recovery tools.

Appendix: Test Configuration Information

SolidFire Configuration

Figure 12, the SolidFire Element GUI screen, shows the access groups configured for the Cassandra data nodes. Each access group has one initiator to a host and one active volume.

Figure 12) SolidFire access groups for Cassandra nodes.



ID	Name	Active Volumes	Compression	Deduplication	Thin Provisioning	Overall Efficiency
22	cassandra-node7	1	2.08x	1.00x	52.22x	108.68x
21	cassandra-node6	1	1.82x	1.03x	61.33x	114.82x
20	cassandra-node5	1	2.72x	1.04x	31.24x	88.25x
18	cassandra-node4	1	2.81x	1.00x	2,027.89x	5,703.39x
17	cassandra-node3	1	1.06x	1.01x	4.59x	4.90x
16	cassandra-node2	1	1.05x	1.05x	4.58x	5.05x
15	cassandra-node1	1	1.05x	1.00x	3.62x	3.79x

Figure 13 shows the SolidFire volumes for a Cassandra cluster. All Cassandra nodes in this test were configured on SolidFire with the same paramters.

Figure 13) Cassandra node SolidFire volume.

ID	Name	Account	Access Groups	Access	Used	Size	Snapshots	Min IOPS	Max IOPS	Burst IOPS	Attributes	512e
599	cassandra-node7	datos-mongo	cassandra-node7	Read / Write	1.56%	500.0 GB	0	50	100,000	100,000	-	Yes
598	cassandra-node6	datos-mongo	cassandra-node6	Read / Write	1.63%	500.0 GB	0	50	100,000	100,000	-	Yes
597	cassandra-node5	datos-mongo	cassandra-node5	Read / Write	1.43%	500.0 GB	0	50	100,000	100,000	-	Yes

Refer to the Interoperability Matrix Tool (IMT) on the NetApp Support site to validate that the exact product and feature versions described in this document are supported for your specific environment. The NetApp IMT defines the product components and versions that can be used to construct configurations that are supported by NetApp. Specific results depend on each customer's installation in accordance with published specifications.

TR-4635-1017