Technical Report

NetApp FAS and Cassandra

Akshay Patil, Karthikeyan Nagalingam
July 2016 | TR-4527

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 Introduction

Apache Cassandra is a massively scalable open source nonrelational database that offers continuous availability, linearly scalable performance, operational simplicity, and easy data distribution across multiple data centers and cloud availability zones. It is a popular column-oriented database. Many modern applications use Cassandra as a database in the back end due to its rich features, including, but not limited to, high availability, partition tolerance, and no single point of failure. Key use cases of Cassandra are web applications with large data requirements, analytics applications, recommendation systems, fraud detection systems, Internet of Things (IoT) applications, and so on.

Cassandra provides high availability by having a user-defined replication factor, where a database administrator can set a custom replication factor, which makes sure of automatic failover within the cluster. Even if one node in the cluster is down, data is reconstructed using the replicated data from the cluster. Cassandra supports seamless horizontal scalability. It also supports tunable consistency; it makes sure that data is always available. It follows a peer-to-peer distributed architecture with no single point of failure. Data access is done by using Cassandra Query Language(CQL), which is similar to SQL.

This reference architecture demonstrates a validated solution design for efficiently deploying a Cassandra NoSQL database on the Data Fabric enabled by NetApp® using NetApp FAS8080. In this solution, the scale-out NetApp FAS array hosts a Cassandra cluster. Backup and disaster recovery services are provided by NetApp Snap Creator®. The NetApp solution offers the following key benefits:

- Better performance as compared to DAS
- Backup and recovery using NetApp Snapshot® technology using Snap Creator

2 Solution Overview

In the end-to-end solution, Cassandra is hosted on FAS8080 storage running NetApp Data ONTAP® 8.3.1. Each Cassandra node was installed on a physical RHEL server. The NetApp FAS8080 is a shared array that is optimized for use with pure flash, and the system is designed for applications that demand low latency with all of the reliability and data services for which NetApp is known. NetApp Snap Creator backup software creates instant NetApp Snapshot copies and clones for creating copies of the entire environment for use in backup, restore, and creating space-efficient copies of the database.

2.1 NetApp FAS

NetApp FAS is a combination of high-performance hardware and adaptive storage software. FAS can unify your SAN and NAS storage needs. The NetApp FAS storage solution, when used with Cassandra provides manageability, high performance, efficiency, and data protection in the cloud and on the premises. It creates NetApp Snapshot copies, which are point-in-time images of the file system. Using NetApp SnapRestore®, you can restore the entire file system using the Snapshot copies in seconds. In terms of performance, FAS storage controller provides low latency and high throughput for enterprise workloads such as Cassandra.

For more information about NetApp FAS, see NetApp FAS Hybrid Storage Array.

2.2 Snap Creator Framework

The NetApp Snap Creator framework allows you to standardize and simplify backup, restore, and disaster recovery in any environment. It's a unified data protection solution for standard and custom applications.

The Snap Creator framework provides:

- **Application-consistent data protection.** Provides a centralized solution for backing up critical information, integrating with existing application architectures for data consistency and to reduce operating costs.
- **Extensibility.** Achieve fast integration using NetApp modular architecture and policy-based automation.
- **Cloud readiness.** OS-independent Snap Creator functionality supports physical and virtual platforms and interoperates with IT-as-a-service and cloud environments.

2.3 Apache Cassandra Architecture Overview

Apache Cassandra handles big data workloads by employing a peer-to-peer distributed architecture across homogeneous nodes. Data is distributed among all nodes in the cluster, and all nodes are in communication with each other. The write activities to a node are first written to a commit log and then indexed and written to the memtable. After the memtable is full, the data is flushed to a sorted string table (SSTable) data file. All writes are automatically partitioned and replicated throughout the cluster.

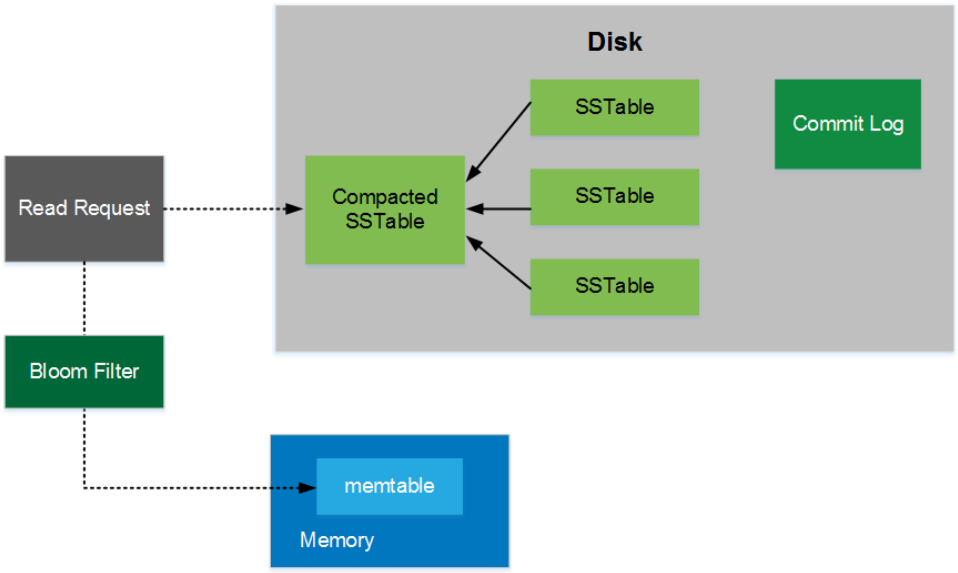
This section explains how read and write requests are handled by Cassandra.

Cassandra Read Path

Figure 1 shows read operation by Cassandra when a read request is received on a node.

When a read request is sent from a client to a Cassandra node, the node uses the bloom filter to identify the SSTable that contains the key. Depending on the key, the node determines where the data lies and serves the data from that location.

Figure 1) Cassandra read path.



Cassandra Write Path

When a write/update request is sent from a client to a Cassandra node, it does the following operations:

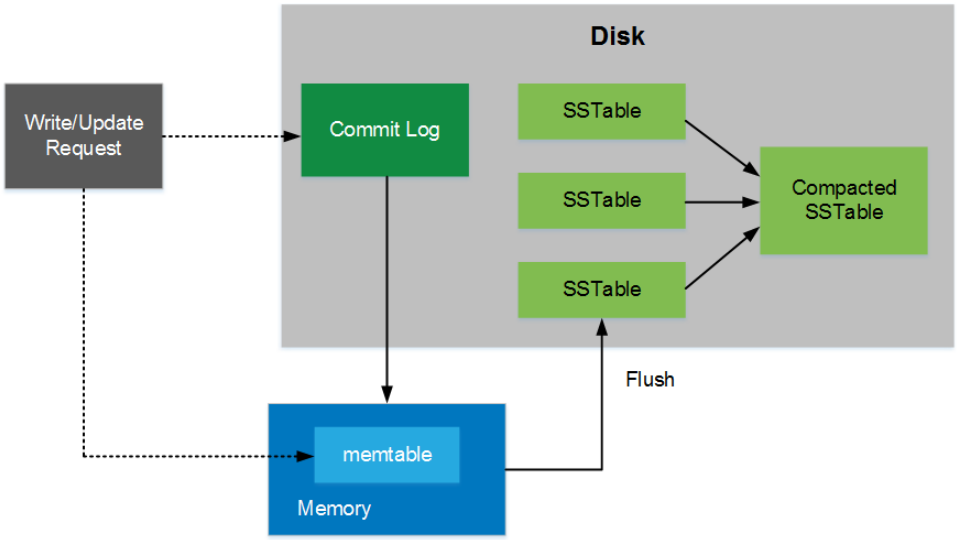
1. Writes the data to an in-memory memtable.
2. Appends the mutation in a commit log.

Eventually memtables are flushed to the disk-based SSTables, thereby moving data from memory to disk.

SSTables are immutable files of key/value string pairs, sorted by keys. SSTables also support compression. After a default size has been reached for the SSTable, they are compacted in Cassandra to save disk space.

Figure 2 depicts the write operation by Cassandra when a write request is received on a node.

Figure 2) Cassandra write path.



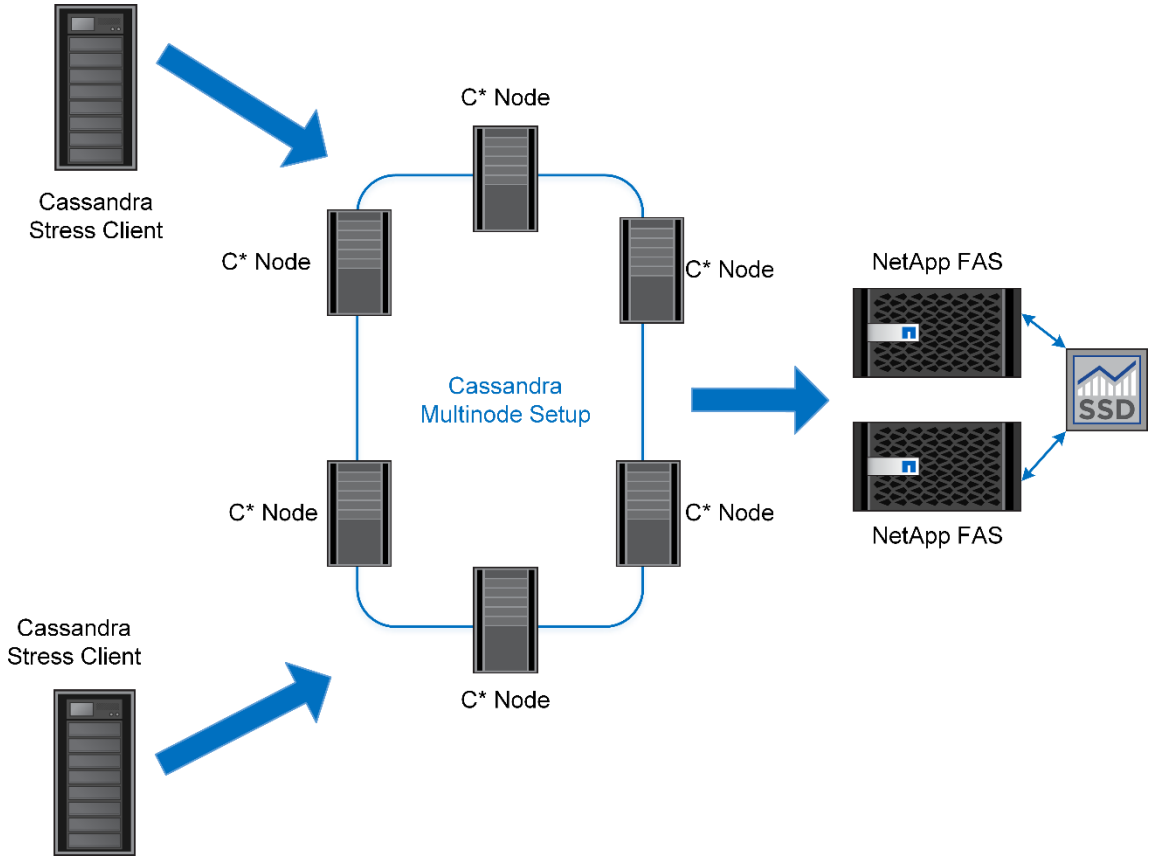
3 Solution Architecture

This section describes the Cassandra architecture, including its components and layout. It also describes the validated architecture and the storage architecture.

3.1 Cassandra Cluster Architecture

Cassandra clusters employ a ring topology. Each node carries some amount of data, depending upon the replication factor and replication strategy defined by the database administrator. There is no single point of failure in a Cassandra cluster; if one node is down, the other nodes can rebuild the data by using the replica data stored in the cluster. Figure 3 shows an example of horizontally scalable Cassandra architecture, in which multiple nodes can be added to the cluster even in a live environment. The newly added node automatically bootstraps itself with the cluster, and it is added to the existing cluster ring by the automatically generated tokens by Cassandra.

Figure 3) Cassandra solution architecture.



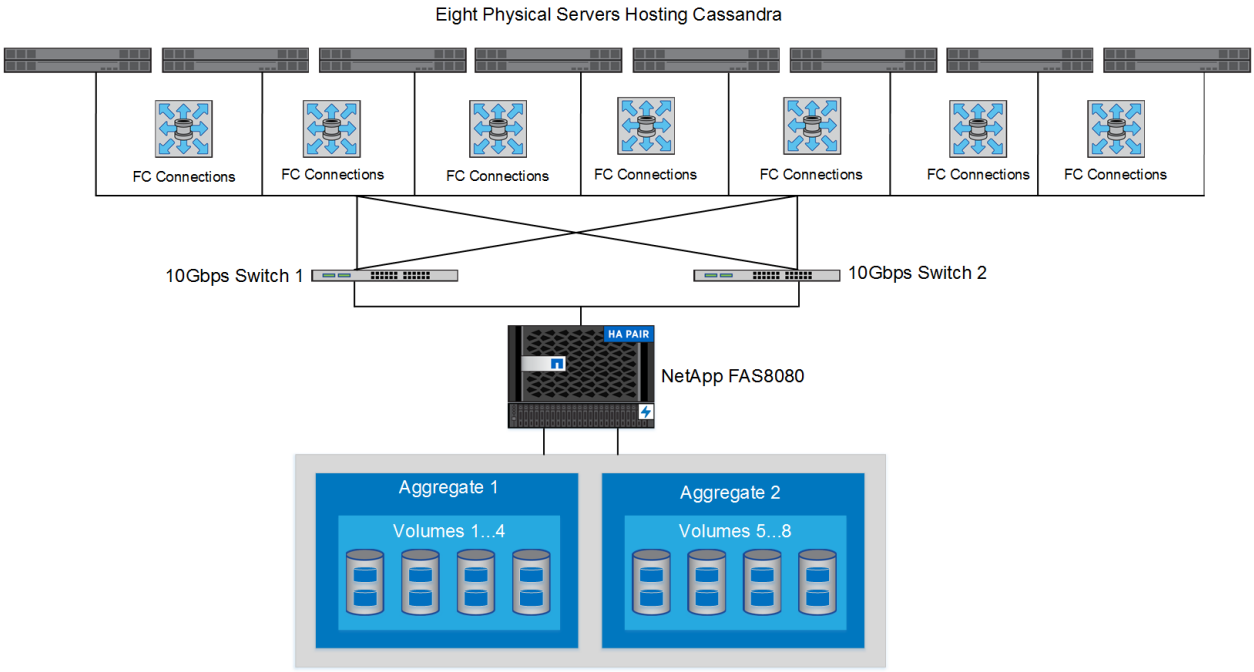
3.2 Network Architecture

Figure 4 depicts the network architecture. A zone with eight servers and the two switches was created to communicate with the NetApp storage. The host server and the NetApp storage are connected through a 10Gbps FC switch. Four FC ports per switch were used in this setup.

In addition to zoning, multipathing was enabled for high availability and to provide multiple paths for writing data to the NetApp LUNs. For details about zoning and multipathing, see the appendix.

The storage side includes two aggregates (one for each of the FAS8080 storage controllers), each containing 22 SSDs. Each aggregate contains four volumes, with each volume containing two LUNs. These LUNs are mounted on the host server by presenting these LUNs to our C* hosts with an EXT4 file system on them. The result of this configuration is that the database files are spread evenly across both storage controllers, allowing for each to contribute evenly to the overall performance of the system.

Figure 4) Network architecture.



4 Solution Validation

4.1 Hardware and Software Prerequisites

Table 1 lists the hardware and software used in our setup.

Table 1) Hardware and software requirements.

Hardware	Details
8 x host servers	2.4GHz CPU, 256GB RAM
Storage	NetApp FAS8080 with 22 x SSD disks
Network	2 x 10Gbps FC switches
Software	**Details**
Operating system	RHEL 6.6
Database	Apache Cassandra 2.1.0
Benchmarking tool	Cassandra stress tool

4.2 Setting Up Multinode Cassandra Cluster

Cassandra utilizes a ring topology, in which multiple nodes can be added to the cluster without needing to restart the cluster. A multinode cluster can be created by configuring the cassandra.yaml file in each of the Cassandra nodes. There should be a seed node mentioned to configure a multinode cluster. This parameter makes sure that the node that wants to joins a cluster knows at least one other node in the ring.

1. Download Cassandra from http://cassandra.apache.org/download/.
2. Extract the package on each node.
3. To set up a multinode cluster, the `install_location/conf/ cassandra.yaml` file needs to be configured as shown in Table 2.

Table 2) `cassandra.yaml` configuration.

Parameter	Default Value	Recommended Value	Description
Cluster name	Test cluster	Any_Name	Name of the cluster. This is mainly used to prevent the Cassandra nodes in one logical cluster from joining another. All the nodes in a Cassandra cluster should have the same name.
Data directory	Install_location/data/data	/Path_to_NetApp_Lun/data	Path to directories where Cassandra should store data.
Commitlog directory	Install_location/data/commitlog	/Path_to_NetApp_Lun/commitlog	Path to directories where Cassandra should store the commit logs.
Saved cache directory	Install_location/data/saved_caches	/Path_to_NetApp_Lun/saved_caches	Path to directories where Cassandra should store saved cache files.
Seed node	127.0.0.1	IP/host name of seed node	Seeds are used during startup to discover the cluster. The seed node bootstraps the gossip process for new nodes joining the cluster. There can be multiple seeds in a cluster, but not every node should be a seed node.
Listen address	localhost	IP/host name of the node	An address or interface to bind to and tell other Cassandra nodes where they should connect to form a cluster. It should be the IP address or host name of the node.
RPC address	localhost	IP/host name of the node	Allows broadcast to drivers and other Cassandra nodes.
Endpoint snitch	SimpleSnitch	GossipingFileProperty Snitch	Allows Cassandra to spread replicas around your cluster to avoid correlated failures.

4. Open all the Cassandra ports from the firewall on each host. See the appendix for all the ports used by Cassandra.
5. Start the seed node first by running the following command:

```
> /install_location/bin/Cassandra
```

6. Start the other nodes one by one after the seed node is up.
7. Check the cluster status by running the following command:

```
> /install_location/bin/nodetool status
```

The output of the previous command displays whether the status of the node is UP or DOWN: that is, if the node is included in our Cassandra ring or not. It also displays the state of the node as one of the following:

- **Normal.** This state indicates that the node is in an optimal state and can send and receive data.
- **Leaving.** This state indicates that the node is leaving the ring and is currently redistributing its tokens to other nodes in the cluster.

- **Joining.** This state indicates that the node has just joined the ring by contacting the seed node and is waiting for its share of tokens.
- **Moving.** This state indicates that the node is moving its data to other nodes before leaving the ring or is getting data from some other node after joining the Cassandra ring.

Ideally, every node in the cluster should be in the UN state: that is, Up and Normal. The following screenshot shows a sample output for the `nodetool status` command.

```
[root@stlrx2540m1-68 apache-cassandra-2.1.0]# bin/nodetool status
Note: Ownership information does not include topology; for complete information, specify a keyspace
Datacenter: DC1
===============
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address         Load       Tokens  Owns    Host ID                               Rack
UN  10.63.150.190   28.92 GB   256     13.6%   15d97774-b4ce-4330-bdb8-53b8ce132e1a  RAC1
UN  10.63.150.165   45.94 GB   256     15.3%   e0ca50a4-549e-4058-813b-83ad794b5e3c  RAC1
UN  10.63.150.163   43.68 GB   256     14.8%   c1f01fba-2d2e-464e-816c-817ee462dead  RAC1
UN  10.63.150.178   19.17 GB   256     14.1%   7543e777-daa6-463b-b38b-3a9c00bf00eb  RAC1
UN  10.63.150.161   43.7 GB    256     15.0%   a55d429d-5218-4407-9b23-a53d802a2c25  RAC1
UN  10.63.150.177   1.3 GB     256     13.2%   de6d6d26-5d85-45f8-81ba-ae8ab29b8c5e  RAC1
UN  10.63.150.176   42.13 GB   256     14.0%   7a037f16-6c59-47b2-8e4e-75bf27a11d3b  RAC1
[root@stlrx2540m1-68 apache-cassandra-2.1.0]#
```

4.3 Cassandra Performance Validation

The Cassandra stress tool is a Java-based stress-testing utility for benchmarking and load testing a Cassandra cluster. The Cassandra stress tool is bundled with the Apache Cassandra and Datastax Enterprise Cassandra versions.

The following are its modes of operation:

- **Inserting.** Loads test data.
- **Reading.** Reads test data.
- **Mixed.** Writes and reads the test data as per user specification.

The Cassandra stress tool creates a keyspace called Keyspace1. Within this keyspace, tables named Standard1, Super1, Counter1, and SuperCounter1 are automatically created the first time you run the stress test and are reused on subsequent runs unless you drop the keyspace using CQL or CLI. The stress tool uses these to load the data that is used for testing. We ran the Cassandra stress tool for validating our architecture and measured the IOPS and median latency. More information about the Cassandra stress tool can be found here.

We performed these tests using the following two setups:

- Four-node Cassandra cluster
- Eight-node Cassandra cluster

Four-Node Cassandra Cluster

In this testing, we inundated data on a Cassandra cluster using the Cassandra stress tool, and the results of the test are recorded. For testing we used 5 million OperationCounts, which are used for read and write operations.

Table 3 shows the test results for a workload of 0% reads or 100% writes. A throughput of around 141k IOPS with a median latency of 1.4ms at 500 threads was achieved. At 500 threads, we encountered a bottleneck at the CPU on the host side because the CPU utilization was ~80% while the Cassandra stress tool was running on the host.

Table 3) Test results for Cassandra on four-node cluster with 100% writes.

No. of Threads	% Reads	No. of Operations	IOPS	Latency (Median) (ms)
100	0	5,000,000	102,889	0.712
250	0	5,000,000	124,556	1.074

No. of Threads	% Reads	No. of Operations	IOPS	Latency (Median) (ms)
500	0	5,000,000	141,886	1.422

Table 4 shows the test results for a mixed workload of 50% reads and 50% writes. A throughput of around 138k IOPS with a median latency of 1.2ms at 500 threads was achieved. At 500 threads we encountered a bottleneck at the CPU on the host side because the CPU utilization was ~80% while the Cassandra stress tool was running on the host.

Table 4) Test results for Cassandra on four-node cluster with 50% reads and 50% writes.

No. of Threads	% Reads	No. of Operations	IOPS	Latency (Median) (ms)
100	50	5,000,000	101,103	0.770
250	50	5,000,000	127,743	0.912
500	50	5,000,000	138,505	1.208

Table 5 shows the test results for a workload of 100% reads. A throughput of around 143k IOPS with a median latency of 1.3ms at 500 threads was achieved.

Table 5) Test results for Cassandra on four-node cluster with 100% reads.

No. of Threads	% Reads	No. of Operations	IOPS	Latency (Median) (ms)
100	100	5,000,000	107,258	0.603
250	100	5,000,000	129,882	1.004
500	100	5,000,000	143,249	1.396

Eight-Node Cassandra Cluster

Because Cassandra is horizontally scalable, four more nodes were added to the cluster, and the performance was then measured using the same workloads and thread counts to determine if there was an improvement in performance after scaling out the size of the cluster. This section details the results of these tests.

Table shows the test results for a workload of 100% writes. A throughput of around 333k IOPS with a median latency of 1.1ms at 500 threads was achieved with an eight-node Cassandra cluster.

Table 6) Test results for Cassandra on eight-node cluster with 100% writes.

No. of Threads	% Reads	No. of Operations	IOPS	Latency (Median) (ms)
100	0	5,000,000	163,707	0.5
250	0	5,000,000	308,136	0.6
500	0	5,000,000	333,079	1.1

Here it can be seen that with an eight-node cluster, IOPS is almost doubled.

Table shows the test results for a mixed workload of 50% writes and 50% reads. A throughput of around 380k IOPS with a median latency of 0.9ms at 500 threads was achieved,

Table 7) Test results for Cassandra on eight-node cluster with 50% reads and 50% writes.

No. of Threads	% Reads	No. of Operations	IOPS	Latency (Median) (ms)
100	50	5,000,000	200,411	0.5
250	50	5,000,000	323,743	0.6
500	50	5,000,000	380,386	0.9

Table shows the test results for a mixed workload of 100% reads. A throughput of around 337k IOPS with a median latency of 1.0ms at 500 threads was achieved.

Table 8) Test results for Cassandra on eight-node cluster with 100% reads.

No. of Threads	% Reads	No. of Operations	IOPS	Latency (Median) (ms)
100	100	5,000,000	180,707	0.7
250	100	5,000,000	298,441	0.9
500	100	5,000,000	337,026	1.0

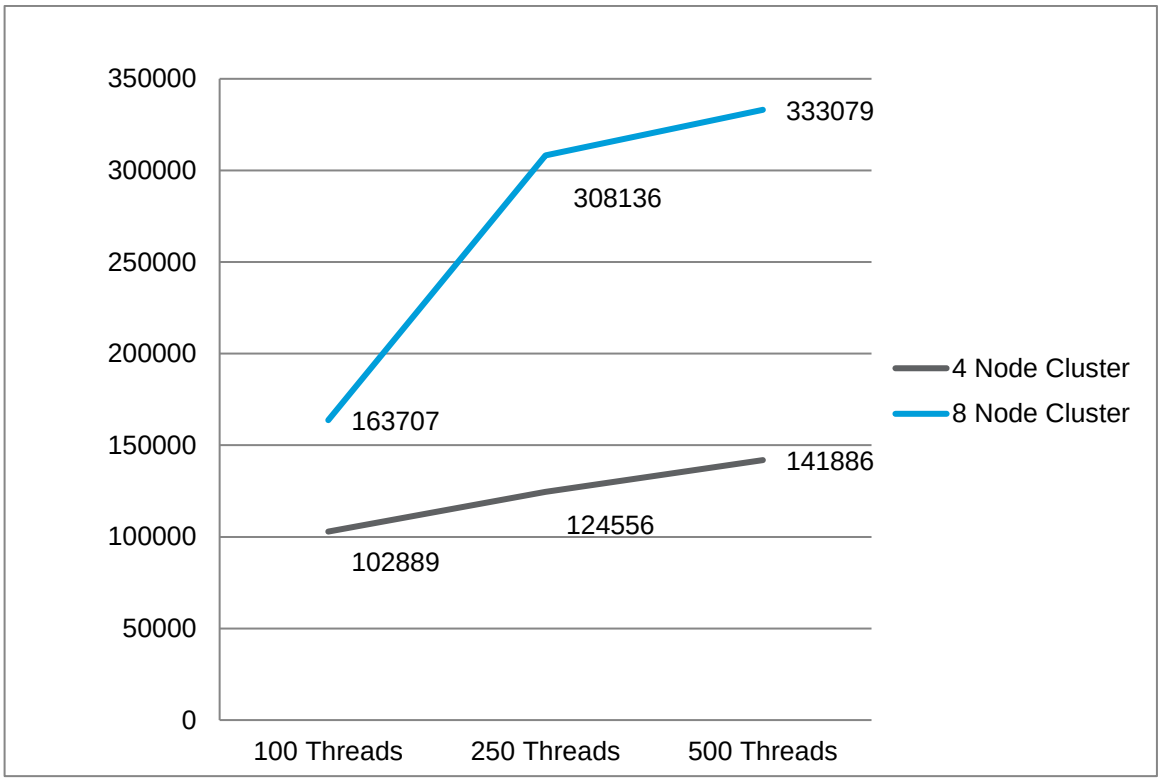
The results showed a near linear performance scaling from an IOPS perspective as a result of scaling out the size of the Cassandra cluster.

4.4 IOPS Results

Figure 5 shows the results of the write operations performed on the database. Approximately 333K IOPS for 500 threads was obtained for a workload of 100% writes.

We recommend that you maintain threads from 250 to 500 for each Cassandra stress client.

Figure 5) IOPS results for 100% writes.



A real-world workload is always a mix of reads and writes. Therefore, a mixed workload of 50% writes and 50% reads was tested to simulate a real-world scenario. As shown in Figure 6 approximately 380K IOPS was obtained for 500 threads. We obtained a maximum throughput at 500 threads at an expense of 80% CPU utilization at the host side.

Figure 6) IOPS results for 50% reads.

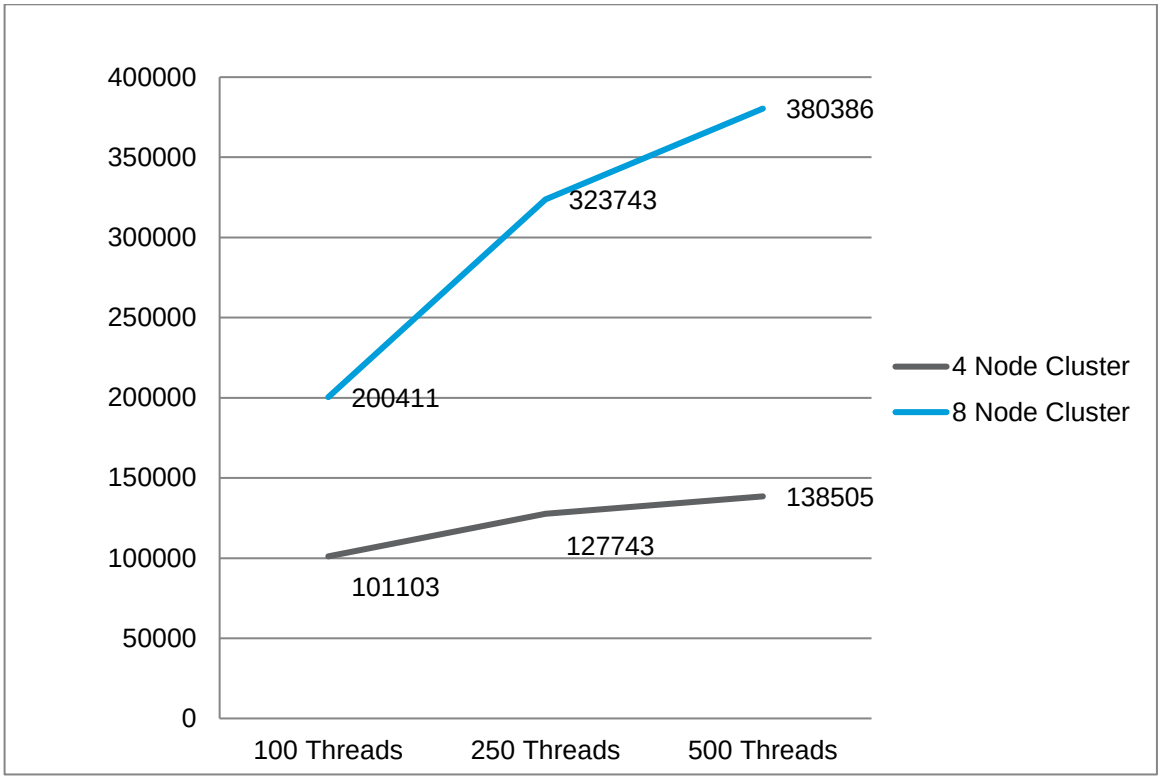
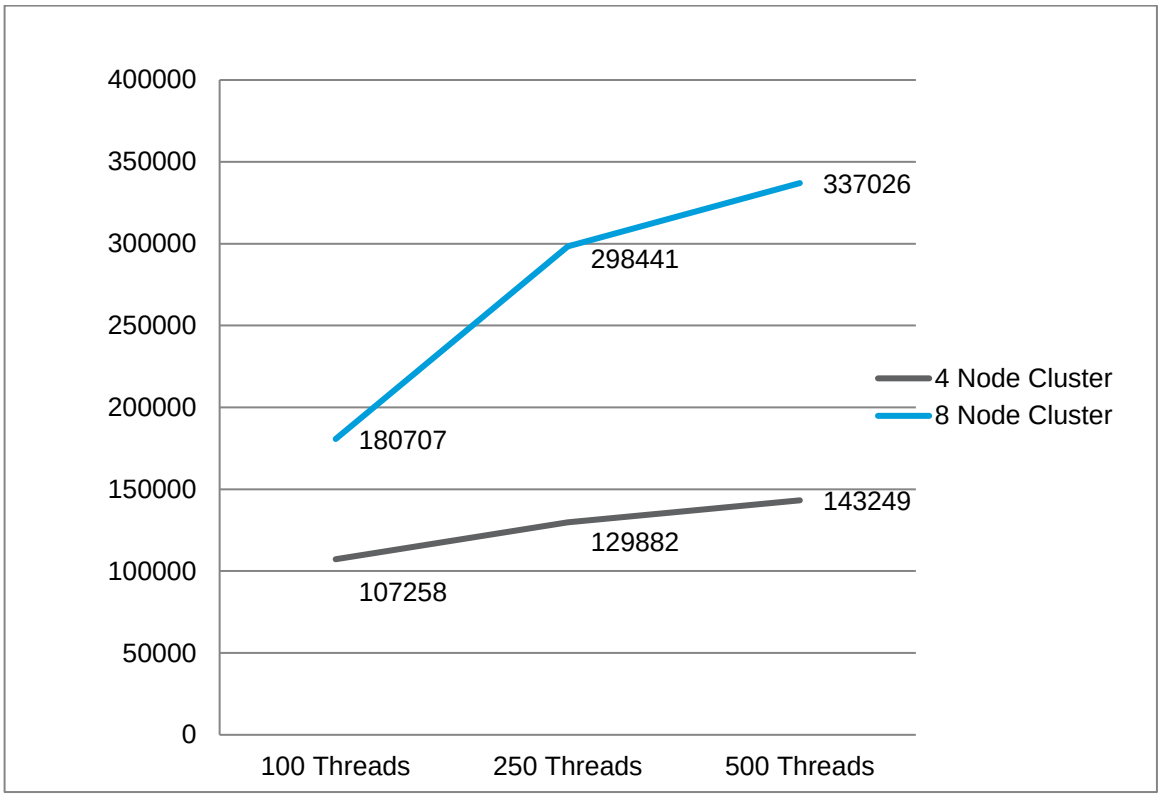


Figure 7 shows the read operations on the database. Approximately 337K IOPS for 500 threads was obtained for a workload of 100% reads. We recommend that you maintain threads from 250 to 500 for each Cassandra stress client.

Figure 7) IOPS results for 100% reads.



4.5 Latency

Figure 8 shows the latency results by a Cassandra stress client while running a mixed workload of 50% reads and 50% writes. We can infer from the graph that we were able to get submillisecond latencies for up to 500 threads on an eight-node Cassandra cluster.

Figure 8) Latency results at 50% reads.

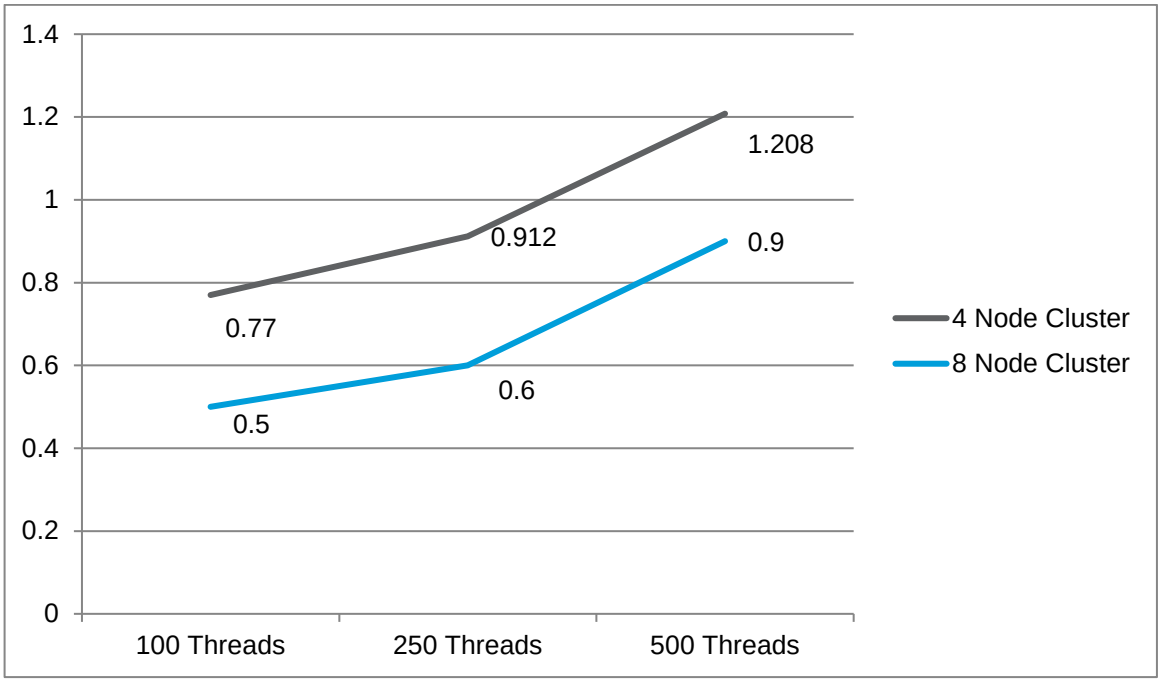


Figure 9 shows latency reported by the Cassandra stress tool while running a workload of 100% reads. As we tested for latency, we recorded the median latency of operations. This makes sure that we avoid noisy recordings. With 500 threads on an 8-node cluster, we were able to achieve latencies around 1ms.

Figure 9) Latency results at 100% reads.

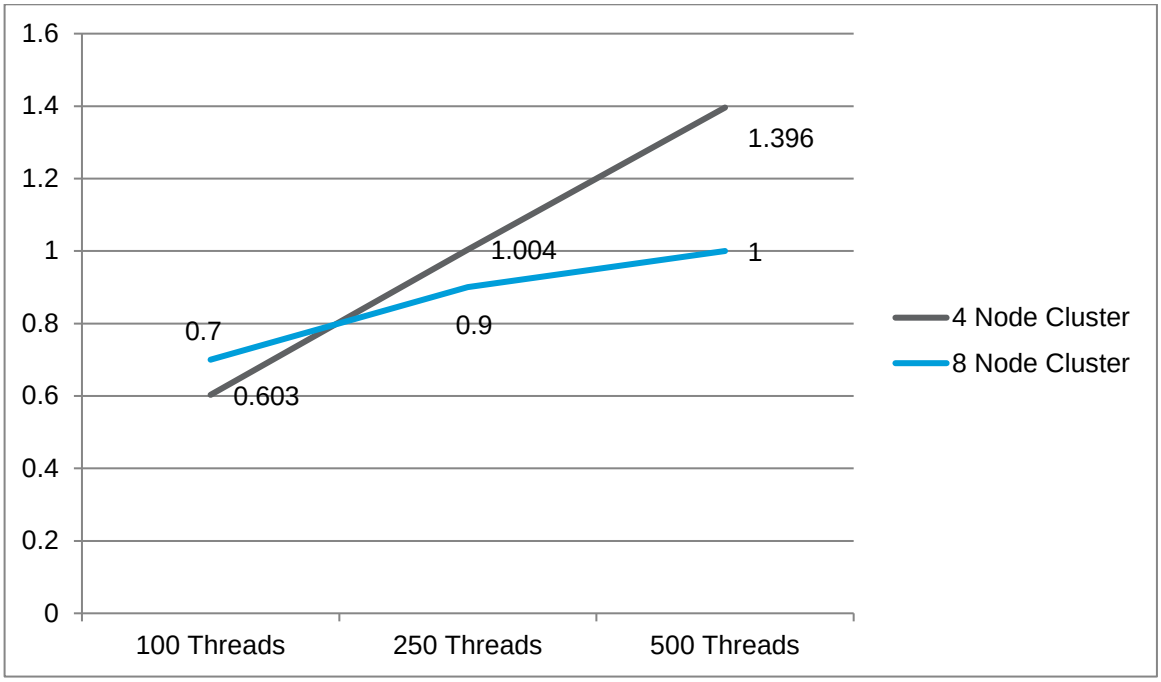
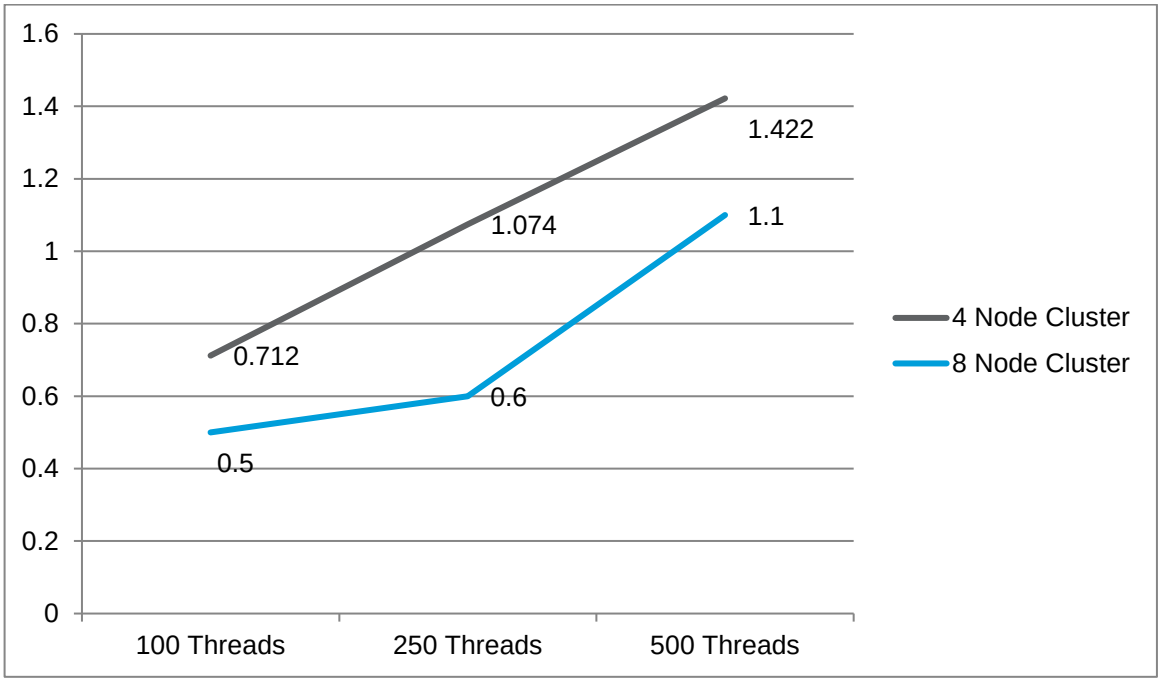


Figure 10 shows latency reported by the Cassandra stress tool while running a workload of 100% writes. Because we tested for latency, we recorded the median latency of operations; this makes sure that we avoid noisy recordings. With 500 threads, we were able to achieve latencies around 1ms.

Figure 10) Latency at 100% writes.



4.6 CPU Utilization

Figure 11 shows the host-side CPU utilization when the Cassandra stress tool is running the workload of 100% writes with 500 threads. The CPU utilization at the host side was more than 75% when running the Cassandra stress tool with 500 threads. We can conclude that the host-side CPU is a bottleneck for delivering more IOPS. The FAS8080 system could have continued to deliver even higher IOPS at consistent ultralow latency if we add more servers to the cluster.

Figure 11) CPU utilization with 100% writes and 500 threads.

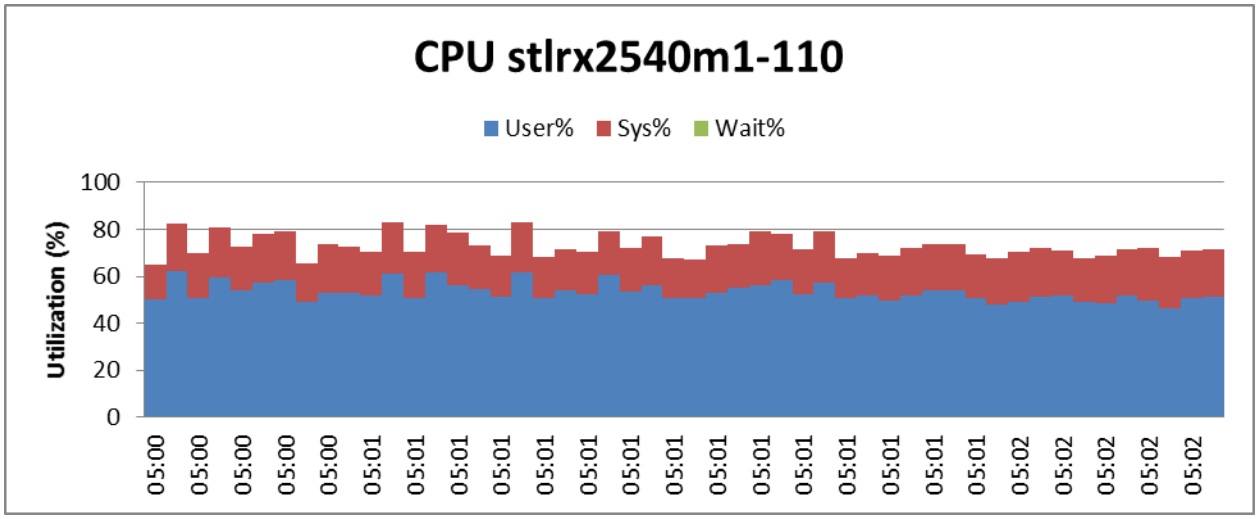


Figure 12 shows the host-side CPU utilization when the Cassandra stress tool is running the workload of 50% reads and 50% writes with 500 threads. The CPU utilization at the host side was >75%. We can conclude that the host-side CPU is a bottleneck for going further.

Figure 12) CPU utilization with 50% reads and 500 threads.

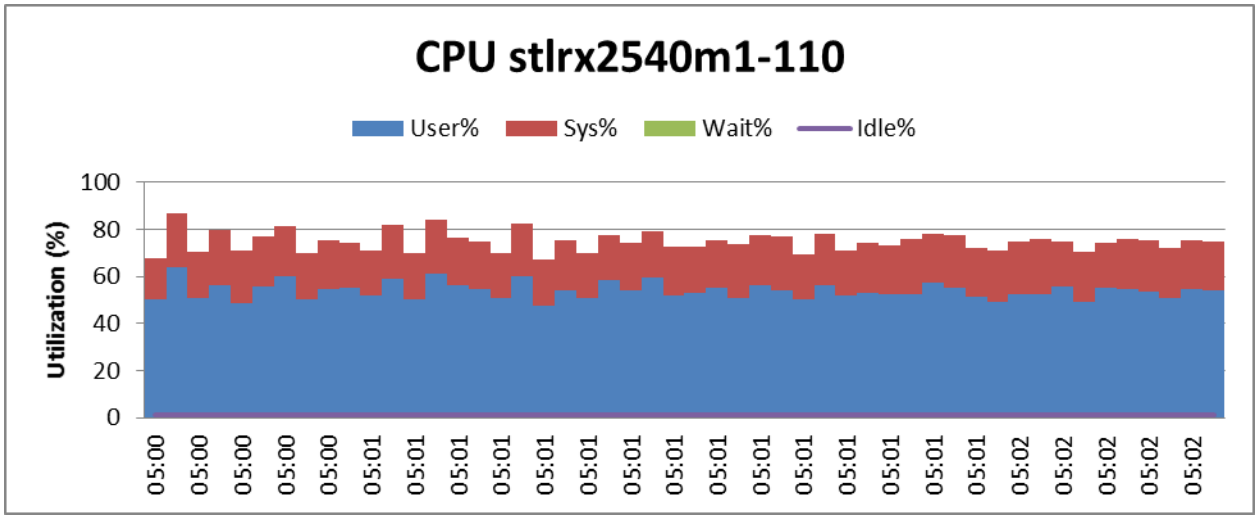
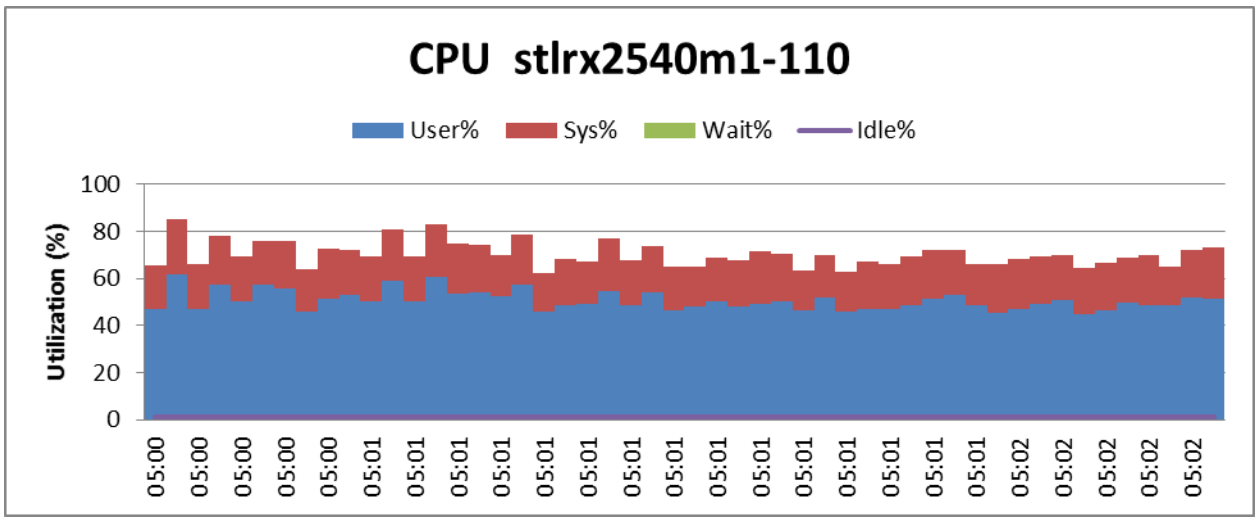


Figure 13 shows the host-side CPU utilization when the Cassandra stress tool is running the workload of 100% reads with 500 threads. The CPU utilization at the host side was >75%. We can conclude that the host-side CPU is a bottleneck for going further.

Figure 13) CPU utilization with 100% reads and 500 threads.



4.7 Backup and Restore Using Snap Creator

This solution leverages the Snap Creator framework to back up and restore the Cassandra database environment by using space-efficient NetApp Snapshot technology. In our testing, we were able to create Snapshot backups for the complete environment in less than one minute. Restore of the NetApp volumes that hosted the Cassandra database also took less than one minute.

For our testing, we configured Snap Creator to carry out the following operations:

- **Quiesce and flush the in-memory data to disk.** Quiescing means temporarily pausing the I/O operations from the database. This makes sure that the database is in a locked state and no new operations can take place when further operations are performed.

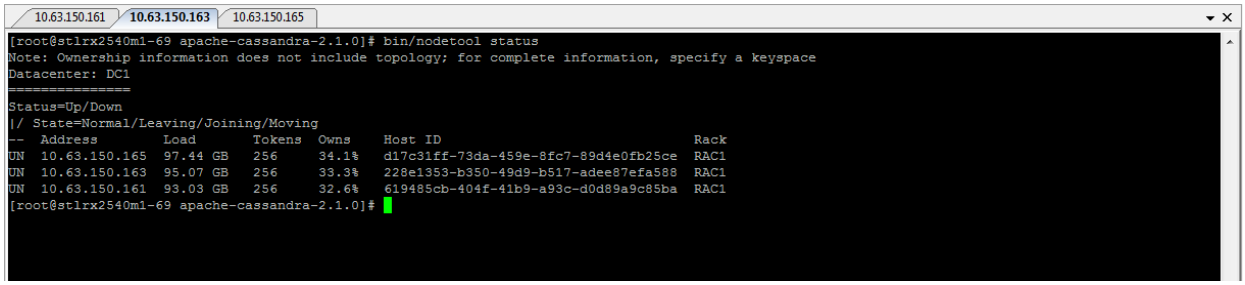
  Because Cassandra supports several in-memory data structures, it can store some data in the memory. However, while it is creating a Snapshot copy, the in-memory data can be lost. Therefore, we first flushed the in-memory data to disk and then created a Snapshot copy in order to prevent data loss.
- **Create NetApp Snapshot copies using Snap Creator.** When a Snapshot copy is created, an instant copy of the state of the volume is obtained. These Snapshot copies can be used as backup copies. These Snapshot copies initially require no additional storage and can be stored in the same volume.

Snap Creator Backup Procedure

To create a Snapshot copy using Snap Creator, complete the following steps:

1. Insert random data into the cluster. We inserted approximately 300GB of random data in our cluster using YCSB (a load generation and benchmarking tool for NoSQL databases).

   The following figure shows the status of our Cassandra cluster after insertion of test data using YCSB.



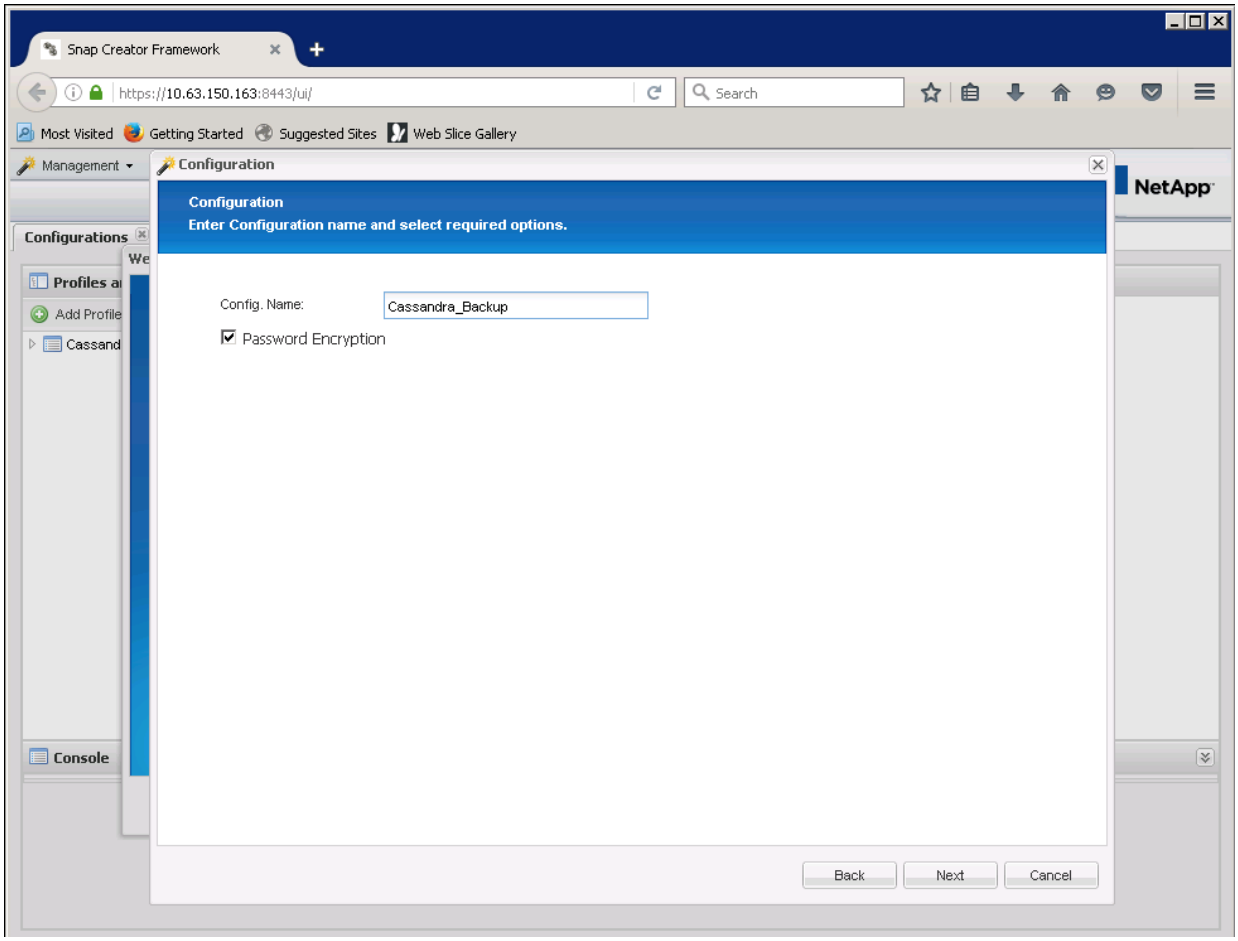
2. Insert a record, which we can query after the restore to check consistency of our cluster.

```
[root@stlrx2540m1-69 apache-cassandra-2.1.0]# bin/cqlsh
Connected to Test Cluster at 10.63.150.163:9042.
[cqlsh 5.0.1 | Cassandra 2.1.0 | CQL spec 3.2.0 | Native protocol v3]
Use HELP for help.
cqlsh> use ycsb;
cqlsh:ycsb> select * from usertable where y_id  = '100000000';

 y_id      | field0 | field1       | field2                           | field3 | field4 | field5 | field6 | field7 | field8 | field9
-----------+--------+--------------+----------------------------------+--------+--------+--------+--------+--------+--------+--------
 100000000 | Akshay | TestingField | Field Available at time of Backup |   null |   null |   null |   null |   null |   null |   null

(1 rows)

cqlsh:ycsb>
```

3. Install and start the Snap Creator server on a node. Go to http://mysupport.netapp.com/NOW/download/software/snapcreator_framework/4.1/.

   Note: By default, the Snap Creator server runs on port 8443.

4. Log in to the Snap Creator server UI.
5. Go to Configurations > Add Configuration > Add Profile.



6. Select the plug-in type as None.
7. Configure the Snap Creator agent on the Cassandra node.
   a. Select Transport as HTTPS with port 443.
   b. Add the Vserver IP and provide the SVM user name and password. This user should have vsadmin privileges. If FCP is being used, create a new network interface with management access. Select and add the Cassandra volumes to back up.

c. Provide a name for the Snapshot copy and define the Snapshot policy. A monthly Snapshot retention policy is enabled by default.

8. Flush all the in-memory data to disk and quiesce the database.

   For example:

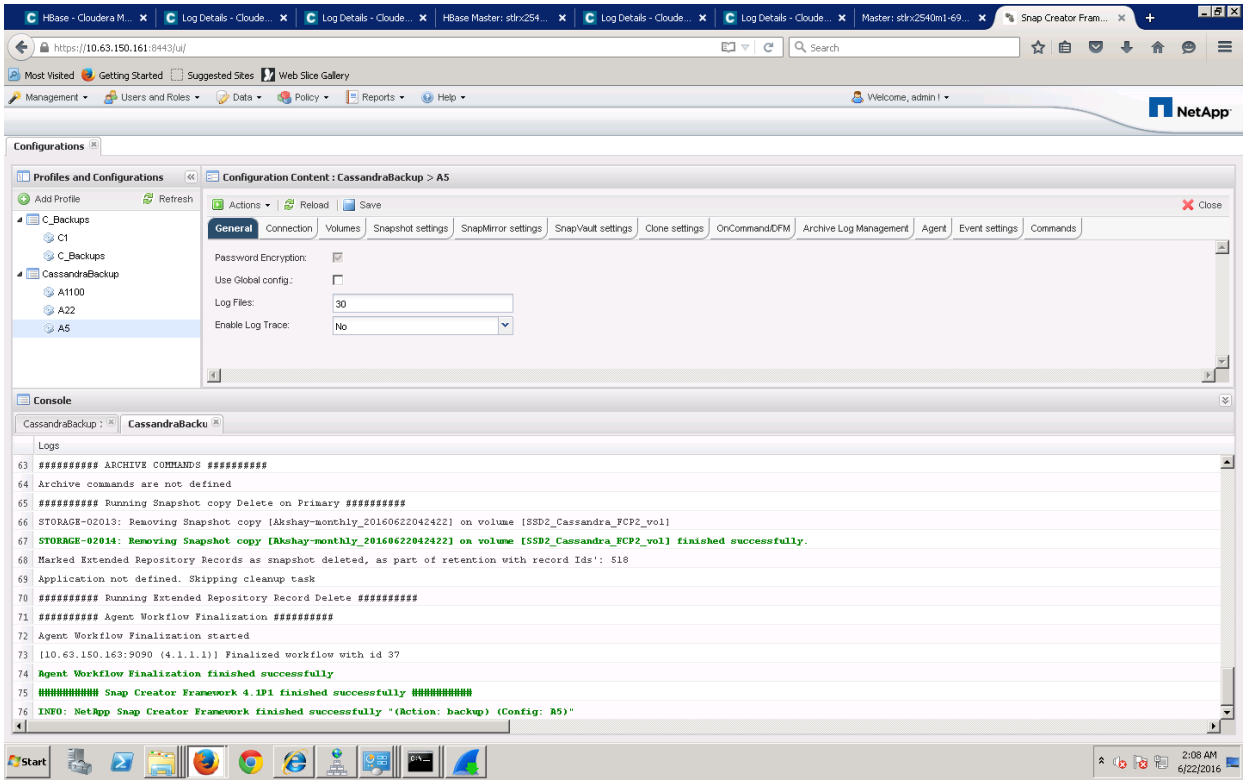
```
>/home/Downloads/apache-cassandra-2.1.0/bin/nodetool -h 10.63.150.163 flush
>/home/Downloads/apache-cassandra-2.1.0/bin/nodetool -h 10.63.150.161 flush
>/home/Downloads/apache-cassandra-2.1.0/bin/nodetool -h 10.63.150.165 flush
>fsfreeze –f /Lun_Path/
```

9. Create a Snapshot copy after the flush is complete from the Snap Creator server UI.
10. Unquiesce the database using the `fsfreeze -u /Lun_Path/` command.

    By using `fsfreeze`, you can suspend and resume access to a file system. After the file system is frozen, all the pending I/O operations are queued until the file system access is resumed by using `fsfreeze`.

Note: Execute `fsfreeze` on each Cassandra node. A parallel SSH tool such as pSSH can be used for this operation. pSSH executes nodetool and communicates to all of the nodes in parallel over IP.

11. To create a Snapshot copy, select Actions > Backup in the Snap Creator Server GUI.



Snap Creator Restore Procedure

Using Snap Creator, we can restore the volumes to their previous states using NetApp Snapshot.

To restore a Cassandra volume using Snap Creator, complete the following steps:

1. Corrupt (by way of deletion) the data Cassandra has stored on the NetApp volume and then restore it using the Snapshot copies.

```
> cd /mnt/SSD_FCP2/
> rm –rf *
```

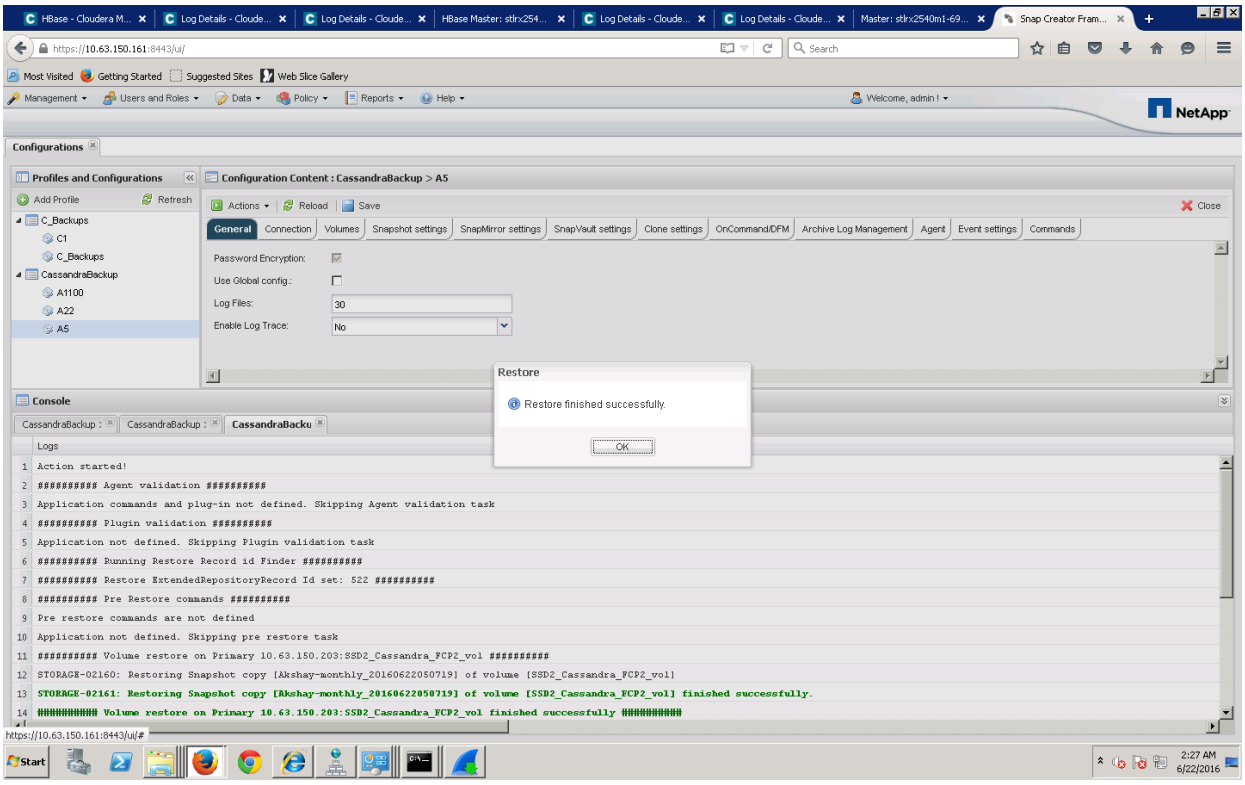
2. Stop Cassandra on all the hosts in the cluster.

```
> ps auwx | grep cassandra
> sudo kill <pid>
```

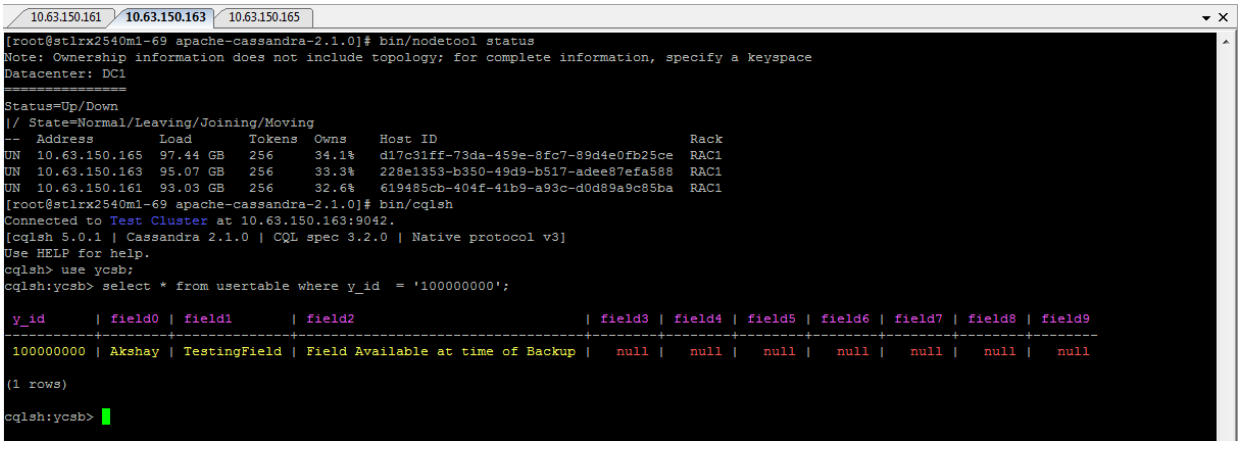
3. Restart the Cassandra bode.
4. Mount the NetApp LUN on the C* node again:

```
mount -t ext4 /dev/mapper/mpatho /mnt/Lun_Path/
```

5. Select the restore volume in Snap Creator. Provide the SVM IP and volume to restore and select the latest Snapshot copy. Select the restore type as volume restore.



6. Restart the seed C* node.
7. Because the restore is complete, our data should be visible again in the Cassandra cluster. To check the consistency of data, we query the known record we inserted just before creating the backup:



With this demonstration we can validate that we are able to restore the volume in seconds and turn the cluster to a normal state without any data loss.

5 Conclusion

The reference architecture described in this document provides an end-to-end solution for efficiently deploying a Cassandra NoSQL database on the Data Fabric enabled by NetApp. In the tested architecture of NetApp storage technology, a Cassandra database was hosted on the scale-out FAS8080. Backups were created by using Snap Creator. The following key benefits were validated in this solution:

- Predictable high performance with consistent low latency, providing very fast response time to the most demanding analytics applications built on Cassandra
- Backup and recovery based on Snapshot technology
- Nondisruptive operations to deliver maximum uptime and high availability

During this validation, NetApp FAS demonstrated excellent test results to deliver high performance on a Cassandra cluster hosted on FAS arrays.

Appendixes

Ports Used by Cassandra

Cassandra uses multiple ports for intercluster communication. By default, these ports are not allowed to communicate to external nodes unless it is defined in the operating system's firewall configuration. Hence, these ports need to be opened for smooth functioning of the Cassandra cluster.

Table 9) Ports used by Cassandra.

Port Number	Description
Public Ports	
22	SSH port
Cassandra Internode Ports	
7000	Cassandra internode cluster communication
7001	Cassandra SSL internode cluster communication
7199	Cassandra JMX monitoring port
Cassandra Client Port	
9042	Cassandra client port
9160	Cassandra client port (Thrift)

Open all the ports mentioned in Table 9 for Cassandra's intercluster communication. To open the ports, run the following commands:

- In CentOS/7:

```
>sudo firewall-cmd --zone=public --add-port=9042/tcp --permanent
>sudo firewall-cmd –reload
```

- RHEL 6.6:

```
> sudo iptables -I INPUT -p tcp -m tcp --dport 9042 -j ACCEPT
> sudo service iptables save
```

FC Zoning

Zoning the FC switches enables the hosts to connect to the storage and limits the number of paths.

Before you begin:

- You must have administrator credentials for the switches.
- You must know the WWPN of each host initiator port and of each FC LIF for the storage virtual machine (SVM) in which you created the LUN.

Steps

1. Log in to the FC switch administration program and then select the zoning configuration option.
2. Create a new zone that includes the first initiator and all of the FC LIFs that connect to the same FC switch as the initiator.
3. Create additional zones for each FC initiator in the host.
4. Save the zones and then activate the new zoning configuration.

Multipathing

Multipathing is a technique of creating multiple paths to read and write data from a host to an external storage device. Following these steps to configure multipathing:

1. Install multipath and sg3_utils packages in RHEL.
2. Configure the multipathing as described in DM-Multipath Configuration.
3. Run the `rescan` script from the sg3utils.

```
>/usr/bin/rescan-scsi-bus.sh
```

4. Verify the configuration by running the following command:

```
multipath -ll
```

Enabling RMX

Nodetool commands are designed to run locally on the C* node. To enable execution of nodetool commands on other nodes from one node, you must enable RMX in the Cassandra configuration. Enable RMX by adding the following line to `install_location/conf/cassandra-env.sh`:

```
JVM_OPTS = "$JVM_OPTS -Djava.rmi.server.hostname=<public name>"
```

This allows remote execution of commands within the cluster.

References

The following references were used in this TR:

- Introduction to Cassandra
  http://docs.datastax.com/en/cassandra/2.0/cassandra/gettingStartedCassandraIntro.html
- A Brief Introduction to Apache Cassandra
  https://academy.datastax.com/resources/brief-introduction-apache-cassandra
- Configuring Multi Node Cassandra Cluster
  https://docs.datastax.com/en/cassandra/2.2/cassandra/initialize/initSingleDS.html
- Snap Creator Framework
  http://www.netapp.com/us/products/management-software/snapcreator-framework.aspx
- NetApp SAN Configuration for Zoning
  https://library.netapp.com/ecm/ecm_download_file/ECMP1196793

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.