NetApp Verified Architecture

Apache Spark Workload with NetApp Storage Solution

NVA Deployment

Karthikeyan Nagalingam, NetApp

March 2021 | NVA-1157-DEPLOY

Abstract

This document describes the performance and functionality validation of Apache Spark SQL on NetApp® NFS AFF storage systems. It reviews the configuration, architecture, and performance testing based on various scenarios, as well as recommendations for using Spark with NetApp ONTAP® data management software. It also covers test results based on just a bunch of disks (JBOD) versus the NetApp AFF A800 storage controller.

TABLE OF CONTENTS

Introduction **4**

Solution overview **4**

Solution technology 4

Use case summary 5

Quantitative finance analytics platform with NetApp storage **5**

Why NetApp for this use case? 6

Technology requirements **6**

Hardware requirements 7

Software requirements 7

Deployment procedures **7**

Hadoop/Spark cluster setup 8

NetApp storage controller setup—ONTAP 8

NetApp storage controller setup—StorageGRID 9

Solution verification overview **9**

Spark SQL with ONTAP 9

Apache Spark with StorageGRID 10

Apache Spark workload through MinIO NFS Gateway 11

Solution verification steps **12**

JBOD versus AFF A800 lab validation for Spark workload **21**

Conclusion **23**

Appendix **23**

Scripts used for ONTAP 23

Scripts used for StorageGRID 25

Scripts used for MinIO 27

Main script for ONTAP, StorageGRID and MinIO 28

Command to run the tests 29

Where to find additional information **30**

Version history **30**

LIST OF TABLES

Table 1) Hardware requirements. 7

Table 2) Software requirements. 7

 © 2021 NetApp, Inc. All rights reserved.

Table 3) Cloud object storage bench load generators on six ESXi hosts each with 1x100Gbps. 9
Table 4) Parquet file creation write query summary—ONTAP. 10
Table 5) Sequential read query—Cache count and groupby filter: ONTAP AFF A800. 10
Table 6) Sequential read query—Cache count and groupby filter: StorageGRID. 11
Table 7) Parquet file creation write query: MinIO NFS Gateway 11
Table 8) Sequential read query—Cache count and groupby filter: MinIO NFS Gateway 11
Table 9) Hardware details used for lab validation. 21

LIST OF FIGURES

Figure 1) Spark solution with NetApp Storage. 5
Figure 2) CPU utilization of Spark worker node. 10
Figure 3) Read query—Count without caching and groupby filter. 13
Figure 4) Write query times for the 2.4TB parquet creation for both NFS and pNFS. 14
Figure 5) Write query and read query results with number of nodes. 15
Figure 6) Write query results with MinIO and number of nodes. 20
Figure 7) Read query results with MinIO and number of nodes. 20
Figure 8) JBOD versus AFF A800 write and read query results on a six-node Spark cluster 23

© 2021 NetApp, Inc. All rights reserved.

Introduction

Apache Spark provides same the benefits as Apache Hadoop, such as scalability, fault tolerance, and distributed computing, and it is one hundred times more efficient and faster than MapReduce by caching data in memory across iterative computations and using lighter weight threads. Spark is mostly used for analysis of real-time stream data, quick results with in-memory computations, chains of parallel operations using iterative algorithms, graph-parallel processing to model data, and machine learning (ML) applications. These use cases guide the customer to move from Hadoop to Spark.

For customers using Spark, this document describes the best practices for using Spark on NetApp® ONTAP®, NetApp StorageGRID®, and MinIO Gateway to NFS by providing the following content:

- NetApp-unique values for using Apache Hadoop/ Apache Spark with NetApp storage systems by moving from just a bunch of disks (JBOD)/white box to enterprise grade.
- Spark performance validation with NetApp AFF A800/A700 storage systems and StorageGRID through NFS, parallel NFS (pNFS), and Simple Storage Service (S3) protocol.
- Spark SQL queries for both read and write operations.
- MinIO Gateway validation to access NFS data through S3a in a secure way for transactional data.
- Best practice guidelines for Spark workloads on NetApp storage systems.
- JBOD versus AFF A800 lab validation for Spark workloads.

Solution overview

This solution addresses the customer challenge in which a Spark worker white box server uses most of the CPU's power for disk operations and the I/O wait time is more than 95% for disk I/O operations. In this case, to provide better performance, the customer must separate the CPU usage used for the disk I/O operations from the Spark worker nodes and reduce the I/O wait time to less than 30%.

NetApp solution architecture provides the storage and compute separation for Modern Big Data Analytics, which separates the disk I/O operations from the server and reduces I/O wait time around 70% to provide better compute power to the Spark workload. Also, AFF systems provide good performance for in-memory workloads such as Spark. In addition to performance, NetApp provides a path to enterprise-grade features with simple management.

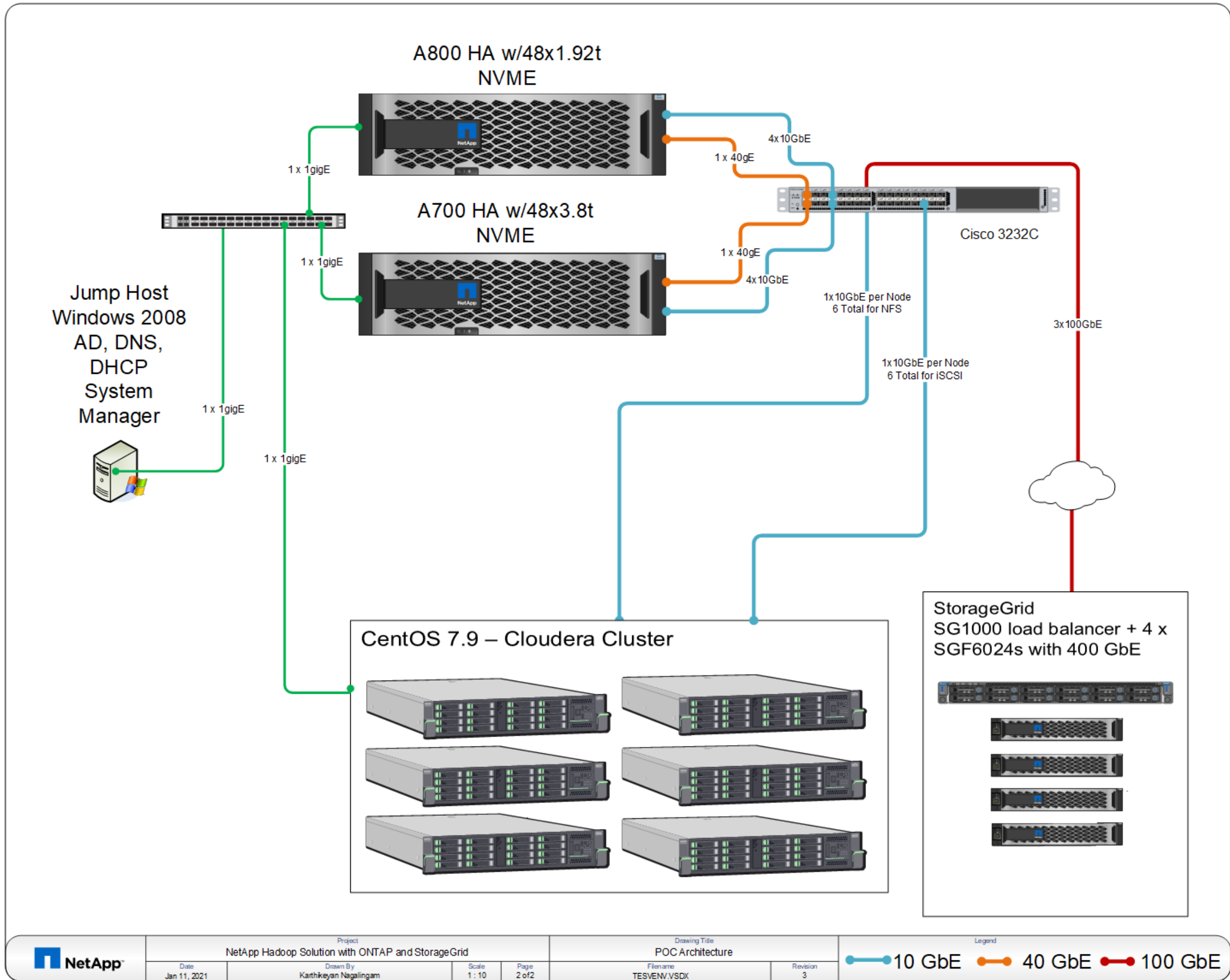
Solution technology

In this solution, we use AFF A800, A700 systems and StorageGRID for accessing parquet files through NFS and S3 protocols, a six node Hadoop cluster with Spark, and YARN and Hive metadata services for analytics operations. The Spark SQL queries are used for both write and read operations. In the read Spark SQL query, we use both caching and no caching to analyze the data from parquet files. From the best practice approach, we use two iSCSI LUNs from AFF A800 and AFF A700 systems for Hadoop temporary space for the Spark jobs. To simulate the customer data, we create parquet files by using a Spark SQL query of around 2.4 TB in size and validated customer requested performance tests with multiple scenarios through 2,4 and 6 Hadoop nodes and then collected the performance data.

Figure 1 shows the technical components of the solution.

© 2021 NetApp, Inc. All rights reserved.

Figure 1) Spark solution with NetApp Storage.



Use case summary

This document is based on a banking customer use case called the Quantitative Finance model. The quantitative finance group at the bank wants to separate the storage and compute and get better performance by reducing I/O wait time on the analytics tools, such as Hadoop and Spark, to run test scenarios, such as simulate investment, potential gains, and risks analysis.

This solution applies to the following use case:

- Quantitative finance analytics platform with NetApp storage

Quantitative finance analytics platform with NetApp storage

The quantitative finance group at the bank leverages fees collected as a result of servicing mortgages; those funds are reinvested by the quantitative finance group to create more revenue for the bank. In this process, they run tests against large amounts of datasets by using algorithms and analytics tools like Hadoop and Spark cluster to simulate the investment, potential gain, risk, and other factors. In the current customer environment, the Hadoop infrastructure used for the analytics platform leverages internal storage from the Hadoop servers. Due to the custom/proprietary nature of the JBOD environment, many internal customers are unable to take advantage of their Monte Carlo model, which is a simulation that

© 2021 NetApp, Inc. All rights reserved.

relies on the repetition of samples to achieve numerical results that are used to understand the effect of uncertainty and randomness in forecasting models. NetApp worked with the customer to design the modern analytics solution with NetApp storage that enables customers to make better informed and real-time decisions.

Why NetApp for this use case?

NetApp has been working closely with a quantitative finance line of business(LoB) group at a large global financial company, to develop a net-new storage environment to support their Hadoop/Spark analytics compute farm. Before moving towards a network attached storage solution, this team had a direct attached storage solution leveraging internal storage from HPE servers.

The NetApp account team has identified several reasons why the customer has decided to make this transition from a direct-attached storage (DAS) solution to an ONTAP NAS storage solution, which encompasses the following:

Transitioning from HDFS to NFS and creating a data lake

With cited issues using Hadoop Distributed File System (HDFS) at scale and the inability to create a data lake in their current environment, the LoB group had an interest in moving towards the NFS protocol. Using this protocol on ONTAP NetApp systems allows the group to have a data lake that can be shared among multiple Hadoop/Spark clusters. This helps to make sure compute and storage resources are not underutilized.

Scaling storage and compute independently

A DAS storage solution has the disadvantage of not allowing the quantitative finance group to scale compute and storage independently in the environment's current state. With a NetApp ONTAP® solution, this team can decouple storage from compute and more effectively bring on infrastructure resources as needed.

Multiprotocol support

With NetApp's best-in class ONTAP operating system, the quantitative finance group can leverage multiprotocol’s in this environment. Protocols of interest include NFS and S3, as well as ISCSI for intermediate storage for Spark workloads.

Trusted partnership

The LoB group that is the subject of this report is a long-time NetApp customer who has grown familiar with ONTAP's products and services. Using ONTAP since the software was identified as 7-mode gives the customer confidence that they can go back to ONTAP when identifying new use cases that require enterprise storage infrastructure.

ONTAP capabilities

NetApp's ONTAP operating system provides capabilities a DAS internal server solution is unable to provide. Space savings with ONTAP storage efficiencies enable the customer to leverage more with less, while ONTAP’s data protection capabilities provide disaster recovery and Encryption at Rest. For these reasons and others, ONTAP provides an optimal solution to make a Hadoop/Spark use case enterprise grade.

Technology requirements

This section covers the technology requirements for the Spark workload with NetApp storage solution.

© 2021 NetApp, Inc. All rights reserved.

Hardware requirements

Table 1 lists the hardware components that are required to implement the solution. The hardware components that are referenced below might vary based on customer requirements.

Table 1) Hardware requirements.

Hardware	Details
NetApp AFF storage array HA pair	• A800 and A700 • ONTAP 9.7P7 • 48 X 1.8 TB SSDs (A800)/3.8TB SSDs (A700) • Single NetApp ONTAP FlexGroup volume on 47 SSD disks
6 x IBM system x3650 M4 servers	• 40 CPUs • Intel(R) Xeon(R) CPU E5-2670 v2 @ 2.50GHz • 256GB physical memory • 1 x 10GbE network port
Networking	• 10GbE
StorageGRID	• 4 x SGF6024s • 4 x 24 x 800 GB SSDs

Software requirements

Table 2 lists the software components that are required to implement the solution. The software components that are referenced below might vary based on customer requirements.

Table 2) Software requirements.

Software	Details
Hadoop / Spark cluster	• Cloudera Distribution Hadoop (CDH)6.3.4 • 6 node cluster
Spark	• Version 2.4.0-cdh6.3.4 • Python 2.7.5 • Pyspark: Scala - 2.11.12, Java HotSpot (TM) 64-bit server virtual machine (VM), 1.8.0_181
MinIO	• RELEASE.2020-11-25T23-04-07Z
Hadoop NodeManager local directories (yarn.nodemanager.local-dirs.)	• /a700_lun/yarn/nm • /a800_lun/yarn/nm (iSCSI)
Hadoop NodeManager container local directories (yarn.nodemanager.log-dirs.)	• /a700_lun/yarn/container-logs • /a800_lun/yarn/container-logs (iSCSI)

Deployment procedures

Deploying the solution involves the following tasks:

- Hadoop/Spark cluster setup
- NetApp storage controller setup
- Spark with ONTAP
- MinIO NFS gateway
- Spark with StorageGRID

© 2021 NetApp, Inc. All rights reserved.

Hadoop/Spark cluster setup

To set up a Hadoop/Spark cluster, complete the following steps:

1. Configure the DNS entries in the DNS server for six nodes and validate the hostname resolve operation with the IP and vice versa by using `nslookup <ip or hostname>`
2. Follow the steps in the Cloudera Proof-of-Concept Installation Guide at https://docs.cloudera.com/documentation/enterprise/latest/topics/poc_installation.html

   Note: You need a Cloudera license for Cloudera 6.3.3 and higher.
3. Install the Cloudera software directly from internet.
4. Access the Cloudera Manager server from your browser and then choose the right components to install for your needs.

   For this validation, we chose Yarn, HDFS, Zookeeper, Spark, and Hive metastore.

   Cloudera 6.3.4 provides Hadoop 3.0.0-cdh6.3.4, which has an NFS driver to access the NFS data directly for MapReduce and Spark workloads. So, we have not modified any configuration in the core-site.xml or hdfs-site.xml for accessing NFS data.

   Note: We recommend that you use LDAP, Kerberos, and Active Directory for centralized user management. For this validation, we used operating system users. Make sure that the UID and GUID is the same for Yarn, Spark, HDFS, and MapRed users across all the servers in the Hadoop cluster. This is because users access the shared NFS data that uses the same UID and GUID. We used the `usermod` command to update the same UID and GUID for Hadoop system users such as Yarn, Spark, HDFS, and MapRed across all the nodes in the Hadoop cluster.
5. Update the Hadoop Node Manager local directories (`yarn.nodemanager.local-dirs`) and the Hadoop NodeManager container local directories (`yarn.nodemanager.log-dirs`) to point to NetApp iSCSI LUNs.

   You can change to FCP, or, if you have large disk space as well as SSD drives on the Hadoop server, you can leverage FCP-based SSD or local SSD drives. Make sure that the disk space is larger than the dataset size.

NetApp storage controller setup—ONTAP

To set up the NetApp storage controller, you must first have completed the following prerequisite tasks:

- Storage system initialization
- Storage virtual machine (SVM) creation
- Assignment of logical network interfaces
- NFS configuration and licensing

Next, complete the following steps:

1. Create one FlexGroup volume for NFSv3 and another for NFSv4.

   We created a Flexgroup volume across 47 SSDs and assigned one SSD for the storage controller root volume.
2. Check the NFS export policy for the FlexGroup volume has read/write permissions for the Hadoop work nodes network. If not, provide the `rw` permissions for the Hadoop nodes network.
3. Change the NFS read and write size to 1MB: `tcp-max-xfer-size` or `v3-tcp-maxwrite/read-size`.

   Note: For the NetApp storage Link Aggregation Control Protocol (LACP) configuration, we assigned three network interfaces per LACP from each storage controller in a two-storage controller HA pair to match six Hadoop worker nodes, then created three network LIFs per LACP from each storage controller.

© 2021 NetApp, Inc. All rights reserved.

Note: For non-LACP configuration, we created three network LIFs from three network interfaces from each storage controller to match six Hadoop worker nodes.

Note: In addition to the default NFS mount option, we included 256KB read (rsize) and write (wsize) options, which provides better performance compared to other rsize and wsize values.

4. Create the same folder name on all Hadoop worker nodes and mount the FlexGroup volume by using unique LIFs from each Hadoop server on the folder.

NetApp storage controller setup—StorageGRID

To complete the storage controller setup for StorageGRID, follow these steps:

1. To build the StorageGRID setup, use an SG1000 load balancer and four SGF6024s.
2. Create a bucket for Spark SQL parquet files.
3. You can use https for the validation.

   For our validation, we used http.
4. Create and collect the secret key, access ID, and S3a endpoint.

Before we started the validation with Spark SQL, we completed benchmark testing on the StorageGRID setup.

Table 3 shows the test results for cloud object storage bench load generators on six ESXi hosts each with 1x100Gbps uplink.

Table 3) Cloud object storage bench load generators on six ESXi hosts each with 1x100Gbps.

Object and working set	Test results
6MB objects, 286GB working set:	6.3GB/sec PUT @ 1,024 threads 18.3GB/sec GET @ 256 threads
300MB objects, 2.9TB working set:	7.9GB/sec PUT @ 96 threads 14.1GB/sec GET @ 60 threads

Solution verification overview

We completed three types of validation:

- **Spark SQL on ONTAP**. Provides easy management and better performance.
- **MinIO Gateway for NFS**. Provides security access to NFS data through S3a protocol.
- **Spark SQL with StorageGRID**. Provides access to low cost storage.

The following sections describe each one of these validation types in detail.

Spark SQL with ONTAP

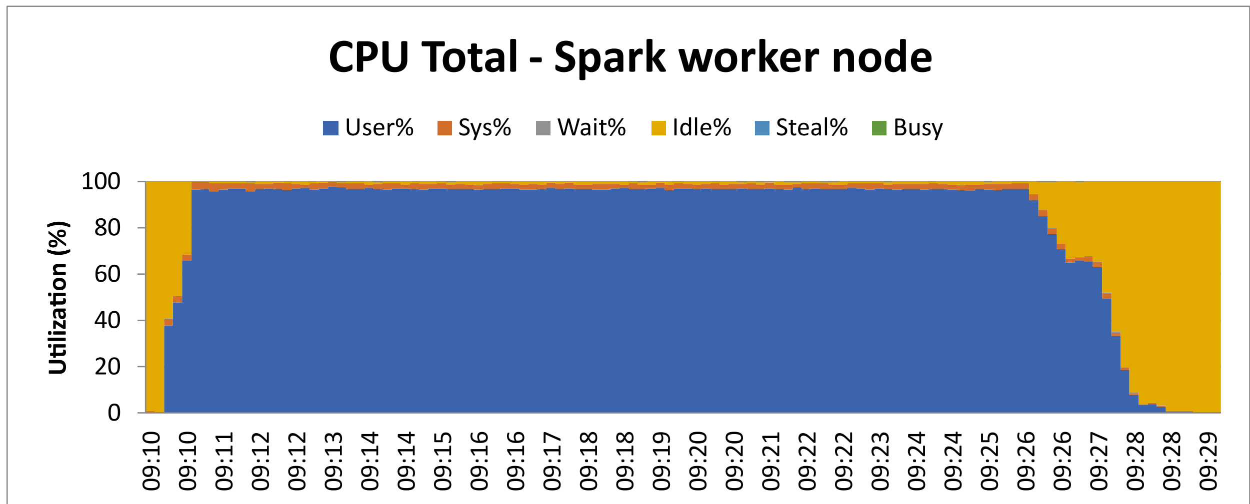
In this validation, we executed an SQL query to create and read parquet files that are in ONTAP FlexGroup NFS volumes on AFF A800 and A700 systems. The customer expectation is to see `iowait` at less than 25% on the Spark work nodes. As shown in Table 4 and Figure 2, we still found room for more throughput from the AFF A800, and the server's CPU utilization is almost 100%. By increasing the number of servers, we can get more throughput from the AFF A800.

© 2021 NetApp, Inc. All rights reserved.

Table 4) Parquet file creation write query summary—ONTAP.

Permutation	I/O wait (%)	Server CPU utilization (%)	Storage CPU utilization (%)	Query time (minutes)	NAS throughput (GB/sec)	Hadoop / Spark cluster throughput (GB/sec)
A700 (2 nodes)	~0	100	11	58	1.65	0.763
A700 (4 nodes)			25	28	2.4	1.5
A700 (6 nodes)			34	19	3.5	2.4
A800 (2 nodes)			9	56	1.8	0. 800
A800 (4 nodes)			20	28	2.57	1.5
A800 (6 nodes)			28	19	3.29	2.3

Figure 2) CPU utilization of Spark worker node.



Apache Spark with StorageGRID

For this validation, we used the configuration described in “NetApp storage controller setup—StorageGRID” to build the setup. The StorageGRID setup provides closer performance than the AFF A800, as shown in Table 5 and Table 6.

Table 5) Sequential read query—Cache count and groupby filter: ONTAP AFF A800.

Permutation	I/O wait (cache count, group by controller) %	Server CPU utilization (%)	Query time (minutes)	NAS throughput (GB/sec)	Hadoop / Spark cluster throughput (GB/sec)
A800 (2 node)	~0, ~32	~96	107	1	1.9
A800 (4 node)	~0, ~32	~95	64	2.2	3.2
A800 (6 node)	~1, ~29	~95	39	4	5.3

© 2021 NetApp, Inc. All rights reserved.

Table 6) Sequential read query—Cache count and groupby filter: StorageGRID.

Permutation	I/O wait (cache count, group by controller) %	Server CPU utilization	Query time (minutes)	StorageGRID network throughput (GB/sec)	Hadoop / Spark cluster throughput (GB/sec)
SGD (2 nodes)	~0, ~34	~95	142	2	1.2
SGD (4 nodes)	~0, ~35	~96	73	2.3	3.7
SGD (6 nodes)	~0, ~34	~95	49	3.5	8.3

Apache Spark workload through MinIO NFS Gateway

The banking customer requested to use the MinIO NFS Gateway with NFS data from the high performance NetApp storage controller because they wanted to access NFS data in a secure way by using credentials.

When you create parquet files through S3A protocol, the Spark or Hadoop cluster writes the data to a temporary location of the destination folder and then copies the files from the temporary folder to the destination folder, which takes more than twice the time to write the files to the 3 targets. However, in NFS protocol, the temporary folder moved to the destination folder that provides the job to complete the write operation in NFS is much faster than S3A protocol. The following tables show write and read query results based on 2.4TB parquet files using a Spark SQL query.

Table 7) Parquet file creation write query: MinIO NFS Gateway

Permutation	I/O wait %	Hadoop cluster CPU%	Storage CPU%	Write query time (minutes)	Storage throughp ut (GB/sec)	Hadoop / Spark cluster throughput (GB/sec
A800 (2-node)	~0	~100	9	176	1.79	1.7
A800 (4-node)			20	129	2.6	2.7
A800 (6-node)			28	92	3.28	3.3

Table 8) Sequential read query—Cache count and groupby filter: MinIO NFS Gateway

Permutation	I/O wait (cache count, group by controller) %	Hadoop cluster CPU%	Storage CPU%	Read query time (Minutes)	Storage throughp ut (GB/sec)	Hadoop / Spark cluster throughput (GB/sec
A700 (2-node)	~2, ~28	~83	~39	147	0.6	1
A700 (4-node)	~3, ~33	~90	~23	70	2	3.3
A700 (6-node)	~3, ~33	~87	~17	46	2.3	5.3
A800 (2-node)	~1, ~33	~89	~15	148	0.5	0.755
A800 (4-node)	~2, ~35	~87	~15	74	2	2.3
A800 (6-node)	~2, ~35	~85	~14	50	3.5	3.5

Based on the above three test validations, for the write operation, the CPU resource is almost 100% and the throughput on the Spark cluster is equal to or lower than the NAS storge controller throughput. For the read operation with cache, the Spark cluster processing in-memory, which leverages server memory and

© 2021 NetApp, Inc. All rights reserved.

CPU, and the throughput on the Spark cluster is equal to or greater than the NAS storge controller throughput.

Solution verification steps

In this section, we describe the detailed steps that are used for the solution verification for ONTAP and StorageGRID.

Pyspark SQL query for sequential write and read in ONTAP NFS and pNFS.

To run the Spark SQL query, follow these steps:

1. Export the environmental variable SPARK_LOCAL_DIRS to point to the NetApp iSCSI location and HADOOP_CONF_DIR to point to /etc/Hadoop/conf.

```
[root@linux4 a800_lun]# df -h | grep lun
/dev/mapper/3600a09803830466b322b512d68484e7a1  1007G  271G  685G  29% /a800_lun
/dev/mapper/3600a0980383041345924513951432f51p1 1007G  1.7G  955G   1% /a700_lun
```

   For A800 testcases:

```
[root@linux4 a800_lun]# export SPARK_LOCAL_DIRS=/a700_lun/spark_local_dir
```

   For A700 testcases:

```
[root@linux4 a800_lun]# export SPARK_LOCAL_DIRS=/a800_lun/spark_local_dir
```

2. Update the yarn-site.xml file for Hadoop NodeManager local directories and container local directories properties to point to NetApp iSCSI LUNs.

```
<property>
       <name>yarn.nodemanager.local-dirs</name>
       <value>/a700_lun/yarn/nm,/a800_lun/yarn/nm</value>
</property>
<property>
       <name>yarn.nodemanager.log-dirs</name>
       <value>/a700_lun/yarn/container-logs,/a800_lun/yarn/container-logs</value>
</property>
```

3. Make sure that the NetApp ONTAP NFS volume is mounted in all Spark worker nodes with the same folder and unique NetApp LIF IP as follows:

```
[root@linux5 script]# for i in 1 2 3 4 5 6; do ssh linux${i} "hostname; df -h | grep a800_nfs/
;"; done
linux1.cpoc.local
192.168.3.120:/a800                               19T  3.4T   16T  18% /a800_nfs/node1
linux2.cpoc.local
192.168.3.121:/a800                               19T  3.4T   16T  18% /a800_nfs/node1
linux3.cpoc.local
192.168.3.122:/a800                               19T  3.4T   16T  18% /a800_nfs/node1
linux4.cpoc.local
192.168.3.123:/a800                               19T  3.4T   16T  18% /a800_nfs/node1
linux5.cpoc.local
192.168.3.124:/a800                               19T  3.4T   16T  18% /a800_nfs/node1
linux6.cpoc.local
192.168.3.125:/a800                               19T  3.4T   16T  18% /a800_nfs/node1
[root@linux5 script]#
```

4. Force an immediate write of all cached data to disk by using the `sync` command
5. Clean memory, such as free page cache and free reclaimable slab objects, from slap objects and page cache using echo 1, 2, or 3 to `the /proc/sys/vm/drop_cache` file.
6. Turn the swap file system off and then on by using `swapoff -a` and `swapon -a`.
7. To collect the server statistics, start the nmon process.

© 2021 NetApp, Inc. All rights reserved.

8. Collect the storage statistics by using the `statistics show-periodic` command on the storage controller.
9. Run the Python query: `pyspark –master yarn`.

   The Spark SQL requests the resource manager to process the request and run containers in NodeManager.

   We used most of the resources (CPU and memory) from NodeManager to run the query. See the query in the “Appendix” section.

   – For the sequential write, we ran the Spark SQL query (cross join) to create parquet files around 2.4TB and 34 billion records by using Python script. (See the detailed script in “Appendix”.)

```
tdf1=df2.crossJoin(df1).drop(df1.id)
#For A800 write test
tdf1.write.mode("overwrite").parquet("file:///a800_nfs/node1/cj_nfs/")
#For A700 write test
#tdf1.write.mode("overwrite").parquet("file:///a700_nfs/node1/cj_nfs/")
#For A800 pNFS write test
#tdf1.write.mode("overwrite").parquet("file:///a800_nfs_pnfs/cj_pnfs/")
```

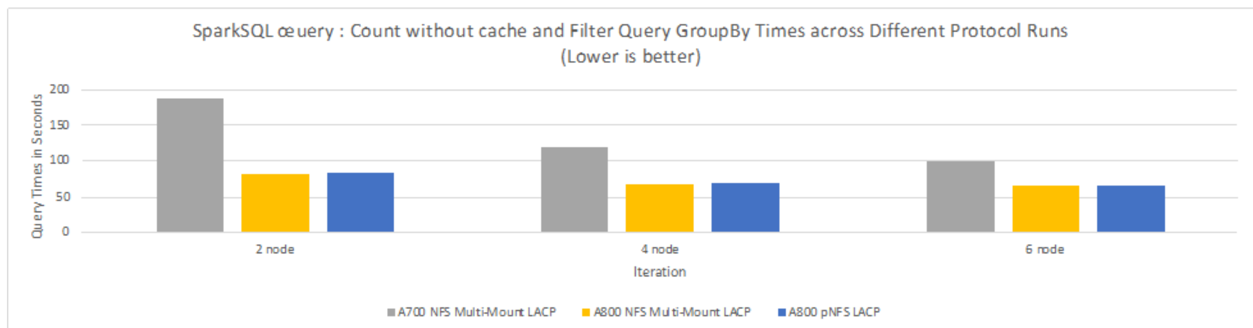
   – For the sequential read, we ran the Spark SQL query to count the number of rows with / without caching and groupby query.

```
#For A800 read test
df3 = sqlContext.read.parquet("file:///a800_nfs/node1/cj_nfs")
#For A700 read test
#df3 = sqlContext.read.parquet("file:///a700_nfs/node1/cj_nfs")
#For A800 pNFS read test
#df3 = sqlContext.read.parquet("file:///a800_nfs_pnfs/cj_pnfs")
# count query with caching
df3.cache().count()
# count query without caching
#df3.count()
#filter by groupby query
df3.filter( (df3.id == 1) | (df3.id == 2)).groupBy(df3.id).count().show()
```

   – We reused the same Spark SQL query for NFS, pNFS, and S3 (StorageGRID) locations.
10. Check and collect the storage CPU utilization and throughput data from the storage controller.
11. Check the Spark History UI to make sure that the Spark job is running and has no errors reported.
12. Note down the time to finish the job and stop the nmon process
13. Create graphs and reports based on the collected data.

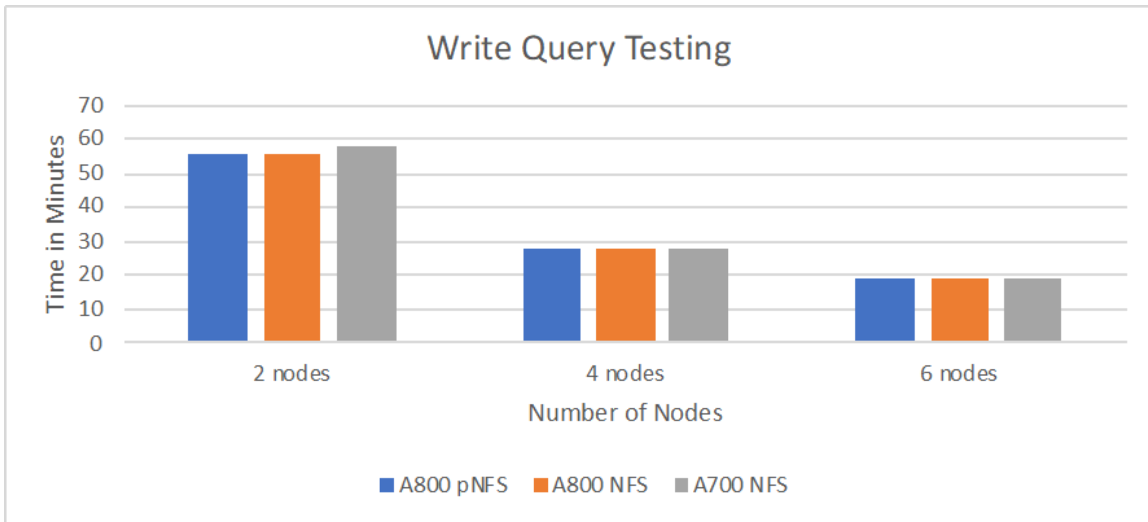
   Figure 3 and Figure 4 show the results of read and write query times collected in our solution verification.

Figure 3) Read query—Count without caching and groupby filter.



© 2021 NetApp, Inc. All rights reserved.

Figure 4) Write query times for the 2.4TB parquet creation for both NFS and pNFS.

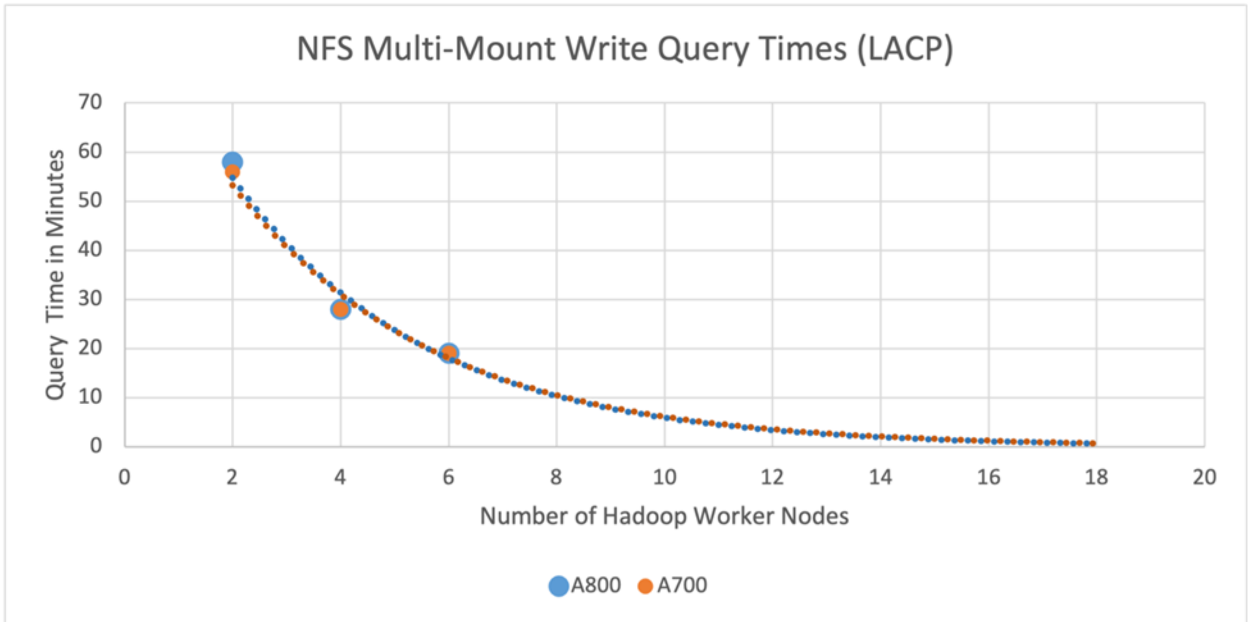


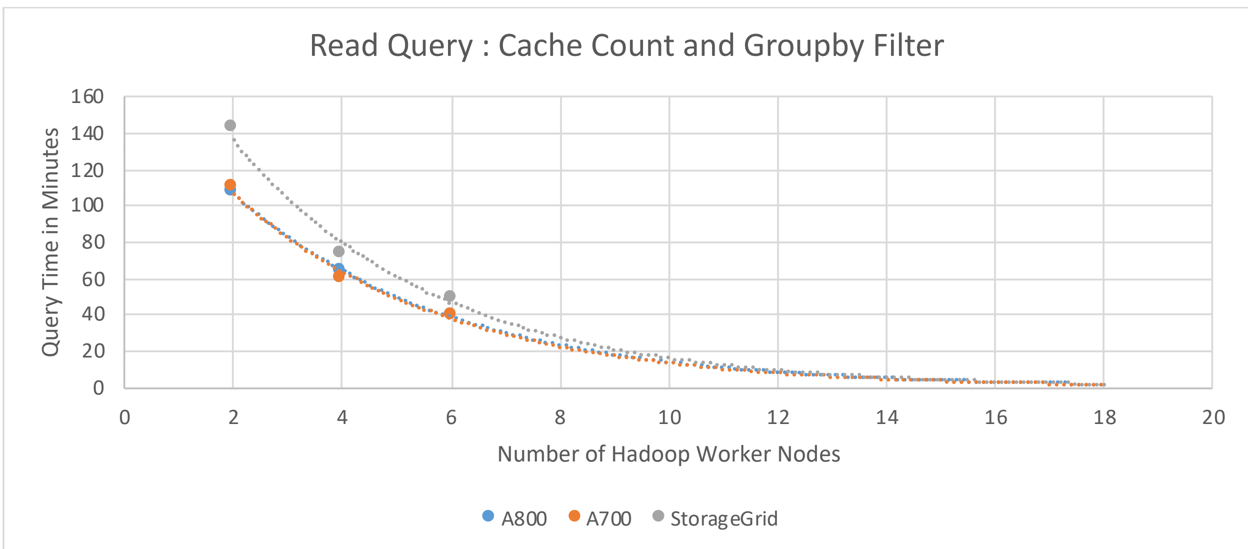
Our solution verification yielded the following results:

- Based on 2-, 4-, and 6-node iterations, extrapolated results to ten or more worker nodes will yield sub 10 minutes for a cache count and filter query.
- More than 34 billion rows were counted and filtered in sequential read queries.
- AFF A700/A800 systems produce similar query time results for 2-,4-, and 6-node iterations.
- StorageGRID query time results are different than AFF A700/A800 results for 2-,4-, and 6-node iterations.
- 256KB block size was used for both rsize and wsize in NFS mount options.
- StorageGRID and MinIO write tests are almost similar and take twice or more time to complete the operation, so we have not included the StorageGRID write result in the following graph.
- For the Spark SQL workload, the performance is almost equal for pNFS and NFS with multimount.

© 2021 NetApp, Inc. All rights reserved.

Figure 5) Write query and read query results with number of nodes.





Pyspark SQL query for sequential write and read in MinIO NFS Gateway on NFS

In this section, we describe the steps to access NFS data by using the MinIO S3 Gateway.

1. Mount the NetApp NFS export on each Spark worker node.

In our configuration, we used a separate port for each NetApp LIF and assigned separate IPs for each LIF for NFS data access.

```
[root@linux1 testbucket]# for i in 1 2 3 4 5 6; do ssh linux${i} "iptables -F ;hostname;  df -h |
grep /a800_nfs/node2 ";done
linux1.cpoc.local
192.168.3.121:/a800                                19T   6.3T   13T  34% /a800_nfs/node2
linux2.cpoc.local
192.168.3.120:/a800                                19T   6.3T   13T  34% /a800_nfs/node2
linux3.cpoc.local
```


© 2021 NetApp, Inc. All rights reserved.

```
192.168.3.121:/a800                              19T  6.3T   13T  34% /a800_nfs/node2
linux4.cpoc.local
192.168.3.121:/a800                              19T  6.3T   13T  34% /a800_nfs/node2
linux5.cpoc.local
192.168.3.121:/a800                              19T  6.3T   13T  34% /a800_nfs/node2
linux6.cpoc.local
192.168.3.121:/a800                              19T  6.3T   13T  34% /a800_nfs/node2
[root@linux1 testbucket]#
```

2. Download the MinIO software from https://min.io/, provide execute permission, and then copy the binary to all servers.

```
[root@linux4 script]# wget https://dl.min.io/server/minio/release/linux-amd64/minio
--2020-11-29 09:26:27--  https://dl.min.io/server/minio/release/linux-amd64/minio
Resolving dl.min.io (dl.min.io)... 178.128.69.202
Connecting to dl.min.io (dl.min.io)|178.128.69.202|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 48754688 (46M) [application/octet-stream]
Saving to: 'minio'

100%[================================================================================================
=================================================================================================
====================================================================================>]
48,754,688   583KB/s   in 88s

2020-11-29 09:27:55 (541 KB/s) - 'minio' saved [48754688/48754688]

[root@linux4 script]# chmod +x minio

[root@linux4 script]# for i in 1 2 3 4 5 6; do scp minio  linux${i}:/usr/local/bin; done
minio
100%  349   531.9KB/s   00:00
minio
100%  349   506.7KB/s   00:00
minio
100%  349   402.7KB/s   00:00
minio
100%  349   433.8KB/s   00:00
minio
100%  349   497.7KB/s   00:00
minio
100%  349   607.4KB/s   00:00
[root@linux4 script]#
```

3. Access the MinIO server through the load balancer.

```
[root@linux1 linux2_201126_1322.nmon]# cat /etc/hosts
#127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
#::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
10.61.114.151        linux1.cpoc.local linux1 minio-1 minIO
10.61.114.152        linux2.cpoc.local linux2 minio-2 minIO
10.61.114.153        linux3.cpoc.local linux3 minio-3 minIO
10.61.114.154        linux4.cpoc.local linux4 minio-4 minIO
10.61.114.155        linux5.cpoc.local linux5 minio-5 minIO
10.61.114.156        linux6.cpoc.local linux6 minio-6 minIO

[root@linux1 linux2_201126_1322.nmon]#
```

4. Create a `minio.service` file and copy to all Spark worker nodes.

```
[root@linux1 testbucket]# cat  /lib/systemd/system/minio.service

[Unit]

Description=minio

Documentation=https://docs.min.io

Wants=network-online.target

After=network-online.target
```

 © 2021 NetApp, Inc. All rights reserved.

```
AssertFileIsExecutable=/usr/local/bin/minio


[Service]

WorkingDirectory=/usr/local/

User=root

Group=root

EnvironmentFile=/etc/default/minio

ExecStartPre=/bin/bash -c "if [ -z \"${MINIO_VOLUMES}\" ]; then echo \"Variable MINIO_VOLUMES not set in /etc/default/minio\"; exit 1; fi"

ExecStart=/usr/local/bin/minio gateway nas $MINIO_OPTS $MINIO_VOLUMES

Restart=always

LimitNOFILE=65536

TimeoutStopSec=infinity

SendSIGKILL=no

StandardOutput=journal
StandardError=inherit


[Install]

WantedBy=multi-user.target

[root@linux1 testbucket]#
```

5. Update the `/etc/default/minio` file for NFS and in all Spark worker nodes.

```
[root@linux1 testbucket]# cat  /etc/default/minio
MINIO_VOLUMES="/a800_nfs/node2"
MINIO_OPTS="--address :9002"
MINIO_ACCESS_KEY=minioadmin
MINIO_SECRET_KEY=minioadmin
[root@linux1 ~]# for i in 1 2 3 4 5 6; do scp /etc/default/minio  linux${i}:/etc/default/minio; done
minio
100%  349   531.9KB/s   00:00
minio
100%  349   506.7KB/s   00:00
minio
100%  349   402.7KB/s   00:00
minio
100%  349   433.8KB/s   00:00
minio
100%  349   497.7KB/s   00:00
minio
100%  349   607.4KB/s   00:00
[root@linux1 ~]#
```

6. Enable, start, and stop the MinIO service.

```
[root@linux1 ~]# for i in 1 2 3 4 5 6; do ssh linux${i} "iptables -F ;hostname; systemctl daemon-reload"; done
linux1.cpoc.local
linux2.cpoc.local
linux3.cpoc.local
linux4.cpoc.local
linux5.cpoc.local
linux6.cpoc.local
```

 © 2021 NetApp, Inc. All rights reserved.

```
[root@linux1 ~]# for i in 1 2 3 4 5 6; do ssh linux${i} "iptables -F ;hostname;  systemctl enable minio"; done
linux1.cpoc.local
Created symlink from /etc/systemd/system/multi-user.target.wants/minio.service to /usr/lib/systemd/system/minio.service.
linux2.cpoc.local
Created symlink from /etc/systemd/system/multi-user.target.wants/minio.service to /usr/lib/systemd/system/minio.service.
linux3.cpoc.local
Created symlink from /etc/systemd/system/multi-user.target.wants/minio.service to /usr/lib/systemd/system/minio.service.
linux4.cpoc.local
Created symlink from /etc/systemd/system/multi-user.target.wants/minio.service to /usr/lib/systemd/system/minio.service.
linux5.cpoc.local
Created symlink from /etc/systemd/system/multi-user.target.wants/minio.service to /usr/lib/systemd/system/minio.service.
linux6.cpoc.local
Created symlink from /etc/systemd/system/multi-user.target.wants/minio.service to /usr/lib/systemd/system/minio.service.
[root@linux1 ~]# for i in 1 2 3 4 5 6; do ssh linux${i} "iptables -F ;hostname;  systemctl stop minio.service"; done
linux1.cpoc.local
linux2.cpoc.local
linux3.cpoc.local
linux4.cpoc.local
linux5.cpoc.local
linux6.cpoc.local
[root@linux1 ~]# for i in 1 2 3 4 5 6; do ssh linux${i} "iptables -F ;hostname;  systemctl start minio.service"; done
linux1.cpoc.local
linux2.cpoc.local
linux3.cpoc.local
linux4.cpoc.local
linux5.cpoc.local
linux6.cpoc.local
[root@linux1 ~]#
```

7. Update the core-site.xml file for MinIO S3 configuration by using Cloudera Manager.

```
  <property>
    <name>fs.s3a.endpoint</name>
    <value>http://minIO:9002/</value>
  </property>
  <property>
    <name>fs.s3a.access.key</name>
    <value>minioadmin</value>
  </property>
  <property>
    <name>fs.s3a.secret.key</name>
    <value>minioadmin</value>
  </property>
  <property>
    <name>fs.s3a.path.style.access</name>
    <value>true</value>
  </property>
  <property>
    <name>fs.s3a.impl</name>
    <value>org.apache.hadoop.fs.s3a.S3AFileSystem</value>
  </property>
  <property>
    <name>fs.s3a.connection.ssl.enabled</name>
    <value>false</value>
  </property>
```

8. Perform basic testing to create a folder and then list the folder.

```
[root@linux1 node2]# hadoop fs -mkdir  s3a://miniotest/miniocreatefolder
20/11/30 13:51:24 WARN impl.MetricsConfig: Cannot locate configuration: tried hadoop-metrics2-s3a-file-system.properties,hadoop-metrics2.properties
20/11/30 13:51:24 INFO impl.MetricsSystemImpl: Scheduled Metric snapshot period at 10 second(s).
```


© 2021 NetApp, Inc. All rights reserved.

```
20/11/30 13:51:24 INFO impl.MetricsSystemImpl: s3a-file-system metrics system started
20/11/30 13:51:26 INFO impl.MetricsSystemImpl: Stopping s3a-file-system metrics system...
20/11/30 13:51:26 INFO impl.MetricsSystemImpl: s3a-file-system metrics system stopped.
20/11/30 13:51:26 INFO impl.MetricsSystemImpl: s3a-file-system metrics system shutdown complete.
[root@linux1 node2]# hadoop fs -ls  s3a://miniotest/
20/11/30 13:51:37 WARN impl.MetricsConfig: Cannot locate configuration: tried hadoop-metrics2-
s3a-file-system.properties,hadoop-metrics2.properties
20/11/30 13:51:37 INFO impl.MetricsSystemImpl: Scheduled Metric snapshot period at 10 second(s).
20/11/30 13:51:37 INFO impl.MetricsSystemImpl: s3a-file-system metrics system started
Found 1 items
drwxrwxrwx   - root root          0 2020-11-30 13:51 s3a://miniotest/miniocreatefolder
20/11/30 13:51:38 INFO impl.MetricsSystemImpl: Stopping s3a-file-system metrics system...
20/11/30 13:51:38 INFO impl.MetricsSystemImpl: s3a-file-system metrics system stopped.
20/11/30 13:51:38 INFO impl.MetricsSystemImpl: s3a-file-system metrics system shutdown complete.
[root@linux1 node2]#
```

We used the Python script to create a parquet file and ran the read query by counting the number of rows with or without cache and groupby filter query.

– For the sequential write, we ran the Spark SQL query (cross join) to create parquet files around 2.4TB and 34 billion records by using the Python script. (See the detailed script in “Appendix”.)

```
tdf1=df2.crossJoin(df1).drop(df1.id)
#For minio write test
tdf1.write.mode("overwrite").parquet("s3a://miniowrite/parq")
```

For sequential read, we ran the Spark SQL query to count the number of rows with / without caching and groupby query.

```
#For minio read test
df3 = sqlContext.read.parquet("s3a://miniowrite/parq/")
# count query with caching
df3.cache().count()
# count query without caching
#df3.count()
#filter by groupby query
df3.filter( (df3.id == 1) | (df3.id == 2)).groupBy(df3.id).count().show()
```

Figure 6 shows the number of servers required to reach the maximum throughput of the single AFF A800 HA pair as well as the time to complete the parquet file’s creation by using MinIO NFS gateway. Our testing yielded the following results:

– Based on 2-, 4-, and 6-node iterations, extrapolated results to 16 or more worker nodes will yield sub 20 minutes for write query times.

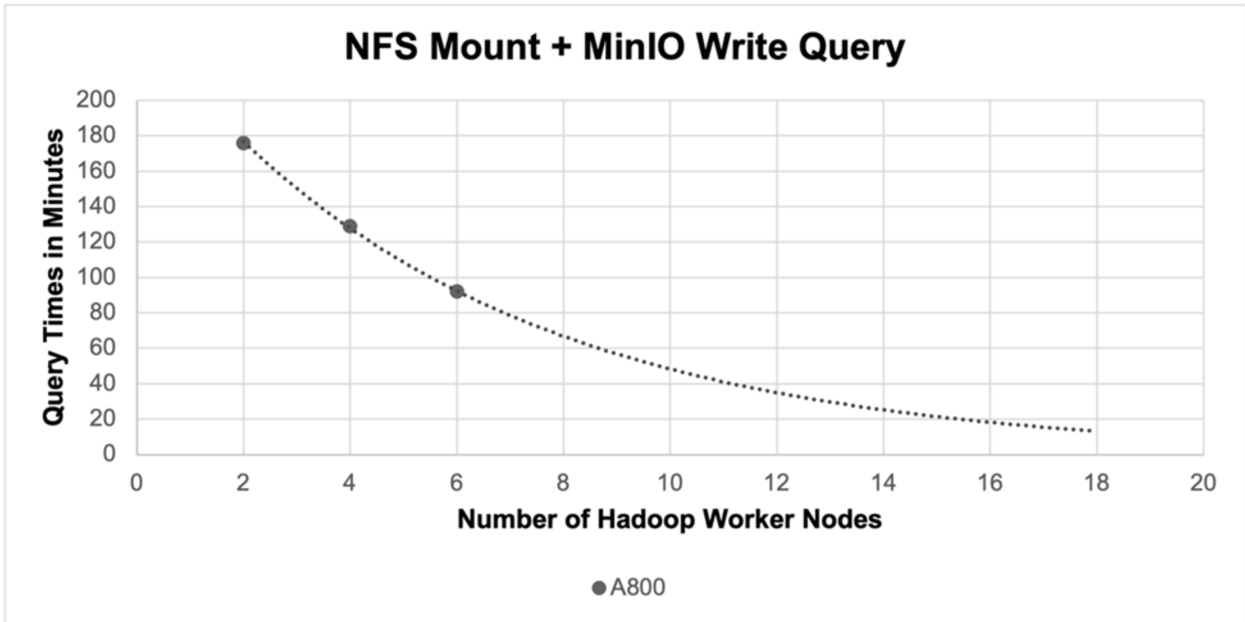
– Parquet file created 2.4TB file dataset size

– 256KB block size used for both rsize and wsize in NFS mount options

– The MinIO write result for AFF A700 is almost similar to A800, so we did not test it.

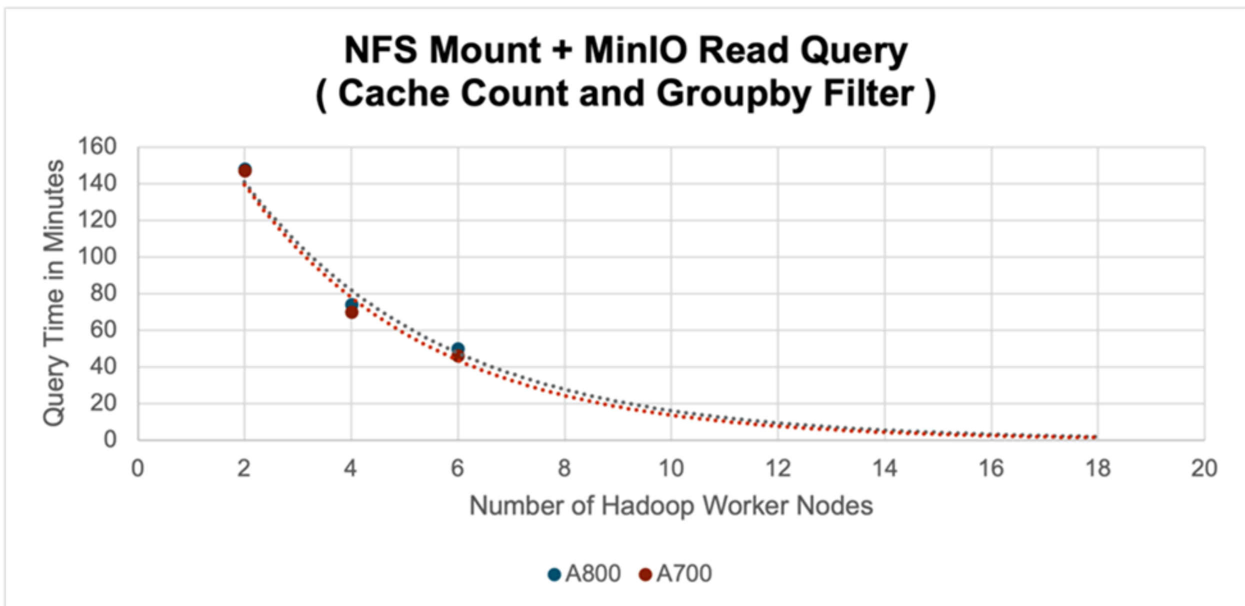
© 2021 NetApp, Inc. All rights reserved.

Figure 6) Write query results with MinIO and number of nodes.



- Based off 2-, 4-, and 6-node iterations, extrapolated results to ten or more worker nodes will yield sub 10 minutes for a cache count and filter query
- More than 34 billion rows were counted and filtered in sequential read queries
- AFF A700/A800 arrays produce similar query time results for 2-,4-, and 6-node iterations

Figure 7) Read query results with MinIO and number of nodes.



© 2021 NetApp, Inc. All rights reserved.

JBOD versus AFF A800 lab validation for Spark workload

This section provides the details about the JBOD versus the AFF A800 storage array comparison for Spark workloads based on a six-node Spark cluster.

Table 9) Hardware details used for lab validation.

Hardware	Details
NetApp AFF storage array HA pair	• AFF A800 • ONTAP 9.7P7X3 • 48 x 3.49TB SSDs • Single FlexGroup volume on 48 SSD disks
6 x FUJITSU Server PRIMERGY RX2540 M2 systems	• 48 CPUs • Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz • 256GB physical memory • 1 x 10GbE network port • 8 x SATA disks per server. (48 disks)
Networking	• 10GbE

The following scripts were used for the validation:

For JBOD, we used eight SATA disks from each server:

```
[root@n136 ~]# pssh -h /root/hosts -i "df -h /data[1-8]/"
[1] 11:48:21 [SUCCESS] 10.63.150.140
Filesystem      Size  Used Avail Use% Mounted on
/dev/sdb1       3.7T  182G  3.5T   5% /data1
/dev/sdc1       3.7T  182G  3.5T   5% /data2
/dev/sdd1       3.7T  182G  3.5T   5% /data3
/dev/sde1       3.7T  182G  3.5T   5% /data4
/dev/sdf1       3.7T  183G  3.5T   5% /data5
/dev/sdg1       3.7T  182G  3.5T   5% /data6
/dev/sdh1       3.7T  182G  3.5T   5% /data7
/dev/sdi1       3.7T  183G  3.5T   5% /data8
[2] 11:48:21 [SUCCESS] 10.63.150.138
Filesystem      Size  Used Avail Use% Mounted on
/dev/sdb1       3.7T  182G  3.5T   5% /data1
/dev/sdc1       3.7T  182G  3.5T   5% /data2
/dev/sdd1       3.7T  182G  3.5T   5% /data3
/dev/sde1       3.7T  182G  3.5T   5% /data4
/dev/sdf1       3.7T  182G  3.5T   5% /data5
/dev/sdg1       3.7T  182G  3.5T   5% /data6
/dev/sdh1       3.7T  182G  3.5T   5% /data7
/dev/sdi1       3.7T  182G  3.5T   5% /data8
[3] 11:48:21 [SUCCESS] 10.63.150.142
Filesystem      Size  Used Avail Use% Mounted on
/dev/sdb1       3.7T  176G  3.5T   5% /data1
/dev/sdc1       3.7T  176G  3.5T   5% /data2
/dev/sdd1       3.7T  176G  3.5T   5% /data3
/dev/sde1       3.7T  177G  3.5T   5% /data4
/dev/sdf1       3.7T  174G  3.5T   5% /data5
/dev/sdg1       3.7T  173G  3.5T   5% /data6
/dev/sdh1       3.7T  170G  3.5T   5% /data7
/dev/sdi1       3.7T  172G  3.5T   5% /data8
[4] 11:48:21 [SUCCESS] 10.63.150.146
Filesystem      Size  Used Avail Use% Mounted on
/dev/sdb1       3.7T  141G  3.6T   4% /data1
/dev/sdc1       3.7T  137G  3.6T   4% /data2
/dev/sdd1       3.7T  139G  3.6T   4% /data3
/dev/sde1       3.7T  145G  3.5T   4% /data4
/dev/sdf1       3.7T  144G  3.5T   4% /data5
/dev/sdg1       3.7T  140G  3.6T   4% /data6
/dev/sdh1       3.7T  139G  3.6T   4% /data7
```


© 2021 NetApp, Inc. All rights reserved.

```
/dev/sdi1       3.7T  141G  3.6T   4% /data8
[5] 11:48:21 [SUCCESS] 10.63.150.144
Filesystem      Size  Used Avail Use% Mounted on
/dev/sdb1       3.7T  148G  3.5T   4% /data1
/dev/sdc1       3.7T  145G  3.5T   4% /data2
/dev/sdd1       3.7T  144G  3.5T   4% /data3
/dev/sde1       3.7T  143G  3.5T   4% /data4
/dev/sdf1       3.7T  140G  3.6T   4% /data5
/dev/sdg1       3.7T  142G  3.5T   4% /data6
/dev/sdh1       3.7T  143G  3.5T   4% /data7
/dev/sdi1       3.7T  145G  3.5T   4% /data8
[6] 11:48:21 [SUCCESS] 10.63.150.136
Filesystem      Size  Used Avail Use% Mounted on
/dev/sdb1       3.7T   92G  3.6T   3% /data1
/dev/sdc1       3.7T   95G  3.6T   3% /data2
/dev/sdd1       3.7T   97G  3.6T   3% /data3
/dev/sde1       3.7T   97G  3.6T   3% /data4
/dev/sdf1       3.7T   96G  3.6T   3% /data5
/dev/sdg1       3.7T   92G  3.6T   3% /data6
/dev/sdh1       3.7T   95G  3.6T   3% /data7
/dev/sdi1       3.7T   98G  3.6T   3% /data8
[root@n136 ~]#
```

For AFF A800 NFS, we used a single FlexGroup volume from 48 SSDs and mounted on different logical IPs from a NetApp storage controller:

```
[root@n136 ~]# pssh -h /root/hosts -i "df -h /xcphdfstest"
[1] 11:50:43 [SUCCESS] 10.63.150.136
Filesystem                 Size  Used Avail Use% Mounted on
10.63.150.90:/xcphdfstest  3.4T  656G  2.8T  19% /xcphdfstest
[2] 11:50:43 [SUCCESS] 10.63.150.138
Filesystem                 Size  Used Avail Use% Mounted on
10.63.150.91:/xcphdfstest  3.4T  656G  2.8T  19% /xcphdfstest
[3] 11:50:43 [SUCCESS] 10.63.150.144
Filesystem                 Size  Used Avail Use% Mounted on
10.63.150.96:/xcphdfstest  3.4T  656G  2.8T  19% /xcphdfstest
[4] 11:50:43 [SUCCESS] 10.63.150.140
Filesystem                 Size  Used Avail Use% Mounted on
10.63.150.92:/xcphdfstest  3.4T  656G  2.8T  19% /xcphdfstest
[5] 11:50:43 [SUCCESS] 10.63.150.142
Filesystem                 Size  Used Avail Use% Mounted on
10.63.150.93:/xcphdfstest  3.4T  656G  2.8T  19% /xcphdfstest
[6] 11:50:43 [SUCCESS] 10.63.150.146
Filesystem                 Size  Used Avail Use% Mounted on
10.63.150.97:/xcphdfstest  3.4T  656G  2.8T  19% /xcphdfstest
[root@n136 ~]#
```

The scripts in the section “Scripts used for ONTAP” were used for AFF A800 and JBOD validation with the below modifications.

- For sequential write operations, we ran the Spark SQL query (cross join) to create parquet files around 2.4TB and 34 billion records by using Python script (See the detailed script in Appendix)

```
tdf1=df2.crossJoin(df1).drop(df1.id)
#For A800 write test
tdf1.write.mode("overwrite").parquet("file:///xcphdfstest/cj_nfs")
#For JBOD write test
#tdf2.write.mode("overwrite").parquet("cj_hdfs")
```

- For sequential read operations, we ran the Spark SQL query to count the number of rows with and without caching and filter groupby query.

```
#For a800 read test
df3 = sqlContext.read.parquet("file:///xcphdfstest/cj_nfs")
#For JBOD read test
#df3 = sqlContext.read.parquet("cj_hdfs")
# count query with caching
df3.cache().count()
# count query without caching
#df3.count()
```


© 2021 NetApp, Inc. All rights reserved.

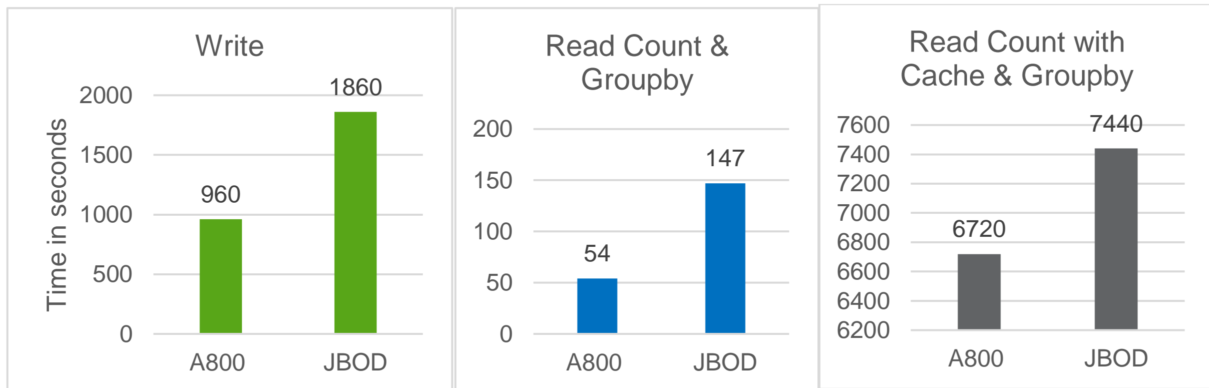
```
#filter by groupby query
df3.filter( (df3.id == 1) | (df3.id == 2)).groupBy(df3.id).count().show()
```

The script in the section “Main script for ONTAP, StorageGRID and MinIO” was used to collect the nmon, storage statistics and run the Spark SQL write and read queries for AFF A800 and JBOD.

The CLI in the section “Command to run the tests” was used to run the tests.

Based on the Pyspark test parquet files results from our lab, the AFF A800 NFS performance is better than the JBOD results.

Figure 8) JBOD versus AFF A800 write and read query results on a six-node Spark cluster.



The above results might vary based on the infrastructure, hardware, software, and type of data used.

Conclusion

In this document, we have provided the performance and functionality validation of Apache Spark on NetApp NFS AFF all-flash storage. We have also covered the configuration, architecture, performance testing based on the test scenarios, and takeaway recommendations for using Spark with ONTAP data management software.

You can use ONTAP with NFS to separate disk I/O operations from the server and reduce I/O wait time by around 70%, providing better compute power and better performance to Spark workloads and enterprise-grade features with simple management. MinIO Gateway provides access to NFS data through S3 protocol and secure access. StorageGrid provides an additional file system and low cost storage option for Spark workloads. If the customer data is in the cloud, the customer can leverage Azure NetApp Files, NetApp Cloud Volumes Service, and Cloud Volumes ONTAP. Based on our validation for Spark SQL, the server CPUs were almost used. In this case, you can either leverage GPUs, such as GPUDirect, by using RAPID, or increase the number of servers with CPUs.

We ran this solution in the NetApp Customer Proof Of Concept lab and presented the results to the customer. We would like to share this solution to other customers and field engineers who might have similar challenges.

Appendix

Scripts used for ONTAP

```
[root@n136 linux5_usr_src_with_old_linux4]# cat
/usr/src/linux5_usr_src_with_old_linux4/create_parquet_on_a800.py
#!/usr/bin/env python
import sys
```


© 2021 NetApp, Inc. All rights reserved.

```
import os
from pyspark.sql import SparkSession
from pyspark.sql.types import *
from pyspark.sql import functions as F


# Cambiamos el encoding a UTF-8 para que no de problemas
reload(sys)
sys.setdefaultencoding('utf8')

## Creamos la sesion de SPARK (Spark2)
spark = SparkSession.builder \
    .master("yarn") \
    .appName("create_POC_dataset_JBOD") \
    .config("spark.executor.cores", "1") \
    .config('spark.executor.memory', '5gb') \
    .config('spark.executor.memoryOverhead','1000')\
    .config('spark.driver.memoryOverhead','1000')\
    .config("spark.sql.shuffle.partitions", "480")\
    .getOrCreate()

# Quitamos el verbosity en la salida de la ejecucion de SPARK
spark.sparkContext.setLogLevel('WARN')

#instead of 240 files, 480 files.
#spark.sparkContext.setConf("spark.sql.shuffle.partitions", "480")

df1 = spark.range(1 << 18).toDF("id")\
    .withColumn("1a", F.rand())\
    .withColumn("a1", F.rand())\
    .withColumn("a2", F.rand())\
    .withColumn("a3", F.rand())\
    .withColumn("a4", F.rand())\
    .withColumn("a5", F.rand())\
    .withColumn("a6", F.rand())\
    .withColumn("a7", F.rand())\
    .withColumn("a8", F.rand())

df2 = spark.range(1 << 17).toDF("id")\
    .withColumn("b", F.rand())\
    .withColumn("b1", F.rand())\
    .withColumn("b2", F.rand())\
    .withColumn("b3", F.rand())\
    .withColumn("b4", F.rand())\
    .withColumn("b5", F.rand())\
    .withColumn("b6", F.rand())\
    .withColumn("b7", F.rand())\
    .withColumn("c1", F.rand())\
    .withColumn("c2", F.rand())\
    .withColumn("c3", F.rand())\
    .withColumn("c4", F.rand())\
    .withColumn("c5", F.rand())\
    .withColumn("c6", F.rand())\
    .withColumn("c7", F.rand())\
    .withColumn("c8", F.rand())\
    .withColumn("c9", F.rand())\
    .withColumn("d1", F.rand())\
    .withColumn("d2", F.rand())\
    .withColumn("d3", F.rand())\
    .withColumn("d4", F.rand())\
    .withColumn("d5", F.rand())\
    .withColumn("d6", F.rand())\
    .withColumn("d7", F.rand())


tdf1=df2.crossJoin(df1).drop(df1.id)

#tdf2 = tdf1.repartition(480)
tdf1.write.mode("overwrite").parquet("file:///a800_nfs/node1/cj_nfs_yarn")
#tdf1.write.mode("overwrite").parquet("file:///a700_nfs/node1/cj_nfs_yarn")
#tdf1.write.mode("overwrite").parquet("file:///a800_nfs_pnfs/cj_pnfs_yarn")
```

 © 2021 NetApp, Inc. All rights reserved.

```
SparkSession.stop


[root@n136 linux5_usr_src_with_old_linux4]#
[root@n136 linux5_usr_src_with_old_linux4]# cat /usr/src/linux5_usr_src_with_old_linux4/pyspark-
customer_a800.py
from pyspark.sql import SparkSession

spark = SparkSession.builder \
    .master('yarn') \
    .appName('BofaPySpark') \
    .config("spark.executor.cores", "1") \
    .config('spark.executor.memory', '5gb') \
    .config('spark.executor.memoryOverhead','1000')\
    .config('spark.driver.memoryOverhead','1000')\
    .getOrCreate()
    #.config("spark.executor.cores", "2") \
    #.config('spark.executor.memory', '14gb') \
    #.config("spark.executor.cores", "5") \
    #.config('spark.executor.memory', '19gb') \


#.config('spark.yarn.executor.memoryOverhead','')

sc = spark.sparkContext

# using SQLContext to read parquet file
from pyspark.sql import SQLContext
sqlContext = SQLContext(sc)

#nfs based
#df3 = sqlContext.read.parquet("file:///a800_nfs_pnfs/cj_pnfs_yarn")
#df3 = sqlContext.read.parquet("file:///a700_nfs_pnfs/cj_nfs_yarn")
df3 = sqlContext.read.parquet("file:///a800_nfs/node1/cj_nfs_yarn")

#cache count query
df3.cache().count()

#no cache count query
#df3.count()

df3.filter( (df3.id == 1) | (df3.id == 2)).groupBy(df3.id).count().show()
[root@n136 linux5_usr_src_with_old_linux4]#
```

Scripts used for StorageGRID

```
[root@n136 linux5_usr_src_with_old_linux4]# cat create_parquet_on_sgd.py
#!/usr/bin/env python
import sys
import os
from pyspark.sql import SparkSession
from pyspark.sql.types import *
from pyspark.sql import functions as F


# Cambiamos el encoding a UTF-8 para que no de problemas
reload(sys)
sys.setdefaultencoding('utf8')

## Creamos la sesion de SPARK (Spark2)
spark = SparkSession.builder \
    .master("yarn") \
    .appName("create_dataset_sgd") \
    .config("spark.executor.cores", "1") \
    .config("spark.dynamicAllocation.enabled", "false") \
    .config("spark.executor.heartbeatInterval", "200000") \
    .config("spark.network.timeout", "300000") \
    .config("spark.tmp.dir", "/a700_lun/spark_local_dir,/a800_lun/spark_local_dir") \
```

 © 2021 NetApp, Inc. All rights reserved.

```
  .config("spark.hadoop.fs.s3a.buffer.dir","/a700_lun/spark_local_dir,/a800_lun/spark_local_dir")
\
    .config("spark.hadoop.fs.s3a.endpoint", "http://webscalenext.netapp.com:10080") \
    .config("spark.hadoop.fs.s3a.access.key", "1D5HZTTZZ7XIJ12NF9ZF") \
    .config("spark.hadoop.fs.s3a.secret.key", "GXN6MaLW6lIVzjrIGVyitycCfbUjFJKrA47sXWvP") \
    .config("spark.hadoop.fs.s3a.path.style.access", True) \
    .config("spark.hadoop.fs.s3a.impl", "org.apache.hadoop.fs.s3a.S3AFileSystem") \
    .getOrCreate()

# Quitamos el verbosity en la salida de la ejecucion de SPARK
spark.sparkContext.setLogLevel('WARN')
#spark.sparkContext.setLogLevel('DEBUG')


#from pyspark import SparkContext, SparkConf, SQLContext
#import os
#conf = (
#    SparkConf()
#    .setAppName("Spark minIO Test")
#    .set("spark.hadoop.fs.s3a.endpoint", "http://minIO:9002")
#    .set("spark.hadoop.fs.s3a.access.key", "minioadmin")
#    .set("spark.hadoop.fs.s3a.secret.key", "minioadmin")
#    .set("spark.hadoop.fs.s3a.path.style.access", True)
#    .set("spark.hadoop.fs.s3a.impl", "org.apache.hadoop.fs.s3a.S3AFileSystem")
#)
#sc = SparkContext(conf=conf).getOrCreate()
#sqlContext = SQLContext(sc)



#df1 = spark.range(1 << 18).toDF("id")\
df1 = spark.range(1 << 10).toDF("id")\
    .withColumn("1a", F.rand())\
    .withColumn("a1", F.rand())\
    .withColumn("a2", F.rand())\
    .withColumn("a3", F.rand())\
    .withColumn("a4", F.rand())\
    .withColumn("a5", F.rand())\
    .withColumn("a6", F.rand())\
    .withColumn("a7", F.rand())\
    .withColumn("a8", F.rand())

#df2 = spark.range(1 << 17).toDF("id")\
df2 = spark.range(1 << 10).toDF("id")\
    .withColumn("b", F.rand())\
    .withColumn("b1", F.rand())\
    .withColumn("b2", F.rand())\
    .withColumn("b3", F.rand())\
    .withColumn("b4", F.rand())\
    .withColumn("b5", F.rand())\
    .withColumn("b6", F.rand())\
    .withColumn("b7", F.rand())\
    .withColumn("c1", F.rand())\
    .withColumn("c2", F.rand())\
    .withColumn("c3", F.rand())\
    .withColumn("c4", F.rand())\
    .withColumn("c5", F.rand())\
    .withColumn("c6", F.rand())\
    .withColumn("c7", F.rand())\
    .withColumn("c8", F.rand())\
    .withColumn("c9", F.rand())\
    .withColumn("d1", F.rand())\
    .withColumn("d2", F.rand())\
    .withColumn("d3", F.rand())\
    .withColumn("d4", F.rand())\
    .withColumn("d5", F.rand())\
    .withColumn("d6", F.rand())\
    .withColumn("d7", F.rand())


tdf1=df2.crossJoin(df1).drop(df1.id)
```


© 2021 NetApp, Inc. All rights reserved.

```
tdf1.write.mode("overwrite").parquet("s3a://sgdtest/cj_sgd")

#df3 = sqlContext.read.parquet("s3a://minioperftest/")

SparkSession.stop


[root@n136 linux5_usr_src_with_old_linux4]#
[root@n136 linux5_usr_src_with_old_linux4]# cat sgd_pyspark-test2.py
from pyspark import SparkConf,SparkContext
from pyspark.sql import SparkSession

spark = SparkSession.builder.getOrCreate()
conf=SparkConf()
sc = spark.sparkContext
from pyspark.sql import SQLContext
hadoopConf = sc._jsc.hadoopConfiguration()
hadoopConf.set("fs.s3a.secret.key","GXN6MaLW6lIVzjrIGVyitycCfbUjFJKrA47sXWvP")
hadoopConf.set("fs.s3a.access.key", "1D5HZTTZZ7XIJ12NF9ZF")
hadoopConf.set("fs.s3a.endpoint","http://webscalenext.netapp.com:10080")
hadoopConf.set("fs.s3a.path.style.access", "true")
hadoopConf.set("fs.s3a.impl", "org.apache.hadoop.fs.s3a.S3AFileSystem")
hadoopConf.set("fs.s3a.connection.ssl.enabled", "false")
sqlContext = SQLContext(sc)
df3 = sqlContext.read.parquet("s3a://sgdtest/cj_sgd")

df3.cache().count()
df3.filter( (df3.id == 1) | (df3.id == 2)).groupBy(df3.id).count().show()

[root@n136 linux5_usr_src_with_old_linux4]#
```

Scripts used for MinIO

```
[root@n136 linux5_usr_src_with_old_linux4]# cat create_parquet_on_nfs_via_minio.py
#!/usr/bin/env python
import sys
import os
from pyspark.sql import SparkSession
from pyspark.sql.types import *
from pyspark.sql import functions as F


# Cambiamos el encoding a UTF-8 para que no de problemas
reload(sys)
sys.setdefaultencoding('utf8')

## Creamos la sesion de SPARK (Spark2)
spark = SparkSession.builder \
    .master("yarn") \
    .appName("create_POC_dataset_NFS_via_minio_s3") \
    .config("spark.executor.cores", "1") \
    .config("spark.dynamicAllocation.enabled", "false") \
    .config("spark.executor.heartbeatInterval", "200000") \
    .config("spark.network.timeout", "300000") \
    .config("spark.tmp.dir", "/a700_lun/spark_local_dir,/a800_lun/spark_local_dir") \
.config("spark.hadoop.fs.s3a.buffer.dir","/a700_lun/spark_local_dir,/a800_lun/spark_local_dir") \
    .config("spark.hadoop.fs.s3a.endpoint", "http://minIO:9002") \
    .config("spark.hadoop.fs.s3a.access.key", "minioadmin") \
    .config("spark.hadoop.fs.s3a.secret.key", "minioadmin") \
    .config("spark.hadoop.fs.s3a.path.style.access", True) \
    .config("spark.hadoop.fs.s3a.impl", "org.apache.hadoop.fs.s3a.S3AFileSystem") \
    .getOrCreate()

# Quitamos el verbosity en la salida de la ejecucion de SPARK
spark.sparkContext.setLogLevel('WARN')
#spark.sparkContext.setLogLevel('DEBUG')

df1 = spark.range(1 << 10).toDF("id")\
    .withColumn("1a", F.rand())\
```

 © 2021 NetApp, Inc. All rights reserved.

```
    .withColumn("a1", F.rand())\
    .withColumn("a2", F.rand())\
    .withColumn("a3", F.rand())\
    .withColumn("a4", F.rand())\
    .withColumn("a5", F.rand())\
    .withColumn("a6", F.rand())\
    .withColumn("a7", F.rand())\
    .withColumn("a8", F.rand())

df2 = spark.range(1 << 10).toDF("id")\
    .withColumn("b", F.rand())\
    .withColumn("b1", F.rand())\
    .withColumn("b2", F.rand())\
    .withColumn("b3", F.rand())\
    .withColumn("b4", F.rand())\
    .withColumn("b5", F.rand())\
    .withColumn("b6", F.rand())\
    .withColumn("b7", F.rand())\
    .withColumn("c1", F.rand())\
    .withColumn("c2", F.rand())\
    .withColumn("c3", F.rand())\
    .withColumn("c4", F.rand())\
    .withColumn("c5", F.rand())\
    .withColumn("c6", F.rand())\
    .withColumn("c7", F.rand())\
    .withColumn("c8", F.rand())\
    .withColumn("c9", F.rand())\
    .withColumn("d1", F.rand())\
    .withColumn("d2", F.rand())\
    .withColumn("d3", F.rand())\
    .withColumn("d4", F.rand())\
    .withColumn("d5", F.rand())\
    .withColumn("d6", F.rand())\
    .withColumn("d7", F.rand())


tdf1=df2.crossJoin(df1).drop(df1.id)

tdf1.write.mode("overwrite").parquet("s3a://miniowrite/parq")

SparkSession.stop

[root@n136 linux5_usr_src_with_old_linux4]#
[root@n136 linux5_usr_src_with_old_linux4]# cat minio_pyspark-test.py

from pyspark import SparkContext, SparkConf, SQLContext
import os
conf = (
    SparkConf()
    .setAppName("Spark minIO Test")
    .set("spark.hadoop.fs.s3a.endpoint", "http://minIO:9002")
    .set("spark.hadoop.fs.s3a.access.key", "minioadmin")
    .set("spark.hadoop.fs.s3a.secret.key", "minioadmin")
    .set("spark.hadoop.fs.s3a.path.style.access", True)
    .set("spark.hadoop.fs.s3a.impl", "org.apache.hadoop.fs.s3a.S3AFileSystem")
)
sc = SparkContext(conf=conf).getOrCreate()
sqlContext = SQLContext(sc)
df3 = sqlContext.read.parquet("s3a://miniowrite/parq/")

#df3.cache().count()
df3.count()
df3.filter( (df3.id == 1) | (df3.id == 2)).groupBy(df3.id).count().show()
[root@n136 linux5_usr_src_with_old_linux4]#
```

Main script for ONTAP, StorageGRID and MinIO

```
[root@n136 linux5_usr_src_with_old_linux4]# cat runtests_demo_ai.sh
#!/bin/bash
```


© 2021 NetApp, Inc. All rights reserved.

```
for i in 5
do
        #numer of executors
        executors=$3

        #storage statistics collection
        ontap_statistics_show_periodic="ontap_statistics_show_periodic_$1"
        nmon_nodes="$1"

        datetime=$2
        exee_memory=$i
        exec_cores=1

        #temp space for spark local dirs
        export SPARK_LOCAL_DIRS=/data8/spark_localdir
        export HADOOP_CONF_DIR=/etc/hadoop/conf

        #clear  the caches from system
        bash +x /usr/src/linux5_usr_src_with_old_linux4/clearcache.sh

        #start the nmon prcesses
        bash +x /usr/src/linux5_usr_src_with_old_linux4/start_nmon_all_workers.sh
${nmon_nodes}_${i}g $datetime

        #collect storage show periodic
        storagefile="/nmon_output/${nmon_nodes}_${i}g/${datetime}/${nmon_nodes}_${i}g/${ontap_sta
tistics_show_periodic}_${datetime}"
        echo $storagefile

        logfiles_location="/nmon_output/${nmon_nodes}_${i}g/${datetime}/${nmon_nodes}_${i}g/"
        mkdir -p  /nmon_output/${nmon_nodes}_${i}g/${datetime}/${nmon_nodes}_${i}g

        nohup ssh admin@10.63.150.62 "set adv;statistics show-periodic" >  $storagefile &

        #query to use with pyspark
        #for A800 and A700
        query="/usr/src/linux5_usr_src_with_old_linux4/create_parquet_on_a800.py"
        #query="/usr/src/linux5_usr_src_with_old_linux4/pyspark-customer_a800.py"

        #for minio on a800
        #query="/usr/src/linux5_usr_src_with_old_linux4/create_parquet_on_a800.py"
        #query="/usr/src/linux5_usr_src_with_old_linux4/minio_pyspark-test.py”

        #for sgd
        #query="/usr/src/linux5_usr_src_with_old_linux4/create_parquet_on_sgd.py"
        #query="/usr/src/linux5_usr_src_with_old_linux4/sgd_pyspark-test2.py”

        cat $query

        echo "time cat $query | sudo -u yarn pyspark --master yarn --num-executors $executors --
executor-memory ${exee_memory}g --executor-cores $exec_cores -v"
        time cat $query | sudo -u yarn pyspark --master yarn  --num-executors $executors --
executor-memory ${exee_memory}g --executor-cores $exec_cores -v

        #kill the running nmon process
        for i in 136 138 140 142 144 146 ; do ssh n${i} "iptables -F ;hostname; killall -9 nmon";
done

        #kill storage statisc show-periodic collection process
        kill -9 `lsof $storagefile | grep -v ^COMMAND | awk '{ print $2 }'`
        echo "logfiles folder location: $logfiles_location"

done
```

Command to run the tests

```
[root@n136 linux5_usr_src_with_old_linux4]#
test="6nodes_a800_count_cache_groupby_1cpu_5g_240_1000mb_overhead_480repartition"; executors=240;
datetime=`date +%Y%m%d_%H%M%S`; nohup bash +x
```

 © 2021 NetApp, Inc. All rights reserved.

```
/usr/src/linux5_usr_src_with_old_linux4/runtests_demo_ai.sh $test $datetime $executors >
/nmon_output/nohup_${datetime}_${test}.out
```

Where to find additional information

To learn more about the information that is described in this document, review the following documents and/or websites:

- Apache Spark documentation
  http://spark.apache.org/
- Cloudera Distribution Hadoop
  https://www.cloudera.com/downloads/cdh.html
- NetApp Product Documentation
  https://docs.netapp.com

Version history

Version	Date	Document Version History
Version 1.0	March 2021	Initial release.

© 2021 NetApp, Inc. All rights reserved.

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.

Copyright Information

Copyright © 2020–2021 NetApp, Inc. All Rights Reserved. Printed in the U.S. No part of this document covered by copyright may be reproduced in any form or by any means—graphic, electronic, or mechanical, including photocopying, recording, taping, or storage in an electronic retrieval system—without prior written permission of the copyright owner.

Software derived from copyrighted NetApp material is subject to the following license and disclaimer:

THIS SOFTWARE IS PROVIDED BY NETAPP "AS IS" AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, WHICH ARE HEREBY DISCLAIMED. IN NO EVENT SHALL NETAPP BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

NetApp reserves the right to change any products described herein at any time, and without notice. NetApp assumes no responsibility or liability arising from the use of products described herein, except as expressly agreed to in writing by NetApp. The use or purchase of this product does not convey a license under any patent rights, trademark rights, or any other intellectual property rights of NetApp.

The product described in this manual may be protected by one or more U.S. patents, foreign patents, or pending applications.

Data contained herein pertains to a commercial item (as defined in FAR 2.101) and is proprietary to NetApp, Inc. The U.S. Government has a non-exclusive, non-transferrable, non-sublicensable, worldwide, limited irrevocable license to use the Data only in connection with and in support of the U.S. Government contract under which the Data was delivered. Except as provided herein, the Data may not be used, disclosed, reproduced, modified, performed, or displayed without the prior written approval of NetApp, Inc. United States Government license rights for the Department of Defense are limited to those rights identified in DFARS clause 252.227-7015(b).

Trademark Information

NETAPP, the NETAPP logo, and the marks listed at http://www.netapp.com/TM are trademarks of NetApp, Inc. Other company and product names may be trademarks of their respective owners.

NVA-1157-DEPLOY-0321

NetApp