现代数据分析
NetApp Solutions

NetApp
April 12, 2024

目录

NetApp 现代数据分析解决方案

借助NetApp文件对象双处理能力和AWS SageMaker实现云数据管理

TR-4967：《使用NetApp文件对象双处理和AWS SageMaker进行云数据管理》

NetApp 公司 Karthikeyan Nagalingam

数据科学家和工程师通常需要访问以NFS格式存储的数据、但在AWS SageMaker中直接从S3协议访问此数据可能会面临挑战、因为AWS仅支持S3存储分段访问。但是、NetApp ONTAP 通过为NFS和S3启用双协议访问来提供解决方案。借助此解决方案 、数据科学家和工程师可以通过NetApp Cloud Volumes ONTAP 的S3存储分段从AWS SageMaker笔记本电脑访问NFS数据。这种方法可以轻松地从NFS和S3访问和共享相同的数据、而无需额外的软件。

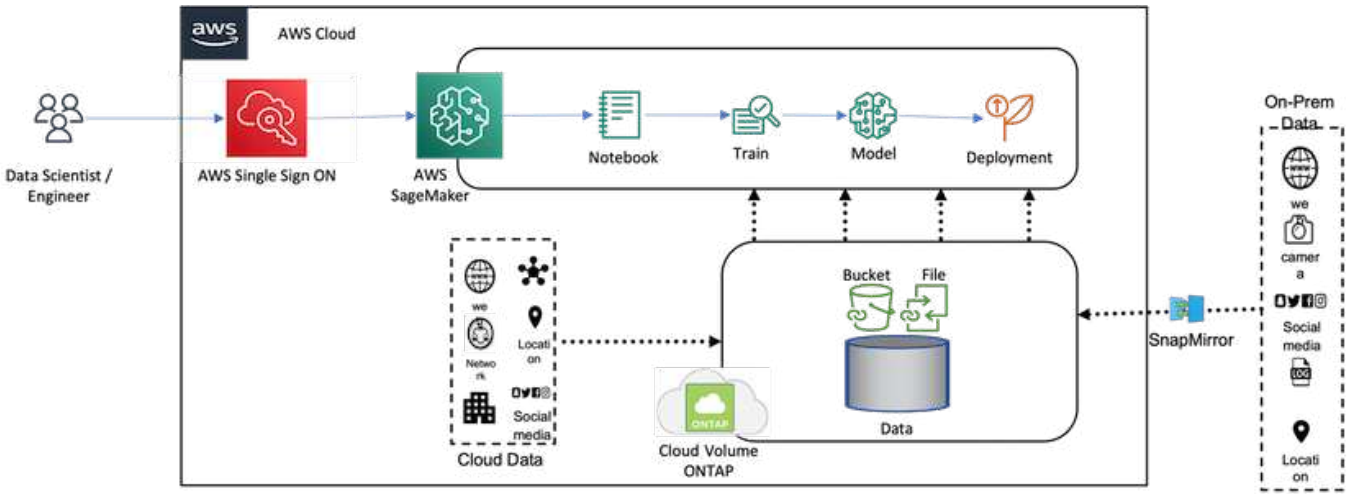
解决方案技术

此解决方案 利用以下技术：

- * AWS SageMaker Notebook.*为开发人员和数据科学家提供机器学习功能、帮助他们高效地创建、训练和部署高质量的ML模型。
- * NetApp BlueXp。*支持发现、部署和操作内部以及AWS、Azure和Google Cloud上的存储。它可以防止数据丢失、网络威胁和计划外中断、并优化数据存储和基础架构。
- * NetApp Cloud Volumes ONTAP。*在AWS、Azure和Google Cloud上提供采用NFS、SMB/CIFS、iSCSI和S3协议的企业级存储卷、使用户可以更灵活地访问和管理云中的数据。

NetApp Cloud Volumes ONTAP 是从BlueXP创建的、用于存储ML数据。

下图显示了解决方案的技术组件。



使用情形摘要

NFS和S3的双协议访问的一个潜在用例是机器学习和数据科学领域。例如、数据科学家团队可能正在使用AWS SageMaker执行机器学习项目、该项目要求访问以NFS格式存储的数据。但是、可能还需要通过S3存储分段访问和共享数据、以便与其他团队成员协作或与使用S3的其他应用程序集成。

通过使用NetApp Cloud Volumes ONTAP 、该团队可以将数据存储在一个位置、并可通过NFS和S3协议访问这些数据。数据科学家可以直接从AWS SageMaker访问NFS格式的数据、而其他团队成员或应用程序则可以通过S3存储分段访问相同的数据。

这种方法可以轻松高效地访问和共享数据、而无需在不同的存储解决方案之间进行额外的软件或数据迁移。此外、它还可以简化工作流并在团队成员之间进行协作、从而更快、更有效地开发机器学习模型。

数据科学家和其他应用程序的数据双重性

数据在NFS中可用、并可从AWS SageMaker的S3访问。

技术要求

您需要使用NetApp BlueXP、NetApp Cloud Volumes ONTAP 和AWS SageMaker笔记本电脑作为数据双用用例。

软件要求

下表列出了实施此用例所需的软件组件。

软件	数量
BlueXP	1.
NetApp Cloud Volumes ONTAP	1.
AWS SageMaker笔记本电脑	1.

部署过程

部署数据双率解决方案 涉及以下任务：

- BlueXP连接器
- NetApp Cloud Volumes ONTAP
- 用于机器学习的数据
- AWS SageMaker
- 通过Jupyter笔记本电脑经验证的机器学习

BlueXP连接器

在此验证中、我们使用了AWS。它也适用于Azure和Google Cloud。要在AWS中创建BlueXP Connector、请完成以下步骤：

1. 我们根据BlueXP中的mcarl-marketplace-subscription使用了这些凭据。
2. 选择适合您环境的区域(例如us-east-1 [N.Virginia)、然后选择身份验证方法(例如、假设角色或AWS密钥)。

在此验证中、我们将使用AWS密钥。

3. 提供连接器的名称并创建角色。
4. 根据您是否需要公共IP、提供VPC、子网或密钥对等网络详细信息。
5. 提供安全组的详细信息、例如从源类型访问HTTP、HTTPS或SSH、例如Anywhere和IP范围信息。
6. 查看并创建BlueXP Connector。
7. 确认在AWS控制台中运行的是BlueXP EC2实例状态、然后从*网络连接*选项卡中检查IP地址。
8. 从BlueXP门户登录到连接器用户界面、或者您也可以使用IP地址从浏览器进行访问。

NetApp Cloud Volumes ONTAP

要在BlueXP中创建Cloud Volumes ONTAP 实例、请完成以下步骤：

1. 创建新的工作环境、选择云提供商、然后选择Cloud Volumes ONTAP 实例的类型(例如、适用于ONTAP 的单CVO、HA或Amazon FSxN)。
2. 提供Cloud Volumes ONTAP 集群名称和凭据等详细信息。在此验证中、我们创建了一个名为的Cloud Volumes ONTAP 实例 `svm_sagemaker_cvo_sn1`。
3. 选择Cloud Volumes ONTAP 所需的服务。在此验证中、我们选择仅监控、因此禁用了*数据感知与合规性*和*备份到云服务*。
4. 在*位置和连接*部分中、选择AWS区域、VPC、子网、安全组、SSH身份验证方法、 以及密码或密钥对。
5. 选择充电方法。我们使用*专业版*进行此验证。
6. 您可以选择预配置的软件包、例如* POC和小型工作负载*、数据库和应用程序数据生产工作负载、经济高效的灾难恢复*或*最高性能生产工作负载。在此验证中、我们选择了* POC和小型工作负载*。
7. 创建具有特定大小、允许的协议和导出选项的卷。在此验证中、我们创建了一个名为的卷 `vol1`。
8. 选择配置文件磁盘类型和分层策略。在此验证中、我们禁用了*存储效率*和*通用SSD–动态性能*。
9. 最后、查看并创建Cloud Volumes ONTAP 实例。然后等待15到20分钟、让BlueXP创建Cloud Volumes ONTAP 工作环境。
10. 配置以下参数以启用双率协议。ONTAP 9支持双度协议(NFS/S3)。12：1及更高版本。
    a. 在此验证中、我们创建了一个名为的SVM `svm_sagemaker_cvo_sn1` 和卷 `vol1`。
    b. 验证SVM是否支持NFS和S3协议。如果不支持、请修改SVM以支持它们。

```
sagemaker_cvo_sn1::> vserver show -vserver svm_sagemaker_cvo_sn1
                                    Vserver: svm_sagemaker_cvo_sn1
                               Vserver Type: data
                            Vserver Subtype: default
                               Vserver UUID: 911065dd-a8bc-11ed-bc24-
e1c0f00ad86b
                                Root Volume:
svm_sagemaker_cvo_sn1_root
                                  Aggregate: aggr1
                                 NIS Domain: -
                  Root Volume Security Style: unix
                                LDAP Client: -
               Default Volume Language Code: C.UTF-8
                            Snapshot Policy: default
                              Data Services: data-cifs, data-
flexcache,
                                             data-iscsi, data-nfs,
                                             data-nvme-tcp
                                     Comment:
                               Quota Policy: default
                List of Aggregates Assigned: aggr1
 Limit on Maximum Number of Volumes allowed: unlimited
                        Vserver Admin State: running
                  Vserver Operational State: running
   Vserver Operational State Stopped Reason: -
                          Allowed Protocols: nfs, cifs, fcp, iscsi,
ndmp, s3
                       Disallowed Protocols: nvme
            Is Vserver with Infinite Volume: false
                           QoS Policy Group: -
                        Caching Policy Name: -
                                Config Lock: false
                               IPspace Name: Default
                         Foreground Process: -
                    Logical Space Reporting: true
                  Logical Space Enforcement: false
Default Anti_ransomware State of the Vserver's Volumes: disabled
            Enable Analytics on New Volumes: false
    Enable Activity Tracking on New Volumes: false

sagemaker_cvo_sn1::>
```

11. 根据需要创建并安装CA证书。
12. 创建服务数据策略。

```
sagemaker_cvo_sn1::*> network interface service-policy create -vserver
svm_sagemaker_cvo_sn1 -policy sagemaker_s3_nfs_policy -services data-
core,data-s3-server,data-nfs,data-flexcache
sagemaker_cvo_sn1::*> network interface create -vserver
svm_sagemaker_cvo_sn1 -lif svm_sagemaker_cvo_sn1_s3_lif -service-policy
sagemaker_s3_nfs_policy -home-node sagemaker_cvo_sn1-01 -address
172.30.10.41 -netmask 255.255.255.192

Warning: The configured failover-group has no valid failover targets for
the LIF's failover-policy. To view the failover targets for a LIF, use
         the "network interface show -failover" command.

sagemaker_cvo_sn1::*>
sagemaker_cvo_sn1::*> network interface show
Logical    Status     Network            Current       Current Is
Vserver     Interface  Admin/Oper Address/Mask       Node          Port
Home
----------- ---------- ---------- ------------------ -------------
------- ----
sagemaker_cvo_sn1
            cluster-mgmt up/up    172.30.10.40/26    sagemaker_cvo_sn1-
01
                                                                    e0a
true
            intercluster up/up    172.30.10.48/26    sagemaker_cvo_sn1-
01
                                                                    e0a
true
            sagemaker_cvo_sn1-01_mgmt1
                         up/up    172.30.10.58/26    sagemaker_cvo_sn1-
01
                                                                    e0a
true
svm_sagemaker_cvo_sn1
            svm_sagemaker_cvo_sn1_data_lif
                         up/up    172.30.10.23/26    sagemaker_cvo_sn1-
01
                                                                    e0a
true
            svm_sagemaker_cvo_sn1_mgmt_lif
                         up/up    172.30.10.32/26    sagemaker_cvo_sn1-
01
                                                                    e0a
true
            svm_sagemaker_cvo_sn1_s3_lif
                         up/up    172.30.10.41/26    sagemaker_cvo_sn1-
```

```
01
                                                                    e0a
true
6 entries were displayed.

sagemaker_cvo_sn1::*>
sagemaker_cvo_sn1::*> vserver object-store-server create -vserver
svm_sagemaker_cvo_sn1  -is-http-enabled true -object-store-server
svm_sagemaker_cvo_s3_sn1 -is-https-enabled false
sagemaker_cvo_sn1::*> vserver object-store-server show

Vserver: svm_sagemaker_cvo_sn1

           Object Store Server Name: svm_sagemaker_cvo_s3_sn1
               Administrative State: up
                       HTTP Enabled: true
             Listener Port For HTTP: 80
                      HTTPS Enabled: false
     Secure Listener Port For HTTPS: 443
  Certificate for HTTPS Connections: -
                  Default UNIX User: pcuser
               Default Windows User: -
                            Comment:

sagemaker_cvo_sn1::*>
```

13. 检查聚合详细信息。

```
sagemaker_cvo_sn1::*> aggr show


Aggregate     Size Available Used% State   #Vols  Nodes            RAID
Status
--------- -------- --------- ----- ------- ------ ----------------
------------
aggr0_sagemaker_cvo_sn1_01
           124.0GB   50.88GB   59% online       1 sagemaker_cvo_
raid0,
                                                   sn1-01
normal
aggr1      907.1GB   904.9GB    0% online       2 sagemaker_cvo_
raid0,
                                                   sn1-01
normal
2 entries were displayed.

sagemaker_cvo_sn1::*>
```

14. 创建用户和组。

```
sagemaker_cvo_sn1::*> vserver object-store-server user create -vserver
svm_sagemaker_cvo_sn1 -user s3user

sagemaker_cvo_sn1::*> vserver object-store-server user show
Vserver     User            ID        Access Key          Secret Key
----------- --------------- --------- -------------------
-------------------
svm_sagemaker_cvo_sn1
            root            0         -                   -
   Comment: Root User
svm_sagemaker_cvo_sn1
            s3user          1         0ZNAX21JW5Q8AP80CQ2E

PpLs4gA9K0_2gPhuykkp014gBjcC9Rbi3QDX_6rr
2 entries were displayed.

sagemaker_cvo_sn1::*>


sagemaker_cvo_sn1::*> vserver object-store-server group create -name
s3group -users s3user -comment ""

sagemaker_cvo_sn1::*>
sagemaker_cvo_sn1::*> vserver object-store-server group delete -gid 1
-vserver svm_sagemaker_cvo_sn1

sagemaker_cvo_sn1::*> vserver object-store-server group create -name
s3group -users s3user -comment "" -policies FullAccess

sagemaker_cvo_sn1::*>
```

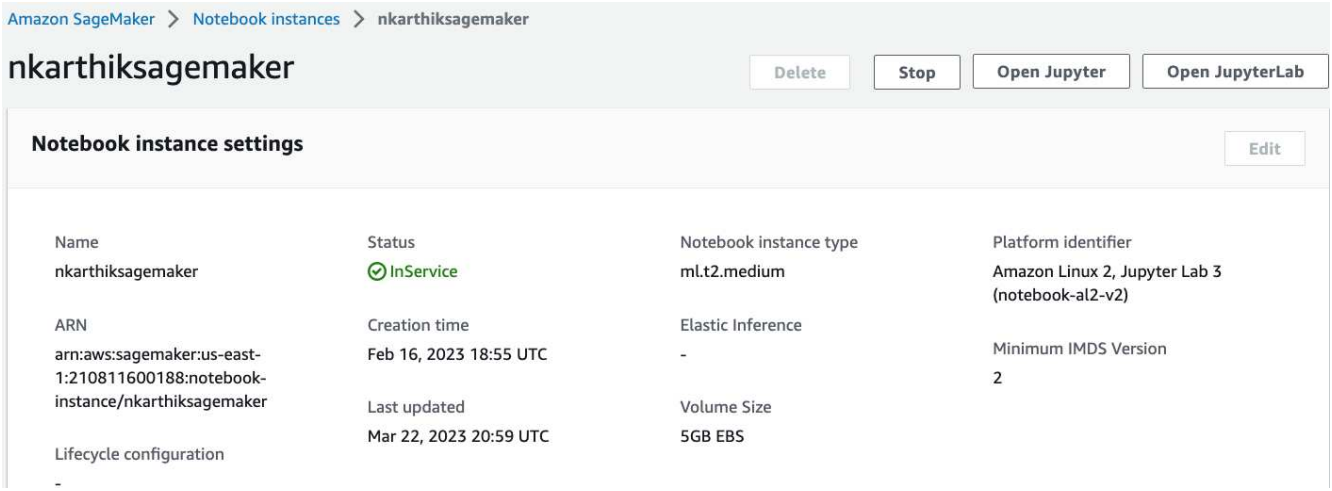
15. 在NFS卷上创建存储分段。

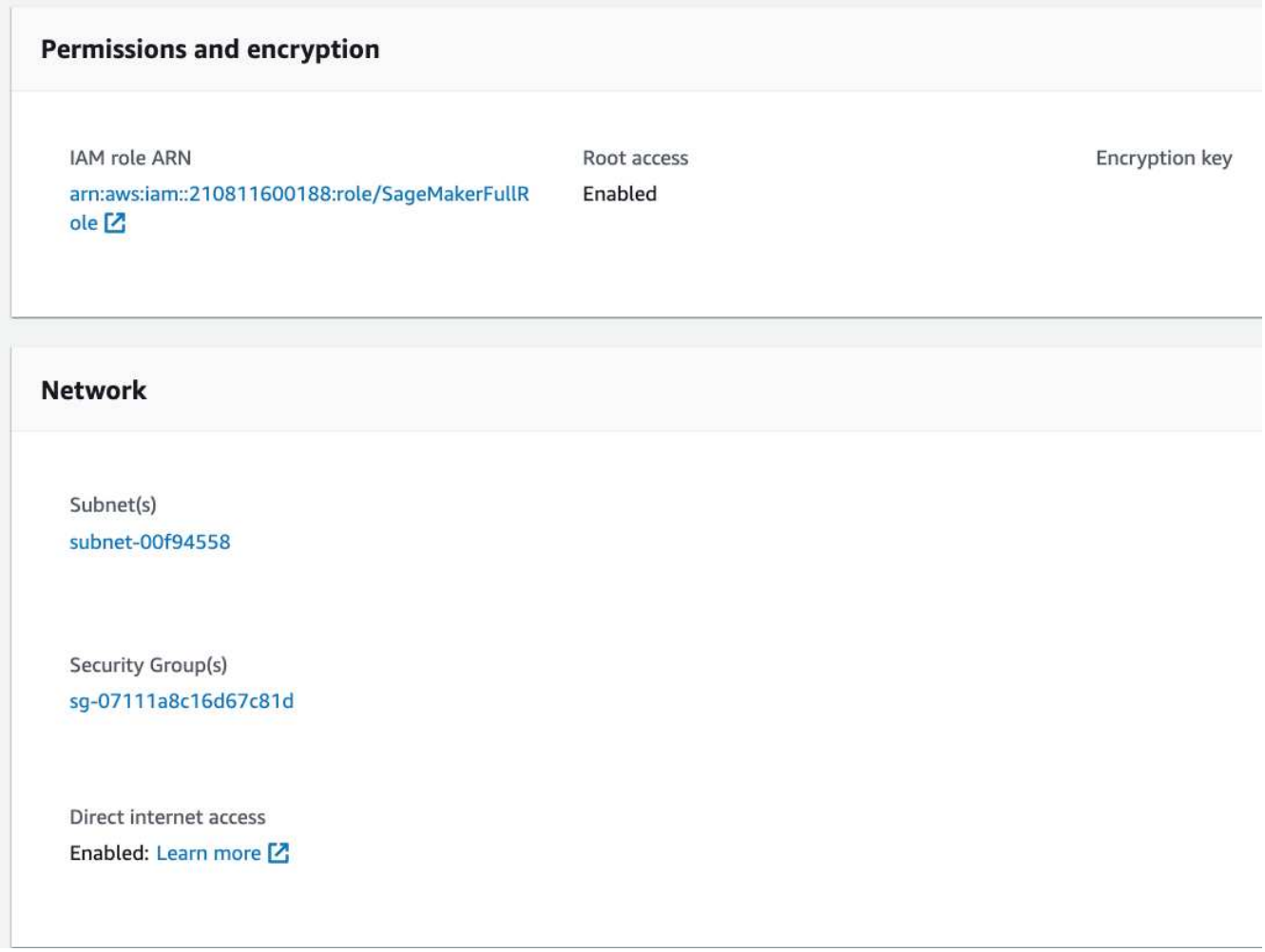
```
sagemaker_cvo_sn1::*> vserver object-store-server bucket create -bucket
ontapbucket1 -type nas -comment "" -vserver svm_sagemaker_cvo_sn1 -nas
-path /vol1
sagemaker_cvo_sn1::*> vserver object-store-server bucket show
Vserver     Bucket          Type     Volume            Size
Encryption Role       NAS Path
----------- --------------- -------- ----------------- ----------
---------- ---------- ----------
svm_sagemaker_cvo_sn1
            ontapbucket1    nas      vol1              -          false
-          /vol1
sagemaker_cvo_sn1::*>
```

AWS SageMaker

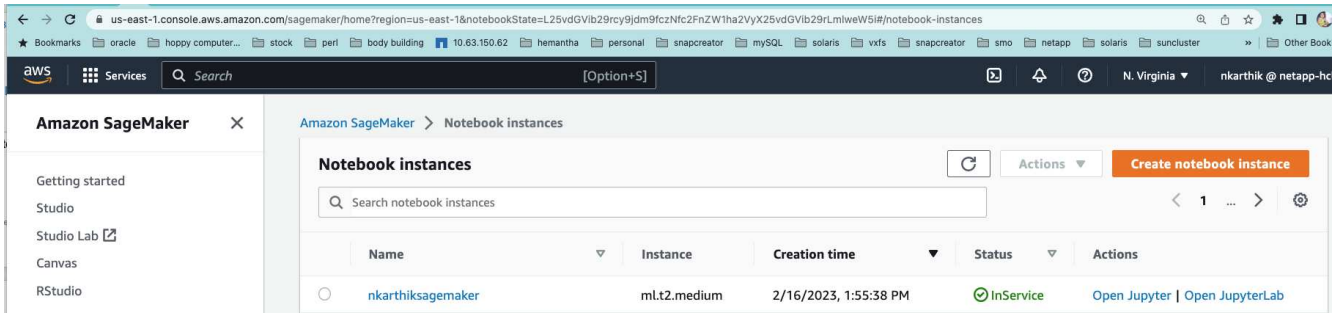
要从AWS SageMaker创建AWS笔记本电脑、请完成以下步骤：

1. 确保正在创建笔记本实例的用户具有AmazonSageMakerFullAccess IAM策略或属于具有AmazonSageMakerFullAccess权限的现有组。在此验证中、用户属于现有组。
2. 请提供以下信息：
   - 笔记本实例名称。
   - 实例类型。
   - 平台标识符。
   - 选择具有AmazonSageMakerFullAccess权限的IAM角色。
   - root访问—启用。
   - 加密密钥-不选择自定义加密。
   - 保留其余默认选项。
3. 在此验证中、SageMaker实例详细信息如下所示：

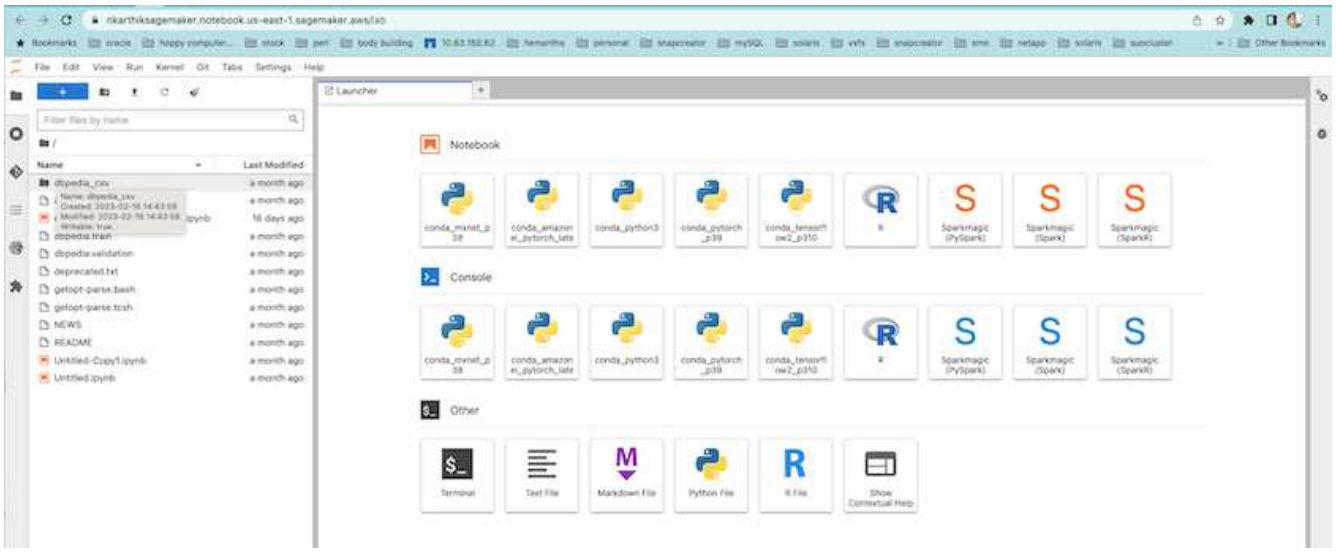




4. 启动AWS笔记本电脑。



5. 打开Jupyter实验室。



6. 登录到终端并挂载Cloud Volumes ONTAP 卷。

```
sh-4.2$ sudo mkdir /vol1; sudo mount -t nfs 172.30.10.41:/vol1 /vol1
sh-4.2$ df -h
Filesystem          Size  Used Avail Use% Mounted on
devtmpfs            2.0G     0  2.0G   0% /dev
tmpfs               2.0G     0  2.0G   0% /dev/shm
tmpfs               2.0G  624K  2.0G   1% /run
tmpfs               2.0G     0  2.0G   0% /sys/fs/cgroup
/dev/xvda1          140G  114G   27G  82% /
/dev/xvdf           4.8G   72K  4.6G   1% /home/ec2-user/SageMaker
tmpfs               393M     0  393M   0% /run/user/1001
tmpfs               393M     0  393M   0% /run/user/1002
tmpfs               393M     0  393M   0% /run/user/1000
172.30.10.41:/vol1  973M  189M  785M  20% /vol1
sh-4.2$
```

7. 使用AWS命令行界面命令检查在Cloud Volumes ONTAP 卷上创建的存储分段。

```
sh-4.2$ aws configure --profile netapp
AWS Access Key ID [None]: 0ZNAX21JW5Q8AP80CQ2E
AWS Secret Access Key [None]: PpLs4gA9K0_2gPhuykkp014gBjcC9Rbi3QDX_6rr
Default region name [None]: us-east-1
Default output format [None]:
sh-4.2$

sh-4.2$ aws s3 ls --profile netapp --endpoint-url
2023-02-10 17:59:48 ontapbucket1

sh-4.2$ aws s3 ls --profile netapp --endpoint-url  s3://ontapbucket1/


2023-02-10 18:46:44       4747 1
2023-02-10 18:48:32         96 setup.cfg

sh-4.2$
```

用于机器学习的数据

在此次验证中、我们使用了一个来自大众社区的DBipedia的数据集、从各种Wikimedia项目中创建的信息中提取结构化内容。

1. 从DBipedia GitHub位置下载数据并提取数据。请使用上一节中使用的相同终端。

```
sh-4.2$ wget
--2023-02-14 23:12:11--
Resolving github.com (github.com)... 140.82.113.3
Connecting to github.com (github.com)|140.82.113.3|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location:  [following]
--2023-02-14 23:12:11--
Resolving raw.githubusercontent.com (raw.githubusercontent.com)...
185.199.109.133, 185.199.110.133, 185.199.111.133, ...
Connecting to raw.githubusercontent.com
(raw.githubusercontent.com)|185.199.109.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 68431223 (65M) [application/octet-stream]
Saving to: ‘dbpedia_csv.tar.gz’

100%[===================================================================
===================================================================
===================>] 68,431,223  56.2MB/s   in 1.2s

2023-02-14 23:12:13 (56.2 MB/s) - ‘dbpedia_csv.tar.gz’ saved
[68431223/68431223]

sh-4.2$ tar -zxvf dbpedia_csv.tar.gz
dbpedia_csv/
dbpedia_csv/test.csv
dbpedia_csv/classes.txt
dbpedia_csv/train.csv
dbpedia_csv/readme.txt
sh-4.2$
```

2. 将数据复制到Cloud Volumes ONTAP 位置、并使用AWS命令行界面从S3存储分段中进行检查。

```
sh-4.2$ df -h
Filesystem          Size  Used Avail Use% Mounted on
devtmpfs            2.0G     0  2.0G   0% /dev
tmpfs               2.0G     0  2.0G   0% /dev/shm
tmpfs               2.0G  628K  2.0G   1% /run
tmpfs               2.0G     0  2.0G   0% /sys/fs/cgroup
/dev/xvda1          140G  114G   27G  82% /
/dev/xvdf           4.8G   52K  4.6G   1% /home/ec2-user/SageMaker
tmpfs               393M     0  393M   0% /run/user/1002
tmpfs               393M     0  393M   0% /run/user/1001
tmpfs               393M     0  393M   0% /run/user/1000
172.30.10.41:/vol1  973M  384K  973M   1% /vol1
sh-4.2$ pwd
/home/ec2-user
sh-4.2$ cp -ra dbpedia_csv /vol1
sh-4.2$ aws s3 ls --profile netapp --endpoint-url  s3://ontapbucket1/
                           PRE dbpedia_csv/
2023-02-10 18:46:44       4747 1
2023-02-10 18:48:32         96 setup.cfg
sh-4.2$
```

3. 执行基本验证以确保读取/写入功能在S3存储分段上正常工作。

```
sh-4.2$ aws s3 cp  --profile netapp --endpoint-url  /usr/share/doc/util-
linux-2.30.2 s3://ontapbucket1/ --recursive
upload: ../../../usr/share/doc/util-linux-2.30.2/deprecated.txt to
s3://ontapbucket1/deprecated.txt
upload: ../../../usr/share/doc/util-linux-2.30.2/getopt-parse.bash to
s3://ontapbucket1/getopt-parse.bash
upload: ../../../usr/share/doc/util-linux-2.30.2/README to
s3://ontapbucket1/README
upload: ../../../usr/share/doc/util-linux-2.30.2/getopt-parse.tcsh to
s3://ontapbucket1/getopt-parse.tcsh
upload: ../../../usr/share/doc/util-linux-2.30.2/AUTHORS to
s3://ontapbucket1/AUTHORS
upload: ../../../usr/share/doc/util-linux-2.30.2/NEWS to
s3://ontapbucket1/NEWS
sh-4.2$ aws s3 ls --profile netapp --endpoint-url
s3://ontapbucket1/s3://ontapbucket1/

An error occurred (InternalError) when calling the ListObjectsV2
operation: We encountered an internal error. Please try again.
sh-4.2$ aws s3 ls --profile netapp --endpoint-url  s3://ontapbucket1/
                           PRE dbpedia_csv/
```

```
2023-02-16 19:19:27      26774 AUTHORS
2023-02-16 19:19:27      72727 NEWS
2023-02-16 19:19:27       4493 README
2023-02-16 19:19:27       2825 deprecated.txt
2023-02-16 19:19:27       1590 getopt-parse.bash
2023-02-16 19:19:27       2245 getopt-parse.tcsh
sh-4.2$ ls -ltr /vol1
total 132
drwxrwxr-x 2 ec2-user ec2-user  4096 Mar 29  2015 dbpedia_csv
-rw-r--r-- 1 nobody   nobody    2245 Apr 10 17:37 getopt-parse.tcsh
-rw-r--r-- 1 nobody   nobody    2825 Apr 10 17:37 deprecated.txt
-rw-r--r-- 1 nobody   nobody    4493 Apr 10 17:37 README
-rw-r--r-- 1 nobody   nobody    1590 Apr 10 17:37 getopt-parse.bash
-rw-r--r-- 1 nobody   nobody   26774 Apr 10 17:37 AUTHORS
-rw-r--r-- 1 nobody   nobody   72727 Apr 10 17:37 NEWS
sh-4.2$ ls -ltr /vol1/dbpedia_csv/
total 192104
-rw------- 1 ec2-user ec2-user 174148970 Mar 28  2015 train.csv
-rw------- 1 ec2-user ec2-user  21775285 Mar 28  2015 test.csv
-rw------- 1 ec2-user ec2-user       146 Mar 28  2015 classes.txt
-rw-rw-r-- 1 ec2-user ec2-user      1758 Mar 29  2015 readme.txt
sh-4.2$ chmod -R 777 /vol1/dbpedia_csv
sh-4.2$ ls -ltr /vol1/dbpedia_csv/
total 192104
-rwxrwxrwx 1 ec2-user ec2-user 174148970 Mar 28  2015 train.csv
-rwxrwxrwx 1 ec2-user ec2-user  21775285 Mar 28  2015 test.csv
-rwxrwxrwx 1 ec2-user ec2-user       146 Mar 28  2015 classes.txt
-rwxrwxrwx 1 ec2-user ec2-user      1758 Mar 29  2015 readme.txt
sh-4.2$ aws s3 cp --profile netapp --endpoint-url http://172.30.2.248/
s3://ontapbucket1/ /tmp --recursive
download: s3://ontapbucket1/AUTHORS to ../../tmp/AUTHORS
download: s3://ontapbucket1/README to ../../tmp/README
download: s3://ontapbucket1/NEWS to ../../tmp/NEWS
download: s3://ontapbucket1/dbpedia_csv/classes.txt to
../../tmp/dbpedia_csv/classes.txt
download: s3://ontapbucket1/dbpedia_csv/readme.txt to
../../tmp/dbpedia_csv/readme.txt
download: s3://ontapbucket1/deprecated.txt to ../../tmp/deprecated.txt
download: s3://ontapbucket1/getopt-parse.bash to ../../tmp/getopt-
parse.bash
download: s3://ontapbucket1/getopt-parse.tcsh to ../../tmp/getopt-
parse.tcsh
download: s3://ontapbucket1/dbpedia_csv/test.csv to
../../tmp/dbpedia_csv/test.csv
download: s3://ontapbucket1/dbpedia_csv/train.csv to
../../tmp/dbpedia_csv/train.csv
```

```
sh-4.2$
sh-4.2$ aws s3 ls --profile netapp --endpoint-url  s3://ontapbucket1/
                           PRE dbpedia_csv/
2023-02-16 19:19:27      26774 AUTHORS
2023-02-16 19:19:27      72727 NEWS
2023-02-16 19:19:27       4493 README
2023-02-16 19:19:27       2825 deprecated.txt
2023-02-16 19:19:27       1590 getopt-parse.bash
2023-02-16 19:19:27       2245 getopt-parse.tcsh
sh-4.2$
```

验证从**Jupyter**笔记本电脑学习机器的情况

以下验证使用以下SageMaker BlazingText示例、通过文本分类提供机器学习构建、培训和部署模型：

1. 安装boto3和SageMaker软件包。

```
In [1]:  pip install --upgrade boto3 sagemaker
```

输出：

```
Looking in indexes: https://pypi.org/simple,
https://pip.repos.neuron.amazo naws.com
Requirement already satisfied: boto3 in /home/ec2-
user/anaconda3/envs/pytho n3/lib/python3.10/site-packages (1.26.44)
Collecting boto3
  Downloading boto3-1.26.72-py3-none-any.whl (132 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
132.7/132.7 kB 14.6 MB/s eta 0: 00:00
Requirement already satisfied: sagemaker in /home/ec2-
user/anaconda3/envs/p ython3/lib/python3.10/site-packages (2.127.0)
Collecting sagemaker
  Downloading sagemaker-2.132.0.tar.gz (668 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
668.0/668.0 kB 12.3 MB/s eta 0:
00:0000:01
  Preparing metadata (setup.py) ... done
Collecting botocore<1.30.0,>=1.29.72
  Downloading botocore-1.29.72-py3-none-any.whl (10.4 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
10.4/10.4 MB 44.3 MB/s eta 0: 00:0000:010:01
Requirement already satisfied: s3transfer<0.7.0,>=0.6.0 in /home/ec2-
user/a naconda3/envs/python3/lib/python3.10/site-packages (from boto3)
(0.6.0)
```

```
Requirement already satisfied: jmespath<2.0.0,>=0.7.1 in /home/ec2-
user/ana conda3/envs/python3/lib/python3.10/site-packages (from boto3)
(0.10.0)
Requirement already satisfied: attrs<23,>=20.3.0 in /home/ec2-
user/anaconda
3/envs/python3/lib/python3.10/site-packages (from sagemaker) (22.1.0)
Requirement already satisfied: google-pasta in /home/ec2-
user/anaconda3/env s/python3/lib/python3.10/site-packages (from
sagemaker) (0.2.0)
Requirement already satisfied: numpy<2.0,>=1.9.0 in /home/ec2-
user/anaconda
3/envs/python3/lib/python3.10/site-packages (from sagemaker) (1.22.4)
Requirement already satisfied: protobuf<4.0,>=3.1 in /home/ec2-
user/anacond a3/envs/python3/lib/python3.10/site-packages (from
sagemaker) (3.20.3)
Requirement already satisfied: protobuf3-to-dict<1.0,>=0.1.5 in
/home/ec2-u ser/anaconda3/envs/python3/lib/python3.10/site-packages
(from sagemaker)
(0.1.5)
Requirement already satisfied: smdebug_rulesconfig==1.0.1 in /home/ec2-
use r/anaconda3/envs/python3/lib/python3.10/site-packages (from
sagemaker) (1.
0.1) Requirement already satisfied: importlib-metadata<5.0,>=1.4.0 in
/home/ec2user/anaconda3/envs/python3/lib/python3.10/site-packages (from
sagemaker)
(4.13.0)
Requirement already satisfied: packaging>=20.0 in /home/ec2-
user/anaconda3/ envs/python3/lib/python3.10/site-packages (from
sagemaker) (21.3)
Requirement already satisfied: pandas in /home/ec2-
user/anaconda3/envs/pyth on3/lib/python3.10/site-packages (from
sagemaker) (1.5.1)
Requirement already satisfied: pathos in /home/ec2-
user/anaconda3/envs/pyth on3/lib/python3.10/site-packages (from
sagemaker) (0.3.0)
Requirement already satisfied: schema in /home/ec2-
user/anaconda3/envs/pyth on3/lib/python3.10/site-packages (from
sagemaker) (0.7.5) Requirement already satisfied: python-
dateutil<3.0.0,>=2.1 in /home/ec2-use
r/anaconda3/envs/python3/lib/python3.10/site-packages (from
botocore<1.30.
0,>=1.29.72->boto3) (2.8.2)
Requirement already satisfied: urllib3<1.27,>=1.25.4 in /home/ec2-
user/anac onda3/envs/python3/lib/python3.10/site-packages (from
botocore<1.30.0,>=1.2
9.72->boto3) (1.26.8) Requirement already satisfied: zipp>=0.5 in
```

```
/home/ec2-user/anaconda3/envs/p ython3/lib/python3.10/site-packages
(from importlib-metadata<5.0,>=1.4.0->s agemaker) (3.10.0)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /home/ec2-
user/a naconda3/envs/python3/lib/python3.10/site-packages (from
packaging>=20.0->s agemaker) (3.0.9)
Requirement already satisfied: six in /home/ec2-
user/anaconda3/envs/python
3/lib/python3.10/site-packages (from protobuf3-to-dict<1.0,>=0.1.5-
>sagemak er) (1.16.0)
Requirement already satisfied: pytz>=2020.1 in /home/ec2-
user/anaconda3/env s/python3/lib/python3.10/site-packages (from pandas-
>sagemaker) (2022.5)
Requirement already satisfied: ppft>=1.7.6.6 in /home/ec2-
user/anaconda3/en vs/python3/lib/python3.10/site-packages (from pathos-
>sagemaker) (1.7.6.6) Requirement already satisfied:
multiprocess>=0.70.14 in /home/ec2-user/anac
onda3/envs/python3/lib/python3.10/site-packages (from pathos->sagemaker)
(0.70.14)
Requirement already satisfied: dill>=0.3.6 in /home/ec2-
user/anaconda3/env s/python3/lib/python3.10/site-packages (from pathos-
>sagemaker) (0.3.6)
Requirement already satisfied: pox>=0.3.2 in /home/ec2-
user/anaconda3/envs/ python3/lib/python3.10/site-packages (from pathos-
>sagemaker) (0.3.2) Requirement already satisfied: contextlib2>=0.5.5 in
/home/ec2-user/anacond a3/envs/python3/lib/python3.10/site-packages
(from schema->sagemaker) (21.
6.0) Building wheels for collected packages: sagemaker
  Building wheel for sagemaker (setup.py) ... done
  Created wheel for sagemaker: filename=sagemaker-2.132.0-py2.py3-none-
any. whl size=905449
sha256=f6100a5dc95627f2e2a49824e38f0481459a27805ee19b5a06ec
83db0252fd41
  Stored in directory: /home/ec2-
user/.cache/pip/wheels/60/41/b6/482e7ab096
520df034fbf2dddd244a1d7ba0681b27ef45aa61
Successfully built sagemaker
Installing collected packages: botocore, boto3, sagemaker
  Attempting uninstall: botocore     Found existing installation:
botocore 1.24.19
    Uninstalling botocore-1.24.19:       Successfully uninstalled
botocore-1.24.19
  Attempting uninstall: boto3     Found existing installation: boto3
1.26.44
    Uninstalling boto3-1.26.44:
      Successfully uninstalled boto3-1.26.44
  Attempting uninstall: sagemaker     Found existing installation:
```

```
sagemaker 2.127.0
    Uninstalling sagemaker-2.127.0:
      Successfully uninstalled sagemaker-2.127.0
ERROR: pip's dependency resolver does not currently take into account
all t he packages that are installed. This behaviour is the source of
the followi ng dependency conflicts.
awscli 1.27.44 requires botocore==1.29.44, but you have botocore 1.29.72
wh ich is incompatible.
aiobotocore 2.0.1 requires botocore<1.22.9,>=1.22.8, but you have
botocore 1.29.72 which is incompatible. Successfully installed boto3-
1.26.72 botocore-1.29.72 sagemaker-2.132.0 Note: you may need to restart
the kernel to use updated packages.
```

2. 在以下步骤中、将显示数据 (`dbpedia_csv`)下载 `ontapbucket1` 机器学习中使用的Jupyter笔记本实例。

```
In [2]: import sagemaker
In [3]: from sagemaker import get_execution_role
In [4]:
import json
import boto3
sess = sagemaker.Session()
role = get_execution_role()
print(role)
bucket = "ontapbucket1"
print(bucket)
sess.s3_client = boto3.client('s3',region_name='',aws_access_key_id =
'0ZNAX21JW5Q8AP80CQ2E',  aws_secret_access_key =
'PpLs4gA9K0_2gPhuykkp014gBjcC9Rbi3QDX_6rr',
                              use_ssl = False, endpoint_url =
'http://172.30.10.41',

config=boto3.session.Config(signature_version='s3v4',
s3={'addressing_style':'path'}) )
sess.s3_resource = boto3.resource('s3',region_name='',aws_access_key_id
= '0ZNAX21JW5Q8AP80CQ2E', aws_secret_access_key =
'PpLs4gA9K0_2gPhuykkp014gBjcC9Rbi3QDX_6rr',
                              use_ssl = False, endpoint_url =
'http://172.30.10.41',

config=boto3.session.Config(signature_version='s3v4',
s3={'addressing_style':'path'}) )
prefix = "blazingtext/supervised"
import os
my_bucket = sess.s3_resource.Bucket(bucket)
my_bucket = sess.s3_resource.Bucket(bucket)
#os.mkdir('dbpedia_csv')
for s3_object in my_bucket.objects.all():
    filename = s3_object.key
#    print(filename)
#    print(s3_object.key)
    my_bucket.download_file(s3_object.key, filename)
```

3. 以下代码将创建从整数索引到类标签的映射、用于在推理期间检索实际类名称。

```
index_to_label = {}
with open("dbpedia_csv/classes.txt") as f:
    for i,label in enumerate(f.readlines()):
        index_to_label[str(i + 1)] = label.strip()
```

输出将列出中的文件和文件夹 `ontapbucket1` 用于AWS SageMaker机器学习验证的数据分段。

```
arn:aws:iam::210811600188:role/SageMakerFullRole ontapbucket1
AUTHORS
AUTHORS
NEWS
NEWS
README README
dbpedia_csv/classes.txt dbpedia_csv/classes.txt dbpedia_csv/readme.txt
dbpedia_csv/readme.txt dbpedia_csv/test.csv dbpedia_csv/test.csv
dbpedia_csv/train.csv dbpedia_csv/train.csv deprecated.txt
deprecated.txt getopt-parse.bash getopt-parse.bash getopt-parse.tcsh
getopt-parse.tcsh
In [5]: ls
AUTHORS       deprecated.txt     getopt-parse.tcsh  NEWS
Untitled.ipynb dbpedia_csv/  getopt-parse.bash  lost+found/
README
In [6]: ls -l dbpedia_csv
total 191344
-rw-rw-r-- 1 ec2-user ec2-user       146 Feb 16 19:43 classes.txt
-rw-rw-r-- 1 ec2-user ec2-user      1758 Feb 16 19:43 readme.txt
-rw-rw-r-- 1 ec2-user ec2-user  21775285 Feb 16 19:43 test.csv
-rw-rw-r-- 1 ec2-user ec2-user 174148970 Feb 16 19:43 train.csv
```

4. 启动数据预处理阶段、将训练数据预处理为空格分隔的令牌化文本格式、BlazingText算法和Nultk库可以使用该格式来标记DBPedia数据集的输入语句。下载nltk令牌程序和其他库。。 `transform_instance` 并行应用于每个数据实例时、会使用Python多处理模块。

```
ln [7]: from random import shuffle
import multiprocessing
from multiprocessing import Pool
import csv
import nltk
nltk.download("punkt")
def transform_instance(row):
    cur_row = []
    label ="__label__" + index_to_label [row[0]] # Prefix the index-ed
label with __label__
    cur_row.append (label)
    cur_row.extend(nltk.word_tokenize(row[1].lower ()))
    cur_row.extend(nltk.word_tokenize(row[2].lower ()))
    return cur_row
def preprocess(input_file, output_file, keep=1):
    all_rows = []
    with open(input_file,"r") as csvinfile:
```

```
        csv_reader = csv.reader(csvinfile, delimiter=",")
        for row in csv_reader:
            all_rows.append(row)
    shuffle(all_rows)
    all_rows = all_rows[: int(keep * len(all_rows))]
    pool = Pool(processes=multiprocessing.cpu_count())
    transformed_rows = pool.map(transform_instance, all_rows)
    pool.close()
    pool. join()
    with open(output_file, "w") as csvoutfile:
        csv_writer = csv.writer (csvoutfile, delimiter=" ",
lineterminator="\n")
        csv_writer.writerows (transformed_rows)

# Preparing the training dataset
# since preprocessing the whole dataset might take a couple of minutes,
# we keep 20% of the training dataset for this demo.
# Set keep to 1 if you want to use the complete dataset
preprocess("dbpedia_csv/train.csv","dbpedia.train", keep=0.2)
# Preparing the validation dataset
preprocess("dbpedia_csv/test.csv","dbpedia.validation")
sess = sagemaker.Session()
role = get_execution_role()
print (role) # This is the role that sageMaker would use to leverage Aws
resources (S3,  Cloudwatch) on your behalf
bucket = sess.default_bucket() # Replace with your own bucket name if
needed
print("default Bucket::: ")
print(bucket)
```

输出：

```
[nltk_data] Downloading package punkt to /home/ec2-user/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
arn:aws:iam::210811600188:role/SageMakerFullRole default Bucket:::
sagemaker-us-east-1-210811600188
```

5. 将格式化的训练数据集上传到S3、以便SageMaker可以使用它来执行训练作业。然后、使用Python SDK将两个文件上传到存储分段和前缀位置。

```
ln [8]: %%time
train_channel = prefix + "/train"
validation_channel = prefix + "/validation"
sess.upload_data(path="dbpedia.train", bucket=bucket,
key_prefix=train_channel)
sess.upload_data(path="dbpedia.validation", bucket=bucket,
key_prefix=validation_channel)
s3_train_data = "s3://{}/{}".format(bucket, train_channel)
s3_validation_data = "s3://{}/{}".format(bucket, validation_channel)
```

输出：

```
CPU times: user 546 ms, sys: 163 ms, total: 709 ms
Wall time: 1.32 s
```

6. 在S3上设置加载模型项目的输出位置、以便项目可以作为算法训练作业的输出。创建 `sageMaker.estimator.Estimator` 用于启动培训作业的对象。

```
In [9]: s3_output_location = "s3://{}/{}/output".format(bucket, prefix)
In [10]: region_name = boto3.Session().region_name
In [11]: container =
sagemaker.amazon.amazon_estimator.get_image_uri(region_name,
"blazingtext","latest")
print("Using SageMaker BlazingText container: {} ({})".format(container,
region_name))
```

输出：

```
The method get_image_uri has been renamed in sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
Defaulting to the only supported framework/algorithm version: 1.
Ignoring f ramework/algorithm version: latest.
Using SageMaker BlazingText container: 811284229777.dkr.ecr.us-east-
1.amazo naws.com/blazingtext:1 (us-east-1)
```

7. 定义SageMaker `Estrimator` 通过资源配置和超参数、在c4.4xlarge实例上使用受监控模式在DBPedia数据集上训练文本分类。

```
In [12]: bt_model = sagemaker.estimator.Estimator(
container,
role,
instance_count=1,
instance_type="ml.c4.4xlarge",
volume_size=30,
max_run=360000,
input_mode="File",
output_path=s3_output_location,
hyperparameters={
        "mode": "supervised",
        "epochs": 1,
        "min_count": 2,
        "learning_rate": 0.05,
        "vector_dim": 10,
        "early_stopping": True,
        "patience": 4,
        "min_epochs": 5,
        "word_ngrams": 2,
 },
     )
```

8. 准备数据通道和算法之间的握手。为此、请创建 `sagemaker.session.s3_input` 数据通道中的对象、并将其保留在词典中、以供算法使用。

```
ln [13]: train_data = sagemaker.inputs.TrainingInput(
    s3_train_data,
    distribution="FullyReplicated",
    content_type="text/plain",
    s3_data_type="S3Prefix",
)
validation_data = sagemaker.inputs.TrainingInput(
    s3_validation_data,
    distribution="FullyReplicated",
    content_type="text/plain",
    s3_data_type="S3Prefix",
)
data_channels = {"train": train_data, "validation": validation_data}
```

9. 作业完成后、将显示作业完成消息。您可以在设置为的S3存储分段中找到经过培训的型号 `output_path` 在估算器中。

```
ln [14]: bt_model.fit(inputs=data_channels, logs=True)
```

输出：

```
INFO:sagemaker:Creating training-job with name: blazingtext-2023-02-16-
20-3
7-30-748
2023-02-16 20:37:30 Starting - Starting the training job......
2023-02-16 20:38:09 Starting - Preparing the instances for
training......
2023-02-16 20:39:24 Downloading - Downloading input data
2023-02-16 20:39:24 Training - Training image download completed.
Training in progress... Arguments: train
[02/16/2023 20:39:41 WARNING 140279908747072] Loggers have already been
set up. [02/16/2023 20:39:41 WARNING 140279908747072] Loggers have
already been set up.
[02/16/2023 20:39:41 INFO 140279908747072] nvidia-smi took:
0.0251793861389
16016 secs to identify 0 gpus
[02/16/2023 20:39:41 INFO 140279908747072] Running single machine CPU
Blazi ngText training using supervised mode.
Number of CPU sockets found in instance is  1
[02/16/2023 20:39:41 INFO 140279908747072] Processing
/opt/ml/input/data/tr ain/dbpedia.train . File size: 35.0693244934082 MB
[02/16/2023 20:39:41 INFO 140279908747072] Processing
/opt/ml/input/data/va lidation/dbpedia.validation . File size:
21.887572288513184 MB
Read 6M words
Number of words:  149301
Loading validation data from
/opt/ml/input/data/validation/dbpedia.validati on
Loaded validation data.
-------------- End of epoch: 1 ##### Alpha: 0.0000  Progress: 100.00%
Million Words/sec: 10.39 ##### Training finished.
Average throughput in Million words/sec: 10.39
Total training time in seconds: 0.60
#train_accuracy: 0.7223
Number of train examples: 112000
#validation_accuracy: 0.7205
Number of validation examples: 70000
2023-02-16 20:39:55 Uploading - Uploading generated training model
2023-02-16 20:40:11 Completed - Training job completed
Training seconds: 68
Billable seconds: 68
```

10. 培训完成后、将经过培训的模型部署为Amazon SageMaker实时托管端点、以进行预测。

```
In [15]: from sagemaker.serializers import JSONSerializer
 text_classifier = bt_model.deploy(
     initial_instance_count=1, instance_type="ml.m4.xlarge",
serializer=JSONS
)
```

输出:

```
INFO:sagemaker:Creating model with name: blazingtext-2023-02-16-20-41-
33-10
0
INFO:sagemaker:Creating endpoint-config with name blazingtext-2023-02-
16-20
-41-33-100
INFO:sagemaker:Creating endpoint with name blazingtext-2023-02-16-20-41-
33-
100
-------!
```

```
In [16]: sentences = [
    "Convair was an american aircraft manufacturing company which later
expanded into rockets and spacecraft.",
       "Berwick secondary college is situated in the outer melbourne
metropolitan suburb of berwick .",
]
# using the same nltk tokenizer that we used during data preparation for
training
tokenized_sentences = [" ".join(nltk.word_tokenize(sent)) for sent in
sentences]
payload = {"instances": tokenized_sentences} response =
text_classifier.predict(payload)
predictions = json.loads(response)
print(json.dumps(predictions, indent=2))
```

```
[
  {
    "label": [
      "__label__Artist"
    ],
    "prob": [
      0.4090951681137085
    ]
  },
  {
    "label": [
      "__label__EducationalInstitution"
    ],
    "prob": [
      0.49466073513031006
    ]
  }
]
```

11. 默认情况下、模型返回一个概率最高的预测。以检索顶部 `k` 预测、设置 `k` 在配置文件中。

```
In [17]: payload = {"instances": tokenized_sentences, "configuration":
{"k": 2}}
 response = text_classifier.predict(payload)

 predictions = json.loads(response)
 print(json.dumps(predictions, indent=2))
```

```
[
  {
    "label": [
      "__label__Artist",
      "__label__MeanOfTransportation"
    ],
    "prob": [
      0.4090951681137085,
      0.26930734515190125
    ]
  },
  {
    "label": [
      "__label__EducationalInstitution",
      "__label__Building"
    ],
    "prob": [
      0.49466073513031006,
      0.15817692875862122
    ]
  }
]
```

12. 请先删除端点、然后再关闭此笔记本。

```
In [18]: sess.delete_endpoint(text_classifier.endpoint)
WARNING:sagemaker.deprecations:The endpoint attribute has been renamed
in s agemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
INFO:sagemaker:Deleting endpoint with name: blazingtext-2023-02-16-20-
41-33
-100
```

结论

根据此验证、数据科学家和工程师可以通过NetApp Cloud Volumes ONTAP 的S3存储分段从AWS SageMaker Jupyter笔记本电脑访问NFS数据。这种方法可以轻松地从NFS和S3访问和共享相同的数据、而无需额外的软件。

从何处查找追加信息

要了解有关本文档中所述信息的更多信息，请查看以下文档和 / 或网站：

- 使用SageMaker BlazingText进行文本分类

"https://sagemaker-examples.readthedocs.io/en/latest/introduction_to_amazon_algorithms/blazingtext_text_classification_dbpedia/blazingtext_text_classification_dbpedia.html"

- S3 对象存储的 ONTAP 版本支持

  "https://docs.netapp.com/us-en/ontap/s3-config/ontap-version-support-s3-concept.html"

Apache Kafka工作负载与NetApp NFS存储

TR-4947：Apache Kafka工作负载与NetApp NFS存储—功能验证和性能

NetApp公司Shantanu Chakole、Karthikeyan Nagalingam和Joe Scott

Kafka是一个分布式发布订阅消息传送系统、具有一个强大的队列、可以接受大量消息数据。借助Kafka、应用程序可以非常快速地对主题进行数据写入和读取。由于Kafka具有容错能力和可扩展性、因此通常在大数据空间中使用它作为快速载入和移动多个数据流的可靠方式。使用情形包括流处理、网站活动跟踪、指标收集和监控、日志聚合、实时分析等。

虽然NFS上的正常Kafka操作运行良好、但 "重命名操作不明智" 在调整NFS上运行的Kafka集群的大小或重新分区期间、问题描述 会使应用程序崩溃。这是一个重要的问题描述 、因为出于负载平衡或维护目的、必须调整Kafka集群的大小或对其进行重新分区。您可以找到其他详细信息 "此处"。

本文档介绍了以下主题：

- 错误重命名问题和解决方案 验证
- 降低CPU利用率以缩短I/O等待时间
- Kafka代理恢复时间更快
- 云端和内部环境中的性能

为什么要对**Kafka**工作负载使用**NFS**存储?

生产应用程序中的Kafka工作负载可以在应用程序之间流式传输大量数据。此数据会保留并存储在Kafka集群中的Kafka代理节点中。Kafka也以可用性和并行性而闻名、它通过将主题划分为分区、然后在整个集群中复制这些分区来实现这一点。这最终意味着、流经Kafka集群的大量数据通常会成倍增加。NFS可以随着代理数量的变化轻松快速地重新平衡数据。对于大型环境、在代理数量发生变化时跨DAS重新平衡数据非常耗时、在大多数Kafka环境中、代理数量经常发生变化。

其他优势包括：

- 成熟度。 NFS是一种成熟的协议、这意味着实施、保护和使用NFS的大部分方面都已被充分理解。
- 开放式 NFS是一种开放式协议、其持续开发已作为一种免费的开放式网络协议记录在互联网规格中。
- 经济高效。 NFS是一种用于网络文件共享的低成本解决方案 、由于它使用现有网络基础架构、因此易于设置。
- *集中管理。*集中管理NFS可减少单个用户系统上对添加软件和磁盘空间的需求。
- 分布式。 NFS可用作分布式文件系统、从而减少了对可移动介质存储设备的需求。

为什么选择**NetApp**来处理**Kafka**工作负载?

NetApp NFS实施被视为该协议的黄金标准、用于无数企业级NAS环境。除了NetApp的信誉之外、它还具有以下优势：

- 可靠性和效率
- 可扩展性和性能
- 高可用性(NetApp ONTAP 集群中的HA配对节点)
- 数据保护
  - *灾难恢复(NetApp SnapMirror)。*您的站点发生故障、或者您希望从其他站点跳转并从您离开的位置继续。
  - 存储系统的易管理性(使用NetApp OnCommand 进行管理和管理)。
  - *负载平衡。*集群允许您从不同节点上托管的数据LIF访问不同的卷。
  - 无中断操作。 LIF或卷移动对NFS客户端是透明的。

NetApp解决方案 **for fly**将问题描述 **for NFS**重命名为**Kafka**工作负载

Kafka的构建假定底层文件系统符合POSIX标准：例如XFS或ext4。Kafka资源重新平衡会在应用程序仍在使用文件时删除这些文件。符合POSIX的文件系统允许取消链接以继续。但是、它仅会在对文件的所有引用均消失后删除该文件。如果底层文件系统已通过网络连接、则NFS客户端将截获取消链接调用并管理工作流。由于正在取消链接的文件存在待定打开状态、因此NFS客户端会向NFS服务器发送重命名请求、并在未链接的文件最后一次关闭时对已重命名的文件执行删除操作。此行为通常称为NFS愚蠢的重命名、它由NFS客户端进行编排。

由于此行为、使用NFSv3服务器中存储的任何Kafka代理都会遇到问题。但是、NFSv4.x协议具有一些功能、可以通过允许服务器对打开的未链接文件负责来解决此问题描述。支持此可选功能的NFS服务器会在文件打开时将所有权功能传递给NFS客户端。然后、当打开并等待处理时、NFS客户端将停止取消链接管理、并允许服务器管理此流。虽然NFSv4规范提供了实施准则、但到目前为止、还没有任何已知的NFS服务器实施支持此可选功能。

NFS服务器和NFS客户端需要进行以下更改、才能处理愚蠢的重命名问题描述：

- *对NFS客户端(Linux)所做的更改。*打开文件时、NFS服务器会通过一个标志进行响应、指示能够处理已打开文件的解除链接。NFS客户端更改允许NFS服务器在存在标志的情况下处理解除链接。NetApp已使用这些更改更新了开源Linux NFS客户端。更新后的NFS客户端现在在RHEL8.7和RHEL9.1中普遍可用。
- 对**NFS**服务器所做的更改。 NFS服务器会跟踪打开情况。现在、服务器会管理现有打开文件的取消链接、以匹配POSIX语义。关闭上次打开的文件后、NFS服务器会启动文件的实际删除、从而避免了愚蠢的重命名过程。ONTAP NFS服务器已在其最新版本ONTAP 9.12.1.中实施此功能。

通过对NFS客户端和服务器进行上述更改、Kafka可以安全地获得网络连接NFS存储的所有优势。

功能验证—错误的重命名修复

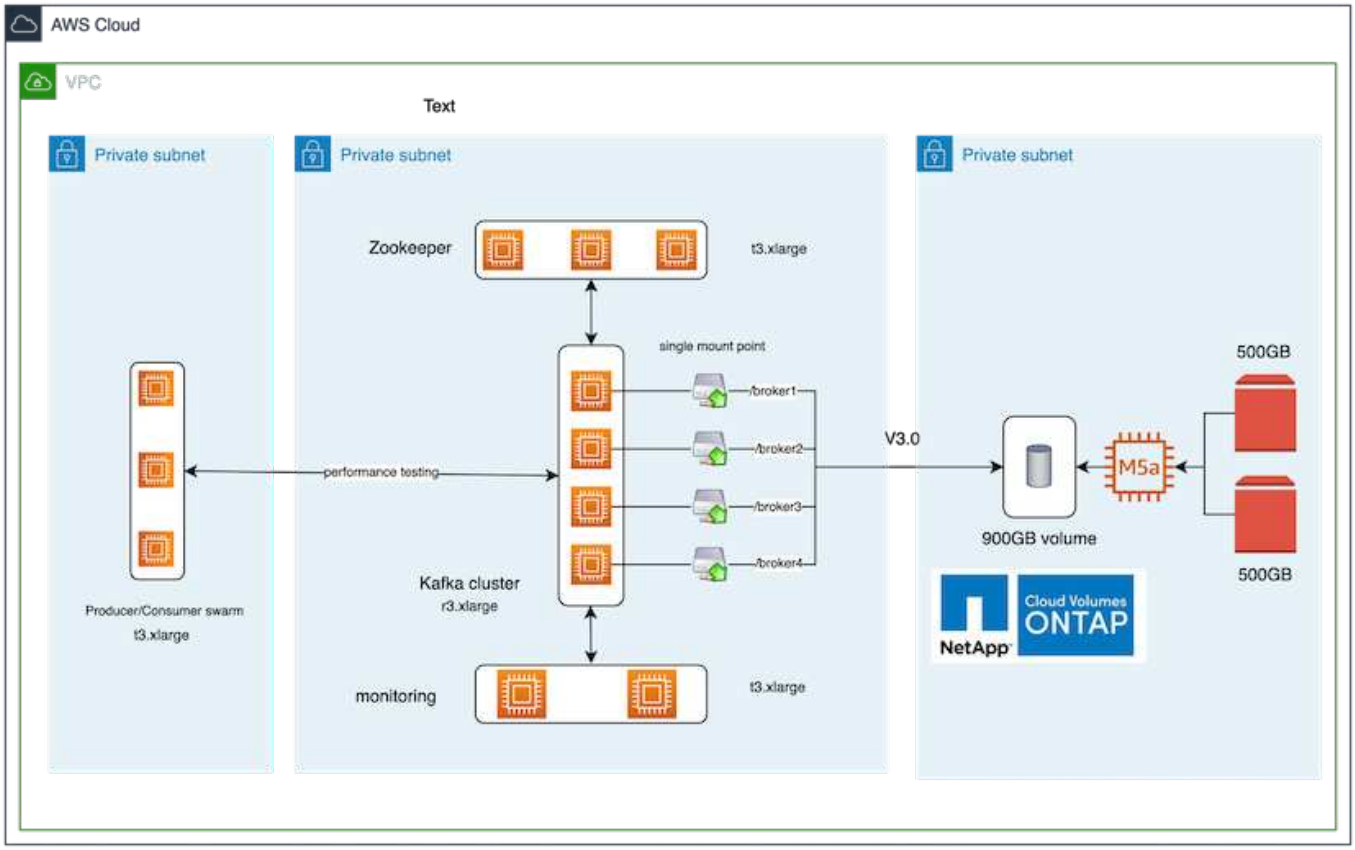
在功能验证方面、我们显示了具有NFSv3存储挂载的Kafka集群无法执行分区重新分配等Kafka操作、而具有修复程序的NFSv4上挂载的另一个集群可以执行相同的操作而不会造成任何中断。

验证设置

此设置将在AWS上运行。下表显示了用于验证的不同平台组件和环境配置。

平台组件	环境配置
Confuent Platform 7.2.1版	• 3个Zookepers—T3.xlarge • 4个代理服务器—r3.xlarge • 1个Grafana—T3.xlarge • 1个控制中心—T3.xlarge • 3个生产者/使用者
所有节点上的操作系统	RHEL8.7或更高版本
NetApp Cloud Volumes ONTAP 实例	单节点实例—M5.2xLarge

下图显示了此解决方案 的架构配置。



架构流程

- *计算。*我们使用了一个四节点Kafka集群、其中三节点Zookeeper集合在专用服务器上运行。
- *监控。*我们将两个节点用于Prometheus-Grafana组合。
- *工作负载。*为了生成工作负载、我们使用了一个单独的三节点集群、该集群可以生成并使用此Kafka集群。

- *存储。*我们使用了一个单节点NetApp Cloud Volumes ONTAP 实例、该实例连接了两个500 GB GP2 AWS-EBS卷。然后、这些卷会通过LIF作为一个NFSv4.1卷公开到Kafka集群中。

已为所有服务器选择Kafka的默认属性。Zookeeper Swarm也是如此。

测试方法

1. 更新 `-is-preserve-unlink-enabled true` 到Kafka卷、如下所示：

```
aws-shantanclastrecall-aws::*> volume create -vserver kafka_svm -volume
kafka_fg_vol01 -aggregate kafka_aggr -size 3500GB -state online -policy
kafka_policy -security-style unix -unix-permissions 0777 -junction-path
/kafka_fg_vol01 -type RW -is-preserve-unlink-enabled true
[Job 32] Job succeeded: Successful
```

2. 创建了两个类似的Kafka集群、但差别如下：
   - *集群1.*运行生产就绪型ONTAP 9.12.1的后端NFS v4.1服务器由NetApp CVO实例托管。这些代理上安装了RHEL 8.7/RHEL 9.1。
   - *集群2.*后端NFS服务器是手动创建的通用Linux NFSv3服务器。
3. 在这两个Kafka集群上都创建了一个演示主题。

   集群1：

```
[root@ip-172-30-0-160 demo]# kafka-topics --bootstrap-server=172.30.0.160:9092,172.30.0.172:9092,172.30.0.188:90
92,172.30.0.123:9092 --describe --topic __a_demo_topic
Topic: __a_demo_topic   TopicId: 2ty29xfhQLq65HKsUQv-pg PartitionCount: 4       ReplicationFactor: 2    Configs:
 min.insync.replicas=1,segment.bytes=1073741824
        Topic: __a_demo_topic   Partition: 0    Leader: 4       Replicas: 4,1   Isr: 4,1        Offline:
        Topic: __a_demo_topic   Partition: 1    Leader: 2       Replicas: 2,4   Isr: 2,4        Offline:
        Topic: __a_demo_topic   Partition: 2    Leader: 3       Replicas: 3,2   Isr: 3,2        Offline:
        Topic: __a_demo_topic   Partition: 3    Leader: 1       Replicas: 1,3   Isr: 1,3        Offline:
```

   集群2：

```
[root@ip-172-30-0-198 demo]# kafka-topics --bootstrap-server=172.30.0.198:9092,172.30.0.163:9092,172.30.0.221:90
92,172.30.0.204:9092 --describe --topic __a_demo_topic
Topic: __a_demo_topic   TopicId: AwQpsZTQShyeMIhaquCG3Q PartitionCount: 4       ReplicationFactor: 2    Configs:
 min.insync.replicas=1,segment.bytes=1073741824
        Topic: __a_demo_topic   Partition: 0    Leader: 2       Replicas: 2,3   Isr: 2,3        Offline:
        Topic: __a_demo_topic   Partition: 1    Leader: 3       Replicas: 3,1   Isr: 3,1        Offline:
        Topic: __a_demo_topic   Partition: 2    Leader: 1       Replicas: 1,4   Isr: 1,4        Offline:
        Topic: __a_demo_topic   Partition: 3    Leader: 4       Replicas: 4,2   Isr: 4,2        Offline:
```

4. 数据已加载到这两个集群的新创建主题中。这是使用默认Kafka软件包中提供的producer-perf-test工具包实现的：

```
./kafka-producer-perf-test.sh --topic __a_demo_topic --throughput -1
--num-records 3000000 --record-size 1024 --producer-props acks=all
bootstrap.servers=172.30.0.160:9092,172.30.0.172:9092,172.30.0.188:9092,
172.30.0.123:9092
```

5. 已使用telnet对每个集群的Broker-1执行运行状况检查：
   - Telnet `172.30.0.160 9092`
   - Telnet `172.30.0.198 9092`

     下一个屏幕截图显示了两个集群上的代理的成功运行状况检查：

```
shantanu@shantanc-mac-0 ~ % telnet 172.30.0.160 9092
Trying 172.30.0.160...
Connected to 172.30.0.160.
Escape character is '^]'.
^[

Connection closed by foreign host.
shantanu@shantanc-mac-0 ~ % telnet 172.30.0.198 9092
Trying 172.30.0.198...
Connected to 172.30.0.198.
Escape character is '^]'.
^[
```

6. 为了触发导致使用NFSv3存储卷的Kafka集群崩溃的故障情况、我们在这两个集群上启动了分区重新分配过程。分区重新分配是使用执行的 `kafka-reassign-partitions.sh`。详细过程如下：
   a. 为了为Kafka集群中的某个主题重新分配分区、我们生成了建议的重新分配配置JSON (这是为这两个集群执行的)。

```
kafka-reassign-partitions --bootstrap
-server=172.30.0.160:9092,172.30.0.172:9092,172.30.0.188:9092,172.30.
0.123:9092 --broker-list "1,2,3,4" --topics-to-move-json-file
/tmp/topics.json --generate
```

   b. 生成的重新分配JSON随后保存在中 `/tmp/reassignment- file.json`。
   c. 实际分区重新分配过程由以下命令触发：

```
kafka-reassign-partitions --bootstrap
-server=172.30.0.198:9092,172.30.0.163:9092,172.30.0.221:9092,172.30.
0.204:9092 --reassignment-json-file /tmp/reassignment-file.json
-execute
```

7. 在完成重新分配几分钟后、对代理进行的另一项运行状况检查显示、使用NFSv3存储卷的集群运行到一个错误的重命名问题描述 中并发生崩溃、而使用NetApp ONTAP NFSv4.1存储卷的集群1则在修复后继续运行、而不会造成任何中断。

```
shantanu@shantanc-mac-0 ~ % telnet 172.30.0.160 9092
Trying 172.30.0.160...
Connected to 172.30.0.160.
Escape character is '^]'.
^[

Connection closed by foreign host.
shantanu@shantanc-mac-0 ~ % telnet 172.30.0.198 9092
Trying 172.30.0.198...
telnet: connect to address 172.30.0.198: Connection refused
telnet: Unable to connect to remote host
```

- cluster1-Broker-1处于活动状态。
- CLUSTER2-Broker-1已失效。

8. 检查Kafka日志目录后、可以明显看出、使用NetApp ONTAP NFSv4.1存储卷并进行修复的集群1分配了干净的分区、而使用通用NFSv3存储的集群2则不是由于错误的重命名问题而导致崩溃。下图显示了集群2的分区重新平衡、这会导致在NFSv3存储上对问题描述 进行重命名、操作很不明智。

```
/demo/broker_demo_1/__a_demo_topic-1.b31a8dd60fd443b283ffda2ecca9c2b9-delete:
total 40
drwxr-xr-x.   2 nobody nobody  4096 Sep 19 10:37 .
drwxr-xr-x. 246 nobody nobody 32768 Sep 19 10:36 ..
-rw-r--r--.   1 nobody nobody     5 Sep 19 10:22 .nfs0000000025f9008400000045
-rw-r--r--.   1 nobody nobody     0 Sep 19 10:25 .nfs0000000025f91d6800000048

/demo/broker_demo_1/__a_demo_topic-2:
total 832592
drwxr-xr-x.   2 nobody nobody      4096 Sep 19 10:26 .
drwxr-xr-x. 246 nobody nobody     32768 Sep 19 10:36 ..
-rw-r--r--.   1 nobody nobody         5 Sep 19 10:22 .nfs0000000025f91d5500000046
-rw-r--r--.   1 nobody nobody         0 Sep 19 10:25 .nfs0000000025f91fce00000047
-rw-r--r--.   1 nobody nobody  10485760 Sep 19 10:24 00000000000000000000.index
-rw-r--r--.   1 nobody nobody 848113134 Sep 19 10:24 00000000000000000000.log
-rw-r--r--.   1 nobody nobody  10485756 Sep 19 10:24 00000000000000000000.timeindex
-rw-r--r--.   1 nobody nobody         0 Sep 19 10:16 leader-epoch-checkpoint
-rw-r--r--.   1 nobody nobody        43 Sep 19 10:16 partition.metadata
```

下图显示了使用NetApp NFSv4.1存储重新平衡集群1的全新分区。

```
/demo/broker_demo_1/__a_demo_topic-0:
total 710932
drwxr-xr-x.  2 nobody nobody      4096 Sep 19 10:26 .
drwxr-xr-x. 85 nobody nobody      8192 Sep 19 10:37 ..
-rw-r--r--.  1 nobody nobody  10485760 Sep 19 10:25 00000000000000000000.index
-rw-r--r--.  1 nobody nobody 724167522 Sep 19 10:25 00000000000000000000.log
-rw-r--r--.  1 nobody nobody  10485756 Sep 19 10:25 00000000000000000000.timeindex
-rw-r--r--.  1 nobody nobody         0 Sep 19 10:15 leader-epoch-checkpoint
-rw-r--r--.  1 nobody nobody        43 Sep 19 10:15 partition.metadata

/demo/broker_demo_1/__a_demo_topic-2:
total 780016
drwxr-xr-x.  2 nobody nobody      4096 Sep 19 10:35 .
drwxr-xr-x. 85 nobody nobody      8192 Sep 19 10:37 ..
-rw-r--r--.  1 nobody nobody  10485760 Sep 19 10:36 00000000000000000000.index
-rw-r--r--.  1 nobody nobody 794575786 Sep 19 10:36 00000000000000000000.log
-rw-r--r--.  1 nobody nobody  10485756 Sep 19 10:36 00000000000000000000.timeindex
-rw-r--r--.  1 nobody nobody         0 Sep 19 10:35 leader-epoch-checkpoint
-rw-r--r--.  1 nobody nobody        43 Sep 19 10:35 partition.metadata
```

为什么选择适用于**Kafka**工作负载的**NetApp NFS**?

现在、在使用Kafka的NFS存储中、有一个解决方案 用于愚蠢地重命名问题描述 、您可以创建强大的部署、利用NetApp ONTAP 存储来处理Kafka工作负载。这样不仅可以显著降低运营开销、还可以为Kafka集群带来以下优势：

- *降低Kafka代理的CPU利用率。*使用分解的NetApp ONTAP 存储可将磁盘I/O操作与代理分离、从而减少其CPU占用空间。
- *代理恢复时间更快。*由于分离式NetApp ONTAP 存储在Kafka代理节点之间共享、因此与传统Kafka部署相比、新的计算实例可以在任意时间点替换损坏的代理、而无需重建数据。
- *存储效率。*由于应用程序的存储层现在通过NetApp ONTAP 进行配置、因此客户可以利用ONTAP 带来的存储效率的所有优势、例如实时数据压缩、重复数据删除和数据缩减。

我们在本节详细讨论的测试案例中对这些优势进行了测试和验证。

降低了**Kafka**代理的**CPU**利用率

我们发现、当我们在两个sperate Kafka集群上运行类似的工作负载时、总CPU利用率低于其DAS对应项、这两个集群的技术规格相同、但存储技术却不同。当Kafka集群使用ONTAP 存储时、不仅整体CPU利用率较低、而且CPU利用率的增加也显示出比基于DAS的Kafka集群更温和的梯度。

架构设置

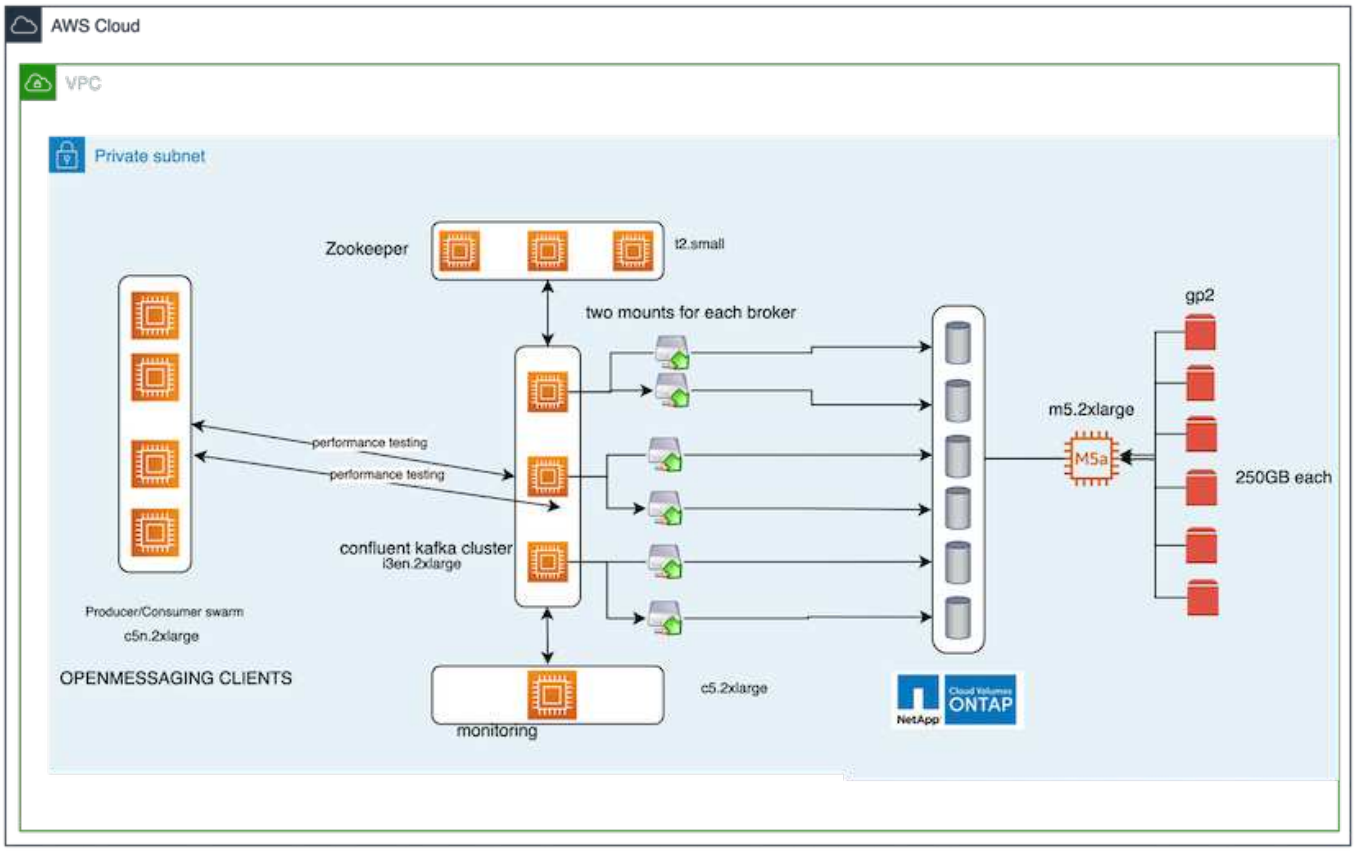
下表显示了用于展示CPU利用率降低情况的环境配置。

平台组件	环境配置
Kafka 3.2.3基准工具：OpenMessaging	• 3个Zookepers—T2.Small • 3个代理服务器—i3en.2xlarge • 1个Grafana—c5n.2xlarge • 4个生产者/使用者—c5n.2xlarge

平台组件	环境配置
所有节点上的操作系统	RHEL 8.7或更高版本
NetApp Cloud Volumes ONTAP 实例	单节点实例—M5.2xLarge

基准测试工具

此测试案例中使用的基准测试工具是 "Open消息 传送" 框架。Openmessaging不受供应商限制、不受语言限制；它为金融、电子商务、物联网和大数据提供行业指导；它有助于跨异构系统和平台开发消息传送和流式传输应用程序。下图展示了Open消息 客户端与Kafka集群的交互。



- *计算。*我们使用了一个三节点Kafka集群、其中三节点Zookeeper集合在专用服务器上运行。每个代理都通过一个专用LIF将两个NFSv4.1挂载点连接到NetApp CVO实例上的一个卷。
- *监控。*我们将两个节点用于Prometheus-Grafana组合。为了生成工作负载、我们提供了一个单独的三节点集群、该集群可以生成并使用此Kafka集群。
- *存储。*我们使用了一个单节点NetApp Cloud Volumes ONTAP 实例、该实例上挂载了六个250 GB GP2 AWS-EBS卷。然后、这些卷会通过专用LIF作为六个NFSv4.1卷公开到Kafka集群中。
- *配置。*本测试用例中的两个可配置元素是Kafka代理和Open消息 工作负载。
  - *代理配置*为Kafka代理选择了以下规格。我们对所有测量结果使用了复制因子3、如下所示。

```
broker.id=1
advertised.listeners=PLAINTEXT://172.30.0.185:9092
log.dirs=/mnt/data-1
zookeeper.connect=172.30.0.13:2181,172.30.0.108:2181,172.30.0.253:2181
num.replica.fetchers=8
message.max.bytes=10485760
replica.fetch.max.bytes=10485760
num.network.threads=8
default.replication.factor=3
replica.lag.time.max.ms=100000000
replica.fetch.max.bytes=1048576
replica.fetch.wait.max.ms=500
num.replica.fetchers=1
replica.high.watermark.checkpoint.interval.ms=5000
fetch.purgatory.purge.interval.requests=1000
producer.purgatory.purge.interval.requests=1000
replica.socket.timeout.ms=30000
replica.socket.receive.buffer.bytes=65536
```

- 提供了以下规格：* Open消息 基准测试(OMB)工作负载配置*。我们指定了一个目标生产者比率、并在下面重点说明了这一比率。

```
name: 4 producer / 4 consumers on 1 topic
topics: 1
partitionsPerTopic: 100
messageSize: 1024
payloadFile: "payload/payload-1Kb.data"
subscriptionsPerTopic: 1
consumerPerSubscription: 4
producersPerTopic: 4
producerRate: 40000
consumerBacklogSizeGB: 0
testDurationMinutes: 5
```

测试方法

1. 创建了两个类似的集群、每个集群都有自己的一组基准集群Swarms。
   - *集群1.*基于NFS的Kafka集群。
   - *集群2.*基于DAS的Kafka集群。
2. 使用Open消息 命令、在每个集群上触发类似的工作负载。

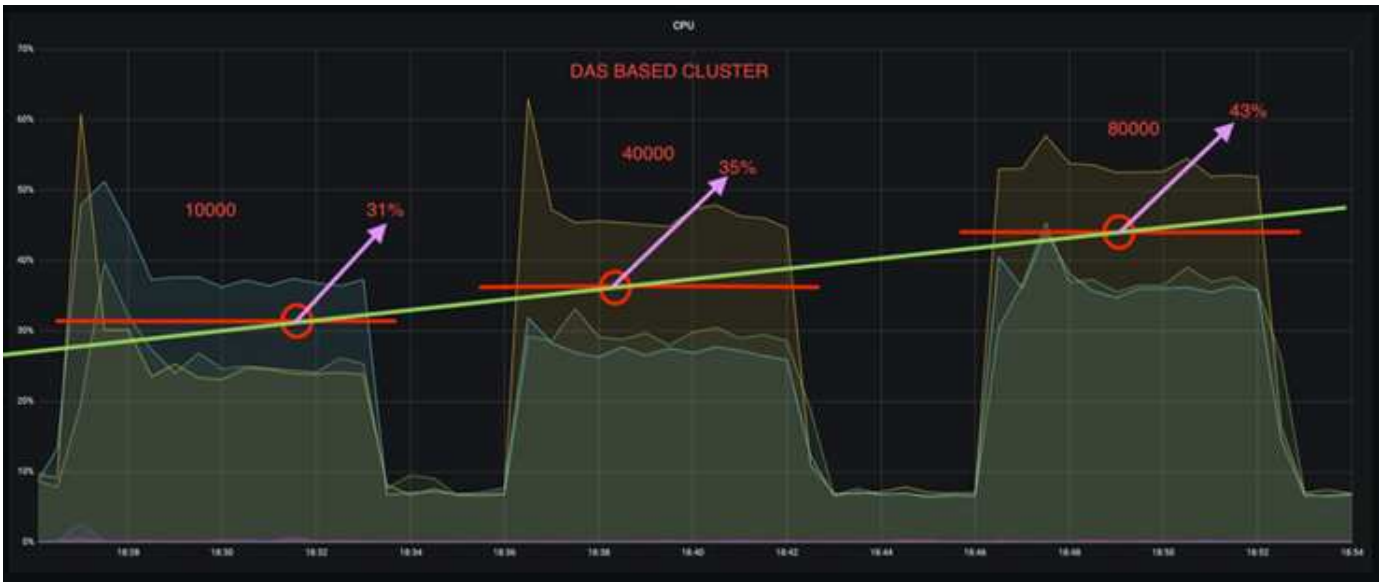
```
sudo bin/benchmark --drivers driver-kafka/kafka-group-all.yaml
workloads/1-topic-100-partitions-1kb.yaml
```

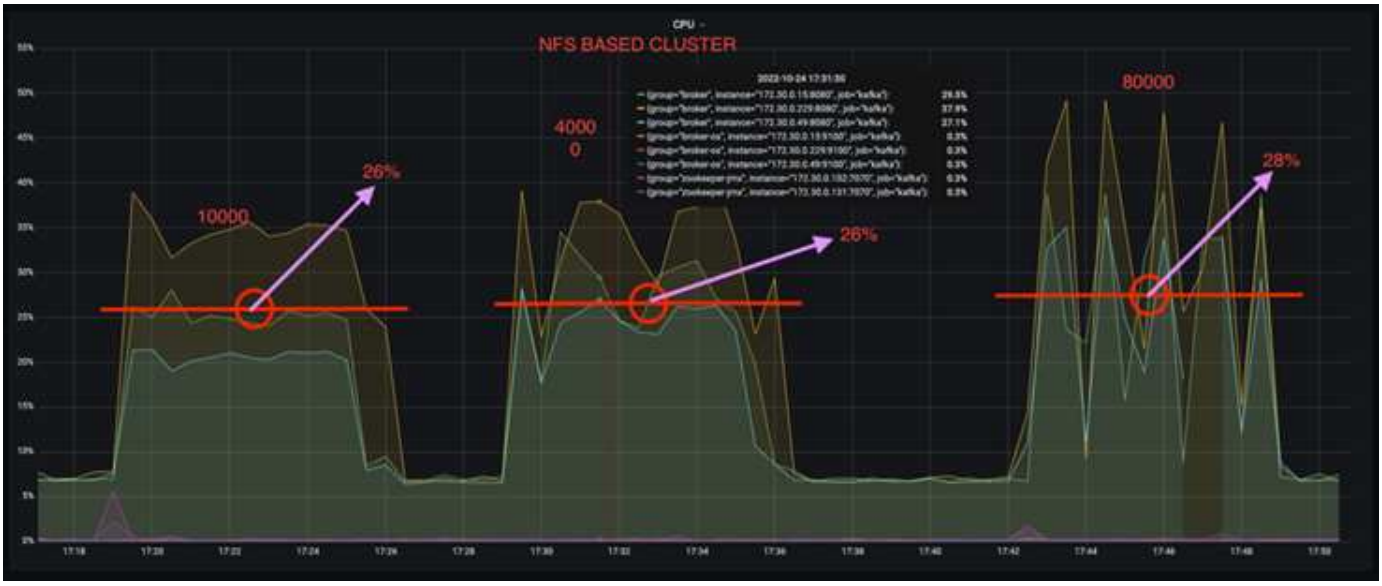
3. 生产率配置在四次迭代中增加、CPU利用率记录在Grafana中。生产率设置为以下级别：
    - 10、000
    - 40,000
    - 80、000
    - 100、000

观察结果

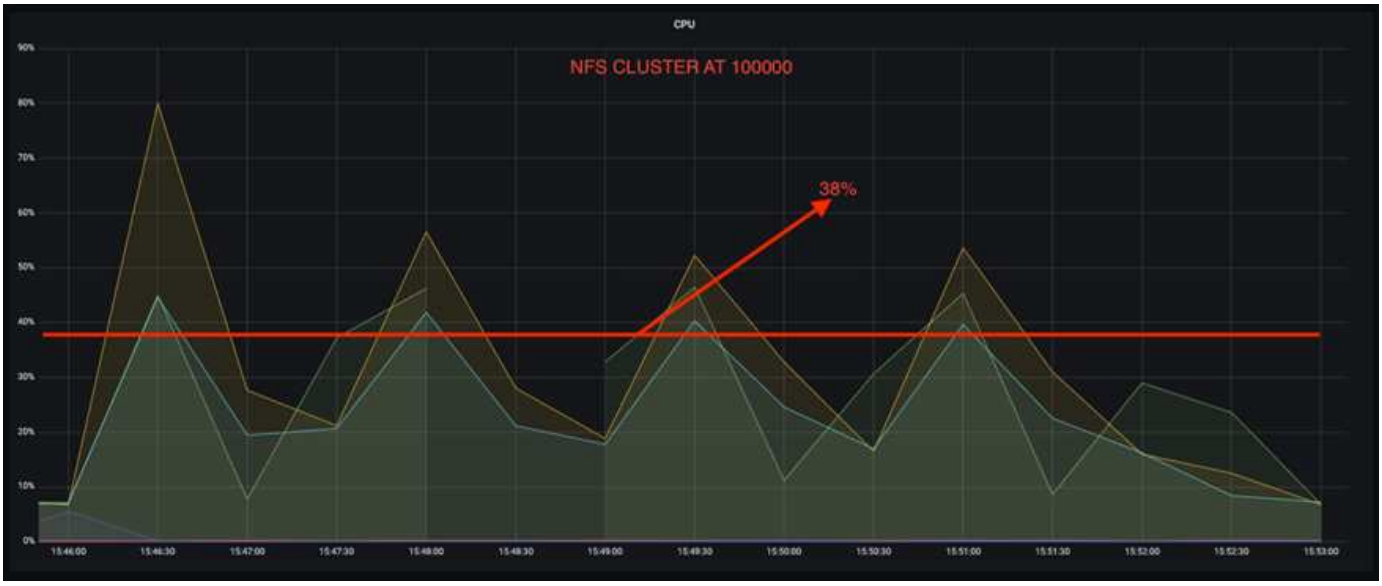
将NetApp NFS存储与Kafka结合使用具有两个主要优势：

- *您可以将CPU利用率降低近三分之一。*与DAS SSD相比、NFS在类似工作负载下的整体CPU利用率更低；节省量从较低生产率的5%到较高生产率的32%不等。
- *在较高的生产率下、CPU利用率漂移减少了三倍。*正如预期的那样、随着生产率的增加、CPU利用率的增加也出现了上升趋势。但是、使用DAS的Kafka代理的CPU利用率从较低生产率的31%上升到较高生产率的70%、即增加39%。但是、在NFS存储后端、CPU利用率从26%上升到38%、增加了12%。





此外、当消息达到100、000时、DAS显示的CPU利用率比NFS集群高。



代理恢复速度更快

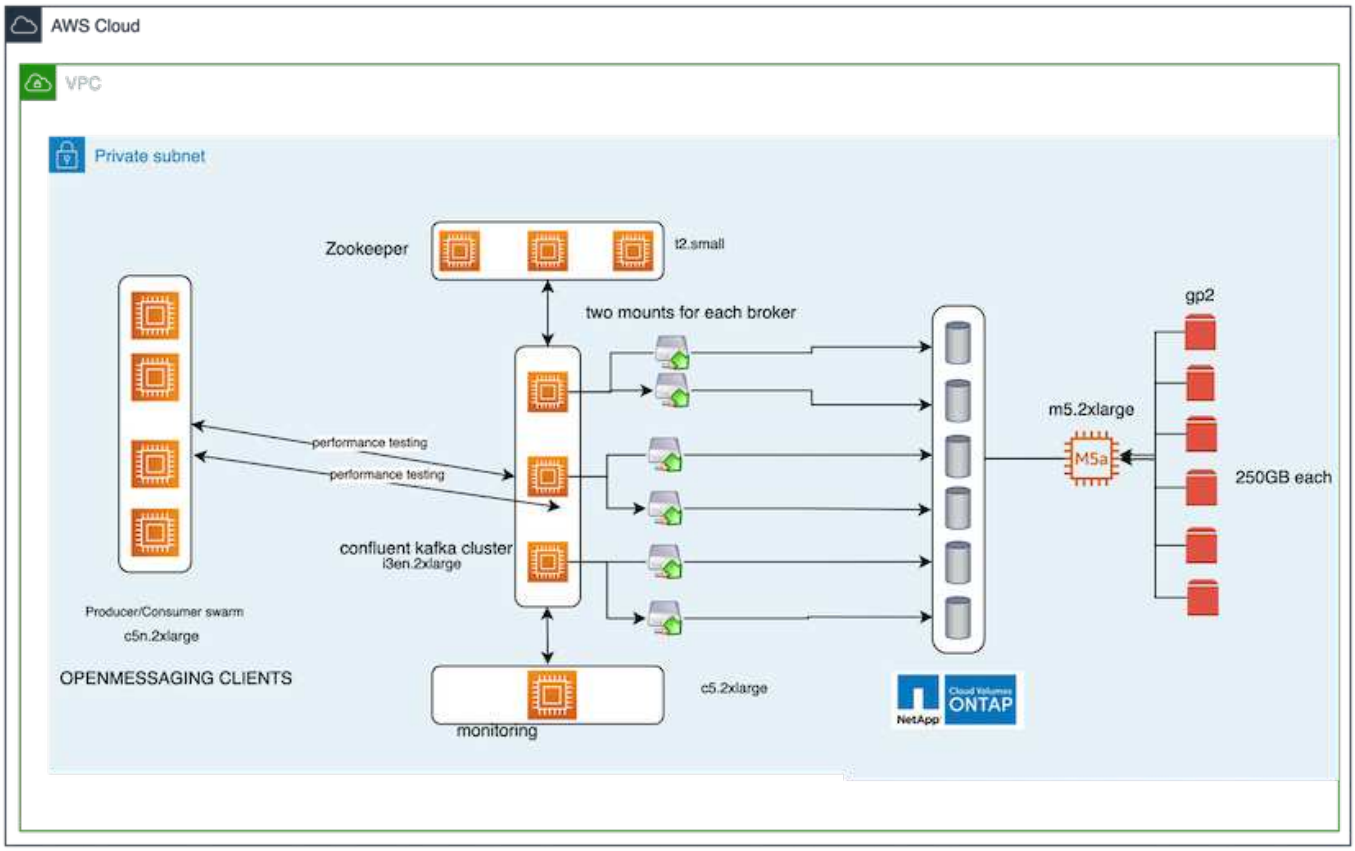
我们发现、Kafka代理在使用共享NetApp NFS存储时恢复速度更快。当Kafka集群中的代理崩溃时、可以使用具有相同代理ID的运行状况良好的代理来替换此代理。执行此测试案例后、我们发现、对于基于DAS的Kafka集群、集群会在新添加的运行状况良好的代理上重建数据、这非常耗时。对于基于NetApp NFS的Kafka集群、替代代理将继续从先前的日志目录读取数据并以更快的速度恢复。

架构设置

下表显示了使用NAS的Kafka集群的环境配置。

平台组件	环境配置
Kafka 3.2.3	• 3个Zookepers—T2.Small • 3个代理服务器—i3en.2xlarge • 1个Grafana—c5n.2xlarge • 4个生产者/使用者—c5n.2xlarge • 1个备份Kafka节点—i3en.2xlarge
所有节点上的操作系统	RHEL8.7或更高版本
NetApp Cloud Volumes ONTAP 实例	单节点实例—M5.2xLarge

下图展示了基于NAS的Kafka集群的架构。



- *计算。*一种三节点Kafka集群、其中三节点zookeeper集合在专用服务器上运行。每个代理都有两个NFS挂载点、可通过专用LIF连接到NetApp CVO实例上的一个卷。
- 监控。 Prometheus-Grafana组合的两个节点。在生成工作负载时、我们会使用一个单独的三节点集群来生成此Kafka集群并将其使用。
- *存储。*一个单节点NetApp Cloud Volumes ONTAP 实例、该实例上挂载了六个250 GB GP2 AWS-EBS卷。然后、这些卷会通过专用LIF作为六个NFS卷公开到Kafka集群中。
- *代理配置。*本测试用例中的一个可配置元素是Kafka代理。为Kafka代理选择了以下规格。。`replica.lag.time.mx.ms` 设置为高值、因为这决定了从ISR列表中删除特定节点的速度。在不良节点和运行状况良好的节点之间切换时、您不希望从ISR列表中排除该代理ID。

```
broker.id=1
advertised.listeners=PLAINTEXT://172.30.0.185:9092
log.dirs=/mnt/data-1
zookeeper.connect=172.30.0.13:2181,172.30.0.108:2181,172.30.0.253:2181
num.replica.fetchers=8
message.max.bytes=10485760
replica.fetch.max.bytes=10485760
num.network.threads=8
default.replication.factor=3
replica.lag.time.max.ms=100000000
replica.fetch.max.bytes=1048576
replica.fetch.wait.max.ms=500
num.replica.fetchers=1
replica.high.watermark.checkpoint.interval.ms=5000
fetch.purgatory.purge.interval.requests=1000
producer.purgatory.purge.interval.requests=1000
replica.socket.timeout.ms=30000
replica.socket.receive.buffer.bytes=65536
```

测试方法

1. 创建了两个类似的集群：
   - 基于EC2的融合集群。
   - 基于NetApp NFS的融合集群。
2. 创建了一个备用Kafka节点、其配置与原始Kafka集群中的节点相同。
3. 在每个集群上创建了一个示例主题、并在每个代理上填充了大约110 GB的数据。
   - 基于* EC2的集群。*已映射Kafka代理数据目录 `/mnt/data-2` (在下图中、为cluster1的Broker-1 (左端子)。
   - 基于**NetApp NFS**的集群。 Kafka代理数据目录挂载在NFS点上 `/mnt/data` (在下图中、为cluster2的Broker-1 [右端子])。

4. 在每个集群中、Broker-1都已终止、以触发失败的代理恢复过程。
5. 代理终止后、代理IP地址将作为二级IP分配给备用代理。之所以需要这样做、是因为Kafka集群中的代理可通过以下方式进行标识：
   - 通过将故障代理IP重新分配给备用代理来分配* IP地址*。
   - 代理**ID**。此ID已在备用代理中配置 `server.properties`。
6. 分配IP后、在备用代理上启动了Kafka服务。
7. 一段时间后、服务器日志被提取、用于检查在集群中的替代节点上构建数据所用的时间。

观察结果

Kafka代理恢复速度几乎是原来的九倍。我们发现、与在Kafka集群中使用DAS SSD相比、使用NetApp NFS共

享存储时、恢复发生故障的代理节点所需的时间要快得多。对于1 TB的主题数据、基于DAS的集群的恢复时间为48分钟、而基于NetApp-NFS的Kafka集群的恢复时间不到5分钟。

我们发现、基于EC2的集群需要10分钟才能在新代理节点上重建110 GB的数据、而基于NFS的集群则需要3分钟才能完成恢复。我们还在日志中观察到、EC2分区的使用者偏移量为0、而在NFS集群上、使用者偏移量是从先前的代理中获取的。

```
[2022-10-31 09:39:17,747] INFO [LogLoader partition=test-topic-51R3EWs-
0000-55, dir=/mnt/kafka-data/broker2] Reloading from producer snapshot and
rebuilding producer state from offset 583999 (kafka.log.UnifiedLog$)
[2022-10-31 08:55:55,170] INFO [LogLoader partition=test-topic-qbVsEZg-
0000-8, dir=/mnt/data-1] Loading producer state till offset 0 with message
format version 2 (kafka.log.UnifiedLog$)
```

基于**DAS**的集群

1. 备份节点从08：55：53、730开始。

```
[2022-10-31 08:55:53,661] INFO Setting -D jdk.tls.rejectClientInitiatedRenegotia
[2022-10-31 08:55:53,727] INFO Registered signal handlers for TERM, INT, HUP (or
[2022-10-31 08:55:53,730] INFO starting (kafka.server.KafkaServer)
[2022-10-31 08:55:53,730] INFO Connecting to zookeeper on 172.30.0.17:2181,172.3
[2022-10-31 08:55:53,755] INFO [ZooKeeperClient Kafka server] Initializing a new
```

2. 数据重建过程于09：05：24、860结束。处理110 GB的数据大约需要10分钟。

```
[2022-10-31 09:05:24,860] INFO [ReplicaFetcherManager on broker 1] Removed fetcher for
partitions HashSet(test-topic-qbVsEZg-0000-95, test-topic-qbVsEZg-0000-5,
test-topic-qbVsEZg-0000-41, test-topic-qbVsEZg-0000-23, test-topic-qbVsEZg-0000-11,
test-topic-qbVsEZg-0000-47, test-topic-qbVsEZg-0000-83, test-topic-qbVsEZg-0000-35,
test-topic-qbVsEZg-0000-89, test-topic-qbVsEZg-0000-71, test-topic-qbVsEZg-0000-53,
test-topic-qbVsEZg-0000-29, test-topic-qbVsEZg-0000-59, test-topic-qbVsEZg-0000-77,
test-topic-qbVsEZg-0000-65, test-topic-qbVsEZg-0000-17)
(kafka.server.ReplicaFetcherManager)
```

基于**NFS**的集群

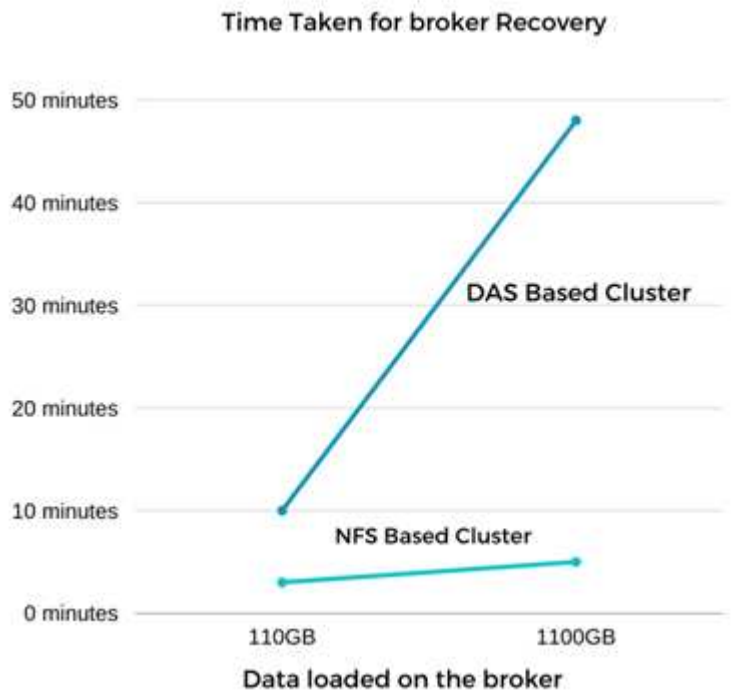
1. 备份节点的启动时间为09：39：17、213。下面突出显示了起始日志条目。

```
[2022-10-31 09:39:17,142] INFO Setting -D jdk.tls.rejectClientInitiatedRenegotiati
[2022-10-31 09:39:17,211] INFO Registered signal handlers for TERM, INT, HUP (org.
[2022-10-31 09:39:17,213] INFO starting (kafka.server.KafkaServer)
[2022-10-31 09:39:17,214] INFO Connecting to zookeeper on 172.30.0.22:2181,172.30.
[2022-10-31 09:39:17,238] INFO [ZooKeeperClient Kafka server] Initializing a new s
[2022-10-31 09:39:17,244] INFO Client environment:zookeeper.version=3.6.3--6401e4a
[2022-10-31 09:39:17,244] INFO Client environment:host.name=ip-172-30-0-110.ec2.in
[2022-10-31 09:39:17,244] INFO Client environment:java.version=11.0.17 (org.apache
```

2. 数据重建过程于09：42：29、115结束。处理110 GB的数据大约需要3分钟。

```
[2022-10-31 09:42:29,115] INFO [GroupMetadataManager brokerId=1] Finished loading offsets
and group metadata from __consumer_offsets-20 in 28478 milliseconds for epoch 3, of which
28478 milliseconds was spent in the scheduler.
(kafka.coordinator.group.GroupMetadataManager)
```

对于包含大约1 TB数据的代理、重复执行此测试、对于DAS、此测试需要大约48分钟、对于NFS、此测试需要3分钟。下图显示了这些结果。



存储效率

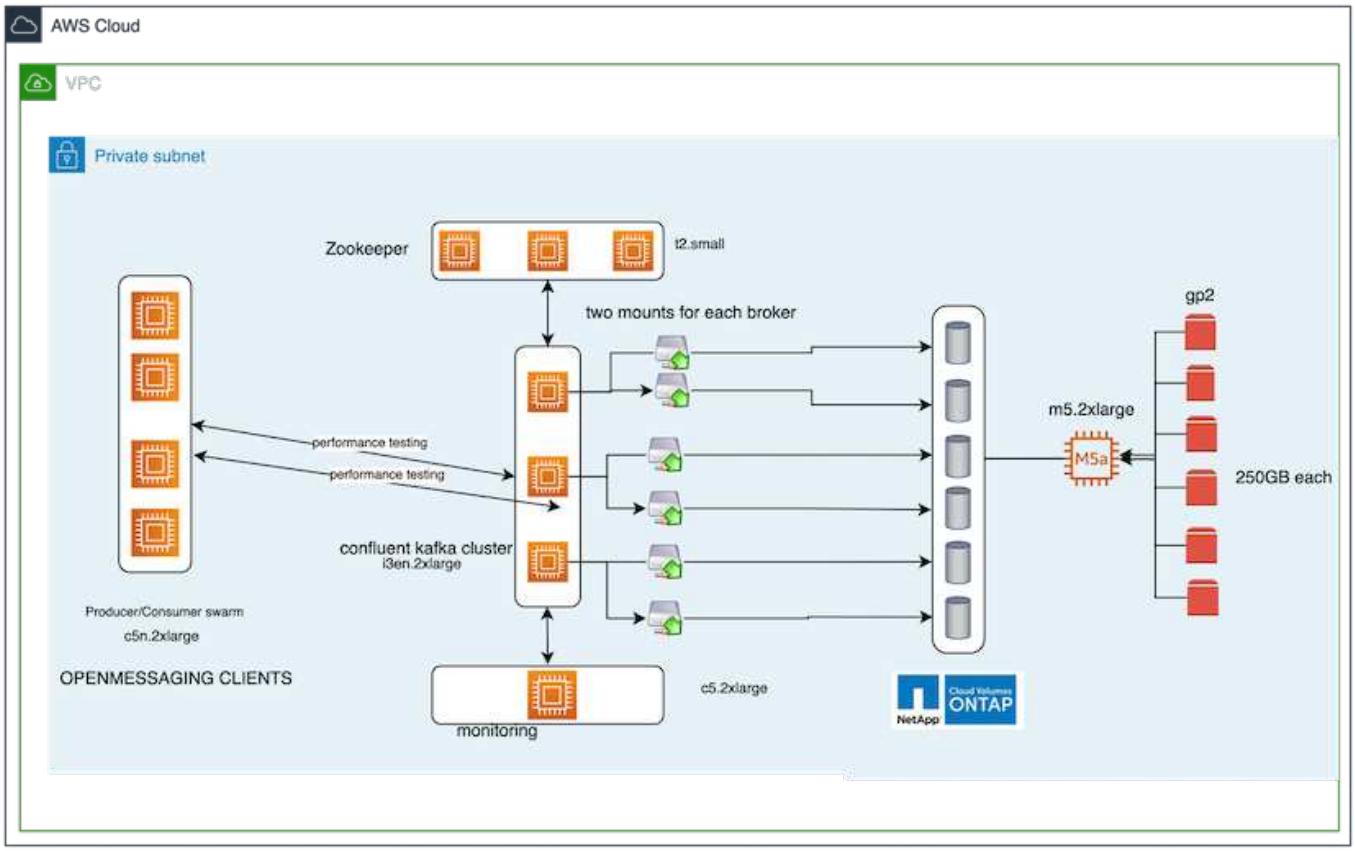
由于Kafka集群的存储层是通过NetApp ONTAP 配置的、因此我们获得了ONTAP 的所有存储效率功能。测试方法是、在Cloud Volumes ONTAP 上配置了NFS存储的Kafka集群上生成大量数据。我们可以看到、由于ONTAP 功能、空间显著减少。

架构设置

下表显示了使用NAS的Kafka集群的环境配置。

平台组件	环境配置
Kafka 3.2.3	• 3个Zookepers—T2.Small • 3个代理服务器—i3en.2xlarge • 1个Grafana—c5n.2xlarge • 4个生产者/使用者—c5n.2xlarge *
所有节点上的操作系统	RHEL8.7或更高版本
NetApp Cloud Volumes ONTAP 实例	单节点实例—M5.2xLarge

下图展示了基于NAS的Kafka集群的架构。



- *计算。*我们使用了一个三节点Kafka集群、其中三节点Zookeeper集合在专用服务器上运行。每个代理都通过一个专用LIF在NetApp CVO实例上有两个NFS挂载点到一个卷。
- *监控。*我们将两个节点用于Prometheus-Grafana组合。为了生成工作负载、我们使用了一个单独的三节点集群、该集群可能会生成此Kafka集群并将其占用。
- *存储。*我们使用了一个单节点NetApp Cloud Volumes ONTAP 实例、该实例上挂载了六个250 GB GP2 AWS-EBS卷。然后、这些卷会通过专用LIF作为六个NFS卷公开到Kafka集群中。
- *配置。*此测试案例中可配置的元素是Kafka代理。

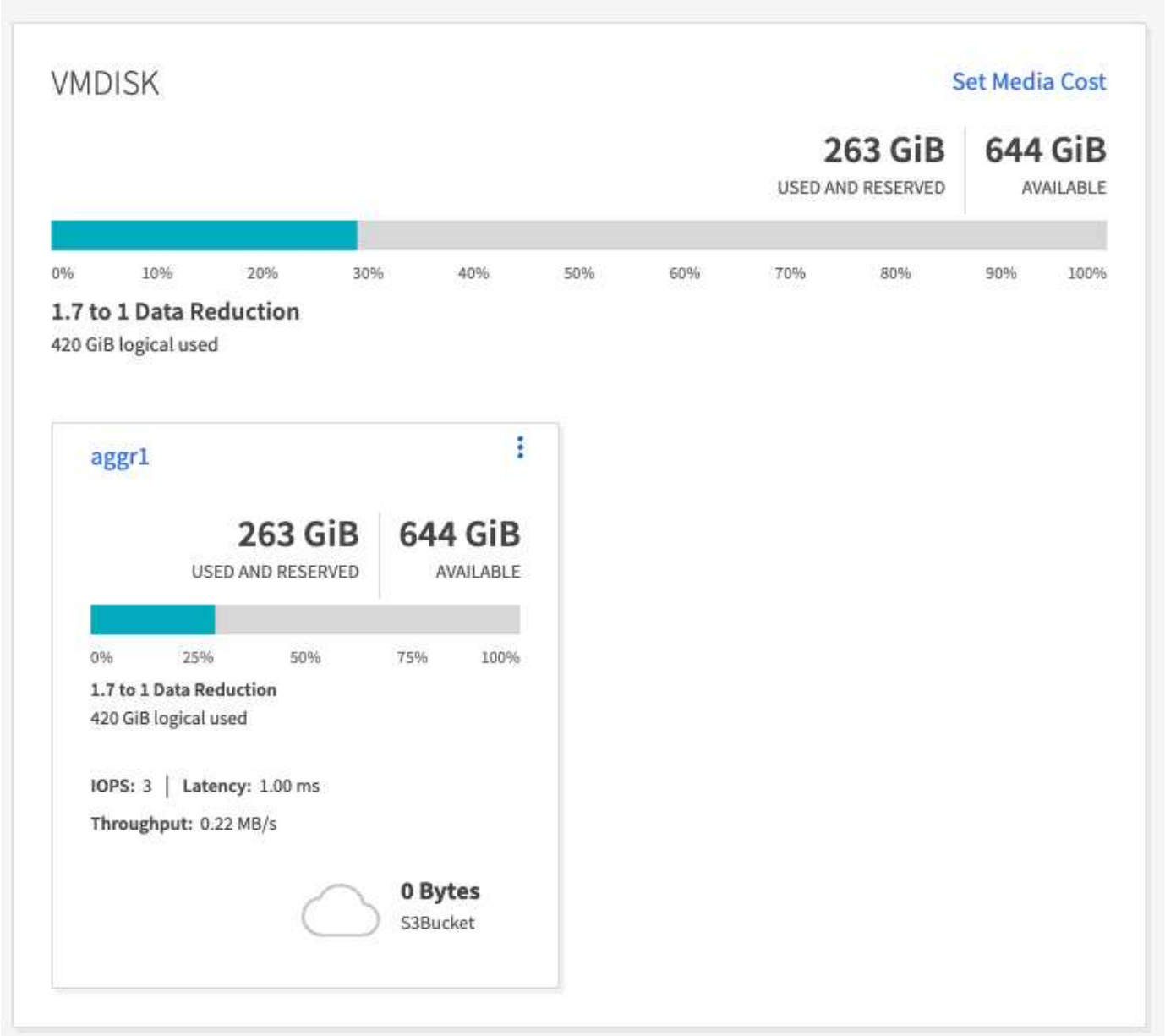
在生产商端关闭了数据压缩、从而使生产商能够生成高吞吐量。而是由计算层处理存储效率。

测试方法

1. 已按照上述规格配置Kafka集群。
2. 在集群上、使用Open消息 基准工具生成了大约350 GB的数据。
3. 工作负载完成后、将使用ONTAP 系统管理器和命令行界面收集存储效率统计信息。

观察结果

对于使用OMB工具生成的数据、我们发现空间节省~33%、存储效率比率为1.70：1。如下图所示、生成的数据所使用的逻辑空间为420.3 GB、用于存放数据的物理空间为281.7 GB。




```
shantanuCVOinstancenew::> df -h -S

Warning: The "-S" parameter is deprecated and may be removed in a future release. To show the efficiency ratio use "aggr show-efficiency"
         command.
Filesystem                 used       total-saved    %total-saved    deduplicated    %deduplicated    compressed    %compressed  Vserver
/vol/vol0/               7319MB                0B              0%              0B               0%            0B             0%  shantanuCVOinstancenew-01
/vol/kafka_vol/           281GB             138GB             33%           138GB              33%            0B             0%  svm_shantanuCVOinstancenew
/vol/svm_shantanuCVOinstancenew_root/
                          660KB                0B              0%              0B               0%            0B             0%  svm_shantanuCVOinstancenew
3 entries were displayed.
```

```
                    Name of the Aggregate: aggr1
             Node where Aggregate Resides: shantanuCVOinstancenew-01
           Total Storage Efficiency Ratio: 1.70:1
Total Data Reduction Efficiency Ratio Without Snapshots: 1.70:1
Total Data Reduction Efficiency Ratio without snapshots and flexclones: 1.70:1
        Logical Space Used for All Volumes: 420.3GB
       Physical Space Used for All Volumes: 281.7GB
```

AWS中的性能概述和验证

基于AWS云中的性能对存储层挂载在NetApp NFS上的Kafka集群进行了基准测试。以下各节将介绍这些基准测试示例。

采用NetApp Cloud Volumes ONTAP 的AWS云中的Kafka (高可用性对和单节点)

采用NetApp Cloud Volumes ONTAP (HA对)的Kafka集群已通过AWS云性能基准测试。以下各节将介绍此基准测试。

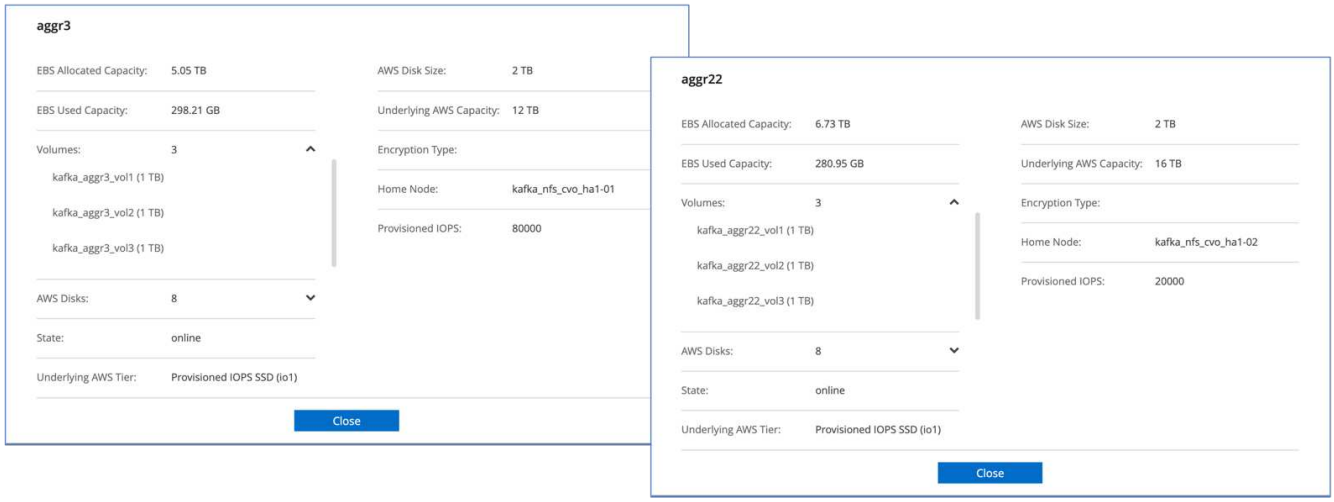
架构设置

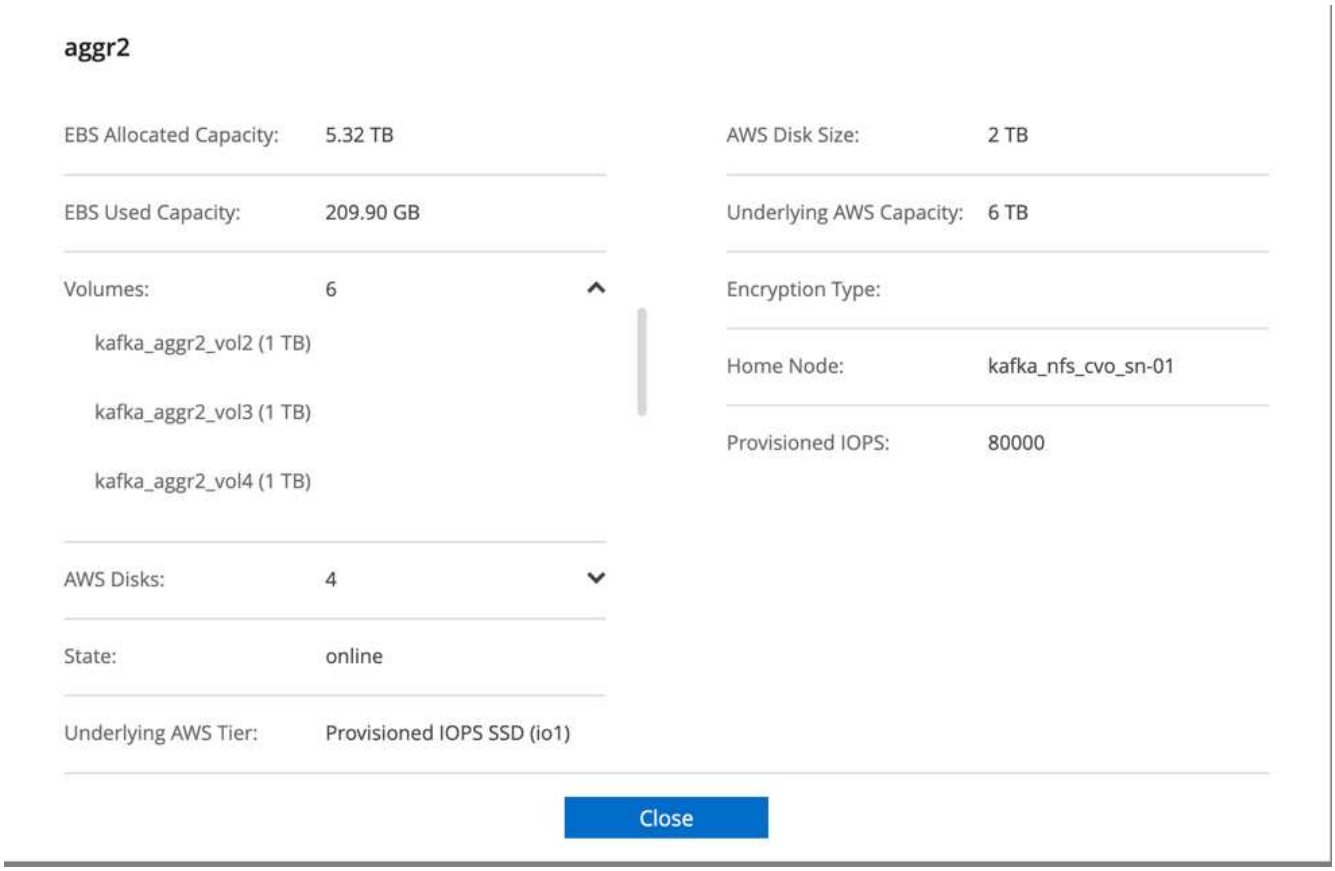
下表显示了使用NAS的Kafka集群的环境配置。

平台组件	环境配置
Kafka 3.2.3	• 3个Zookepers—T2.Small • 3个代理服务器—i3en.2xlarge • 1个Grafana—c5n.2xlarge • 4个生产者/使用者—c5n.2xlarge *
所有节点上的操作系统	RHEL8.6
NetApp Cloud Volumes ONTAP 实例	HA对实例—m5dn.12x插入x双节点单节点实例—m5dn.12x插入x 1个节点

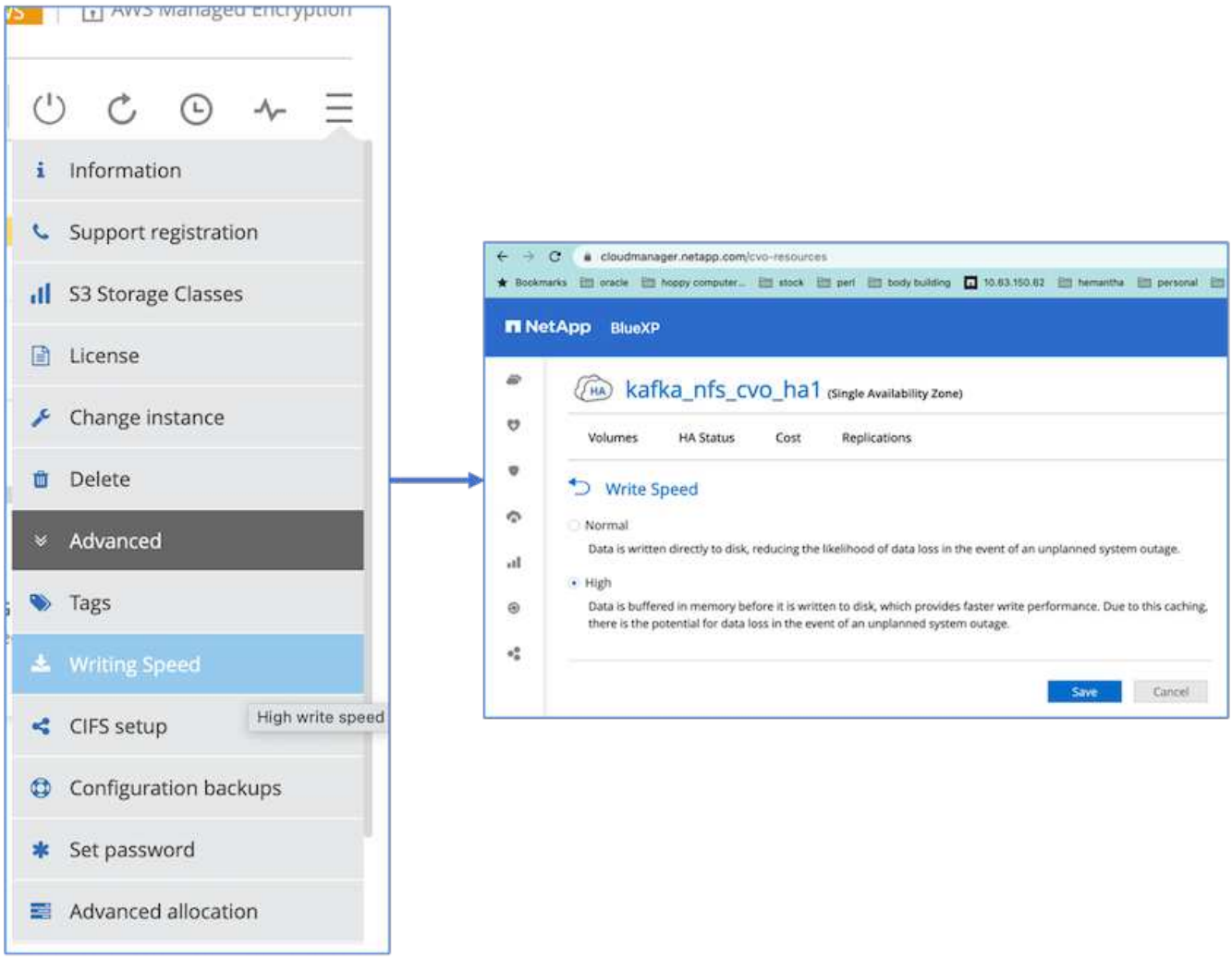
NetApp集群卷ONTAP 设置

1. 对于Cloud Volumes ONTAP HA对、我们在每个存储控制器的每个聚合上创建了两个聚合、其中包含三个卷。对于单个Cloud Volumes ONTAP 节点、我们会在一个聚合中创建六个卷。



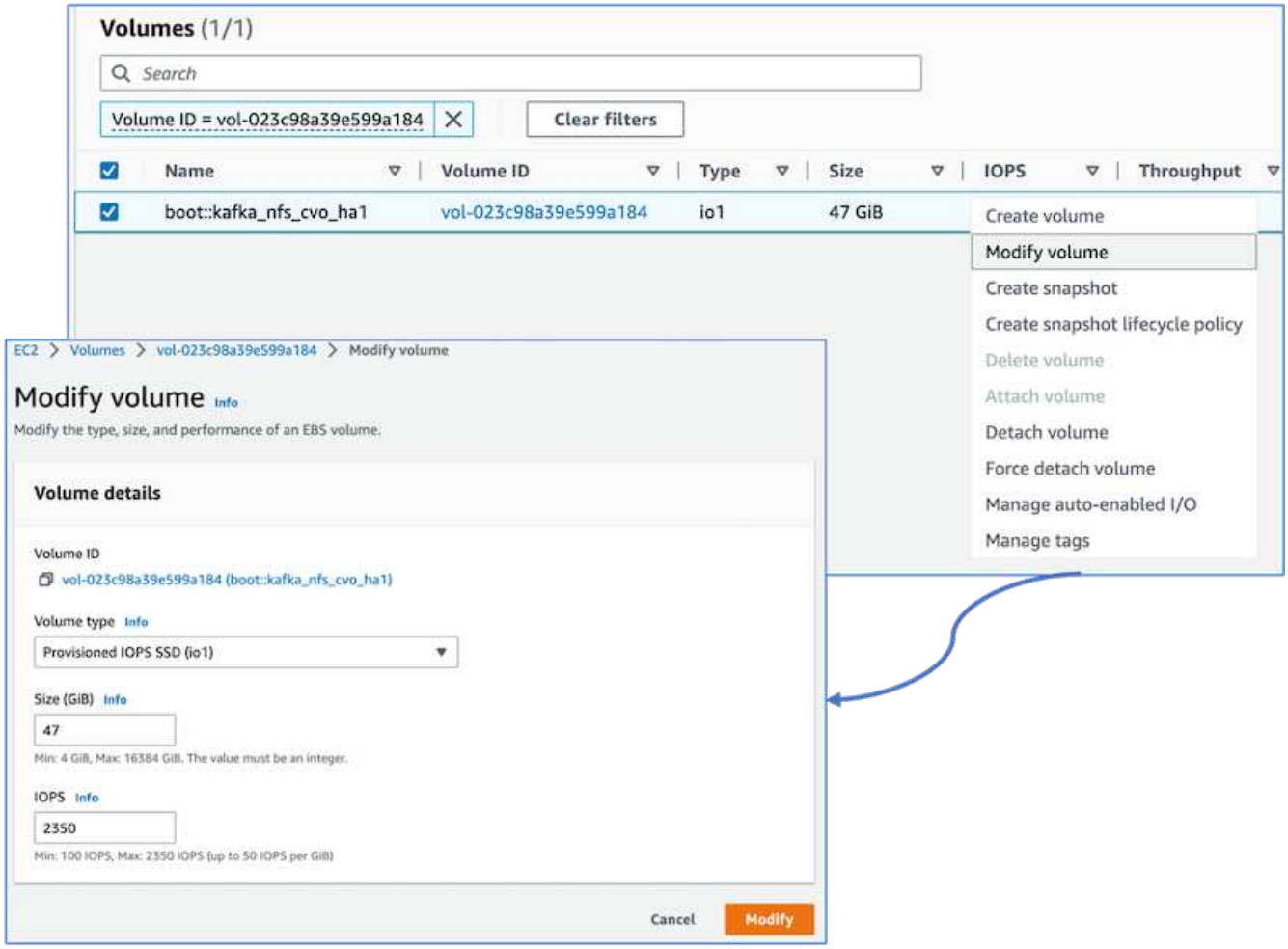


2. 为了提高网络性能、我们为HA对和单个节点启用了高速网络连接。



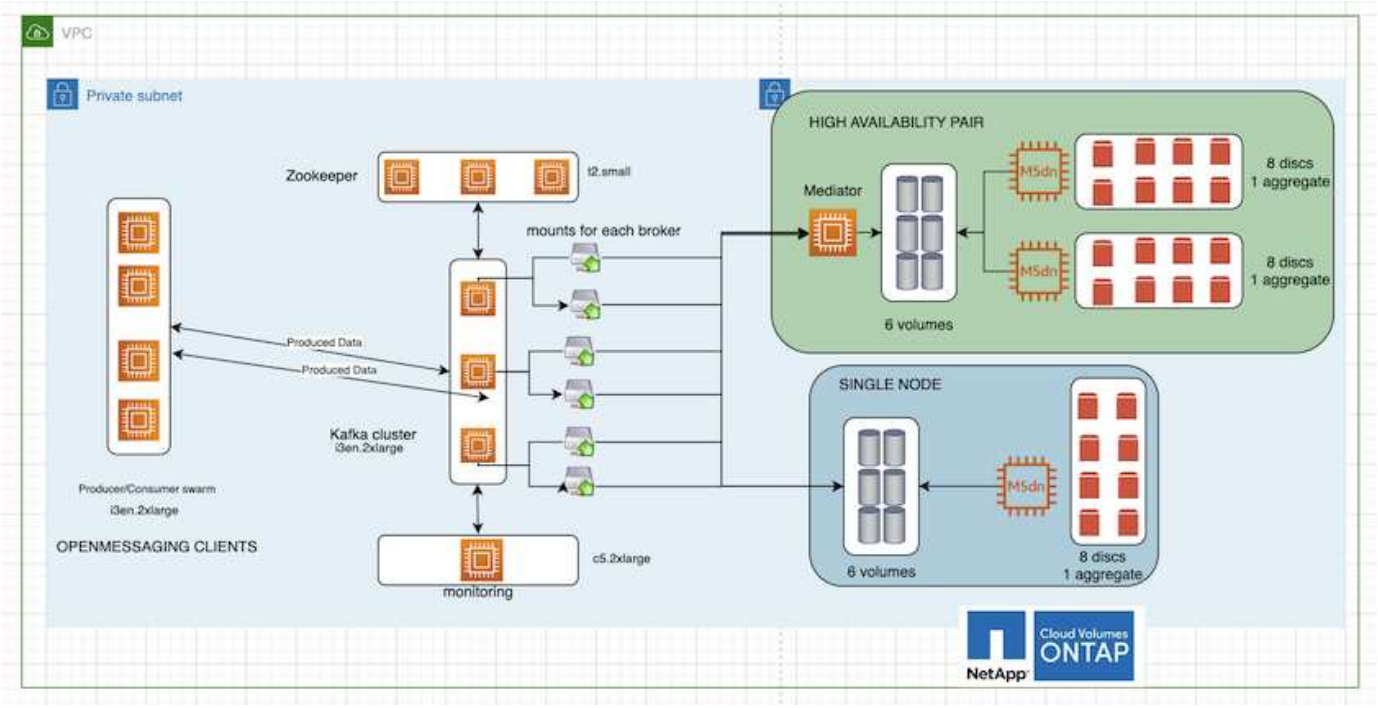
3. 我们注意到ONTAP NVRAM的IOPS较多、因此将Cloud Volumes ONTAP 根卷的IOPS更改为2350。Cloud Volumes ONTAP 中的根卷磁盘大小为47 GB。以下ONTAP 命令适用于HA对、同一步骤适用于单个节点。

```
statistics start -object vnvram -instance vnvram -counter
backing_store_iops -sample-id sample_555
kafka_nfs_cvo_ha1::*> statistics show -sample-id sample_555
Object: vnvram
Instance: vnvram
Start-time: 1/18/2023 18:03:11
End-time: 1/18/2023 18:03:13
Elapsed-time: 2s
Scope: kafka_nfs_cvo_ha1-01
    Counter                                                     Value
    -------------------------------- --------------------------------
    backing_store_iops                                           1479
Object: vnvram
Instance: vnvram
Start-time: 1/18/2023 18:03:11
End-time: 1/18/2023 18:03:13
Elapsed-time: 2s
Scope: kafka_nfs_cvo_ha1-02
    Counter                                                     Value
    -------------------------------- --------------------------------
    backing_store_iops                                           1210
2 entries were displayed.
kafka_nfs_cvo_ha1::*>
```



下图展示了基于NAS的Kafka集群的架构。

- *计算。*我们使用了一个三节点Kafka集群、其中三节点Zookeeper集合在专用服务器上运行。每个代理都通过一个专用LIF与Cloud Volumes ONTAP 实例上的一个卷具有两个NFS挂载点。
- *监控。*我们将两个节点用于Prometheus-Grafana组合。为了生成工作负载、我们使用了一个单独的三节点集群、该集群可能会生成此Kafka集群并将其占用。
- *存储。*我们使用了一个HA对Cloud Volumes ONTAP 实例、该实例上挂载了一个6 TB的GP3 AWS-EBS卷。然后、该卷会通过NFS挂载导出到Kafka代理。



OpenMessage基准配置

1. 为了提高NFS性能、我们需要在NFS服务器和NFS客户端之间建立更多的网络连接、这些连接可以使用nconnect来创建。运行以下命令、使用nconnect选项在代理节点上挂载NFS卷：

```
[root@ip-172-30-0-121 ~]# cat /etc/fstab
UUID=eaa1f38e-de0f-4ed5-a5b5-2fa9db43bb38/xfsdefaults00
/dev/nvme1n1 /mnt/data-1 xfs defaults,noatime,nodiscard 0 0
/dev/nvme2n1 /mnt/data-2 xfs defaults,noatime,nodiscard 0 0
172.30.0.233:/kafka_aggr3_vol1 /kafka_aggr3_vol1 nfs
defaults,nconnect=16 0 0
172.30.0.233:/kafka_aggr3_vol2 /kafka_aggr3_vol2 nfs
defaults,nconnect=16 0 0
172.30.0.233:/kafka_aggr3_vol3 /kafka_aggr3_vol3 nfs
defaults,nconnect=16 0 0
172.30.0.242:/kafka_aggr22_vol1 /kafka_aggr22_vol1 nfs
defaults,nconnect=16 0 0
172.30.0.242:/kafka_aggr22_vol2 /kafka_aggr22_vol2 nfs
defaults,nconnect=16 0 0
172.30.0.242:/kafka_aggr22_vol3 /kafka_aggr22_vol3 nfs
defaults,nconnect=16 0 0
[root@ip-172-30-0-121 ~]# mount -a
[root@ip-172-30-0-121 ~]# df -h
Filesystem                       Size  Used Avail Use% Mounted on
devtmpfs                          31G     0   31G   0% /dev
tmpfs                             31G  249M   31G   1% /run
tmpfs                             31G     0   31G   0% /sys/fs/cgroup
/dev/nvme0n1p2                    10G  2.8G  7.2G  28% /
/dev/nvme1n1                     2.3T  248G  2.1T  11% /mnt/data-1
/dev/nvme2n1                     2.3T  245G  2.1T  11% /mnt/data-2
172.30.0.233:/kafka_aggr3_vol1   1.0T   12G 1013G   2% /kafka_aggr3_vol1
172.30.0.233:/kafka_aggr3_vol2   1.0T  5.5G 1019G   1% /kafka_aggr3_vol2
172.30.0.233:/kafka_aggr3_vol3   1.0T  8.9G 1016G   1% /kafka_aggr3_vol3
172.30.0.242:/kafka_aggr22_vol1  1.0T  7.3G 1017G   1%
/kafka_aggr22_vol1
172.30.0.242:/kafka_aggr22_vol2  1.0T  6.9G 1018G   1%
/kafka_aggr22_vol2
172.30.0.242:/kafka_aggr22_vol3  1.0T  5.9G 1019G   1%
/kafka_aggr22_vol3
tmpfs                            6.2G     0  6.2G   0% /run/user/1000
[root@ip-172-30-0-121 ~]#
```

2. 在Cloud Volumes ONTAP 中检查网络连接。从单个Cloud Volumes ONTAP 节点使用以下ONTAP 命令。同一步骤也适用于Cloud Volumes ONTAP HA对。

```
Last login time: 1/20/2023 00:16:29
kafka_nfs_cvo_sn::> network connections active show -service nfs*
-fields remote-host
node                cid        vserver              remote-host
------------------- ---------- -------------------- ------------
```

```
kafka_nfs_cvo_sn-01 2315762628 svm_kafka_nfs_cvo_sn 172.30.0.121
kafka_nfs_cvo_sn-01 2315762629 svm_kafka_nfs_cvo_sn 172.30.0.121
kafka_nfs_cvo_sn-01 2315762630 svm_kafka_nfs_cvo_sn 172.30.0.121
kafka_nfs_cvo_sn-01 2315762631 svm_kafka_nfs_cvo_sn 172.30.0.121
kafka_nfs_cvo_sn-01 2315762632 svm_kafka_nfs_cvo_sn 172.30.0.121
kafka_nfs_cvo_sn-01 2315762633 svm_kafka_nfs_cvo_sn 172.30.0.121
kafka_nfs_cvo_sn-01 2315762634 svm_kafka_nfs_cvo_sn 172.30.0.121
kafka_nfs_cvo_sn-01 2315762635 svm_kafka_nfs_cvo_sn 172.30.0.121
kafka_nfs_cvo_sn-01 2315762636 svm_kafka_nfs_cvo_sn 172.30.0.121
kafka_nfs_cvo_sn-01 2315762637 svm_kafka_nfs_cvo_sn 172.30.0.121
kafka_nfs_cvo_sn-01 2315762639 svm_kafka_nfs_cvo_sn 172.30.0.72
kafka_nfs_cvo_sn-01 2315762640 svm_kafka_nfs_cvo_sn 172.30.0.72
kafka_nfs_cvo_sn-01 2315762641 svm_kafka_nfs_cvo_sn 172.30.0.72
kafka_nfs_cvo_sn-01 2315762642 svm_kafka_nfs_cvo_sn 172.30.0.72
kafka_nfs_cvo_sn-01 2315762643 svm_kafka_nfs_cvo_sn 172.30.0.72
kafka_nfs_cvo_sn-01 2315762644 svm_kafka_nfs_cvo_sn 172.30.0.72
kafka_nfs_cvo_sn-01 2315762645 svm_kafka_nfs_cvo_sn 172.30.0.72
kafka_nfs_cvo_sn-01 2315762646 svm_kafka_nfs_cvo_sn 172.30.0.72
kafka_nfs_cvo_sn-01 2315762647 svm_kafka_nfs_cvo_sn 172.30.0.72
kafka_nfs_cvo_sn-01 2315762648 svm_kafka_nfs_cvo_sn 172.30.0.72
kafka_nfs_cvo_sn-01 2315762649 svm_kafka_nfs_cvo_sn 172.30.0.121
kafka_nfs_cvo_sn-01 2315762650 svm_kafka_nfs_cvo_sn 172.30.0.121
kafka_nfs_cvo_sn-01 2315762651 svm_kafka_nfs_cvo_sn 172.30.0.121
kafka_nfs_cvo_sn-01 2315762652 svm_kafka_nfs_cvo_sn 172.30.0.121
kafka_nfs_cvo_sn-01 2315762653 svm_kafka_nfs_cvo_sn 172.30.0.121
kafka_nfs_cvo_sn-01 2315762656 svm_kafka_nfs_cvo_sn 172.30.0.223
kafka_nfs_cvo_sn-01 2315762657 svm_kafka_nfs_cvo_sn 172.30.0.223
kafka_nfs_cvo_sn-01 2315762658 svm_kafka_nfs_cvo_sn 172.30.0.223
kafka_nfs_cvo_sn-01 2315762659 svm_kafka_nfs_cvo_sn 172.30.0.223
kafka_nfs_cvo_sn-01 2315762660 svm_kafka_nfs_cvo_sn 172.30.0.223
kafka_nfs_cvo_sn-01 2315762661 svm_kafka_nfs_cvo_sn 172.30.0.223
kafka_nfs_cvo_sn-01 2315762662 svm_kafka_nfs_cvo_sn 172.30.0.223
kafka_nfs_cvo_sn-01 2315762663 svm_kafka_nfs_cvo_sn 172.30.0.223
kafka_nfs_cvo_sn-01 2315762664 svm_kafka_nfs_cvo_sn 172.30.0.223
kafka_nfs_cvo_sn-01 2315762665 svm_kafka_nfs_cvo_sn 172.30.0.223
kafka_nfs_cvo_sn-01 2315762666 svm_kafka_nfs_cvo_sn 172.30.0.223
kafka_nfs_cvo_sn-01 2315762667 svm_kafka_nfs_cvo_sn 172.30.0.72
kafka_nfs_cvo_sn-01 2315762668 svm_kafka_nfs_cvo_sn 172.30.0.72
kafka_nfs_cvo_sn-01 2315762669 svm_kafka_nfs_cvo_sn 172.30.0.72
kafka_nfs_cvo_sn-01 2315762670 svm_kafka_nfs_cvo_sn 172.30.0.72
kafka_nfs_cvo_sn-01 2315762671 svm_kafka_nfs_cvo_sn 172.30.0.72
kafka_nfs_cvo_sn-01 2315762672 svm_kafka_nfs_cvo_sn 172.30.0.72
kafka_nfs_cvo_sn-01 2315762673 svm_kafka_nfs_cvo_sn 172.30.0.223
kafka_nfs_cvo_sn-01 2315762674 svm_kafka_nfs_cvo_sn 172.30.0.223
kafka_nfs_cvo_sn-01 2315762676 svm_kafka_nfs_cvo_sn 172.30.0.121
kafka_nfs_cvo_sn-01 2315762677 svm_kafka_nfs_cvo_sn 172.30.0.223
```

```
kafka_nfs_cvo_sn-01 2315762678 svm_kafka_nfs_cvo_sn 172.30.0.223
kafka_nfs_cvo_sn-01 2315762679 svm_kafka_nfs_cvo_sn 172.30.0.223
48 entries were displayed.

kafka_nfs_cvo_sn::>
```

3. 我们使用以下Kafka `server.properties` 在Cloud Volumes ONTAP HA对的所有Kafka代理中。。`log.dirs` 每个代理的属性都不同、其余属性对于代理是通用的。对于Broker1、为 `log.dirs` 值如下：

```
[root@ip-172-30-0-121 ~]# cat /opt/kafka/config/server.properties
broker.id=0
advertised.listeners=PLAINTEXT://172.30.0.121:9092
#log.dirs=/mnt/data-1/d1,/mnt/data-1/d2,/mnt/data-1/d3,/mnt/data-
2/d1,/mnt/data-2/d2,/mnt/data-2/d3
log.dirs=/kafka_aggr3_vol1/broker1,/kafka_aggr3_vol2/broker1,/kafka_aggr
3_vol3/broker1,/kafka_aggr22_vol1/broker1,/kafka_aggr22_vol2/broker1,/ka
fka_aggr22_vol3/broker1
zookeeper.connect=172.30.0.12:2181,172.30.0.30:2181,172.30.0.178:2181
num.network.threads=64
num.io.threads=64
socket.send.buffer.bytes=102400
socket.receive.buffer.bytes=102400
socket.request.max.bytes=104857600
num.partitions=1
num.recovery.threads.per.data.dir=1
offsets.topic.replication.factor=1
transaction.state.log.replication.factor=1
transaction.state.log.min.isr=1
replica.fetch.max.bytes=524288000
background.threads=20
num.replica.alter.log.dirs.threads=40
num.replica.fetchers=20
[root@ip-172-30-0-121 ~]#
```

- 对于Broker2、为 `log.dirs` 属性值如下：

```
log.dirs=/kafka_aggr3_vol1/broker2,/kafka_aggr3_vol2/broker2,/kafka_a
ggr3_vol3/broker2,/kafka_aggr22_vol1/broker2,/kafka_aggr22_vol2/broke
r2,/kafka_aggr22_vol3/broker2
```

- 对于Broker3、为 `log.dirs` 属性值如下：

```
log.dirs=/kafka_aggr3_vol1/broker3,/kafka_aggr3_vol2/broker3,/kafka_a
ggr3_vol3/broker3,/kafka_aggr22_vol1/broker3,/kafka_aggr22_vol2/broke
r3,/kafka_aggr22_vol3/broker3
```

4. 对于单个Cloud Volumes ONTAP 节点、为Kafka `servers.properties` 与Cloud Volumes ONTAP HA对相同、但不包括 `log.dirs` 属性。
   - 对于Broker1、为 `log.dirs` 值如下：

```
log.dirs=/kafka_aggr2_vol1/broker1,/kafka_aggr2_vol2/broker1,/kafka_a
ggr2_vol3/broker1,/kafka_aggr2_vol4/broker1,/kafka_aggr2_vol5/broker1
,/kafka_aggr2_vol6/broker1
```

   - 对于Broker2、为 `log.dirs` 值如下：

```
log.dirs=/kafka_aggr2_vol1/broker2,/kafka_aggr2_vol2/broker2,/kafka_a
ggr2_vol3/broker2,/kafka_aggr2_vol4/broker2,/kafka_aggr2_vol5/broker2
,/kafka_aggr2_vol6/broker2
```

   - 对于Broker3、为 `log.dirs` 属性值如下：

```
log.dirs=/kafka_aggr2_vol1/broker3,/kafka_aggr2_vol2/broker3,/kafka_a
ggr2_vol3/broker3,/kafka_aggr2_vol4/broker3,/kafka_aggr2_vol5/broker3
,/kafka_aggr2_vol6/broker3
```

5. OMB中的工作负载配置了以下属性： `(/opt/benchmark/workloads/1-topic-100-partitions-1kb.yaml)`。

```
topics: 4
partitionsPerTopic: 100
messageSize: 32768
useRandomizedPayloads: true
randomBytesRatio: 0.5
randomizedPayloadPoolSize: 100
subscriptionsPerTopic: 1
consumerPerSubscription: 80
producersPerTopic: 40
producerRate: 1000000
consumerBacklogSizeGB: 0
testDurationMinutes: 5
```

。 `messageSize` 可能因使用情形而异。在性能测试中、我们使用了3 K。

我们使用OMB中的两个不同驱动程序Sync或Throughput在Kafka集群上生成工作负载。

- 用于Sync驱动程序属性的YAML文件如下所示 `(/opt/benchmark/driver- kafka/kafka-sync.yaml)`：

```
name: Kafka
driverClass:
io.openmessaging.benchmark.driver.kafka.KafkaBenchmarkDriver
# Kafka client-specific configuration
replicationFactor: 3
topicConfig: |
  min.insync.replicas=2
  flush.messages=1
  flush.ms=0
commonConfig: |

bootstrap.servers=172.30.0.121:9092,172.30.0.72:9092,172.30.0.223:909
2
producerConfig: |
  acks=all
  linger.ms=1
  batch.size=1048576
consumerConfig: |
  auto.offset.reset=earliest
  enable.auto.commit=false
  max.partition.fetch.bytes=10485760
```

- 用于吞吐量驱动程序属性的YAML文件如下所示 `(/opt/benchmark/driver- kafka/kafka-throughput.yaml)`：

```
name: Kafka
driverClass:
io.openmessaging.benchmark.driver.kafka.KafkaBenchmarkDriver
# Kafka client-specific configuration
replicationFactor: 3
topicConfig: |
  min.insync.replicas=2
commonConfig: |

bootstrap.servers=172.30.0.121:9092,172.30.0.72:9092,172.30.0.223:909
2
  default.api.timeout.ms=1200000
  request.timeout.ms=1200000
producerConfig: |
  acks=all
  linger.ms=1
  batch.size=1048576
consumerConfig: |
  auto.offset.reset=earliest
  enable.auto.commit=false
  max.partition.fetch.bytes=10485760
```

测试方法

1. Kafka集群是按照上述规范使用Terraform和Ansible配置的。Terraform用于使用适用于Kafka集群的AWS实例构建基础架构、Ansible在这些实例上构建Kafka集群。
2. 已使用上述工作负载配置和Sync驱动程序触发OMB工作负载。

```
Sudo bin/benchmark –drivers driver-kafka/kafka- sync.yaml workloads/1-
topic-100-partitions-1kb.yaml
```

3. 使用相同工作负载配置的吞吐量驱动程序触发了另一个工作负载。

```
sudo bin/benchmark –drivers driver-kafka/kafka-throughput.yaml
workloads/1-topic-100-partitions-1kb.yaml
```

观察结果

我们使用了两种不同类型的驱动程序来生成工作负载、以便对在NFS上运行的Kafka实例的性能进行基准测试。驱动程序之间的区别在于日志刷新属性。

对于Cloud Volumes ONTAP HA对：

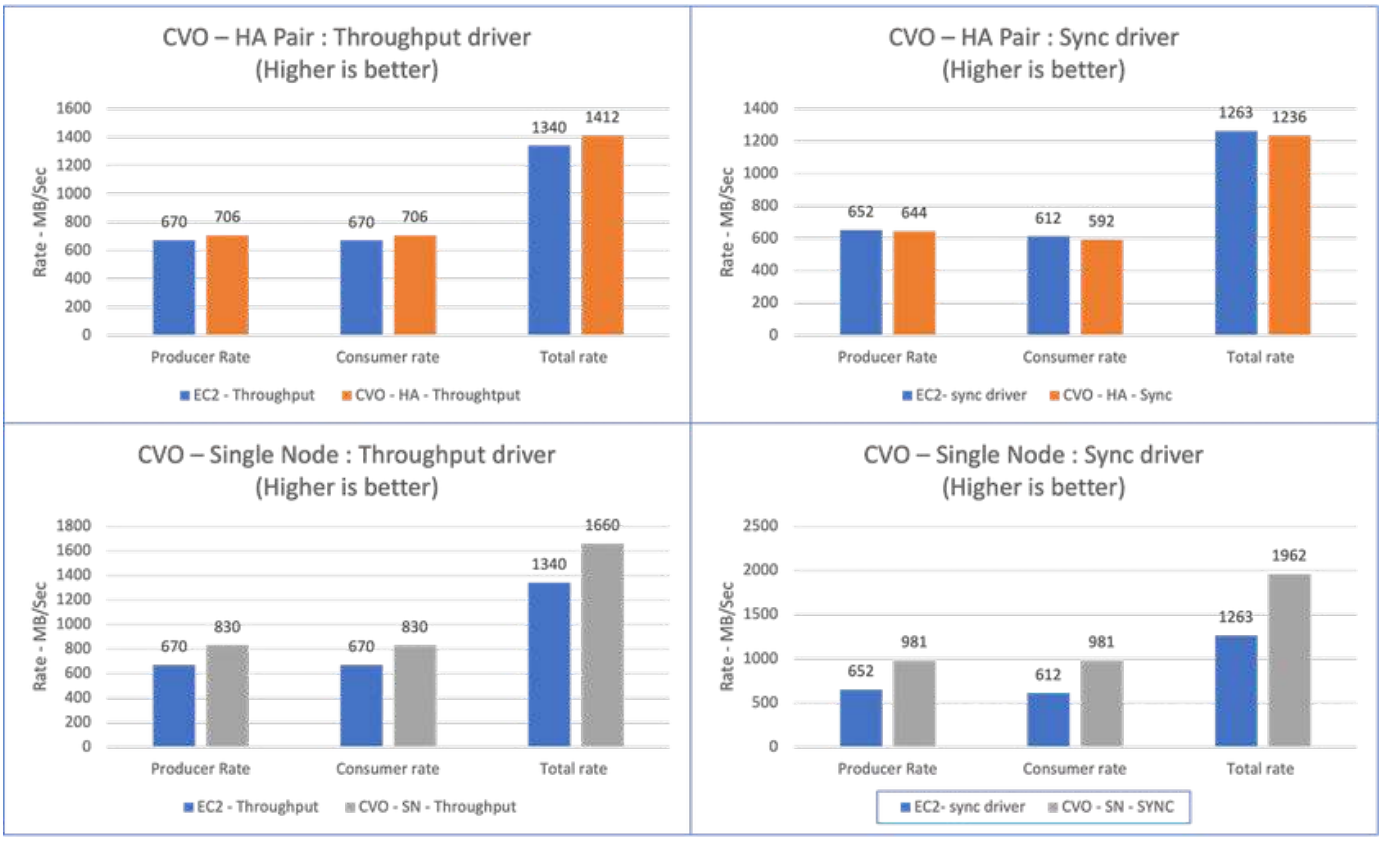
- Sync驱动程序一致生成的总吞吐量：~1236 MBps。
- 为吞吐量驱动程序生成的总吞吐量：峰值~1412 MBps。

对于单个Cloud Volumes ONTAP 节点：

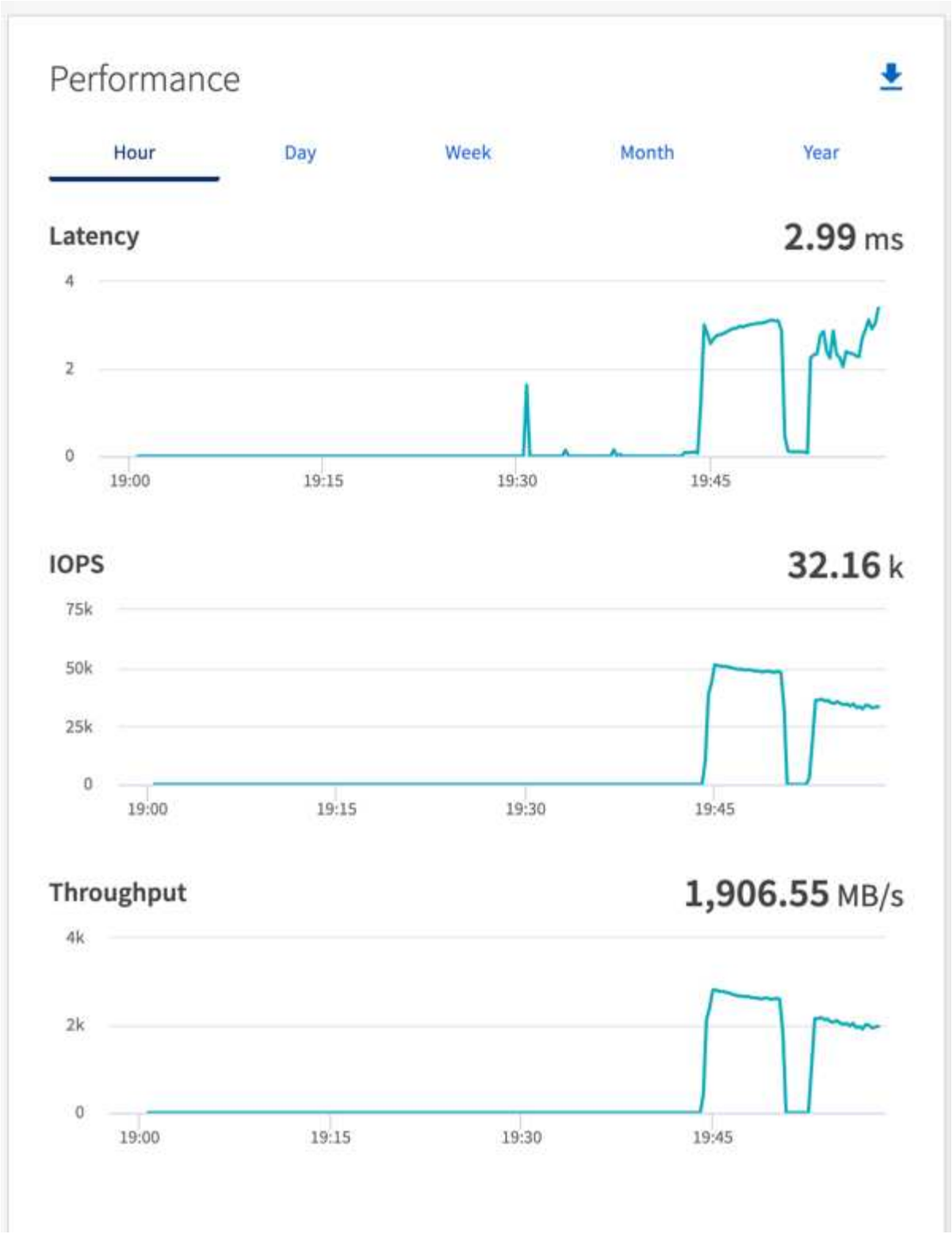
- Sync驱动程序一致生成的总吞吐量：~ 1962MBps。
- 吞吐量驱动程序生成的总吞吐量：峰值~1660MBps

同步驱动程序可以在日志即时转储到磁盘时生成一致的吞吐量、而吞吐量驱动程序则在将日志批量提交到磁盘时生成突发的吞吐量。

这些吞吐量数字是为给定的AWS配置生成的。为了满足更高的性能要求、可以进一步扩展和调整实例类型、以提高吞吐量。总吞吐量或总速率是生产者和使用者速率的组合。



在执行吞吐量或同步驱动程序基准测试时、请务必检查存储吞吐量。



AWS FSx for NetApp ONTAP中的性能概述和验证

在AWS FSx for NetApp ONTAP中、对NetApp NFS上挂载了存储层的Kafka集群进行了性能基准测试。以下各节将介绍这些基准测试示例。

AWS FSx for NetApp ONTAP中的Apache Kafka

网络文件系统(Network File System、NFS)是一种广泛用于存储大量数据的网络文件系统。在大多数企业中、数据越来越多地由Apache Kafka等流式应用程序生成。这些工作负载需要可扩展性、低延迟以及具有现代存储功能的强大数据采集架构。要实现实时分析并提供可指导行动的洞察力、需要一个设计完善且性能高的基础架构。

Kafka的设计支持POSIX兼容文件系统、并依靠文件系统来处理文件操作、但在NFS3文件系统上存储数据时、Kafka代理NFS客户端对文件操作的解释可能与XFS或ext4等本地文件系统不同。一个常见的示例是NFS愚蠢的重命名、该重命名导致Kafka代理在扩展集群和重新分配分区时失败。为了应对这一挑战、NetApp对开源Linux NFS客户端进行了更新、对RHEL8.7和RHEL9.1中的内容进行了一般更改、并从当前FSx for NetApp ONTAP版本ONTAP 9.12.1开始受支持。

Amazon FSx for NetApp ONTAP可在云中提供一个完全托管、可扩展且高性能的NFS文件系统。FSx for NetApp ONTAP上的Kafka数据可以进行扩展、以处理大量数据并确保容错。NFS可为关键和敏感数据集提供集中式存储管理和数据保护。

通过这些增强功能、AWS客户可以在AWS计算服务上运行Kafka工作负载时利用FSx for NetApp ONTAP。这些优势包括：
*降低CPU利用率以缩短I/O等待时间
* Kafka代理恢复时间更快。
*可靠性和效率。
*可扩展性和性能。
*多可用性区域可用性。
*数据保护。

AWS FSx for NetApp ONTAP中的性能概述和验证

基于AWS云中的性能对存储层挂载在NetApp NFS上的Kafka集群进行了基准测试。以下各节将介绍这些基准测试示例。

Kafka位于AWS FSx for NetApp ONTAP中

采用AWS FSx for NetApp ONTAP的Kafka集群已通过AWS云中的性能基准测试。以下各节将介绍此基准测试。

架构设置

下表显示了使用AWS FSx for NetApp ONTAP的Kafka集群的环境配置。

平台组件	环境配置
Kafka 3.2.3	• 3个Zookepers—T2.Small • 3个代理服务器—i3en.2xlarge • 1个Grafana—c5n.2xlarge • 4个生产者/使用者—c5n.2xlarge *
所有节点上的操作系统	RHEL8.6
AWS FSx for NetApp ONTAP	多可用性(AZ)、吞吐量为4 GB/秒、IOPS为160000次

NetApp FSx for NetApp ONTAP设置

1. 在初始测试中、我们为NetApp ONTAP文件系统创建了一个FSx、容量为2 TB、吞吐量为400、000 IOPS、每秒2 GB。

```
[root@ip-172-31-33-69 ~]# aws fsx create-file-system --region us-east-2
--storage-capacity 2048 --subnet-ids <desired subnet 1> subnet-<desired
subnet 2> --file-system-type ONTAP --ontap-configuration
DeploymentType=MULTI_AZ_HA_1,ThroughputCapacity=2048,PreferredSubnetId=<
desired primary subnet>,FsxAdminPassword=<new
password>,DiskIopsConfiguration="{Mode=USER_PROVISIONED,Iops=40000"}
```

   在本示例中、我们将通过AWS命令行界面部署FSx for NetApp ONTAP。您需要根据需要在环境中进一步自定义此命令。此外、FSx for NetApp ONTAP还可以通过AWS控制台进行部署和管理、以减少命令行输入、获得更轻松、更简化的部署体验。

   文档在FSx for NetApp ONTAP中、测试区域(US-East-1)中2 GB/秒吞吐量文件系统可实现的最大IOPS为80、000次IOPS。FSx for NetApp ONTAP文件系统的总最大IOPS为160、000次IOPS、需要部署4 GB/秒吞吐量才能达到此目的、我们将在本文档后面进行演示。

   有关FSx for NetApp ONTAP性能规格的详细信息、请随时访问AWS FSx for NetApp ONTAP文档、网址为： https://docs.aws.amazon.com/fsx/latest/ONTAPGuide/performance.html 。

   有关FSx "crea-File-system"的详细命令行语法、请参见： https://docs.aws.amazon.com/cli/latest/reference/fsx/create-file-system.html

   例如、您可以指定特定的KMS密钥、而不是在未指定KMS密钥时使用的默认AWS FSx主密钥。

2. 创建FSx for NetApp ONTAP文件系统时、请等到JSON返回中的"LifeCycle (生命周期)"状态更改为"Available (可用)"、然后按如下所示描述文件系统：

```
[root@ip-172-31-33-69 ~]# aws fsx describe-file-systems  --region us-
east-1 --file-system-ids fs-02ff04bab5ce01c7c
```

3. 使用fsxadmin用户登录到FSx for NetApp ONTAP SSH以验证凭据：
   FSxadmin是创建时FSx for NetApp ONTAP文件系统的默认管理员帐户。fsxadmin的密码是我们在步骤1中完成的首次在AWS控制台中或使用AWS命令行界面创建文件系统时配置的密码。

```
[root@ip-172-31-33-69 ~]# ssh fsxadmin@198.19.250.244
The authenticity of host '198.19.250.244 (198.19.250.244)' can't be
established.
ED25519 key fingerprint is
SHA256:mgCyRXJfWRc2d/jOjFbMBsUcYOWjxoIky0ltHvVDL/Y.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '198.19.250.244' (ED25519) to the list of
known hosts.
(fsxadmin@198.19.250.244) Password:

This is your first recorded login.
```

4. 验证凭据后、在FSx for NetApp ONTAP文件系统上创建Storage Virtual Machine

```
[root@ip-172-31-33-69 ~]# aws fsx --region us-east-1 create-storage-
virtual-machine --name svmkafkatest --file-system-id fs-
02ff04bab5ce01c7c
```

Storage Virtual Machine (SVM)是一种孤立的文件服务器、具有自己的管理凭据和端点、用于管理和访问FSx for NetApp ONTAP卷中的数据、并提供FSx for NetApp ONTAP多租户功能。

5. 配置主Storage Virtual Machine后、通过SSH连接到新创建的FSx for NetApp ONTAP文件系统、然后使用以下示例命令在Storage Virtual Machine中创建卷、同样、我们会为此验证创建6个卷。根据我们的验证、保留默认成分卷(8)或更少的成分卷、这样可以提高Kafka的性能。

```
FsxId02ff04bab5ce01c7c::*> volume create -volume kafkafsxN1 -state
online -policy default -unix-permissions ---rwxr-xr-x -junction-active
true -type RW -snapshot-policy none  -junction-path /kafkafsxN1 -aggr
-list aggr1
```

6. 我们需要在卷中增加容量以进行测试。将卷的大小扩展到2 TB、然后挂载到接合路径上。

```
FsxId02ff04bab5ce01c7c::*> volume size -volume kafkafsxN1 -new-size +2TB
vol size: Volume "svmkafkatest:kafkafsxN1" size set to 2.10t.

FsxId02ff04bab5ce01c7c::*> volume size -volume kafkafsxN2 -new-size +2TB
vol size: Volume "svmkafkatest:kafkafsxN2" size set to 2.10t.

FsxId02ff04bab5ce01c7c::*> volume size -volume kafkafsxN3 -new-size +2TB
vol size: Volume "svmkafkatest:kafkafsxN3" size set to 2.10t.

FsxId02ff04bab5ce01c7c::*> volume size -volume kafkafsxN4 -new-size +2TB
```

```
vol size: Volume "svmkafkatest:kafkafsxN4" size set to 2.10t.

FsxId02ff04bab5ce01c7c::*> volume size -volume kafkafsxN5 -new-size +2TB
vol size: Volume "svmkafkatest:kafkafsxN5" size set to 2.10t.

FsxId02ff04bab5ce01c7c::*> volume size -volume kafkafsxN6 -new-size +2TB
vol size: Volume "svmkafkatest:kafkafsxN6" size set to 2.10t.

FsxId02ff04bab5ce01c7c::*> volume show -vserver svmkafkatest -volume *
Vserver   Volume       Aggregate    State      Type       Size
Available Used%
--------- ------------ ------------ ---------- ---- ----------
---------- -----
svmkafkatest
          kafkafsxN1   -            online     RW       2.10TB
1.99TB    0%
svmkafkatest
          kafkafsxN2   -            online     RW       2.10TB
1.99TB    0%
svmkafkatest
          kafkafsxN3   -            online     RW       2.10TB
1.99TB    0%
svmkafkatest
          kafkafsxN4   -            online     RW       2.10TB
1.99TB    0%
svmkafkatest
          kafkafsxN5   -            online     RW       2.10TB
1.99TB    0%
svmkafkatest
          kafkafsxN6   -            online     RW       2.10TB
1.99TB    0%
svmkafkatest
          svmkafkatest_root
                       aggr1        online     RW          1GB
968.1MB    0%
7 entries were displayed.

FsxId02ff04bab5ce01c7c::*> volume mount -volume kafkafsxN1 -junction
-path /kafkafsxN1

FsxId02ff04bab5ce01c7c::*> volume mount -volume kafkafsxN2 -junction
-path /kafkafsxN2

FsxId02ff04bab5ce01c7c::*> volume mount -volume kafkafsxN3 -junction
-path /kafkafsxN3
```

```
FsxId02ff04bab5ce01c7c::*> volume mount -volume kafkafsxN4 -junction
-path /kafkafsxN4

FsxId02ff04bab5ce01c7c::*> volume mount -volume kafkafsxN5 -junction
-path /kafkafsxN5

FsxId02ff04bab5ce01c7c::*> volume mount -volume kafkafsxN6 -junction
-path /kafkafsxN6
```

在FSx for NetApp ONTAP中、可以对卷进行精简配置。在我们的示例中、扩展卷总容量超过文件系统总容量、因此我们需要扩展文件系统总容量、以便解锁额外配置的卷容量、我们将在下一步演示这一点。

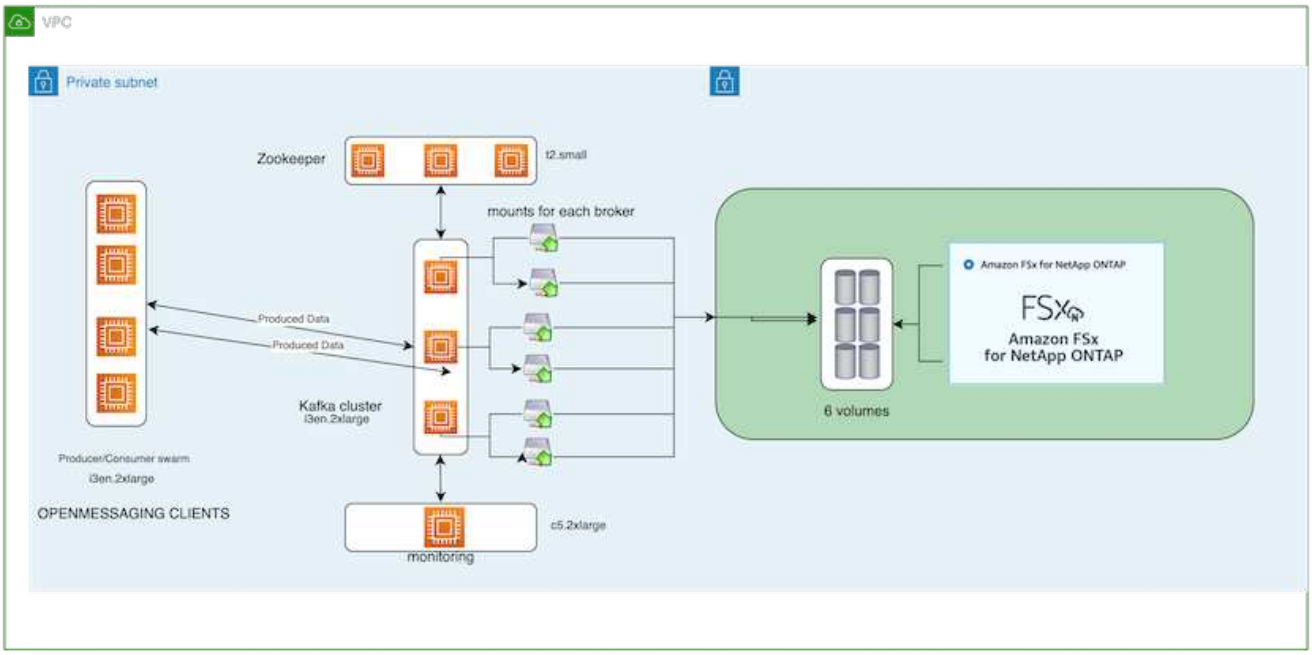
7. 接下来、为了提高性能和容量、我们将FSx的NetApp ONTAP吞吐量容量从2 GB/秒扩展到4 GB/秒、将IOPS扩展到160000、并将容量扩展到5 TB

```
[root@ip-172-31-33-69 ~]# aws fsx update-file-system --region us-east-1
--storage-capacity 5120 --ontap-configuration
'ThroughputCapacity=4096,DiskIopsConfiguration={Mode=USER_PROVISIONED,Io
ps=160000}' --file-system-id fs-02ff04bab5ce01c7c
```

有关FSx "update-file-system"的详细命令行语法、请参见：
https://docs.aws.amazon.com/cli/latest/reference/fsx/update-file-system.html

8. FSx for NetApp ONTAP卷使用nconnect和默认选项挂载在Kafka代理中

下图显示了基于Kafka集群的FSx for NetApp ONTAP的最终架构：



- 计算。我们使用了一个三节点Kafka集群、其中一个三节点Zookeer集合运行在专用服务器上。每个代理都有六个NFS挂载点、指向FSx for NetApp ONTAP实例上的六个卷。

- 监控。我们将两个节点用于Prometheus-Grafana组合。为了生成工作负载、我们使用了一个单独的三节点集群、该集群可能会生成此Kafka集群并将其占用。
- 存储。我们使用FSx for NetApp ONTAP、其中已挂载六个2 TB卷。然后、使用NFS挂载将卷导出到Kafka代理。FSx for NetApp ONTAP卷使用Kafka代理中的16个nconnect会话和默认选项进行挂载。

OpenMessage基准测试配置。

我们使用了与NetApp Cloud Volumes ONTAP相同的配置、其详细信息如下所示- https://docs.netapp.com/us-en/netapp-solutions/data-analytics/kafka-nfs-performance-overview-and-validation-in-aws.html#architectural-setup

测试方法

1. 按照上述规范、我们使用terraform和Ans得 来配置Kafka集群。Terraform用于使用适用于Kafka集群的AWS实例构建基础架构、而Ans可 在这些实例上构建Kafka集群。
2. 已使用上述工作负载配置和Sync驱动程序触发OMB工作负载。

```
sudo bin/benchmark -drivers driver-kafka/kafka-sync.yaml workloads/1-topic-100-partitions-1kb.yaml
```

3. 使用相同工作负载配置的吞吐量驱动程序触发了另一个工作负载。

```
sudo bin/benchmark -drivers driver-kafka/kafka-throughput.yaml workloads/1-topic-100-partitions-1kb.yaml
```

观察结果

我们使用了两种不同类型的驱动程序来生成工作负载、以便对在NFS上运行的Kafka实例的性能进行基准测试。驱动程序之间的区别在于日志刷新属性。

对于Kafka复制因子1和FSx for NetApp ONTAP：

- Sync驱动程序一致生成的总吞吐量：~ 3218 Mbps、峰值性能(~ 3352 Mbps)。
- 吞吐量驱动程序一致生成的总吞吐量：~ 3639 Mbps、峰值性能(~ 3908 Mbps)。

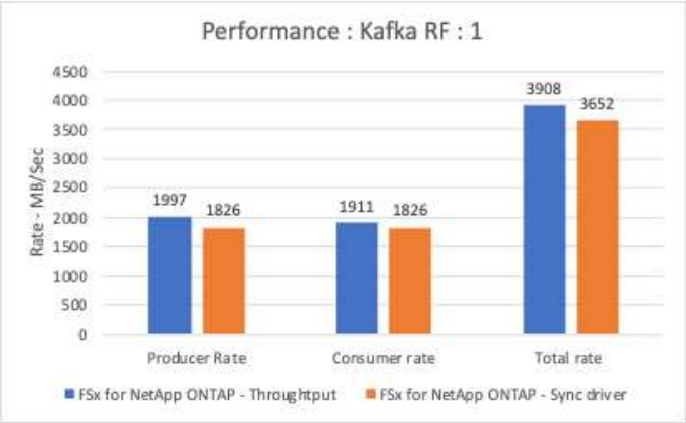
对于复制因子为3的Kafka和FSx for NetApp ONTAP：

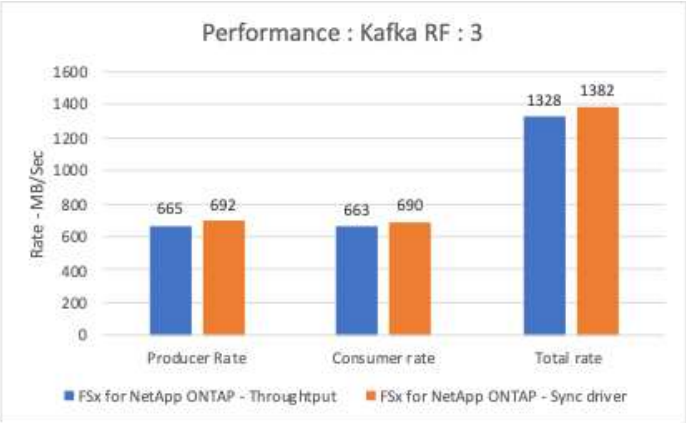
- Sync驱动程序一致生成的总吞吐量：~ 1252 Mbps、峰值性能(~ 1382 Mbps)。
- 吞吐量驱动程序一致生成的总吞吐量：~ 1218 MBps、峰值性能(以~ 1328 MBps为单位)。

在Kafka复制因子3中、读取和写入操作在FSx for NetApp ONTAP上发生了三次；在Kafka复制因子1中、读取和写入操作在FSx for NetApp ONTAP上发生了一次、因此、在这两种验证中、 我们可以达到4 GB/秒的最大吞吐量。

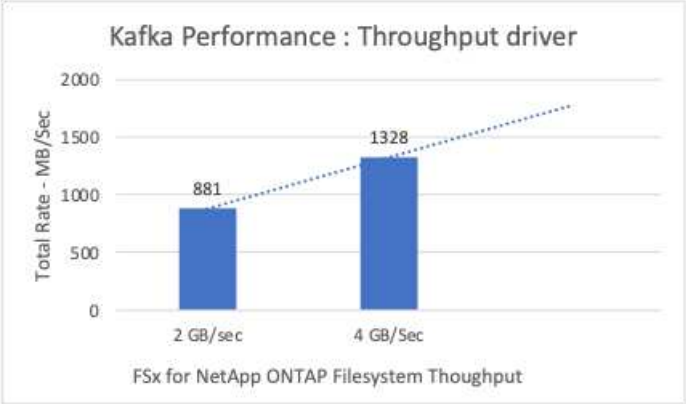
同步驱动程序可以在日志即时转储到磁盘时生成一致的吞吐量、而吞吐量驱动程序则在将日志批量提交到磁盘时生成突发的吞吐量。

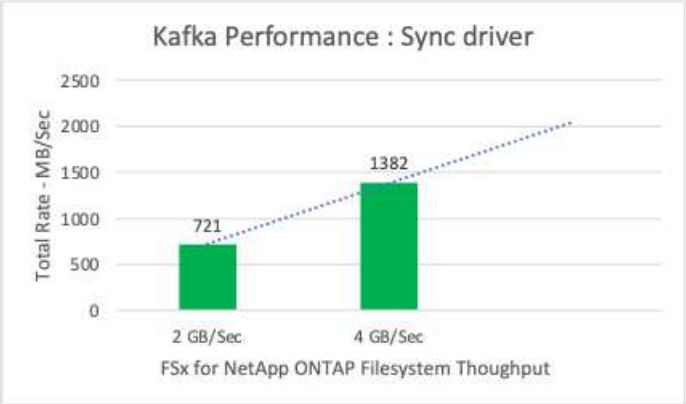
这些吞吐量数字是为给定的AWS配置生成的。为了满足更高的性能要求、可以进一步扩展和调整实例类型、以提高吞吐量。总吞吐量或总速率是生产者和使用者速率的组合。



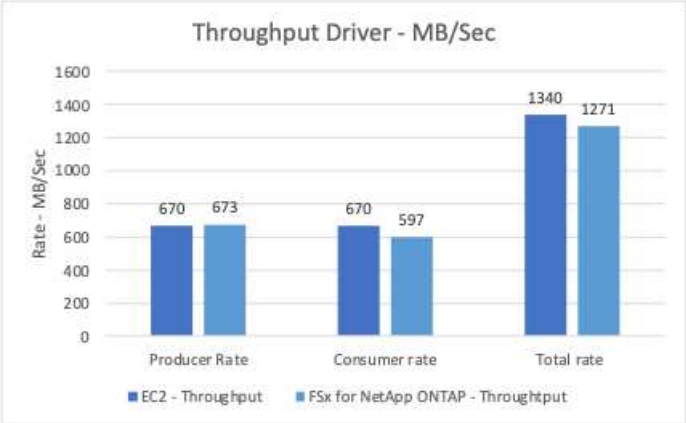


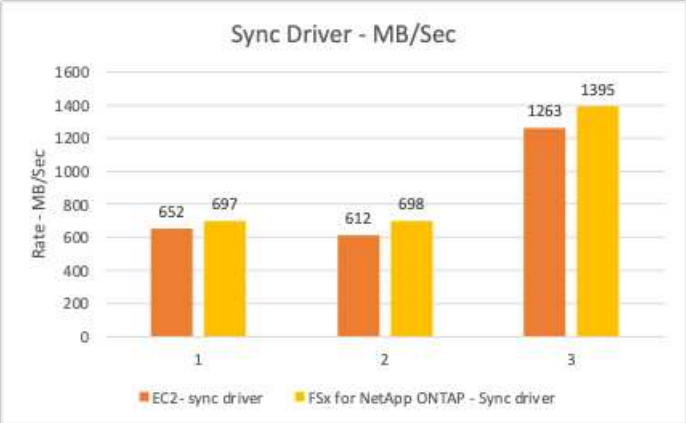
下图显示了NetApp ONTAP的2 GB/秒FSx和Kafka复制因子3的4 GB/秒性能。复制因子3在FSx for NetApp ONTAP存储上执行三次读取和写入操作。吞吐量驱动程序的总速率为881 MB/秒、在NetApp ONTAP文件系统的2 GB/秒FSx上执行读取和写入Kafka操作的速率约为2.64 GB/秒、吞吐量驱动程序的总速率为1328 MB/秒、执行读取和写入Kafka操作的速率约为3.98 GB/秒。Kafka的性能是线性的、可根据FSx for NetApp ONTAP吞吐量进行扩展。





下图显示了EC2实例与FSx for NetApp ONTAP之间的性能(Kafka复制因子：3)

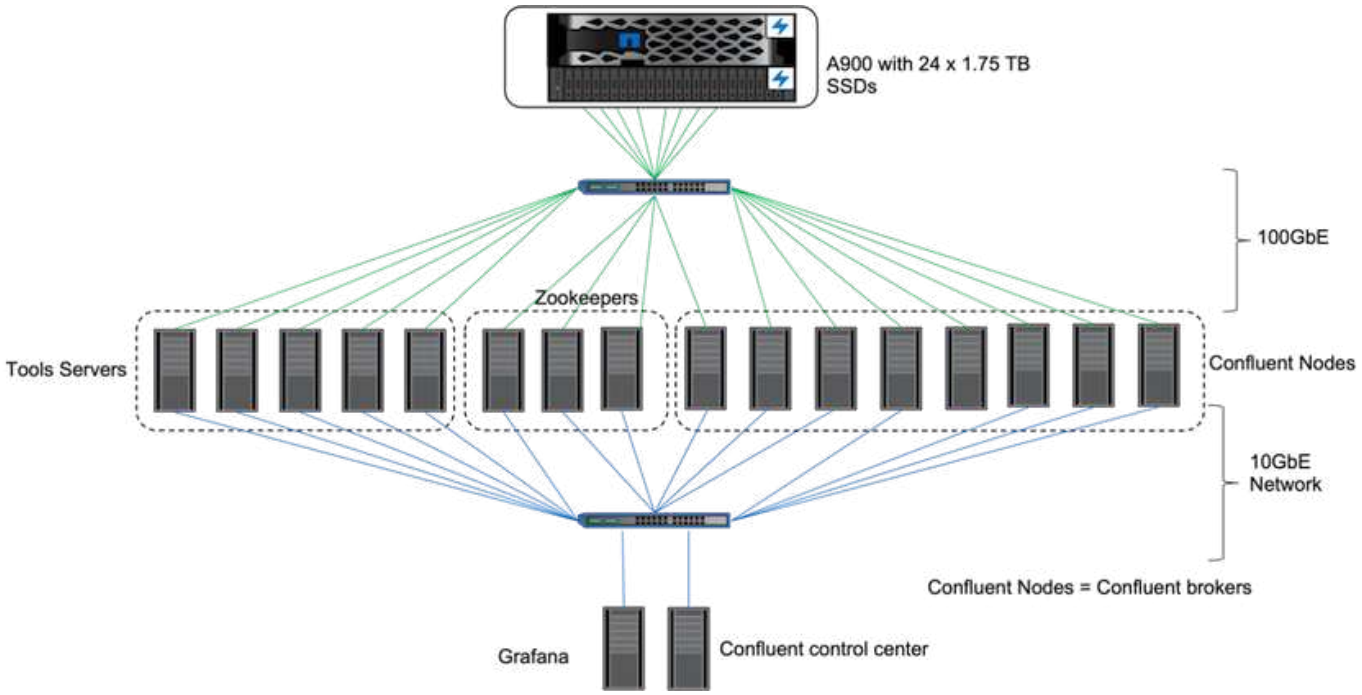




使用**AFF A900**内部部署进行性能概述和验证

在内部部署中、我们使用NetApp AFF A900存储控制器和ONTAP 9.12.1RC1来验证Kafka集群的性能和扩展能力。我们使用的测试平台与先前使用ONTAP 和AFF 的分层存储最佳实践中的测试平台相同。

我们使用Confluent Kafka 6.2.0对AFF A900进行了评估。集群具有八个代理节点和三个Zookeeper节点。在性能测试中、我们使用了五个OMB辅助节点。



存储配置

我们使用NetApp FlexGroup实例为日志目录提供了一个命名空间、从而简化了恢复和配置。我们使用NFSv4.1和pNFS为日志段数据提供直接路径访问。

客户端调整

每个客户端都使用以下命令挂载FlexGroup 实例。

```
mount -t nfs -o vers=4.1,nconnect=16 172.30.0.121:/kafka_vol01
/data/kafka_vol01
```

此外、我们还增加了 `max_session_slots` 默认值 `64` to `180`。这与ONTAP 中的默认会话插槽限制匹配。

Kafka代理调整

为了最大程度地提高测试系统中的吞吐量、我们显著增加了某些关键线程池的默认参数。对于大多数配置、我们建议遵循Confluent Kafka最佳实践。此调整用于最大程度地提高存储未处理I/O的并发性。可以根据代理的计算资源和存储属性调整这些参数。

```
num.io.threads=96
num.network.threads=96
background.threads=20
num.replica.alter.log.dirs.threads=40
num.replica.fetchers=20
queued.max.requests=2000
```

工作负载生成器测试方法

我们使用的OMB配置与对吞吐量驱动程序和主题配置进行云测试相同。

1. 已在AFF 集群上使用Ansible配置FlexGroup 实例。

```
---
- name: Set up kafka broker processes
  hosts: localhost
  vars:
    ntap_hostname: ‘hostname’
    ntap_username: 'user'
    ntap_password: 'password'
    size: 10
    size_unit: tb
    vserver: vs1
    state: present
    https: true
    export_policy: default
    volumes:
      - name: kafka_fg_vol01
        aggr: ["aggr1_a", "aggr2_a", “aggr1_b”, “aggr2_b”]
        path: /kafka_fg_vol01
  tasks:
    - name: Edit volumes
      netapp.ontap.na_ontap_volume:
        state: "{{ state }}"
        name: "{{ item.name }}"
        aggr_list: "{{ item.aggr }}"
        aggr_list_multiplier: 8
        size: "{{ size }}"
        size_unit: "{{ size_unit }}"
        vserver: "{{ vserver }}"
        snapshot_policy: none
        export_policy: default
        junction_path: "{{ item.path }}"
        qos_policy_group: none
        wait_for_completion: True
        hostname: "{{ ntap_hostname }}"
        username: "{{ ntap_username }}"
        password: "{{ ntap_password }}"
        https: "{{ https }}"
        validate_certs: false
      connection: local
      with_items: "{{ volumes }}"
```

2. 已在ONTAP SVM上启用pNFS。

```
vserver modify -vserver vs1 -v4.1-pnfs enabled -tcp-max-xfer-size 262144
```

3. 此工作负载是使用与Cloud Volumes ONTAP 相同的工作负载配置通过吞吐量驱动程序触发的。请参见第节"[稳定状态性能]"。工作负载使用的复制系数为3、这意味着在NFS中维护了三个日志段副本。

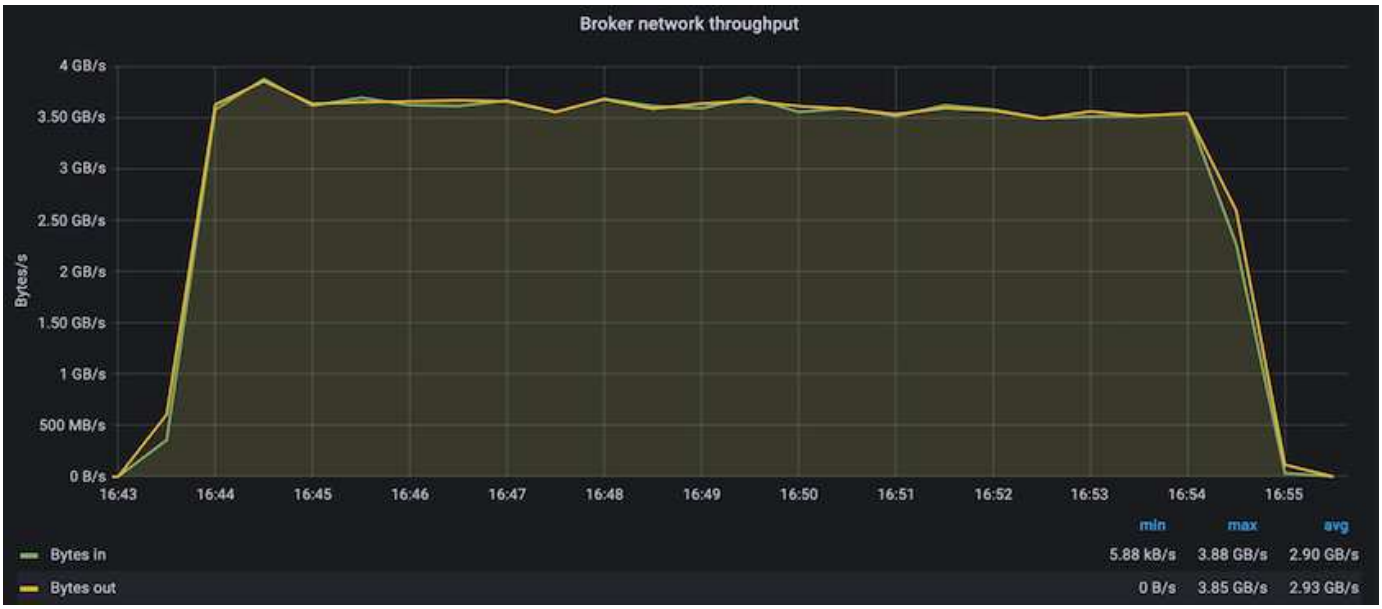
```
sudo bin/benchmark --drivers driver-kafka/kafka-throughput.yaml
workloads/1-topic-100-partitions-1kb.yaml
```

4. 最后、我们使用积压数据完成了测量、以衡量消费者是否能够跟上最新消息的步伐。OMB通过在测量开始期间暂停使用者来构建积压。这会产生三个不同的阶段：创建积压(仅限生产商的流量)、积压耗尽(消费者在一个主题中遇到未完成的事件时会遇到大量耗时的阶段)和稳定状态。请参见第节"探索存储限制"了解更多信息。

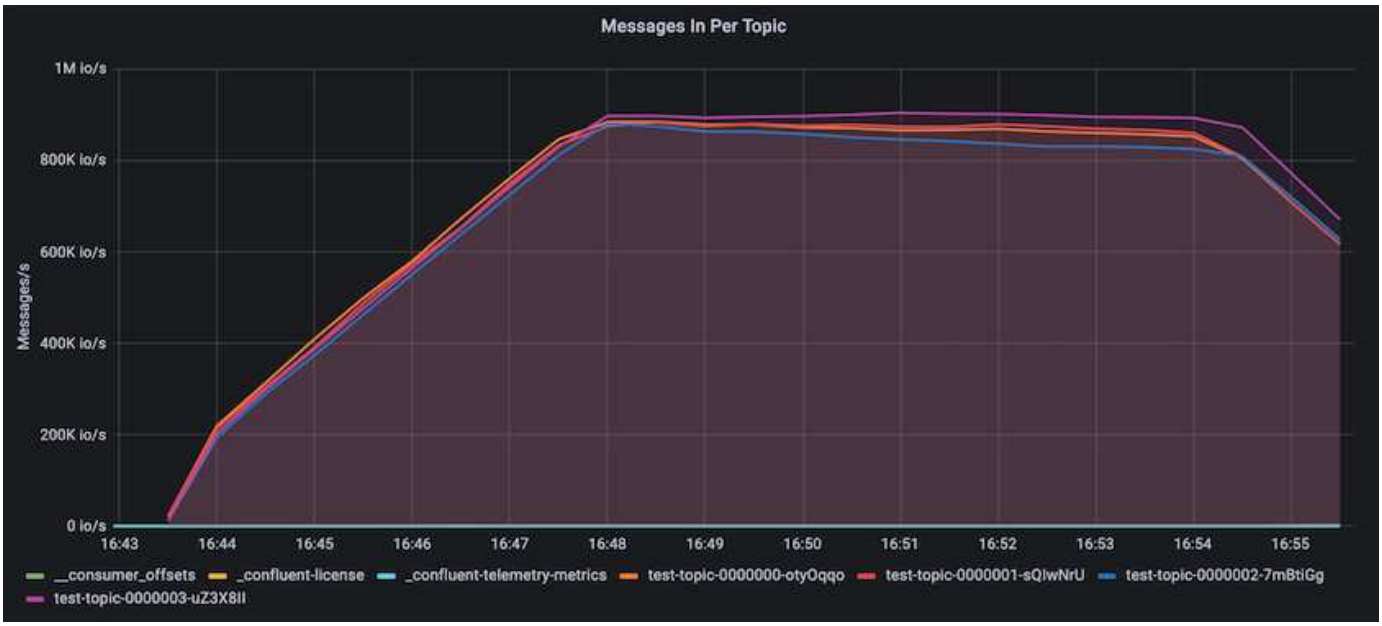
稳定状态性能

我们使用Open消息 基准测试对AFF A900进行了评估、以提供与AWS中的Cloud Volumes ONTAP 和AWS中的DAS类似的比较结果。所有性能值均表示生产商和消费者级别的Kafka-cluster吞吐量。

借助Confluent Kafka和AFF A900实现稳定的状态性能、生产者和使用者的平均吞吐量均超过3.4 GBps。在整个Kafka集群中、此消息超过340万条。通过直观地显示BrokerTopicMetrics的持续吞吐量(以字节/秒为单位)、我们可以看到AFF A900所支持的出色稳定状态性能和流量。



这与每个主题所传送消息的视图非常一致。下图按主题显示了细分情况。在测试的配置中、我们在四个主题中看到了每个主题近900、000条消息。

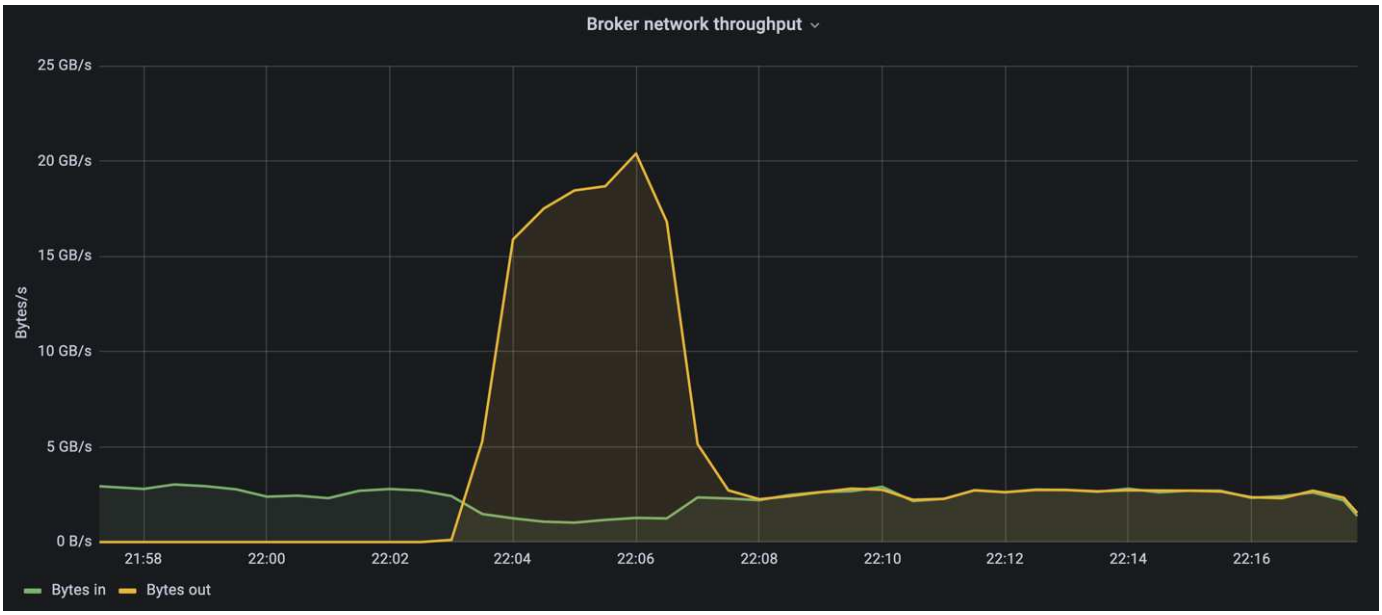


极致性能并探索存储限制

对于AFF 、我们还使用积压功能对OMB进行了测试。在Kafka集群中创建积压事件时、积压功能会暂停使用者订阅。在此阶段、仅会发生生成方流量、此流量会生成提交到日志的事件。这最能模拟批处理或脱机分析工作流；在这些工作流中、用户订阅会启动、并且必须读取已从代理缓存中逐出的历史数据。

为了了解此配置中对使用者吞吐量的存储限制、我们测量了纯生产者阶段、以了解A900可以吸收多少写入流量。请参见下一节"[规模估算指南]了解如何利用这些数据。

在本次测量中、我们发现、在仅用于生产商的部分、峰值吞吐量会突破A900性能的限制(此时、其他代理资源不会饱和地为生产商和消费者提供服务)。



我们将此度量值的消息大小增加到16k、以限制每条消息的开销、并最大程度地提高NFS挂载点的存储吞吐量。

```
messageSize: 16384
consumerBacklogSizeGB: 4096
```

Confluent Kafka集群的生产商吞吐量峰值为4.03 GBps。

```
18:12:23.833 [main] INFO WorkloadGenerator - Pub rate 257759.2 msg/s /
4027.5 MB/s | Pub err     0.0 err/s …
```

在OMB完成事件积压填充后、使用者流量将重新启动。在对积压量进行测量期间、我们在所有主题中观察到消费者峰值吞吐量超过20 Gbps。存储OMB日志数据的NFS卷的总吞吐量接近~30Gbps。

规模估算指南

Amazon Web Services提供了 "规模估算指南" 用于Kafka集群规模估算和扩展。

此规模估算为确定Kafka集群的存储吞吐量要求提供了一个有用的公式：

对于复制因子为r的tcluster集群生成的聚合吞吐量、代理存储收到的吞吐量如下：

```
t[storage] = t[cluster]/#brokers + t[cluster]/#brokers * (r-1)
           = t[cluster]/#brokers * r
```

这一点可以进一步简化：

```
max(t[cluster]) <= max(t[storage]) * #brokers/r
```

使用此公式、您可以根据Kafka热层需求选择合适的ONTAP 平台。

下表说明了A900的预期生产者吞吐量以及不同的复制因素：

复制因子	生产者吞吐量**(GPP)**
3 (测量值)	3.4
2.	5.1
1.	10.2

结论

针对错误重命名问题的NetApp解决方案 为以前与NFS不兼容的工作负载提供了一种简单、廉价且集中管理的存储形式。

这种新模式使客户能够创建更易于管理的Kafka集群、这些集群更易于迁移和镜像、以实现灾难恢复和数据保护。

我们还发现、NFS还可以通过NetApp ONTAP 提供更多优势、例如降低CPU利用率和加快恢复速度、显著提高存储效率以及提高性能。

从何处查找追加信息

要了解有关本文档中所述信息的更多信息，请查看以下文档和 / 或网站：

- 什么是Apache Kafka?

  "https://www.confluent.io/what-is-apache-kafka/"

- 什么是愚蠢的重命名?

  "https://linux-nfs.org/wiki/index.php/Server-side_silly_rename"

- 正在为流式应用程序读取ONATP。

  "https://www.netapp.com/blog/ontap-ready-for-streaming-applications/"

- 愚蠢—将问题描述 重命名为Kafka。

  "https://sbg.technology/2018/07/10/kafka-nfs/"

- NetApp 产品文档

  "https://www.netapp.com/support-and-training/documentation/"

- 什么是NFS?

  "https://en.wikipedia.org/wiki/Network_File_System"

- 什么是Kafka分区重新分配?

  "https://docs.cloudera.com/runtime/7.2.10/kafka-managing/topics/kafka-manage-cli-reassign-overview.html"

- 什么是Open消息 基准?

  "https://openmessaging.cloud/"

- 如何迁移Kafka代理?

  "https://medium.com/@sanchitbansal26/how-to-migrate-kafka-cluster-with-no-downtime-58c216129058"

- 您如何监控与Prometheus的Kafka代理?

  https://www.confluent.io/blog/monitor-kafka-clusters-with-prometheus-grafana-and-confluent/

- 适用于Apache Kafka的托管平台

  https://www.instaclustr.com/platform/managed-apache-kafka/

- 支持Apache Kafka

https://www.instaclustr.com/support-solutions/kafka-support/

- Apache Kafka的咨询服务

  https://www.instaclustr.com/services/consulting/

Kafka与NetApp ONTAP 存储控制器相结合

TR-4941：与NetApp ONTAP 存储控制器相结合

Karthikeyan Nagalingam、Joe Scott、NetApp Rankesh Kumar、Confluent

要使Confluent平台更具可扩展性和弹性、IT必须能够快速扩展和平衡工作负载。分层存储可减轻这种操作负担、从而使在Confluent中存储海量数据变得易于管理。

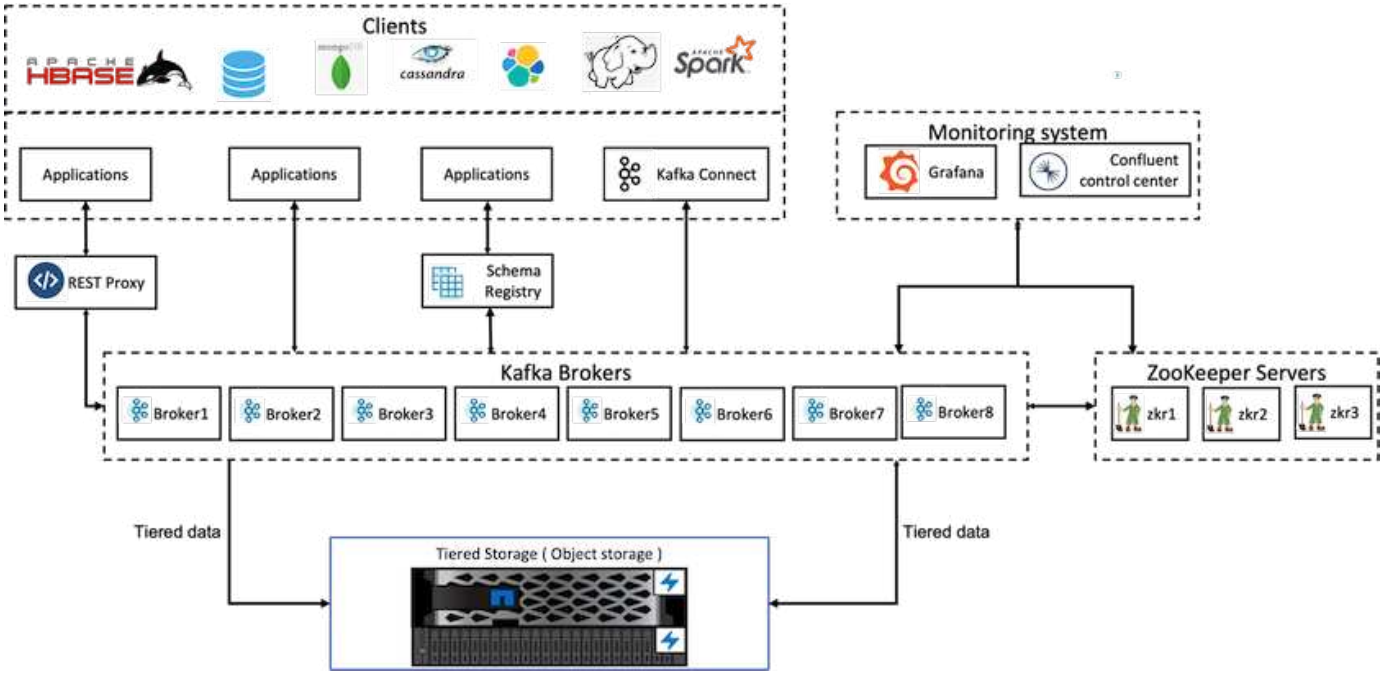
基本理念是将数据存储与数据处理分离、这样可以更轻松地独立扩展每个存储。

NetApp ONTAP 数据管理软件集行业领先的创新技术于一身、无论数据位于何处、均可为Confluent提供诸多优势。

本文档概述了使用分层存储基准测试套件的NetApp ONTAP 上的Confluent平台的性能基准测试。

解决方案

由ONTAP 提供支持的Confuent和NetApp AFF A900存储控制器是专为数据流设计的分布式系统。两者均可水平扩展、容错、并在负载下提供出色的性能。它们通过数据精简技术在分布式数据流和流处理方面相辅相成、并可降低存储成本、从而最大限度地减少数据占用空间。AFF A900存储控制器可提供出色的性能、同时允许分离计算和数据存储资源。这样可以简化系统管理并独立扩展资源。



解决方案架构详细信息

本节介绍在使用NetApp ONTAP 进行分层存储的Confluent Platform部署中用于性能验证的硬件和软件。下表介绍了解决方案 架构和基本组件。

平台组件	环境配置
Confuent Platform 6.2版	• 3个Zookepers • 8个代理服务器 • 5个工具服务器 • 1个Grafana • 1个控制中心
所有节点上的操作系统	Linux （ Ubuntu 18.04 ）
适用于温分段的NetApp ONTAP	• 1个AFF A900高可用性(HA)对 • 4 个 24 x 800 SSD • S3 协议 • 100GbE
15 台 Fujitsu PRIMERGY RX2540 服务器	• 2个CPU；总共16个物理核心 • Intel Xeon • 256 GB物理内存 • 100GbE双端口

技术概述

本节介绍此解决方案中使用的技术。

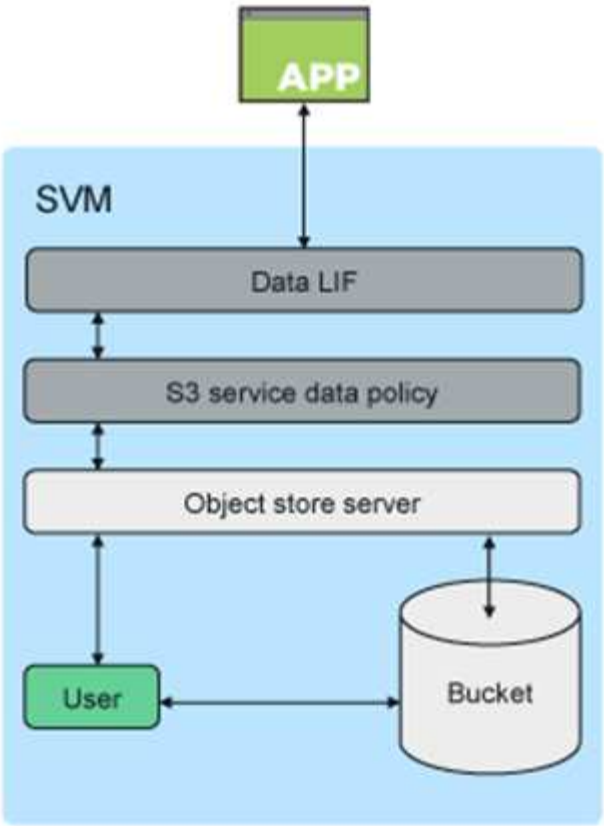
NetApp ONTAP 存储控制器

NetApp ONTAP 是一款高性能企业级存储操作系统。

NetApp ONTAP 9.8引入了对Amazon Simple Storage Service (S3) API的支持。ONTAP 支持部分Amazon Web Services (AWS) S3 API操作、并允许在云提供商(AWS、Azure和GCP)和内部环境中将数据表示为基于ONTAP的系统中的对象。

NetApp StorageGRID 软件是用于对象存储的旗舰级NetApp解决方案。ONTAP 通过在边缘提供载入和预处理点、扩展由NetApp提供支持的对象数据数据数据网络结构以及提高NetApp产品组合的价值、对StorageGRID 进行了补充。

可以通过授权的用户和客户端应用程序访问S3存储分段。下图显示了访问S3存储分段的应用程序。



主要用例

支持S3 API的主要目的是在ONTAP 上提供对象访问。ONTAP 统一存储架构现在支持文件(NFS和SMB)、块(FC和iSCSI)和对象(S3)。

原生 **S3**应用程序

越来越多的应用程序能够使用S3利用ONTAP 支持对象访问。虽然非常适合高容量归档工作负载、但原生 S3应用程序对高性能的需求正在快速增长、其中包括：

- 分析
- 人工智能
- 边缘到核心载入
- 机器学习

现在、客户可以使用熟悉的易管理性工具(如ONTAP System Manager)快速配置高性能对象存储、以便在ONTAP 中进行开发和操作、同时充分利用ONTAP 的存储效率和安全性。

FabricPool 端点

从ONTAP 9.8开始、FabricPool 支持在ONTAP 中分层到分段、从而可以进行ONTAP到ONTAP分层。对于希望将现有FAS 基础架构重新用作对象存储端点的客户来说、这是一个绝佳的选择。

FabricPool 支持通过两种方式分层到ONTAP：

- *本地集群分层。*使用集群LIF将非活动数据分层到位于本地集群上的存储分段。

- *远程集群分层。*非活动数据将采用与传统FabricPool 云层类似的方式分层到远程集群上的存储分层、方法是在FabricPool 客户端上使用IC LIF和在ONTAP 对象存储上使用数据LIF。

如果您需要在现有集群上使用 S3 功能，而无需额外的硬件和管理，则 ONTAP S3 是合适的。对于300 TB以上的部署、NetApp StorageGRID 软件仍然是NetApp对象存储的旗舰级解决方案。使用ONTAP 或StorageGRID作为云层时、不需要FabricPool 许可证。

适用于**Confluent**分层存储的**NetApp ONTAP**

每个数据中心都需要保持业务关键型应用程序持续运行、并确保重要数据的可用性和安全性。全新的NetApp AFF A900系统采用ONTAP 企业版软件和高弹性设计。我们全新的快如闪电的NVMe存储系统可消除任务关键型运营中断、最大限度地降低性能调整、并保护您的数据免受勒索软件攻击。

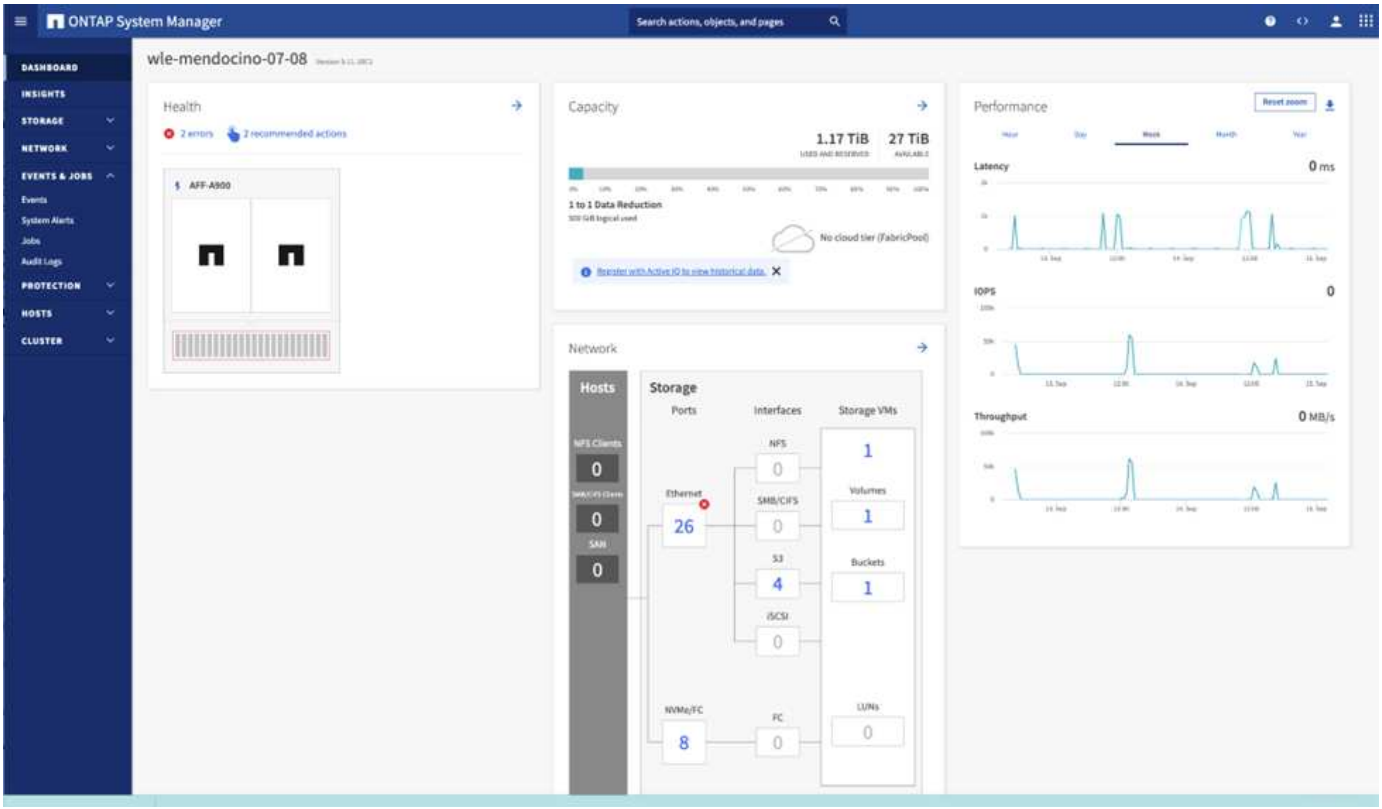
从初始部署到扩展Confluent集群、您的环境需要快速适应不会对业务关键型应用程序造成中断的变化。ONTAP 企业级数据管理、服务质量(Quality of Service、QoS)和性能支持您规划和适应您的环境。

将NetApp ONTAP 和Confluent分层存储结合使用、可将ONTAP 用作横向扩展存储目标、从而简化Apache Kafka集群的管理、并可为Confluent独立扩展计算和存储资源。

ONTAP S3服务器基于ONTAP 成熟的横向扩展存储功能构建。通过扩展S3存储分段、将新添加的节点添加到ONTAP 集群、可以无缝扩展ONTAP 集群。

使用**ONTAP System Manager**进行简单管理

ONTAP System Manager是一个基于浏览器的图形界面、可用于在一个管理平台中跨全球分布位置配置、管理和监控ONTAP 存储控制器。



您可以使用System Manager和ONTAP 命令行界面配置和管理ONTAP S3。当您启用S3并使用System Manager创建存储分段时、ONTAP 会为简化的配置提供最佳实践默认值。如果您从CLI配置S3服务器和存储分段、则仍

可根据需要使用System Manager对其进行管理、反之亦然。

使用 System Manager 创建 S3 存储分段时， ONTAP 会配置系统上可用性最高的默认性能服务级别。例如、在AFF 系统上、默认设置为"Extreme"。性能服务级别是预定义的自适应QoS策略组。您可以指定自定义 QoS 策略组，也可以不指定策略组，而不指定默认服务级别之一。

预定义的自适应QoS策略组包括以下内容：

- 至尊。*用于需要最低延迟和最高性能的应用程序。
- *性能。*用于性能需求和延迟适中的应用程序。
- 值。用于吞吐量和容量比延迟更重要的应用程序。
- *自定义。*指定自定义QoS策略或不指定QoS策略。

如果选择 * 用于分层 * ，则不会选择任何性能服务级别，系统会尝试为分层数据选择具有最佳性能的低成本介质。

ONTAP 会尝试在磁盘最合适的本地层上配置此存储分段，以满足所选的服务级别。但是，如果需要指定要包含在存储分段中的磁盘，请考虑通过指定本地层（聚合）从 CLI 配置 S3 对象存储。如果您通过 CLI 配置 S3 服务器，则仍可根据需要使用 System Manager 对其进行管理。

如果您希望能够指定用于存储分段的聚合，则只能使用命令行界面来执行此操作。

两者结合

Confuent Platform 是一个全面的数据流平台，可让您轻松地以持续的实时流的形式访问，存储和管理数据。Confluent 由 Apache Kafka 的原始创建者构建，通过企业级功能扩展了 Kafka 的优势，同时消除了 Kafka 的管理或监控负担。如今、《财富》100强企业中有80%以上的企业采用数据流技术、大多数企业都使用Confluent。

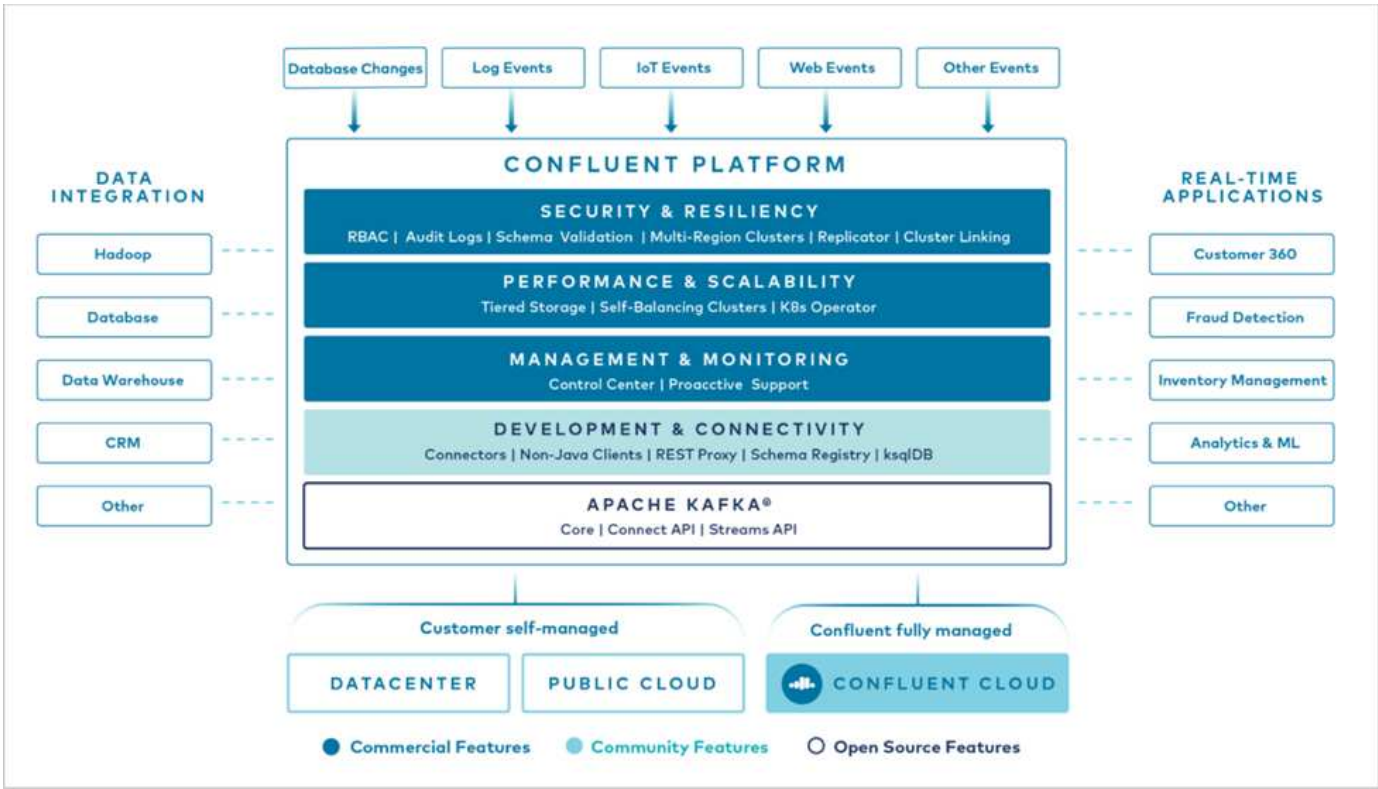
为什么选择 **Confluent** ？

通过将历史数据和实时数据集成到一个统一的中央真相来源中， Confluent 可以轻松构建一个全新的现代化事件驱动型应用程序类别，获得通用数据管道，并充分扩展性，性能和可靠性，释放出强大的新用例。

Confluent 的用途是什么？

借助整合平台，您可以专注于如何从数据中获得业务价值，而不是担心底层机制，例如如何在不同系统之间传输或集成数据。具体而言， Confluent Platform 可简化将数据源连接到 Kafka 的过程，构建流式应用程序，以及保护，监控和管理 Kafka 基础架构。如今、Consfluent Platform已广泛用于各行各业的各种用例、从金融服务、全渠道零售和自动驾驶汽车到欺诈检测、微服务和物联网。

下图显示了Confluent Platform的组件。



Confluent事件流技术概述

Confluent Platform 的核心是 "Kafka"最受欢迎的开源分布式流式平台。Kafka的主要功能包括：

- 发布并订阅记录流。
- 以容错方式存储记录流。
- 处理记录流。

即装即用的 Confluent 平台还包括架构注册表， REST 代理，总共 100 多个预构建的 Kafka 连接器和 ksqlDB 。

Confluent平台企业功能概述

- 流畅控制中心。一种基于UI的系统、用于管理和监控Kafka。您可以通过它轻松管理 Kafka Connect ，以及创建，编辑和管理与其他系统的连接。
- * Kubernetes 的 Confluent 。 * Kubernetes 的 Confluent 是 Kubernetes 的操作员。Kubernetes 操作员通过为特定平台应用程序提供独特的功能和要求，扩展了 Kubernetes 的业务流程功能。对于 Confluent Platform ，这包括大幅简化 Kubernetes 上 Kafka 的部署流程，并自动执行典型的基础架构生命周期任务。
- *卡夫卡连接连接器。*连接器使用Kafka Connect API将Kafka连接到数据库、密钥值存储、搜索索引和文件系统等其他系统。Confluent Hub 提供可下载的连接器，用于最常用的数据源和数据池，包括这些连接器经过全面测试且受支持的版本以及 Confluent 平台。有关更多详细信息，请参见 "此处"。
- * 自平衡集群。 * 提供自动化负载平衡，故障检测和自我修复功能。它还支持根据需要添加或停用代理、而无需手动调整。
- * 流畅集群链接。 * 直接将集群连接在一起，并通过链路网桥将主题从一个集群镜像到另一个集群。集群链接可简化多数据中心，多集群和混合云部署的设置。
- *流畅自动数据平衡器。*监控集群中的代理数量、分区大小、分区数量和导数。它允许您在集群中移动数据

以创建均匀的工作负载，同时限制重新平衡流量，以便在重新平衡的同时最大限度地减少对生产工作负载的影响。

- * 流畅复制器。 * 使在多个数据中心维护多个 Kafka 集群变得比以往任何时候都更轻松。
- * 分层存储。 * 提供了使用您喜欢的云提供商存储大量 Kafka 数据的选项，从而减轻了运营负担并降低了成本。借助分层存储，您只能在需要更多计算资源时，才可以将数据保存在经济高效的对象存储和扩展代理上。
- * 流畅的 jms 客户端。 * 流畅平台包括适用于 Kafka 的与 jms 兼容的客户端。此 Kafka 客户端使用 Kafka 代理作为后端，实施了 Jms 1.1 标准 API 。如果旧版应用程序使用的是 jms ，并且您希望将现有的 jms 消息代理替换为 Kafka ，则此功能非常有用。
- * 流畅的 MQT 代理。 * 提供了一种从 MQT 设备和网关直接向 Kafka 发布数据的方法，而无需在中间使用 MQT 代理。
- * 流畅安全插件。 * 流畅安全插件用于为各种流畅平台工具和产品添加安全功能。目前，可以为 Confluent REST 代理提供一个插件，用于对传入请求进行身份验证，并将经过身份验证的主体传播到 Kafka 请求。这样， Confluent REST 代理客户端便可利用 Kafka 代理的多租户安全功能。

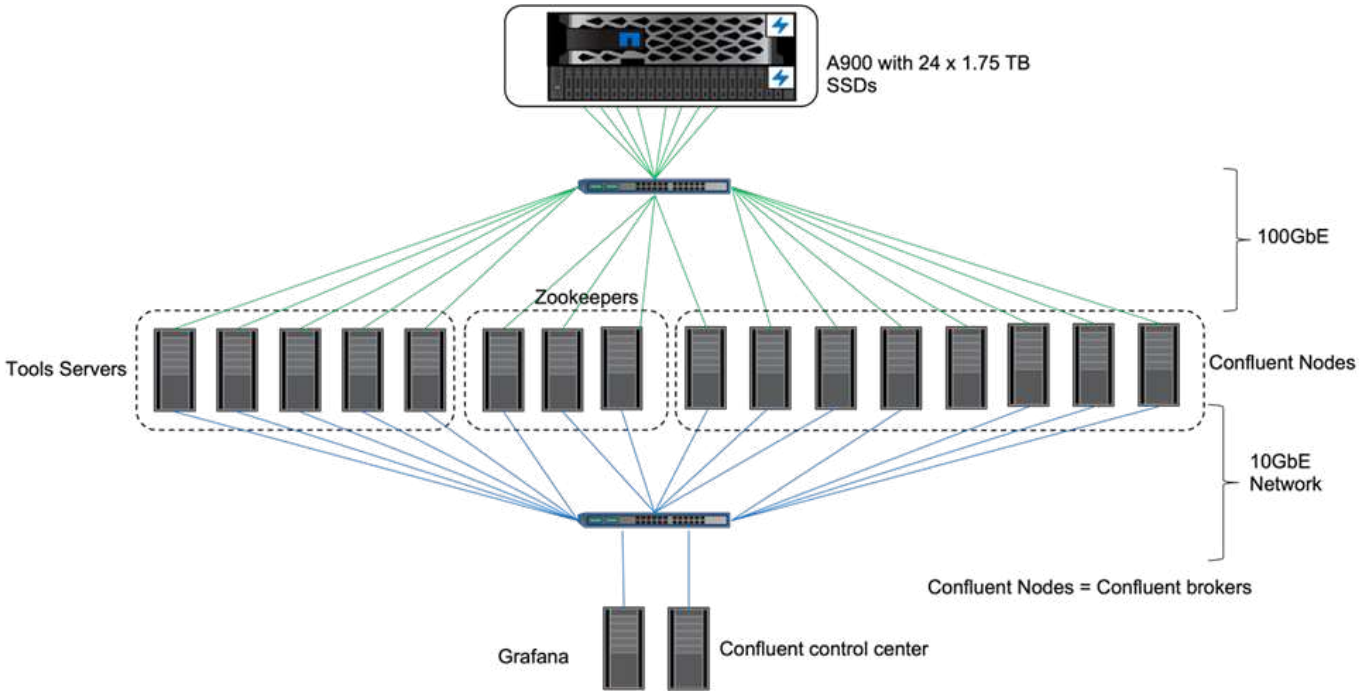
融合性能验证

我们已使用Confluent Platform对NetApp ONTAP 上的分层存储执行了验证。NetApp和Confluent团队共同执行了此验证、并运行了所需的测试用例。

设置冲突

在设置中、我们使用了三个Zookepers、五个代理和五个测试服务器、这些服务器具有256 GB RAM和16个CPU。对于NetApp存储、我们将ONTAP 与AFF A900 HA对结合使用。存储和代理通过100GbE连接进行连接。

下图显示了用于分层存储验证的配置的网络拓扑。



工具服务器充当向Confluent节点发送或从Confluent节点接收事件的应用程序客户端。

融合的分层存储配置

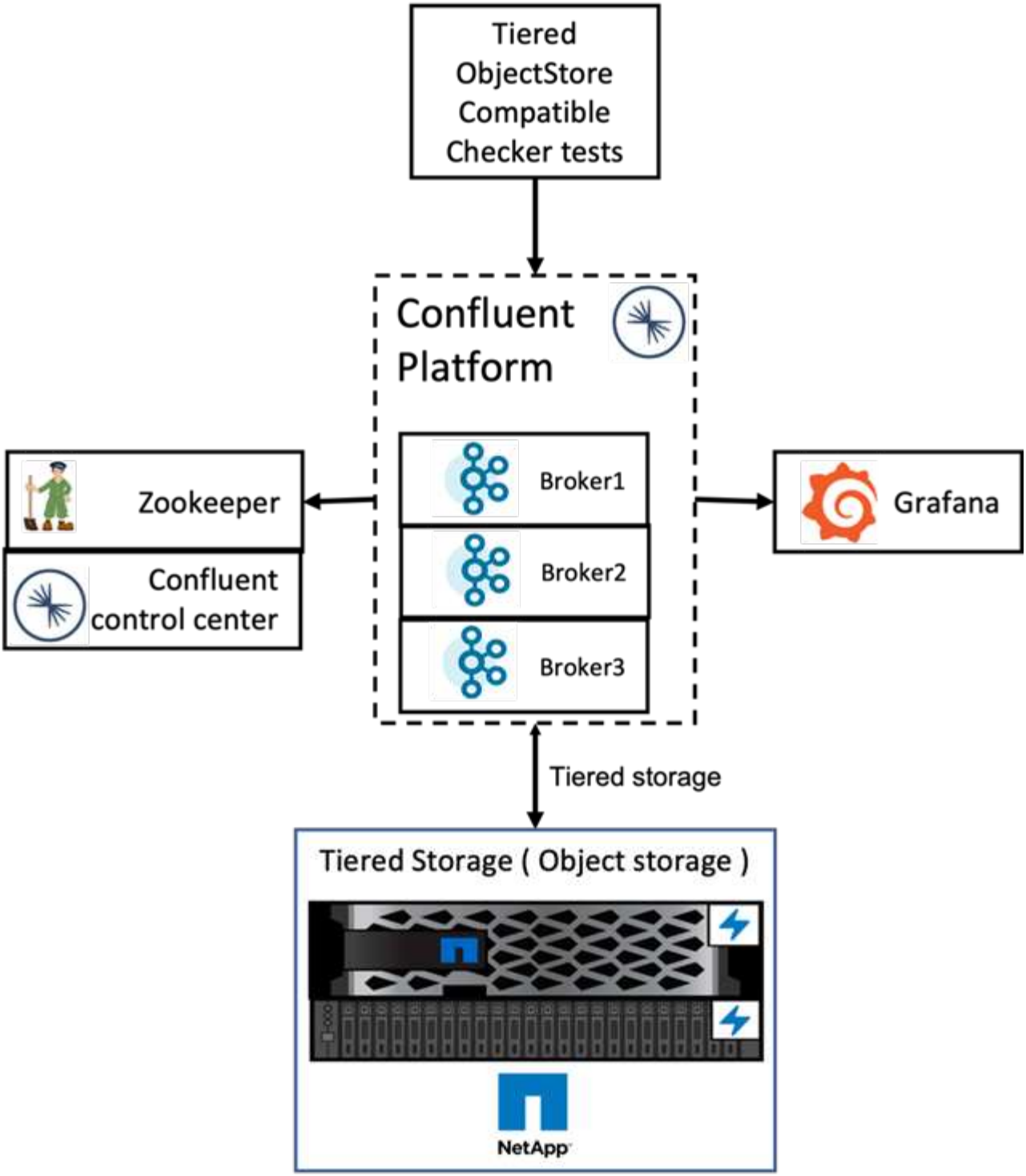
我们使用了以下测试参数：

```
confluent.tier.fetcher.num.threads=80
confluent.tier.archiver.num.threads=80
confluent.tier.enable=true
confluent.tier.feature=true
confluent.tier.backend=S3
confluent.tier.s3.bucket=kafkabucket1-1
confluent.tier.s3.region=us-east-1
confluent.tier.s3.cred.file.path=/data/kafka/.ssh/credentials
confluent.tier.s3.aws.endpoint.override=http://wle-mendocino-07-08/
confluent.tier.s3.force.path.style.access=true
bootstrap.server=192.168.150.172:9092,192.168.150.120:9092,192.168.150.164
:9092,192.168.150.198:9092,192.168.150.109:9092,192.168.150.165:9092,192.1
68.150.119:9092,192.168.150.133:9092
debug=true
jmx.port=7203
num.partitions=80
num.records=200000000
#object PUT size - 512MB and fetch 100MB - netapp
segment.bytes=536870912
max.partition.fetch.bytes=1048576000
#GET size is max.partition.fetch.bytes/num.partitions
length.key.value=2048
trogdor.agent.nodes=node0,node1,node2,node3,node4
trogdor.coordinator.hostname.port=192.168.150.155:8889
num.producers=20
num.head.consumers=20
num.tail.consumers=1
test.binary.task.max.heap.size=32G
test.binary.task.timeout.sec=3600
producer.timeout.sec=3600
consumer.timeout.sec=3600
```

为了进行验证、我们将ONTAP 与HTTP协议结合使用、但HTTPS也可以正常工作。访问密钥和机密密钥存储在 `confuent.tier.s3.cred.file.path` 参数中提供的文件名中。

NetApp存储控制器—ONTAP

我们在ONTAP 中配置了一个HA对配置以进行验证。



验证结果

我们已完成以下五个测试案例以进行验证。前两项是功能测试，其余三项是性能测试。

对象存储正确性测试

此测试使用API调用对用于分层存储的对象存储执行基本操作、例如GET、PUT和DELETE。

分层功能正确性测试

此测试将检查对象存储的端到端功能。它会创建一个主题，为新创建的主题生成一个事件流，等待代理将这些分段归档到对象存储，使用事件流，并验证已用流与已生成流的匹配情况。我们执行此测试时，无论是否注入了对象存储故障。我们通过在ONTAP 中的一个节点中停止服务管理器服务并验证端到端功能是否适用于对象存储来模拟节点故障。

层提取基准测试

此测试验证了分层对象存储的读取性能，并检查了基准测试生成的区块在负载过重时的范围提取读取请求。在此基准测试中， Confluent 开发了自定义客户端来处理层提取请求。

生成并使用工作负载生成器

此测试会通过归档区块间接在对象存储上生成写入工作负载。读取工作负载（区块读取）是在使用者组提取区块时从对象存储生成的。此工作负载由TOCC脚本生成。此测试检查了并行线程中对象存储上的读写性能。与分层功能正确性测试一样，我们测试了是否存在对象存储故障注入。

保留工作负载生成器

此测试检查了在主题保留工作负载繁重的情况下对象存储的删除性能。保留工作负载是使用TOCC脚本生成的、该脚本会与测试主题并行生成许多消息。本测试主题使用主动式基于大小和基于时间的保留设置进行配置，此设置会导致从对象存储中持续清除事件流。然后，这些区块会归档。这导致代理在对象存储中删除了许多内容、并收集了对象存储删除操作的性能。

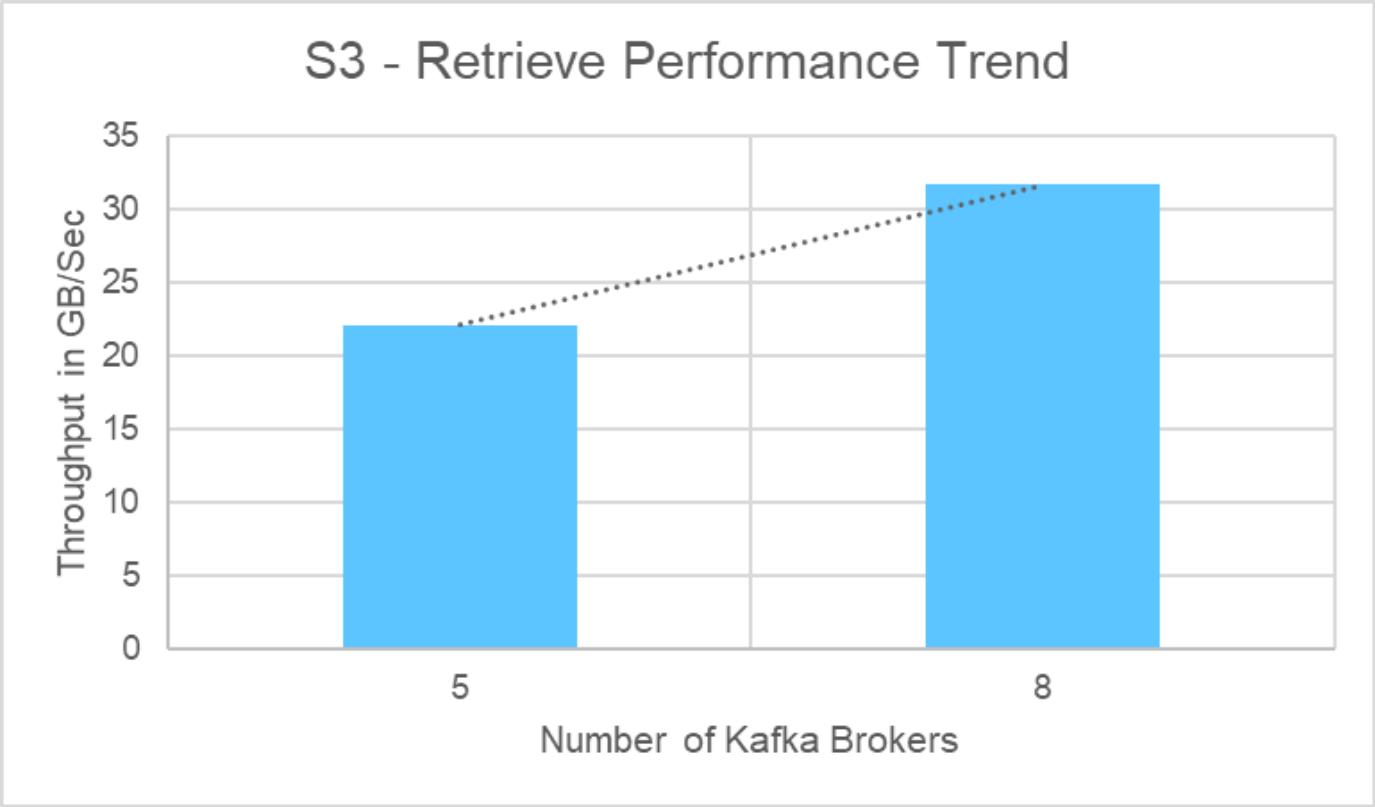
有关验证详细信息、请参见 "两者结合" 网站。

使用生产用工作负载生成器进行性能测试

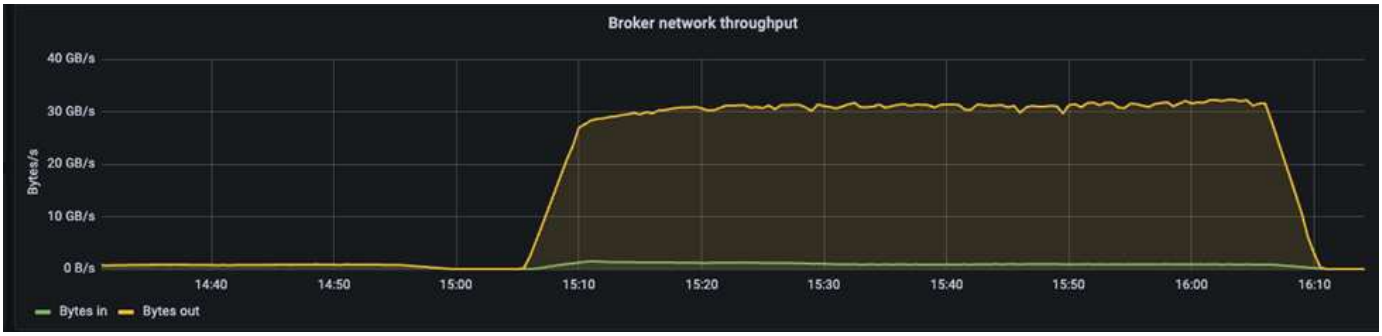
在使用一个AFF A900 HA对NetApp存储控制器的生产用工作负载期间、我们使用五个或八个代理节点执行分层存储测试。根据我们的测试结果、完成时间和性能结果会随着代理节点的数量进行扩展、直到AFF A900资源利用率达到百分之一百为止。ONTAP 存储控制器设置至少需要一个HA对。

根据Confluent代理节点的数量、S3检索操作的性能呈线性增长。ONTAP 存储控制器在一个部署中最多支持12个HA对。

下图显示了具有五个或八个代理节点的S3分层流量组合。我们最大限度地提高了AFF A900单HA对的性能。



下图显示了Kafka吞吐量约为31.74 GBps。



我们还在ONTAP 存储控制器`perfstat`报告中观察到类似的吞吐量。

```
object_store_server:wle-mendocino-07-08:get_data:34080805907b/ s
object_store_server:wle-mendocino-07-08:put_data:484236974b/ s
```

性能最佳实践准则

此页面介绍了在此解决方案 中提高性能的最佳实践。

- 对于ONTAP 、如果可能、请使用GET大小>=1MB。
- 通过在代理节点上的`server.properties`中增加`num.network.threads`和`num.io.threads`、您可以将更多的分层活动推送到S3层。这些结果会在`num.network.threads`和`num.io.threads`设置为32的情况下显示。
- S3存储分段应针对每个成员聚合的八个成分卷。

- 驱动S3流量的以太网链路应尽可能在存储和客户端上使用9k的MTU。

结论

此验证测试在使用NetApp ONTAP 存储控制器的Confluent上达到了31.74 GBps的分层吞吐量。

从何处查找追加信息

要了解有关本文档中所述信息的更多信息，请查看以下文档和 / 或网站：

- 什么是Confluent?

  "https://www.confluent.io/apache-kafka-vs-confluent/"

- S3-sink 参数详细信息

  "https://docs.confluent.io/kafka-connect-s3-sink/current/configuration_options.html#s3-configuration-options"

- Apache Kafka

  "https://en.wikipedia.org/wiki/Apache_Kafka"

- ONTAP 最佳实践中的S3

  https://www.netapp.com/pdf.html?item=/media/17219-tr4814.pdf

- S3对象存储管理

  "https://docs.netapp.com/us-en/ontap/s3-config/s3-support-concept.html"

- NetApp 产品文档

  "https://www.netapp.com/support-and-training/documentation/"

适用于 **Apache Spark** 的 **NetApp** 存储解决方案

TR-4570：《适用于**Apache Spark**的**NetApp**存储解决方案：架构、用例和性能结果》

NetApp公司Rick Huang、Karthikeyan Nagalingam

本文档重点介绍与大数据分析和人工智能(AI)相关的Apache Spark架构、客户用例和NetApp存储产品组合。此外、它还会根据典型的Hadoop系统使用行业标准AI、机器学习(ML)和深度学习(DL)工具显示各种测试结果、以便您可以选择适当的Spark解决方案。首先、您需要一个Spark架构、适当的组件以及两种部署模式(集群和客户端)。

本文档还提供了客户用于解决配置问题的用例、并概述了与大数据分析以及采用Spark的AI、ML和DL相关的NetApp存储产品组合。然后、我们将根据Spark专用用例和NetApp Spark解决方案 产品组合得出测试结果。

客户面临的挑战

本节重点介绍客户在零售、数字营销、银行、离散式制造、流程制造等数据增长行业面临的大数据分析和AI/ML/DL挑战。 政府和专业服务。

性能不可预测

传统Hadoop部署通常使用商用硬件。要提高性能、您必须调整网络、操作系统、Hadoop集群、Spark等生态系统组件和硬件。即使您对每一层进行了调整、也很难达到所需的性能级别、因为Hadoop运行在并非为环境中的高性能而设计的商用硬件上。

介质和节点故障

即使在正常情况下、商用硬件也容易发生故障。如果数据节点上的一个磁盘发生故障、则默认情况下、Hadoop主节点会将该节点视为运行状况不正常。然后、它会通过网络将特定数据从该节点从副本复制到运行正常的节点。此过程会减慢任何Hadoop作业的网络数据包速度。然后、当运行状况不正常的节点恢复正常时、集群必须重新复制数据并删除过度复制的数据。

Hadoop供应商锁定

Hadoop分销商拥有自己的Hadoop分发软件包、并拥有自己的版本控制、这会使客户锁定到这些分发软件包中。但是、许多客户需要支持内存分析、而不会将客户与特定Hadoop分发版联系起来。他们需要自由地更改分发版本、同时仍能利用分析功能。

不支持多种语言

客户通常除了需要MapReduce Java程序支持多种语言之外、还需要支持多种语言才能运行其作业。SQL和脚本等选项可以更灵活地获取答案、提供更多的数据组织和检索选项、以及更快地将数据移动到分析框架中。

难以使用

一段时间以来、人们一直抱怨Hadoop难以使用。尽管Hadoop在每个新版本中变得更加简单、功能更强大、但这一评论仍然存在。Hadoop要求您了解Java和MapReduce编程模式、这是数据库管理员和具有传统脚本编写技能的人员面临的一项挑战。

复杂的框架和工具

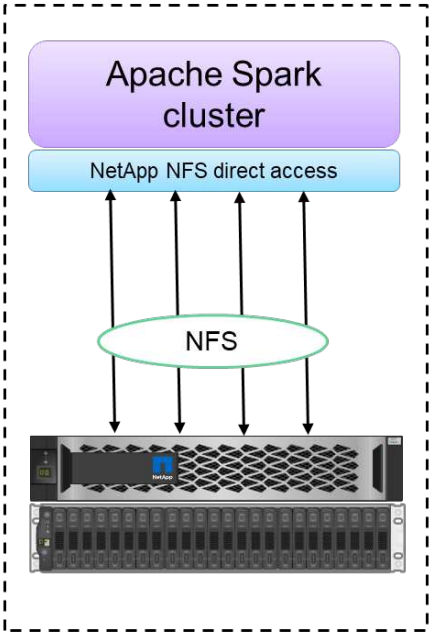
企业AI团队面临多种挑战。即使具备专业的数据科学知识、适用于不同部署生态系统和应用程序的工具和框架也可能不会简单地从一个转变为另一个。数据科学平台应与基于Spark构建的相应大数据平台无缝集成、可轻松移动数据、实现可重复使用的模型、即装即用的代码以及支持原型制作、验证、版本控制、共享、重复使用的最佳实践的工具。 并将模型快速部署到生产环境中。

为什么选择**NetApp**?

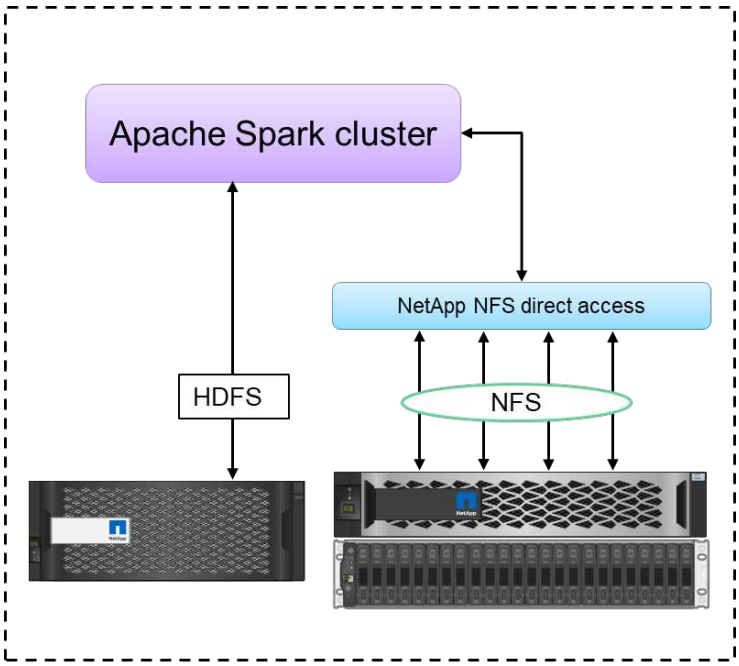
NetApp可以通过以下方式改善您的Spark体验：

- 通过NetApp NFS直接访问(如下图所示)、客户无需移动或复制数据、即可对现有或新的NFSv3或NFSv4数据运行大数据分析作业。它可以防止多个数据副本，并且无需将数据与源进行同步。
- 存储效率更高、服务器复制更少。例如、NetApp E系列Hadoop解决方案 需要两个而非三个数据副本、而FAS Hadoop解决方案 则需要一个数据源、但不需要复制或复制数据。NetApp存储解决方案还可以减少服务器到服务器的流量。
- 在驱动器和节点发生故障期间、Hadoop作业和集群行为会更好。

- 提高数据载入性能。



Configuration 1: NFS as primary storage



Configuration 2: HDFS and NFS in single Spark cluster

例如、在金融和医疗保健领域、数据从一个位置移动到另一个位置必须履行法律义务、这不是一项容易的任务。在此情景中、NetApp NFS直接访问功能会分析其原始位置的财务和医疗保健数据。另一个主要优势是、使用NetApp NFS直接访问可通过原生 Hadoop命令简化对Hadoop数据的保护、并通过NetApp丰富的数据管理产品组合启用数据保护工作流。

NetApp NFS直接访问为Hadoop/Spark集群提供了两种部署选项：

- 默认情况下、Hadoop或Spark集群使用Hadoop分布式文件系统(HDFS)进行数据存储和默认文件系统。NetApp NFS直接访问可以将默认HDFS替换为NFS存储作为默认文件系统、从而可以对NFS数据进行直接分析。
- 在另一种部署选项中、NetApp NFS直接访问支持在单个Hadoop或Spark集群中将NFS与HDFS配置为额外存储。在这种情况下，客户可以通过 NFS 导出共享数据，并从同一集群访问数据以及 HDFS 数据。

使用NetApp NFS直接访问的主要优势包括：

- 从当前位置分析数据、这样可以防止将分析数据移动到HDFS等Hadoop基础架构这一既耗时又耗性能的任务。
- 将副本数量从三个减少为一个。
- 使用户能够分离计算和存储以独立扩展。
- 利用ONTAP 丰富的数据管理功能提供企业数据保护。
- Hortonworks数据平台认证。
- 支持混合数据分析部署。
- 利用动态多线程功能缩短备份时间。

请参见 "TR-4657： NetApp 混合云数据解决方案—基于客户用例的 Spark 和 Hadoop" 用于将Hadoop数据备份、备份和灾难恢复从云备份到内部环境、对现有Hadoop数据启用DevTest、实现数据保护和多云连接、并加

快分析工作负载的速度。

以下各节介绍了对Spark客户非常重要的存储功能。

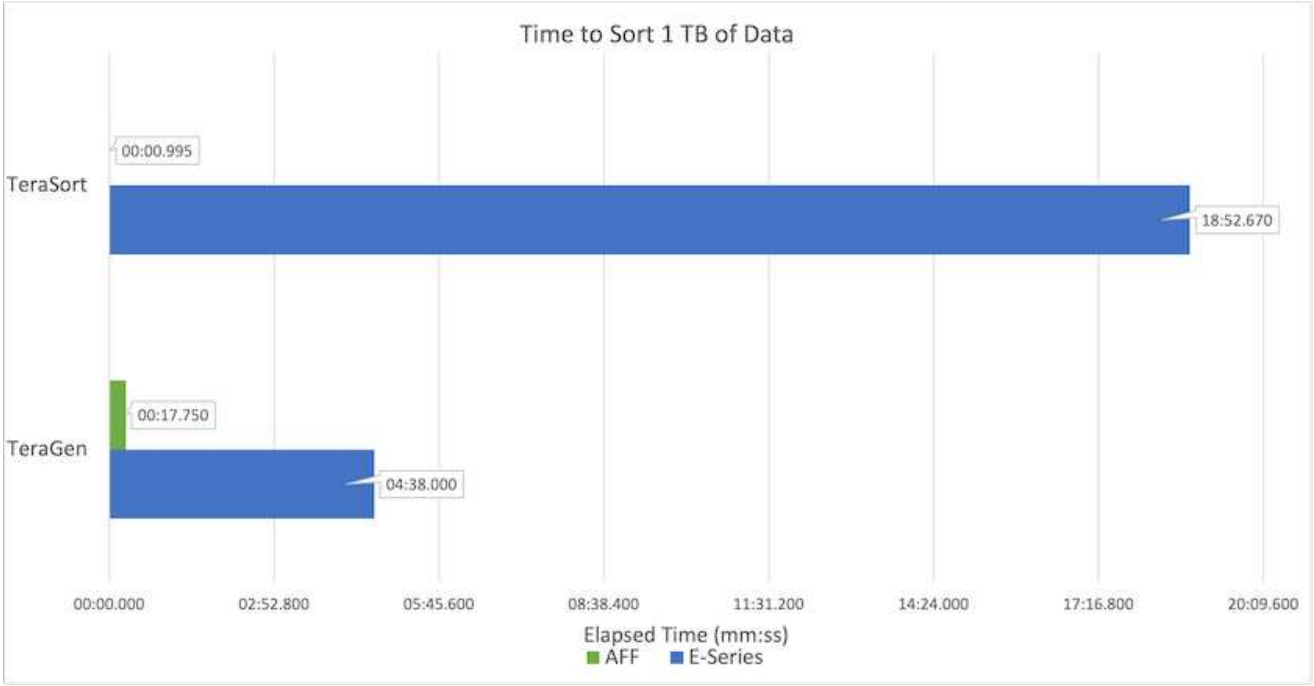
存储分层

使用Hadoop存储分层、您可以根据存储策略存储具有不同存储类型的文件。存储类型包括`Hot`、`Cold`、`Warm`、`All_SSD`、`One_SSD`、 和`lazy_persist`。

<<<<<<<<<HEAD我们对使用不同存储策略的SSD和SAS驱动器的NetApp AFF 存储控制器和E系列存储控制器上的Hadoop存储分层进行了验证。采用AF-A800的Spark集群具有四个计算辅助节点、而采用E系列的集群则具有八个。这主要是为了比较固态驱动器(SSD)与硬盘驱动器(HDD)的性能。

我们对使用不同存储策略的SSD和SAS驱动器的NetApp AFF 存储控制器和E系列存储控制器上的Hadoop存储分层进行了验证。采用AF-A800的Spark集群具有四个计算辅助节点、而采用E系列的集群则具有八个。我们这样做的主要目的是将固态驱动器与硬盘驱动器磁盘的性能进行比较。>>>>>a51c9ddf73ca69e1120ce05edc7b0b9607b96eae.

下图显示了适用于Hadoop SSD的NetApp解决方案的性能。



- 基线NL-SAS配置使用8个计算节点和96个NL-SAS驱动器。此配置在4分38秒内生成1 TB的数据。请参见 "TR-3969适用于Hadoop的NetApp E系列解决方案" 有关集群和存储配置的详细信息。
- 使用TeraGen、SSD配置生成的数据速度比NL-SAS配置快15.66倍。此外、SSD配置使用的计算节点数为计算节点数的一半、磁盘驱动器数的一半(总共24个SSD驱动器)。根据作业完成时间、该速度几乎是NL-SAS配置的两倍。
- 使用TeraSort、SSD配置的1 TB数据排序速度比NL-SAS配置快1138.36倍。此外、SSD配置使用的计算节点数为计算节点数的一半、磁盘驱动器数的一半(总共24个SSD驱动器)。因此、每个驱动器的速度大约是NL-SAS配置的三倍。<<<<<<<<< 标题
- 这种方法正在从旋转磁盘过渡到全闪存、从而提高了性能。计算节点的数量不是瓶颈。借助NetApp的全闪存存储、运行时性能可进行良好扩展。
- 使用NFS时、数据在功能上相当于将全部池化在一起、这样可以根据您的工作负载减少计算节点的数量。更改计算节点数量时、Apache Spark集群用户无需手动重新平衡数据。

- 总之、从旋转磁盘过渡到全闪存可提高性能。计算节点的数量不是瓶颈。借助NetApp全闪存存储、运行时性能可进行良好扩展。
- 使用NFS、数据在功能上相当于将所有数据池在一起、这样可以根据您的工作负载减少计算节点的数量。Apache Spark集群用户在更改计算节点数量时无需手动重新平衡数据。>>>>>a51c9ddf73ca69e1120ce05edc7b0b9607b96eae.

性能扩展—横向扩展

如果您需要AFF 解决方案 中Hadoop集群的更多计算能力、则可以添加具有适当数量存储控制器的数据节点。NetApp建议从每个存储控制器阵列四个数据节点开始、并根据工作负载特征将每个存储控制器的数据节点数增加到八个。

AFF 和FAS 非常适合原位分析。根据计算要求、您可以添加节点管理器、而无中断操作允许您按需添加存储控制器、而无需停机。我们通过AFF 和FAS 提供丰富的功能、例如NVMe介质支持、有保障的效率、数据精简、QoS、预测性分析、 云分层、复制、云部署和安全性。为了帮助客户满足其需求、NetApp提供了文件系统分析、配额和机载负载平衡等功能、无需额外的许可证成本。与竞争对手相比、NetApp在并发作业数量方面的性能更好、延迟更短、操作更简单、每秒吞吐量也更高。此外、NetApp Cloud Volumes ONTAP 还可在所有三个主要云提供商上运行。

性能扩展—纵向扩展

通过纵向扩展功能、您可以在需要更多存储容量时向AFF 、FAS 和E系列系统添加磁盘驱动器。借助Cloud Volumes ONTAP 、将存储扩展到PB级别是两个因素的组合：将不常用的数据从块存储分层到对象存储、以及在不进行额外计算的情况下堆栈Cloud Volumes ONTAP 许可证。

多个协议

NetApp系统支持适用于Hadoop部署的大多数协议、包括SAS、iSCSI、FCP、InfiniBand、 和NFS。

运行和支持的解决方案

NetApp支持本文档中所述的Hadoop解决方案。这些解决方案还通过了主要Hadoop分销商的认证。有关信息、请参见 "MapR" 站点、 "Hortonworks" 站点和Cloudera "认证" 和 "合作伙伴" 站点。

目标受众

分析和数据科学领域涉及IT和业务领域的多个领域：

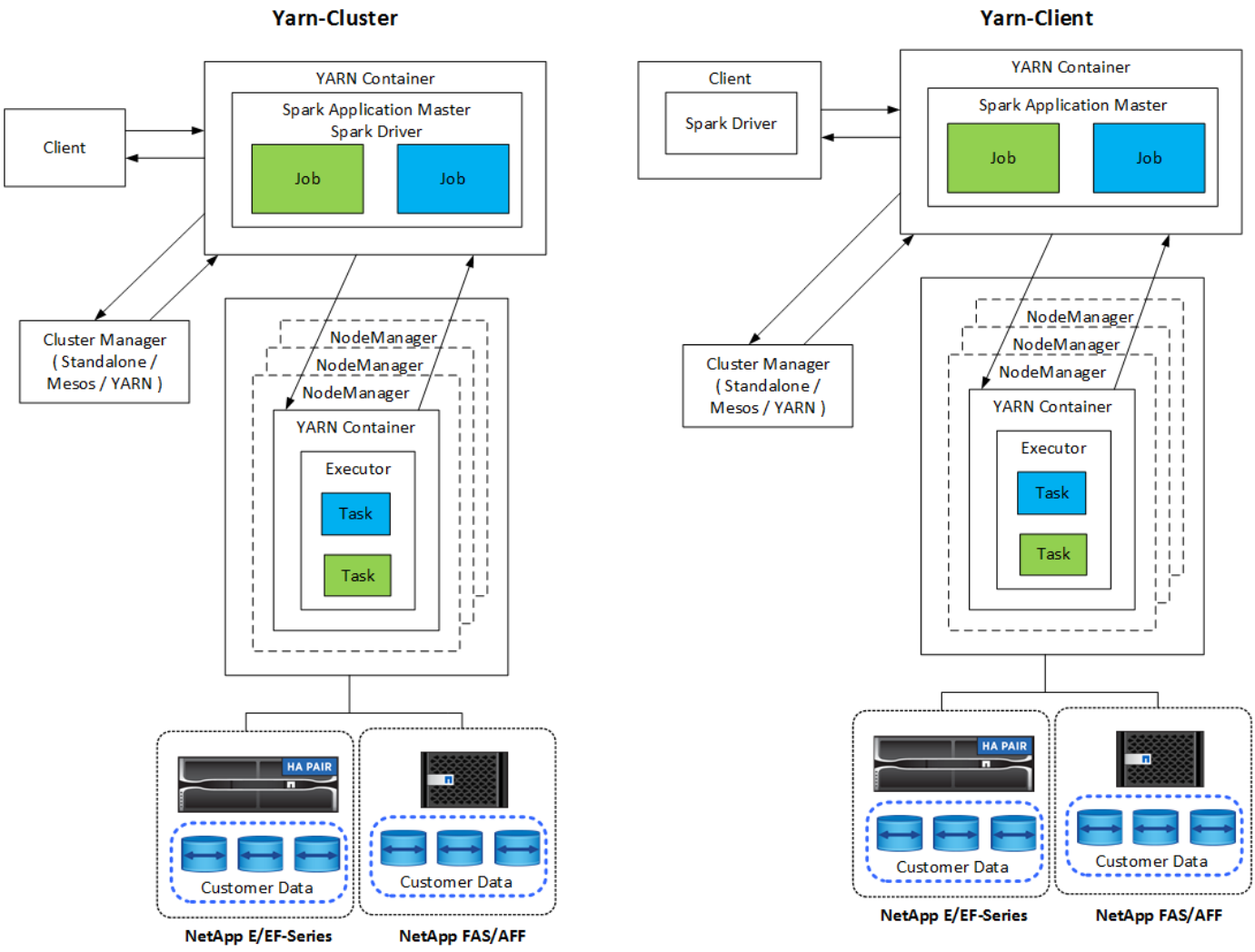
- 数据科学家需要灵活地使用自己选择的工具和库。
- 数据工程师需要了解数据的流动方式及其所在位置。
- DevOps工程师需要使用工具将新的AI和ML应用程序集成到其CI和CD管道中。
- 云管理员和架构师必须能够设置和管理混合云资源。
- 业务用户希望能够访问分析、AI、ML和DL应用程序。

在本技术报告中、我们将介绍NetApp AFF 、E系列、StorageGRID 、NFS直接访问、Apache Spark、 Horovod和Keras可以帮助这些角色为业务创造价值。

解决方案技术

Apache Spark是一种常见的编程框架、用于编写直接与Hadoop分布式文件系统(HDFS)配合使用的Hadoop应用程序。SPARK可随时投入生产、支持流式数据处理、并且速度比MapReduce更快。Spark具有可配置的内存数据缓存功能、可实现高效迭代、而Spark shell可通过交互方式学习和探索数据。借助Spark、您可以在Python、Scala或Java中创建应用程序。激发应用程序由一个或多个具有一个或多个任务的作业组成。

每个Spark应用程序都有一个Spark驱动程序。在yar-Client模式下、驱动程序在本地客户端上运行。在yar-Cluster模式下、驱动程序在应用程序主节点上的集群中运行。在集群模式下、即使客户端断开连接、应用程序也会继续运行。



集群管理器有三种：

- *独立。*此管理器是Spark的一部分、可轻松设置集群。
- * Apache Mesos.*这是一个通用集群管理器、也运行MapReduce和其他应用程序。
- * Hadoop yaryar.*这是Hadoop 3中的资源管理器。

弹性分布式数据集(RDD)是Spark的主要组件。RDD会重新创建存储在集群内存中的数据中的丢失和缺失数据、并存储来自文件或通过编程方式创建的初始数据。可以使用文件、内存中的数据或其他RDD创建RDD。SPARK编程执行两项操作：转型和操作。转型将基于现有RDD创建新的RDD。操作将从RDD返回一个值。

转换和操作也适用于Spark数据集和DataFrame。数据集是一组分布式数据、可提供RDD的优势(强类型、使用lambda函数)以及Spark SQL优化的执行引擎的优势。可以使用JVM对象构建数据集、然后使用功能转换(map、flatMap、filter等)进行操作。DataFrame是按命名列组织的数据集。它在概念上相当于关系数据库中的表或R/Python中的数据帧。DataFrame可以从多种源构建、例如结构化数据文件、Hive/HBase中的表、内部或云中的外部数据库或现有RDD。

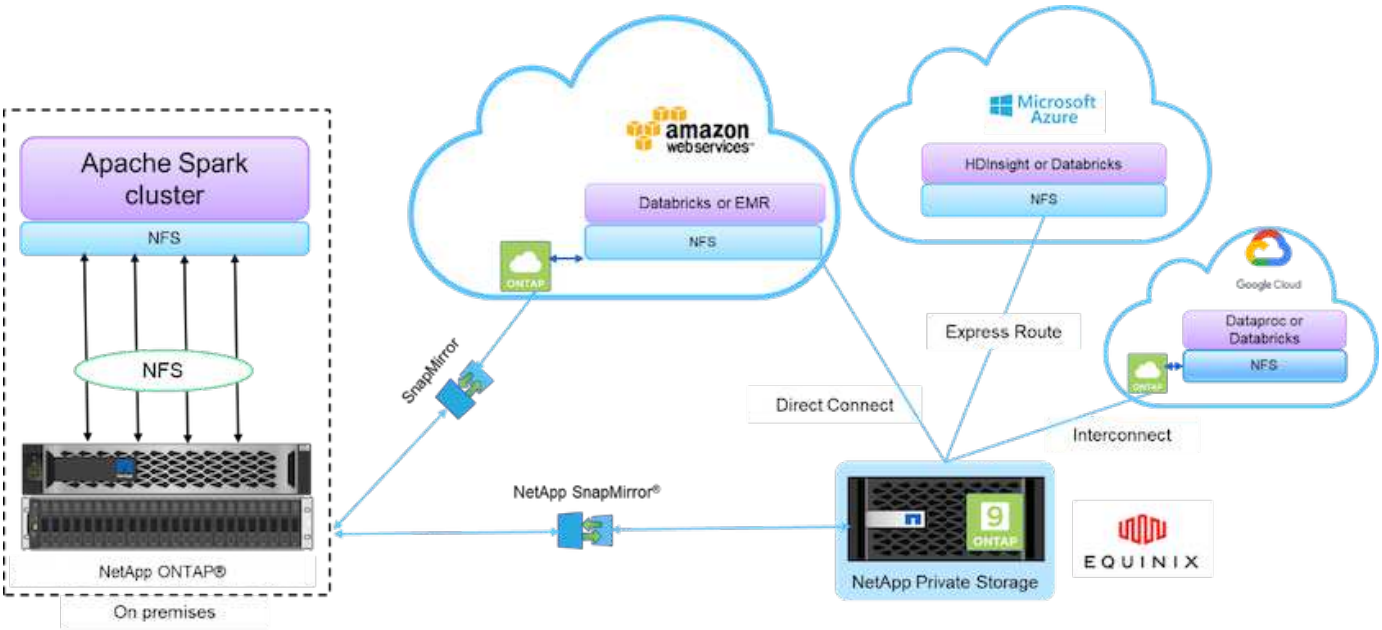
Spark应用程序包括一个或多个Spark作业。作业在执行器中运行任务、而执行器在YARN容器中运行。每个执行

者在一个容器中运行、执行者在应用程序的整个生命周期内都存在。执行者在应用程序启动后得到修复、而yarn不会调整已分配的容器的大小。执行者可以同时对内存数据运行任务。

NetApp Spark解决方案概述

NetApp拥有三种存储产品组合：FAS/AFF、E系列和Cloud Volumes ONTAP。我们已通过Apache Spark验证了适用于Hadoop解决方案的AFF 和采用ONTAP 的E系列存储系统。

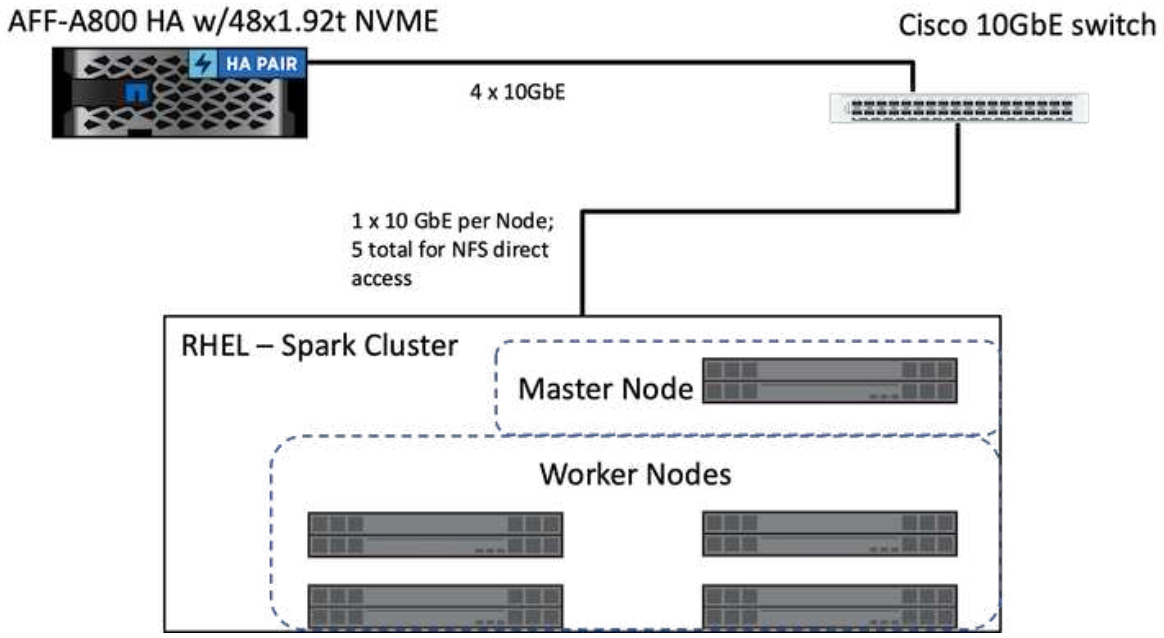
由 NetApp 提供支持的 Data Fabric 集成了数据管理服务和应用程序（组件），用于实现数据访问，控制，保护和安全性，如下图所示。



上图中的组件包括：

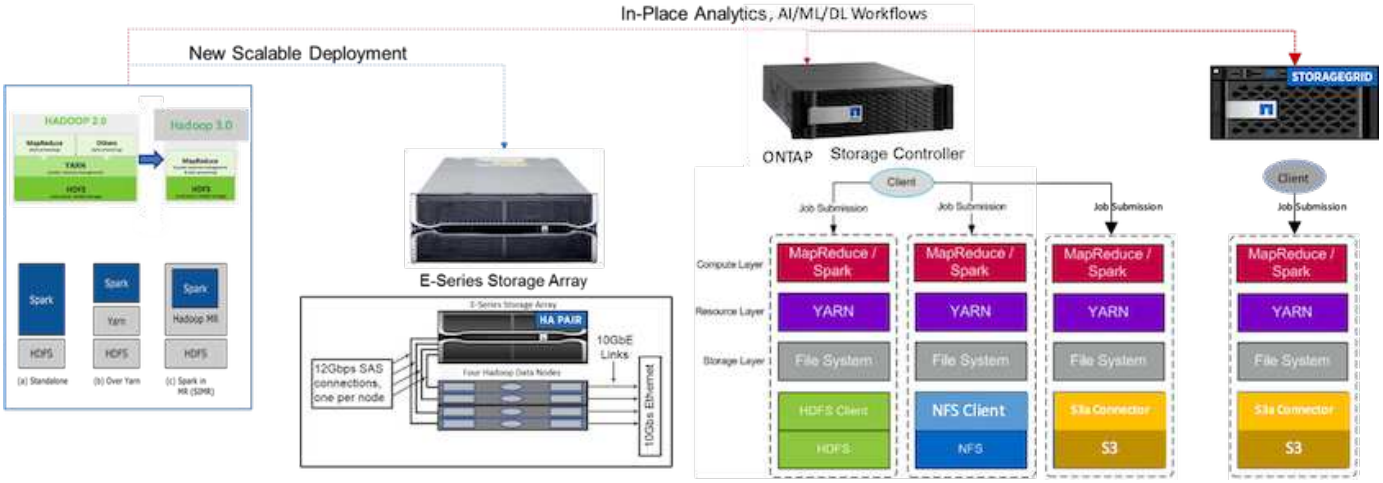
- * NetApp NFS 直接访问。 * 为最新的 Hadoop 和 Spark 集群提供对 NetApp NFS 卷的直接访问，而无需额外的软件或驱动程序要求。
- * NetApp Cloud Volumes ONTAP 和云卷服务。 * 基于在 Microsoft Azure 云服务的 Amazon Web Services （AWS ）或 Azure NetApp Files （ANF ）中运行的 ONTAP 的软件定义的连接存储。
- * NetApp SnapMirror技术。*可在内部部署和ONTAP 云或NPS实例之间提供数据保护功能。
- * 云服务提供商。 * 这些提供商包括 AWS ， Microsoft Azure ， Google Cloud 和 IBM Cloud 。
- * PaaaS 。 * 基于云的分析服务，例如 AWS 中的 Amazon Elastic MapReduce （EMR ）和 Databricks 以及 Microsoft Azure HDInsight 和 Azure Databricks 。

下图展示了采用NetApp存储的Spark解决方案。

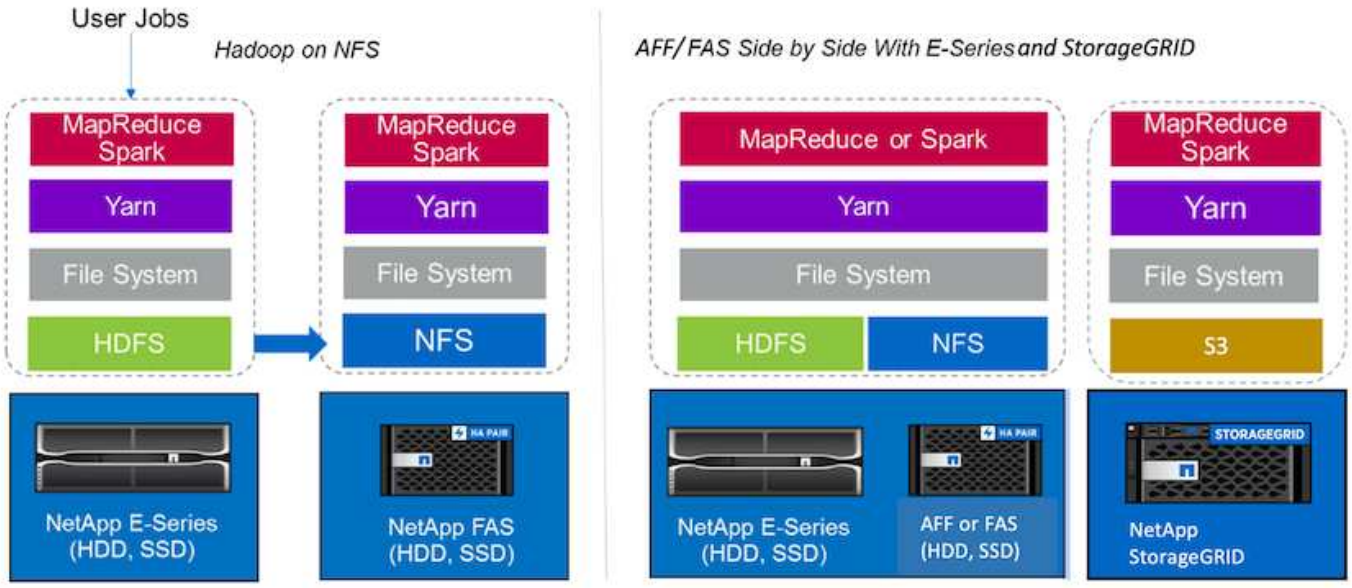


ONTAP Spark解决方案 使用NetApp NFS直接访问协议进行原位分析、并通过访问现有生产数据来访问AI、ML和DL工作流。可供Hadoop节点使用的生产数据会导出、以执行原位分析和AI、ML和DL作业。您可以通过NetApp NFS直接访问或不使用它访问要在Hadoop节点中处理的数据。在具有独立或`yarn`集群管理器的Spark中、您可以使用`配置NFS卷<file:///<target_volume>`。我们验证了使用不同数据集的三个用例。有关这些验证的详细信息、请参见"测试结果"一节。 (Xref)

下图展示了NetApp Apache Spark或Hadoop存储定位。



我们确定了E系列Spark解决方案 、AFF/FAS ONTAP Spark解决方案 和StorageGRID Spark解决方案 的独特功能、并执行了详细的验证和测试。根据我们的观察结果、NetApp建议将E系列解决方案 用于全新安装和全新可扩展部署、并将AFF/FAS解决方案 用于使用现有NFS数据的原位分析、AI、ML和DL工作负载、并在需要对象存储时将StorageGRID 用于AI、ML和DL和现代数据分析。



数据湖是原生 形式的大型数据集的存储库、可用于分析、AI、ML和DL作业。我们为E系列、AFF/FAS和StorageGRID SG6060 Spark解决方案构建了数据湖存储库。E系列系统提供对Hadoop Spark集群的HDFS访问、而现有生产数据则通过NFS直接访问协议访问Hadoop集群。对于驻留在对象存储中的数据集、NetApp StorageGRID 可提供S3和S3a安全访问。

使用情形摘要

此页面介绍了可使用此解决方案 的不同区域。

流式传输数据

Apache Spark可以处理流式数据、用于流式提取、转换和加载(ETL)流程；数据丰富；触发事件检测；以及复杂会话分析：

- *流式ETL*数据在被推入数据存储库之前会持续清理和聚合。Netflix使用Kafka和Spark流式传输构建实时在线电影建议和数据监控解决方案 、每天可以处理来自不同数据源的数十亿个事件。但是、用于批处理的传统ETL的处理方式有所不同。首先读取此数据、然后将其转换为数据库格式、然后再写入数据库。
- 数据丰富。 Spark流可利用静态数据丰富实时数据、实现更实时的数据分析。例如、在线广告公司可以根据客户行为信息提供个性化的有针对性的广告。
- *触发事件检测。*利用Spark流、您可以检测并快速响应可能指示潜在严重问题的异常行为。例如、金融机构使用触发器检测和停止欺诈交易、医院使用触发器检测患者生命迹象中检测到的危险健康变化。
- 会话分析复杂。 Spark流式传输会在登录到网站或应用程序后收集用户活动等事件、然后对这些事件进行分组和分析。例如、Netflix使用此功能提供实时电影建议。

有关流式数据配置、Confluent Kafka验证和性能测试的更多信息、请参见 "TR-4912：《采用 NetApp 的 Confluent Kafka 分层存储最佳实践指南》"。

机器学习

Spark集成框架可帮助您使用机器学习库(MLlib)对数据集重复运行查询。MLlib用于集群、分类和维度缩减等领域、用于某些常见的大数据功能、例如预测性智能、用于营销的客户细分以及情感分析。MLlib用于网络安全领域、用于实时检查数据包是否存在恶意活动迹象。它可以帮助安全提供商了解新威胁、在实时保护客户端的同时保持领先的黑客地位。

深度学习

TensorFlow是一个广泛应用于整个行业的深度学习框架。TensorFlow支持在CPU或GPU集群上进行分布式培训。通过这种分布式培训、用户可以在包含大量深层层的大量数据上运行此培训。

直到最近、如果我们要将TensorFlow与Apache Spark结合使用、我们需要在PySpark中对TensorFlow执行所有必要的ETL、然后将数据写入中间存储。然后、这些数据将加载到TensorFlow集群中、用于实际的培训过程。此工作流要求用户维护两个不同的集群、一个用于ETL、一个用于TensorFlow的分布式培训。运行和维护多个集群通常既繁琐又耗时。

早期版本的Spark中的DataFrame和RDD不适合深度学习、因为随机访问受限。在采用项目"氢"的Spark 3.0中、增加了对深度学习框架的原生 支持。此方法允许在Spark集群上进行非基于MapReduce的计划。

交互式分析

Apache Spark速度非常快、无需使用包括SQL、R和Python在内的其他开发语言进行采样、即可执行探索性查询。SPARK使用可视化工具处理复杂数据并以交互方式将其可视化。利用结构化流技术激发用户对Web分析中的实时数据执行交互式查询、使您能够对Web访客的当前会话运行交互式查询。

建议系统

多年来、推荐系统为我们的生活带来了巨大的变化、因为企业和消费者已经对在线购物、在线娱乐和许多其他行业的巨大变化做出了响应。事实上、这些系统是人工智能在生产领域最明显的成功案例之一。在许多实际使用情形中、推荐系统与对话式AI或与NLP后端交互的聊天机器人相结合、以获取相关信息并生成有用的推断。

如今、许多零售商都在采用新的业务模式、例如在线购买和在店内取货、轮式取件、自助结账、扫描即用等。在COVID-19大流行病期间、这些模式变得更加突出、让消费者购物更加安全、更方便。AI对于这些不断增长的数字趋势至关重要、这些趋势受消费者行为的影响、反之亦然。为了满足消费者不断增长的需求、增强客户体验、提高运营效率并增加收入、NetApp帮助其企业客户和企业使用机器学习和深度学习算法设计更快、更准确的推荐系统。

提供建议时、可以使用多种常见的方法、包括协作筛选、基于内容的系统、深度学习建议模型(DLRM)和混合技术。之前、客户使用PySpark实施协作式筛选来创建建议系统。SPARK MLlib可实施最少交替方形(ALS)来进行协作筛选、这是DLRM兴起之前企业中非常流行的一种算法。

自然语言处理

通过自然语言处理(NLP)实现的对话AI是AI的分支、可帮助计算机与人类进行通信。NLP在从智能助手和聊天机器人到Google搜索和预测性文本的每个行业垂直市场和许多用例中都很普遍。根据 A "Gartner" 预测、到2022年、70%的人将每天与对话式AI平台进行交互。要在人与机器之间进行高质量的对话、响应必须快速、智能且自然。

客户需要大量数据来处理和训练其NLP和自动语音识别(Automatic Speech Recognition、As1)模式。他们还需要跨边缘、核心和云移动数据、并且需要在数毫秒内执行推理的能力、以便与人类建立自然的通信。NetApp AI和Apache Spark是计算、存储、数据处理、模型训练、微调、 和部署。

情感分析是NLP中的一个研究领域、其中从文本中提取积极、负面或中立的情绪。情感分析有多种使用情形、从确定支持中心员工在与调用方对话时的表现到提供适当的自动聊天机器人响应。它还用于根据公司代表与受众在季度收益电话会议上的互动情况预测公司的股票价格。此外、情感分析可用于确定客户对品牌提供的产品、服务或支持的看法。

我们使用了 "激发NLP" 库自 "John Snow Labs" 从Transformer (Bert)型号加载经过预先训练的管道和双向编码器表示、包括 "金融新闻情感" 和 "完成"、大规模执行令牌化、命名实体识别、模型训练、拟合和情感分

析。SPARK NLP是生产中唯一一个开源NLP库、可提供最先进的互感器、例如Bert、B俊 尔特、Electra、XNet、DistillBeert、 Roberta、DeBerta、XLM- Roberta、Longformerk、ELMOA、 通用句子编码器、Google t5、MarianMT和GPt2。该库不仅适用于Python和R、还适用于通过本机扩展Apache Spark实现规模化的JVM生态系统(Java、Scala和Kotlin)。

主要的**AI**、**ML**和**DL**用例和架构

主要的AI、ML和DL用例和方法可分为以下几节：

激发**NLP**管道和**TensorFlow**分布式推理

以下列表列出了数据科学界在不同开发级别下采用的最受欢迎的开源NLP库：

- "自然语言工具包(NLTK)"。适用于所有NLP技术的完整工具包。自21世纪初以来、该系统一直保持不变。
- "文本Blob"。基于NLTK和模式构建的简单易用的NLP工具Python API。
- "斯坦福核心NLP"。由斯坦福NLP集团开发的Java中的NLP服务和软件包。
- "Gensim"。人类主题建模最初是作为一组Python脚本在捷克数字数学库项目中推出的。
- "空间"。采用Python和Cython的端到端工业NLP工作流、并为互感器提供GPU加速。
- "快速文本"。一个免费的轻型开源NLP库、用于由Facebook的AI Research (Ffair)实验室创建的字词学习嵌入和句子分类。

SPARK NLP是一个统一的解决方案 、可满足所有NLP任务和要求、可为实际生产用例提供可扩展、高性能和高准确性的NLP驱动软件。它利用传输学习、在研究领域和跨行业实施最新的一流算法和模型。由于Spark不能全面支持上述库、因此、Spark NLP是基于构建的 "激发ML" 利用Spark的通用内存分布式数据处理引擎作为任务关键型生产工作流的企业级NLP库。其标注器利用基于规则的算法、机器学习和TensorFlow为深度学习实施提供支持。其中包括常见的NLP任务、包括但不限于令牌化、lemization、stemming、部分语音标记、命名实体识别、拼写检查和情感分析。

Transformers (Bert)提供的双向编码器表示是一种基于转换器的机器学习技术、适用于NLP。它推广了预训练和微调的概念。Bert中的转换器架构源自机器翻译、与基于神经网络(RNN)的经常性语言模型相比、该模型更好地模拟长期依赖关系。它还引入了屏蔽语言建模(Masked Language Modeling、MLM)任务、其中随机屏蔽了所有令牌的15%、模型对其进行预测、从而实现了真正的双向性。

由于该领域的专业语言和缺少标记数据、财务情绪分析具有挑战性。 "完成"一种基于经过预先培训的Bert的语言模式、该语言模式已经过域调整 "Reuters TRC2"一种财务资料、并使用标记的数据("金融租赁银行")进行财务状况分类。研究人员从新闻文章中提取了4、500句话、并使用了财务术语。然后、16名具有财务背景的专家和硕士学生将这些句子标记为肯定、中立和否定。我们构建了一个端到端Spark工作流、用于分析2016年至2020年排名前10位的纳斯达克公司收益电话记录的情绪、使用FinBeert和另外两个经过预先培训的管道("金融新闻情感分析"， "说明文档DL")。

适用于Spark NLP的底层深度学习引擎是TensorFlow、它是一个端到端的开源机器学习平台、可轻松构建模型、随时随地进行强大的ML生产以及进行强大的研究试验。因此、在Spark `Yarn集群`模式下执行管道时、我们基本上是在集群上挂载的网络连接存储中运行分布式TensorFlow、并在一个主节点和多个辅助节点之间并行处理数据和模型。

Horovod分布式培训

使用TeraGen、TeraSort、TeraValidate和DFSIO (读写)执行与MapReduce相关的性能的核心Hadoop验证。TeraGen和TeraSort验证结果显示在中 "TR-3969：《适用于Hadoop的NetApp解决方案》" 对于E系列和AFF 的"存储分层"一节(xref)。

根据客户的要求、我们认为使用Spark进行分布式培训是各种使用情形中最重要的一个。在本文档中、我们使用了 "Hovorod on Spark" 使用NetApp全闪存FAS (AFF)存储控制器、Azure NetApp Files 和StorageGRID 验证NetApp内部部署、云原生和混合云解决方案的Spark性能。

Horovod on Spark软件包为Horovod提供了一个方便的包装、使在Spark集群中运行分布式训练工作负载变得简单、从而实现了一个紧密的模型设计环路、其中数据处理、模型训练和模型评估都在训练和推理数据所在的Spark中完成。

在Spark上运行Horovod有两种API：一种是高级估算器API、另一种是运行API。虽然这两种方法都使用相同的底层机制来启动Horovod on Spark执行程序、但Estimator API可对数据处理、模型训练循环、模型检查点、指标收集和分布式培训进行抽象化。我们使用Horovod Spark Estimators、TensorFlow和Keras基于进行端到端数据准备和分布式培训工作流 "Kagle Rossmann商店销售人员" 竞争。

可以在部分中找到脚本`keras_sock_horovod_rossmann_estimator.py` "适用于每个主要用例的Python脚本。" 它包含三个部分：

- 第一部分对Kagger提供并由社区收集的一组初始CSV文件执行各种数据预处理步骤。输入数据将分为一个训练集、其中包含`Validation`子集和一个测试数据集。
- 第二部分定义了具有对数Sigma激活功能和Adam优化器的Keras深度神经网络(DNN)模型、并使用Horovod on Spark对该模型进行分布式培训。
- 第三部分使用最佳模型对测试数据集执行预测、以最大程度地减少验证集的整体平均绝对错误。然后、它将创建一个输出CSV文件。

请参见一节 ""机器学习"" 查看各种运行时比较结果。

利用**Keras**进行**CTR**预测的多员工深度学习

随着 ML 平台和应用程序的最新发展，我们现在非常关注大规模学习。点击率（ CTR ）是指每 100 次在线广告曝光的平均点击次数（以百分比表示）。它已广泛用作各种行业垂直市场和用例的关键指标，包括数字营销，零售，电子商务和服务提供商。请参见我们的 "TR-4904： Azure 中的分布式培训—点击率预测" 有关CTR应用程序以及使用Kubernetes实施的端到端Cloud AI工作流、分布式数据ETL以及使用dask和CUDA ML的模型培训的更多详细信息。

在本技术报告中、我们使用了的变体 "Trigeo Terabyte单击Logs dataset" (请参见TR-4904)、针对使用Keras的多员工分布式深度学习、使用深度和跨网(深度和跨网)模型构建Spark工作流、将其在日志丢失错误功能方面的性能与基线Spark ML物流回归模型进行比较。DCN可以 高效地捕获有界限的有效功能交互、学习高度非线性的交互、无需手动执行功能工程或全面搜索、并且计算成本较低。

网络级推荐系统的数据大多是离散的、分类的、导致功能空间庞大而稀疏、这对功能探索来说是一项挑战。这样、大多数大型系统就只能使用诸如物流回归等线性模型。但是、确定常见的预测功能并同时探索未知或罕见的交叉功能是做出良好预测的关键。线性模型简单、可解释且易于扩展、但其表达能力有限。

另一方面、交叉特征在提高模型的显示能力方面表现出了显著的意义。遗憾的是、它通常需要手动执行功能工程或全面搜索才能识别此类功能。通常很难将功能交互概括为不可见。使用像DCN这样 的交叉神经网络、可以通过自动明确应用功能交叉来避免特定于任务的功能工程。跨网络由多个层组成、其中最大程度的交互可通过层深度来确定。每个层都会根据现有层生成较高顺序的交互、并保留先前层的交互。

深度神经网络(DNN)有望捕获功能之间非常复杂的交互。但是、与DCN相比、它需要的参数数量几乎要多一个数量级、无法明确形成交叉功能、并且可能无法高效地了解某些类型的功能交互。跨网络可节省内存并易于实施。将交叉组件和DNN组件联合训练在一起、有效地捕获预测性功能交互、并在Criteo CTR数据集上提供一流的性能。

一个型号的DCN-首先是一个嵌入层和堆栈层、然后是一个跨网络和一个并行的深度网络。然后是最后一个组合层、该层将两个网络的输出相结合。您的输入数据可以是具有稀疏和密集功能的向量。在Spark中、这两者都是如此 "毫升" 和 "mllib" 库的类型为`SparseVector`。因此、用户必须区分这两者、并在调用各自的功能和方法时要小心谨慎。在CTR预测等网络级建议系统中、输入主要是分类功能、例如`'国家/地区=美国'`。此类功能通常编码为单热向量、例如、`、1、0、…'。使用`SparseVector`的单热编码(OHEE)在处理实际数据集时非常有用、因为这些数据集的词义不断变化且不断增长。我们在中修改了示例 "DeepCTR" 要处理大型卷标、请在我们的DCN"嵌入和堆栈"层中创建嵌入载体。

。 "Criteo显示广告数据集" 预测广告点击率。它具有13个整数功能和26个分类功能、其中每个类别的基数都很高。对于此数据集、由于输入大小较大、光泽度提高了0.001实际上是一项显著的改进。如果对大型用户群的预测准确性稍作提高、可能会导致公司收入大幅增加。该数据集包含7天内的11 GB用户日志、相当于大约4、100万条记录。我们使用Spark `dataFrame.Random拆 分()函数`随机拆分数据进行训练(80%)、交叉验证(10%)、其余10%用于测试。

在与Keras的TensorFlow上实施了DCN。在使用DCN-CN-CN-CN-CN-CN-CN-CN-A实施模型培训过程中、主要包含四个

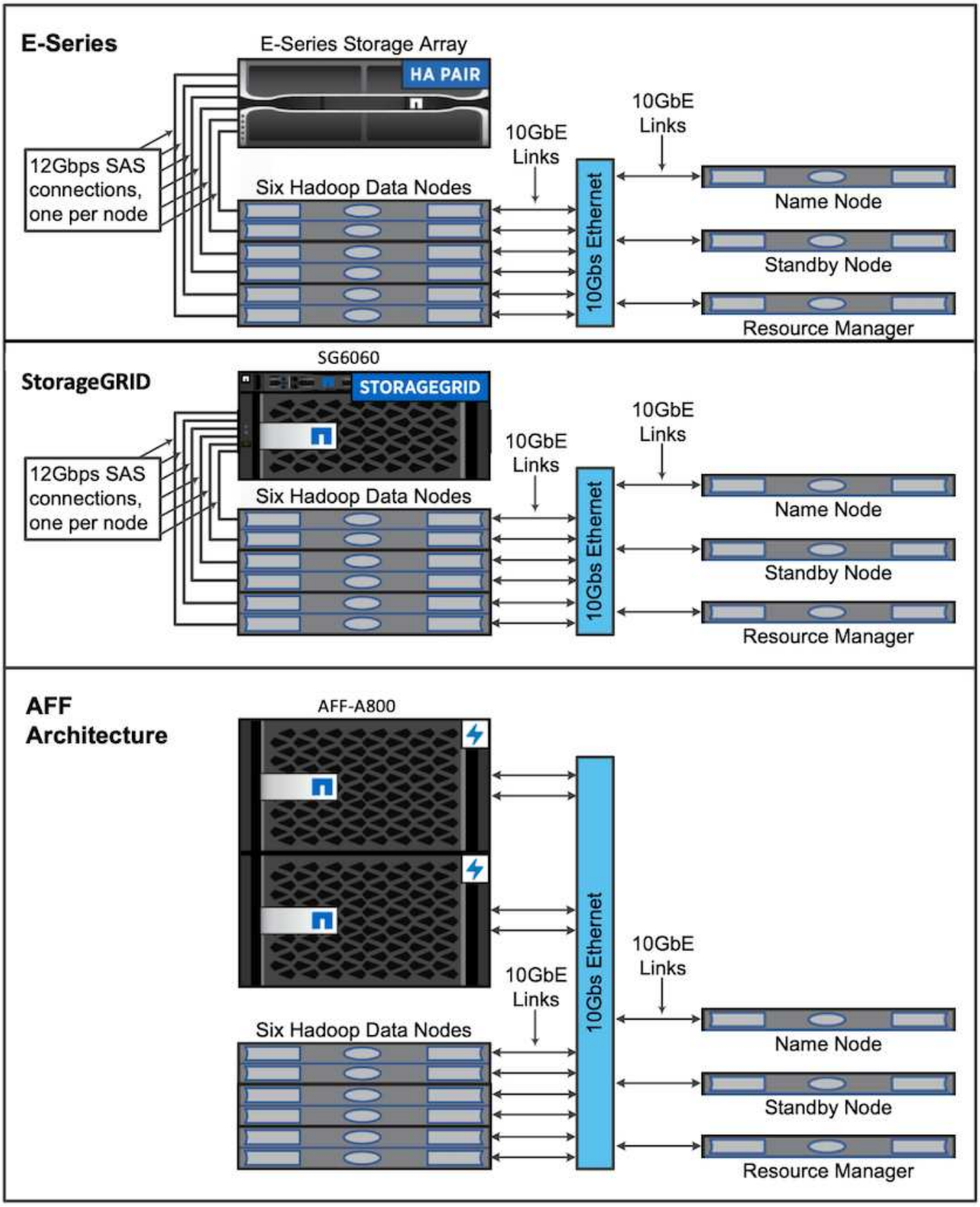
- *数据处理和嵌入。*实际价值的功能通过应用日志转换进行标准化。为了获得明确的功能、我们将这些功能嵌入到维度6×(类别基数) 1/4的密集向量中。将所有嵌入项串联后、将生成维度1026的向量。
- *优化。*我们使用了Adam优化器进行了迷你批处理随机优化。批处理大小设置为512。对深度网络应用了批处理标准化、梯度线夹规范设置为100。
- *规范化。*由于未发现L2规范化或降级有效、我们使用了提前停止的方法。
- *超参数。*我们根据对隐藏层数、隐藏层大小、初始学习速率和跨层数的网格搜索来报告结果。隐藏层的数量从2到5不等、隐藏层大小从32到1024不等。对于DCN、跨层数量为1到6。初始学习速率从0.0001调整为0.001、增量为0.0001。所有实验都在训练步骤150、000时进行了早期停止、超过此步骤后、开始发生过度安装。

除了使用了DCN之外、我们还测试了其他常见的深度学习模型来进行CTR预测、其中包括 "DeepFM"， "xDeepFM"， "自动内置"，和 "DCNv2"。

用于验证的架构

在此验证中、我们使用了四个辅助节点和一个具有AF-A800 HA对的主节点。所有集群成员均通过10GbE网络交换机进行连接。

在此NetApp Spark解决方案 验证中、我们使用了三种不同的存储控制器：E5760、E5724和AFF-A800。E系列存储控制器通过12 Gbps SAS连接连接到五个数据节点。AFF HA对存储控制器通过与Hadoop工作节点的10GbE连接提供导出的NFS卷。Hadoop集群成员通过E系列、AFF 和StorageGRID Hadoop解决方案中的10GbE连接进行连接。



测试结果

我们使用TeraGen基准测试工具中的TeraSort和TeraValidate脚本测量E5760、E5724

和AFF-A800配置下的Spark性能验证。此外、还测试了三个主要用例：SPARK NLP管道和TensorFlow分布式培训、Horovod分布式培训以及使用Keras通过DeepFM进行CTR预测的多员工深度学习。

对于E系列和StorageGRID 验证、我们使用Hadoop复制因子2。对于AFF 验证、我们仅使用一个数据源。

下表列出了用于Spark性能验证的硬件配置。

Type	Hadoop工作节点	驱动器类型	每个节点的驱动器数	存储控制器
SG6060	4.	（SAS）。	12	一个高可用性(HA)对
E5760	4.	（SAS）。	60	单个HA对
E5724	4.	（SAS）。	24	单个HA对
AFF800	4.	SSD	6.	单个HA对

下表列出了软件要求。

软件	version
RHEL	7.9
OpenJDK运行时环境	1.8.0
OpenJDK 64位服务器VM	25.302
Git	2.24.1
GCC或G++	11.2.1
激发	3.2.1
PySpark	3.1.2
SparkNLP	3.4.2
TensorFlow	2.9.0
克罗斯	2.9.0
Horovod	0.24.3.

财务状况分析

我们发布了 "TR-4910：《利用 NetApp AI 进行客户沟通时的情感分析》"、其中使用构建了一个端到端对话AI管道 "NetApp DataOps 工具包"、AFF 存储和NVIDIA DGX系统。该管道利用DataOps工具包执行批量音频信号处理、自动语音识别(Automatic Speech Recognition、As1)、传输学习和情感分析。 "NVIDIA Riva SDK"和 "TAO框架"。将情感分析用例扩展到金融服务行业、我们构建了SparkNLP工作流、加载了三个Bert模型来执行各种NLP任务、例如、命名实体识别、并在纳斯达克排名前10位的公司季度收益电话会议中获得了句子级的感受。

以下脚本`sentiment_analysis _sacp。py`使用FinBet模型处理HDFS中的脚本、并产生积极、中立和负面的情绪计数、如下表所示：

```
-bash-4.2$ time ~/anaconda3/bin/spark-submit
--packages com.johnsnowlabs.nlp:spark-nlp_2.12:3.4.3
--master yarn
--executor-memory 5g
--executor-cores 1
--num-executors 160
--conf spark.driver.extraJavaOptions="-Xss10m -XX:MaxPermSize=1024M"
--conf spark.executor.extraJavaOptions="-Xss10m -XX:MaxPermSize=512M"
/sparkusecase/tr-4570-nlp/sentiment_analysis_spark.py
hdfs:///data1/Transcripts/
> ./sentiment_analysis_hdfs.log 2>&1
real13m14.300s
user557m11.319s
sys4m47.676s
```

下表列出了2016年至2020年纳斯达克排名前10位的公司的收益情况、句子级别的情绪分析。

情感的数量和百分比	所有**10**家公司	**AAPL**	**AMD**	不支持	**CSCO**	**GOOGL**	**INTC**	**MSFT**	**NVDA**
正数	7447	1567	743	290	682	826	824	904	417
空值	64067	6856	7596	5086	6650	5914	6099	5715	6189
负计数	1787年	253.	213.	84.	189.	97	282.	202	89.
未分类计数	196年	0	0	76.	0	0	0	1.	0
(总数)	73497	8676	8552	5536	7521	6837	7205	6822	6695

就百分比而言、CEO和CFO所说的大多数句子都是事实、因此具有中立的情绪。在收益调查期间、分析师会提出可能会表达积极或负面情绪的问题。值得进一步量化调查负面或正面情绪对交易当天或第二天的股票价格有何影响。

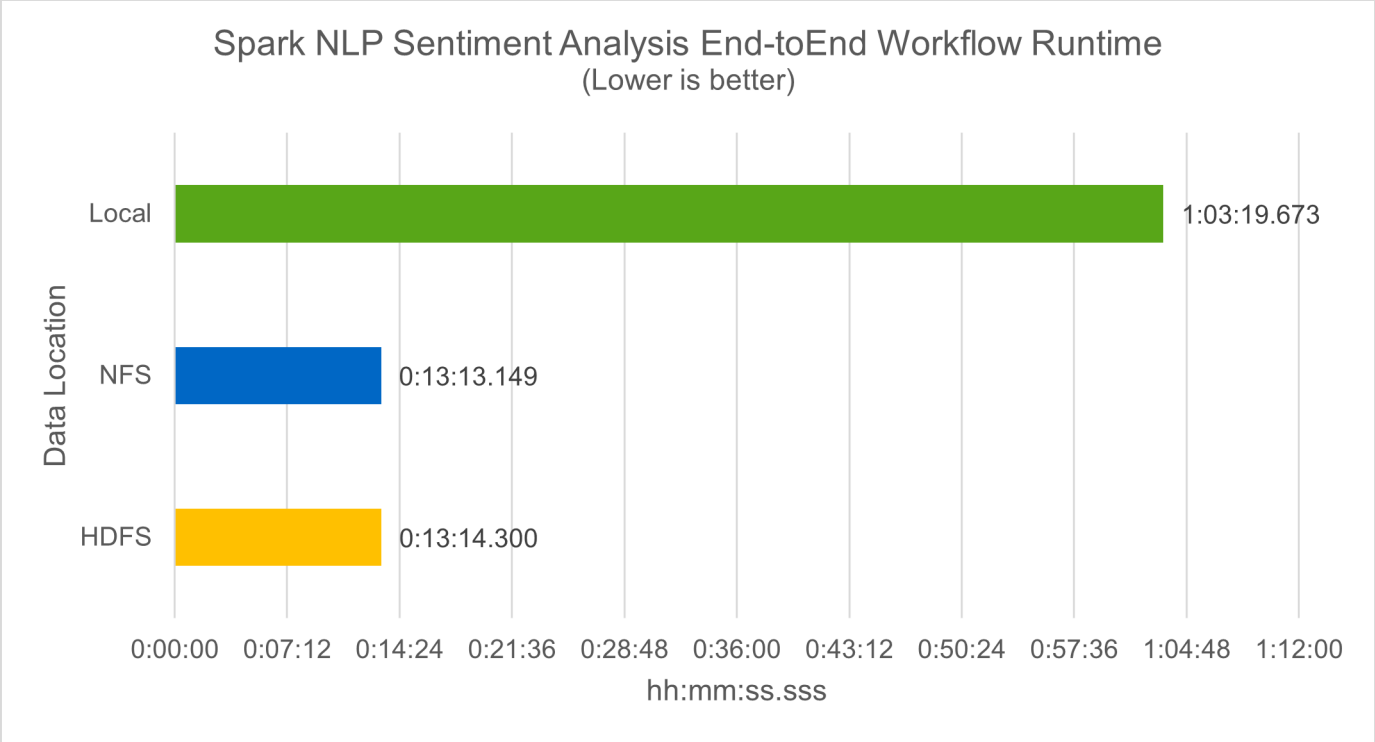
下表列出了纳斯达克排名前10位的公司的句子级别情感分析、以百分比表示。

情绪百分比	所有**10**家公司	**AAPL**	**AMD**	不支持	**CSCO**	**GOOGL**	**INTC**	**MSFT**	**NVDA**
肯定	10.13%	18.06%	8.69%	5.24%	9.07%	12.08%	11.44%	13.25%	6.23%
中立	87.17%	79.02%	88.82%	91.87%	88.42%	86.50%	84.65%	83.77%	92.44%
否定	2.43%	2.92%	2.49%	1.52%	2.51%	1.42%	3.91%	2.96%	1.33%
未分类	0.27%	0%	0%	1.37%	0%	0%	0%	0.01%	0%

在工作流运行时间方面、我们发现从`本地`模式到HDFS中的分布式环境、有4.78倍的显著提升、而利用NFS则进一步提高了0.14%。

```
-bash-4.2$ time ~/anaconda3/bin/spark-submit
--packages com.johnsnowlabs.nlp:spark-nlp_2.12:3.4.3
--master yarn
--executor-memory 5g
--executor-cores 1
--num-executors 160
--conf spark.driver.extraJavaOptions="-Xss10m -XX:MaxPermSize=1024M"
--conf spark.executor.extraJavaOptions="-Xss10m -XX:MaxPermSize=512M"
/sparkusecase/tr-4570-nlp/sentiment_analysis_spark.py
file:///sparkdemo/sparknlp/Transcripts/
> ./sentiment_analysis_nfs.log 2>&1
real13m13.149s
user537m50.148s
sys4m46.173s
```

如下图所示、数据和模型并行性提高了数据处理和分布式TensorFlow模型推理速度。NFS中的数据位置会使运行时间略有提高、因为工作流瓶颈是下载经过预先训练的模型。如果增加脚本数据集大小、NFS的优势就会更加明显。



分布式培训、提供**Horovod**性能

以下命令在Spark集群中使用一个`m`主节点生成运行时信息和日志文件、该节点包含160个执行器、每个执行器具有一个核心。执行器内存限制为5 GB、以避免内存不足错误。请参见一节 "《适用于每个主要用例的Python脚本》" 有关数据处理、模型训练和模型准确性计算的更多详细信息、请参见`keras_sock_horovod_Rossmann_estimator.py`。

```
(base) [root@n138 horovod]# time spark-submit
--master local
--executor-memory 5g
--executor-cores 1
--num-executors 160
/sparkusecase/horovod/keras_spark_horovod_rossmann_estimator.py
--epochs 10
--data-dir file:///sparkusecase/horovod
--local-submission-csv /tmp/submission_0.csv
--local-checkpoint-file /tmp/checkpoint/
> /tmp/keras_spark_horovod_rossmann_estimator_local. log 2>&1
```

由此产生的运行时间为十个训练时长、如下所示：

```
real43m34.608s
user12m22.057s
sys2m30.127s
```

处理输入数据、训练DNN模型、计算准确性以及生成TensorFlow检查点和CSV文件以获得预测结果需要43分钟以上的时间。我们将培训时间限制为10个、实际上通常设置为100个、以确保模型的准确性。训练时间通常随时间间隔的数量呈线性增长。

接下来、我们会使用集群中的四个工作节点、并在`yarn`模式下使用HDFS中的数据执行同一个脚本：

```
(base) [root@n138 horovod]# time spark-submit
--master yarn
--executor-memory 5g
--executor-cores 1 --num-executors 160
/sparkusecase/horovod/keras_spark_horovod_rossmann_estimator.py
--epochs 10
--data-dir hdfs:///user/hdfs/tr-4570/experiments/horovod
--local-submission-csv /tmp/submission_1.csv
--local-checkpoint-file /tmp/checkpoint/
> /tmp/keras_spark_horovod_rossmann_estimator_yarn.log 2>&1
```

生成的运行时间得到了以下改进：

```
real8m13.728s
user7m48.421s
sys1m26.063s
```

借助Horovod在Spark中的模型和数据并行、我们发现运行时速度比`yarn`和`local`模式加快了5.29倍、并有十个训练时长。下图显示了这一点以及图例`HDFS`和`Local`。如果可以使用GPU、则可以进一步加快底

层TensorFlow DNN模型培训的速度。我们计划执行此测试、并在未来的技术报告中公布测试结果。

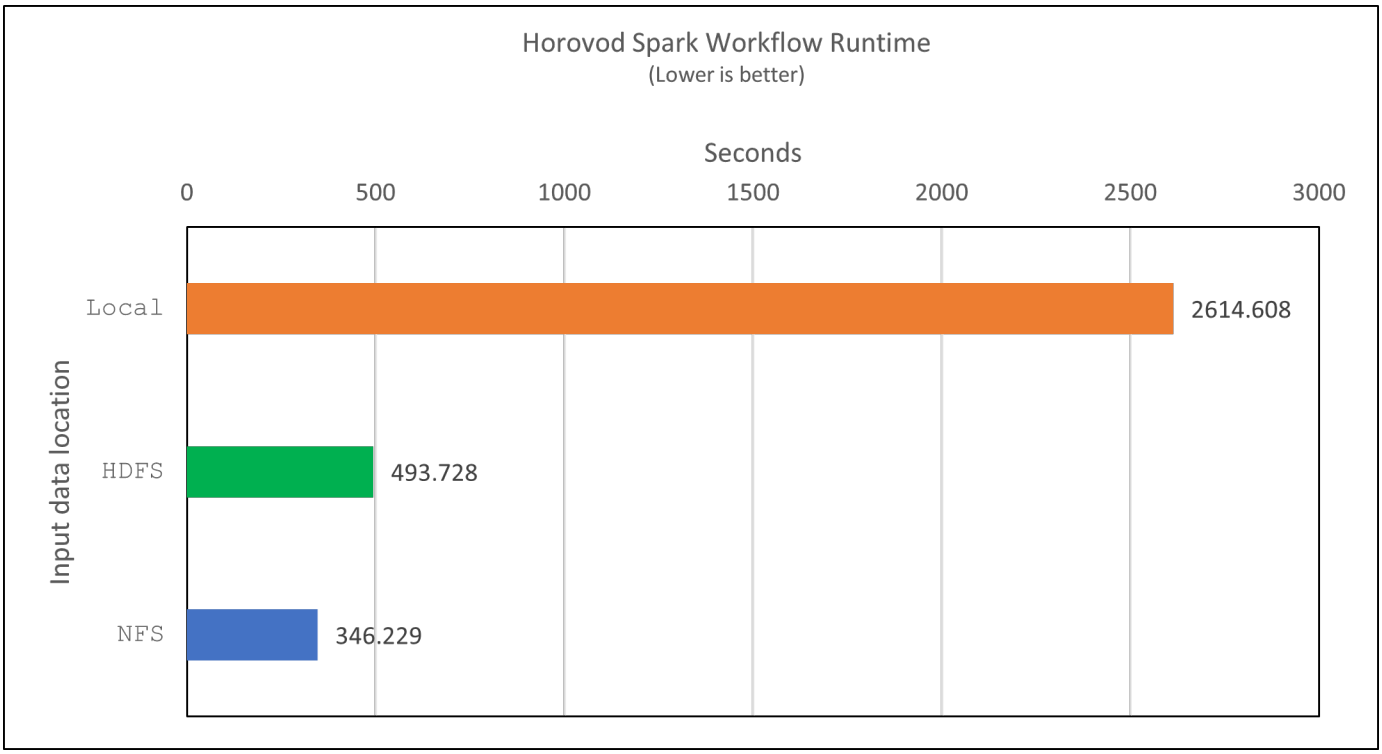
我们的下一个测试将NFS中的输入数据的运行时间与HDFS进行了比较。AFF A800上的NFS卷已挂载在集群的五个节点(一个主节点、四个员工节点)上的`或sparemdemo/horovod`上。我们运行的命令与先前测试类似、其中`-data-dir`参数现在指向NFS挂载：

```
(base) [root@n138 horovod]# time spark-submit
--master yarn
--executor-memory 5g
--executor-cores 1
--num-executors 160
/sparkusecase/horovod/keras_spark_horovod_rossmann_estimator.py
--epochs 10
--data-dir file:///sparkdemo/horovod
--local-submission-csv /tmp/submission_2.csv
--local-checkpoint-file /tmp/checkpoint/
> /tmp/keras_spark_horovod_rossmann_estimator_nfs.log 2>&1
```

使用NFS生成的运行时如下：

```
real 5m46.229s
user 5m35.693s
sys  1m5.615s
```

此外、还实现了1.43倍的加速、如下图所示。因此、在将NetApp全闪存存储连接到集群后、客户可以享受到Horovod Spark工作流快速数据传输和分发的优势、与在单个节点上运行相比、速度加快了7.55倍。



深度学习模型、用于控制器预测性能

对于旨在最大程度地提高CTR的推荐系统、您必须了解用户行为背后的复杂功能交互、这些交互可以从低顺序到高顺序进行数学计算。低顺序和高顺序功能交互对于良好的深度学习模型来说都同样重要、而不是相互影响。深度Factorization Machine (DeepFM)是一种基于面化机器的神经网络、它将面化机器结合在一起、在一个新的神经网络架构中提供建议、并进行深度学习以进行功能学习。

虽然传统的面化机可以模拟成对的功能交互、将其作为功能之间潜在向量的内在产品、并可从理论上捕获高阶信息、但实际上、由于计算和存储复杂性较高、机器学习实践者通常只使用二级功能交互。Google等深度神经网络变体 "宽和高；深模型" 另一方面、通过将线性宽模型和深度模型相结合、可以在混合网络结构中学习复杂的功能交互。

此宽深模型有两个输入、一个用于底层宽模型、另一个用于深度、后者的后半部分仍需要专家级的功能工程、因此、技术在其他领域的推广程度较低。与宽深模型不同、DeepFM可以高效地进行原始功能培训、而无需任何功能工程、因为其宽部分和深部分共享相同的输入和嵌入向量。

首先、我们使用部分中的`run_section_criteo_spark.py`将Criteo `trint.txt`(11GB)文件处理为CSV文件`ctrt_trint.csv`、该文件存储在NFS挂载中`/sparemodem/tr-4570-data` "《适用于每个主要用例的Python脚本》。" 在此脚本中、函数`process_input_file`会执行多种字符串方法来删除选项卡并插入`‘、’`作为分隔符、并将`‘\n ’`作为换行符。请注意、您只需处理原始的`Train .txt`一次、即可将代码块显示为注释。

对于以下不同DL型号的测试、我们使用`ct_Train.csv`作为输入文件。在后续测试运行中、输入的CSV文件会读取到Spark DataFrame中、其架构包含`‘label ’`、整型密集型功能`['I1'、'Ies'、'I3'、…、'I13']`、 和稀疏功能、'c1"、'c2'、'cc3、…、'c26']`。以下`spart-Submit`命令将获取输入CSV、将DeepFM模型分成20%进行交叉验证、并在经过十次训练后选择最佳模型来计算测试集的预测准确性：

```
(base) [root@n138 ~]# time spark-submit --master yarn --executor-memory 5g
--executor-cores 1 --num-executors 160
/sparkusecase/DeepCTR/examples/run_classification_criteo_spark.py --data
-dir file:///sparkdemo/tr-4570-data >
/tmp/run_classification_criteo_spark_local.log 2>&1
```

请注意、由于数据文件`CT_Train.csv`超过11 GB、因此您必须设置一个足够的`spara.driver.maxResult Size`、使其大于数据集大小、以避免出现错误。

```
 spark = SparkSession.builder \
    .master("yarn") \
    .appName("deep_ctr_classification") \
    .config("spark.jars.packages", "io.github.ravwojdyla:spark-schema-
utils_2.12:0.1.0") \
    .config("spark.executor.cores", "1") \
    .config('spark.executor.memory', '5gb') \
    .config('spark.executor.memoryOverhead', '1500') \
    .config('spark.driver.memoryOverhead', '1500') \
    .config("spark.sql.shuffle.partitions", "480") \
    .config("spark.sql.execution.arrow.enabled", "true") \
    .config("spark.driver.maxResultSize", "50gb") \
    .getOrCreate()
```

在上述`SparkSession.Builder`配置中、我们还启用了 "Apache Arrow"、使用`D .parctoandas ()`方法将Spark DataFrame转换为熊猫DataFrame。

```
22/06/17 15:56:21 INFO scheduler.DAGScheduler: Job 2 finished: toPandas at
/sparkusecase/DeepCTR/examples/run_classification_criteo_spark.py:96, took
627.126487 s
Obtained Spark DF and transformed to Pandas DF using Arrow.
```

随机拆分后、训练数据集中的行数超过36M、而测试集中的样本数则超过9M：

```
Training dataset size =  36672493
Testing dataset size =  9168124
```

由于本技术报告重点介绍CPU测试而不使用任何GPU、因此、您必须使用适当的编译器标志构建TensorFlow。此步骤可避免调用任何GPU加速库、并充分利用TensorFlow的高级矢量扩展(Advanced Vector Extension、AVX)和AVX2指令。这些功能专为线性代数计算而设计、例如矢量化添加、前馈或后传播DNN训练中的矩阵乘法。使用256位浮点(FP)注册的AVX2可提供融合乘法添加(FMA)指令、非常适合整数代码和数据类型、从而实现高达2倍的加速。对于FP代码和数据类型、与AVX相比、AVX2实现了8%的加速。

```
2022-06-18 07:19:20.101478: I
tensorflow/core/platform/cpu_feature_guard.cc:151] This TensorFlow binary
is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the
following CPU instructions in performance-critical operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the
appropriate compiler flags.
```

要从源构建TensorFlow、NetApp建议使用 "市场"。对于我们的环境、我们会在shell提示符处执行以下命令来安装`dnF`、`dnf-plugins`和azel。

```
yum install dnf
dnf install 'dnf-command(copr)'
dnf copr enable vbatts/bazel
dnf install bazel5
```

要在构建过程中使用C+17功能、必须启用GCC 5或更高版本、此功能由RHEL和软件收集库(Software Collections Library、SCL)提供。以下命令可在RHEL 7.9集群上安装`devtoolset`和GCC 11.2.1：

```
subscription-manager repos --enable rhel-server-rhscl-7-rpms
yum install devtoolset-11-toolchain
yum install devtoolset-11-gcc-c++
yum update
scl enable devtoolset-11 bash
. /opt/rh/devtoolset-11/enable
```

请注意、最后两个命令会启用`devtoolset-11`、它会使用`/opt/rg/devtoolset-11/root/usr/bin/gcc`(GCC 11.2.1)。此外、请确保您的`git`版本高于1.8.3 (RHEL 7.9随附此版本)。请参见此部分 "文章" 用于将`git`更新到2.24.1。

我们假定您已克隆最新的TensorFlow主报告。然后、使用`workspace`文件创建`workspace`目录、以便使用AVX、AVX2和FMA从源构建TensorFlow。运行`configure`文件并指定正确的Python二进制位置。 "CUDA" 已在测试中禁用、因为我们未使用GPU。将根据您的设置生成`.bazelrc`文件。此外、我们还编辑了该文件并设置`build -def=no_hdfs_support=false`以启用HDFS支持。请参见一节中的`.bazelrc` "《针对每个主要用例的Python脚本》、" 有关设置和标志的完整列表。

```
./configure
bazel build -c opt --copt=-mavx --copt=-mavx2 --copt=-mfma --copt=
-mfpmath=both -k //tensorflow/tools/pip_package:build_pip_package
```

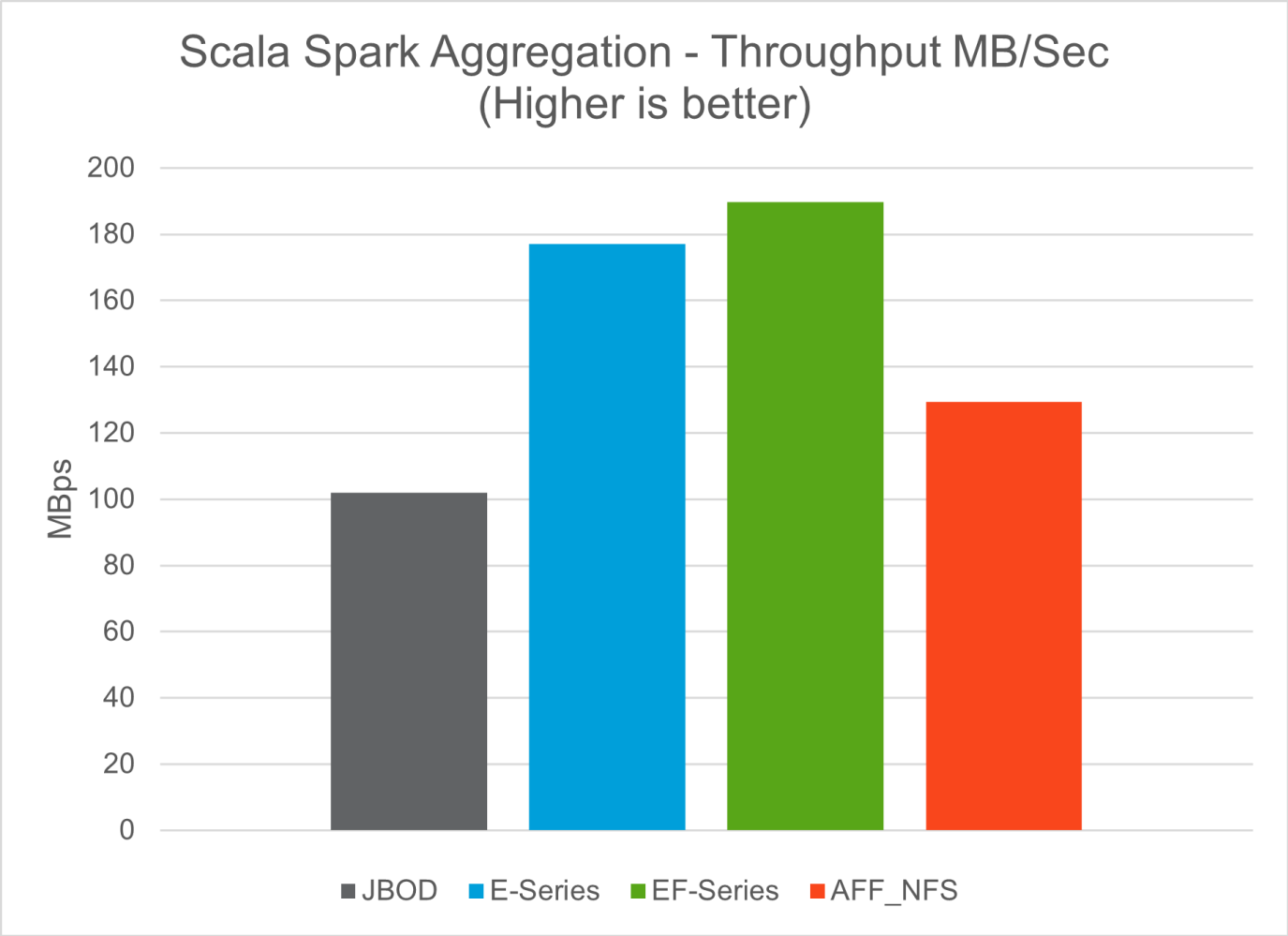
使用正确的标志构建TensorFlow后、运行以下脚本以处理Criteo显示广告数据集、训练DeepFM模型、并根据预测分数计算接收器运行特征曲线(ROC AUC)下的区域。

```
(base) [root@n138 examples]# ~/anaconda3/bin/spark-submit
--master yarn
--executor-memory 15g
--executor-cores 1
--num-executors 160
/sparkusecase/DeepCTR/examples/run_classification_criteo_spark.py
--data-dir file:///sparkdemo/tr-4570-data
> . /run_classification_criteo_spark_nfs.log 2>&1
```

经过十次训练后、我们在测试数据集中获得了AUC分数：

```
Epoch 1/10
125/125 - 7s - loss: 0.4976 - binary_crossentropy: 0.4974 - val_loss:
0.4629 - val_binary_crossentropy: 0.4624
Epoch 2/10
125/125 - 1s - loss: 0.3281 - binary_crossentropy: 0.3271 - val_loss:
0.5146 - val_binary_crossentropy: 0.5130
Epoch 3/10
125/125 - 1s - loss: 0.1948 - binary_crossentropy: 0.1928 - val_loss:
0.6166 - val_binary_crossentropy: 0.6144
Epoch 4/10
125/125 - 1s - loss: 0.1408 - binary_crossentropy: 0.1383 - val_loss:
0.7261 - val_binary_crossentropy: 0.7235
Epoch 5/10
125/125 - 1s - loss: 0.1129 - binary_crossentropy: 0.1102 - val_loss:
0.7961 - val_binary_crossentropy: 0.7934
Epoch 6/10
125/125 - 1s - loss: 0.0949 - binary_crossentropy: 0.0921 - val_loss:
0.9502 - val_binary_crossentropy: 0.9474
Epoch 7/10
125/125 - 1s - loss: 0.0778 - binary_crossentropy: 0.0750 - val_loss:
1.1329 - val_binary_crossentropy: 1.1301
Epoch 8/10
125/125 - 1s - loss: 0.0651 - binary_crossentropy: 0.0622 - val_loss:
1.3794 - val_binary_crossentropy: 1.3766
Epoch 9/10
125/125 - 1s - loss: 0.0555 - binary_crossentropy: 0.0527 - val_loss:
1.6115 - val_binary_crossentropy: 1.6087
Epoch 10/10
125/125 - 1s - loss: 0.0470 - binary_crossentropy: 0.0442 - val_loss:
1.6768 - val_binary_crossentropy: 1.6740
test AUC 0.6337
```

我们采用与先前使用情形类似的方式、将Spark工作流运行时与驻留在不同位置的数据进行了比较。下图比较了Spark工作流运行时的深度学习CTR预测。

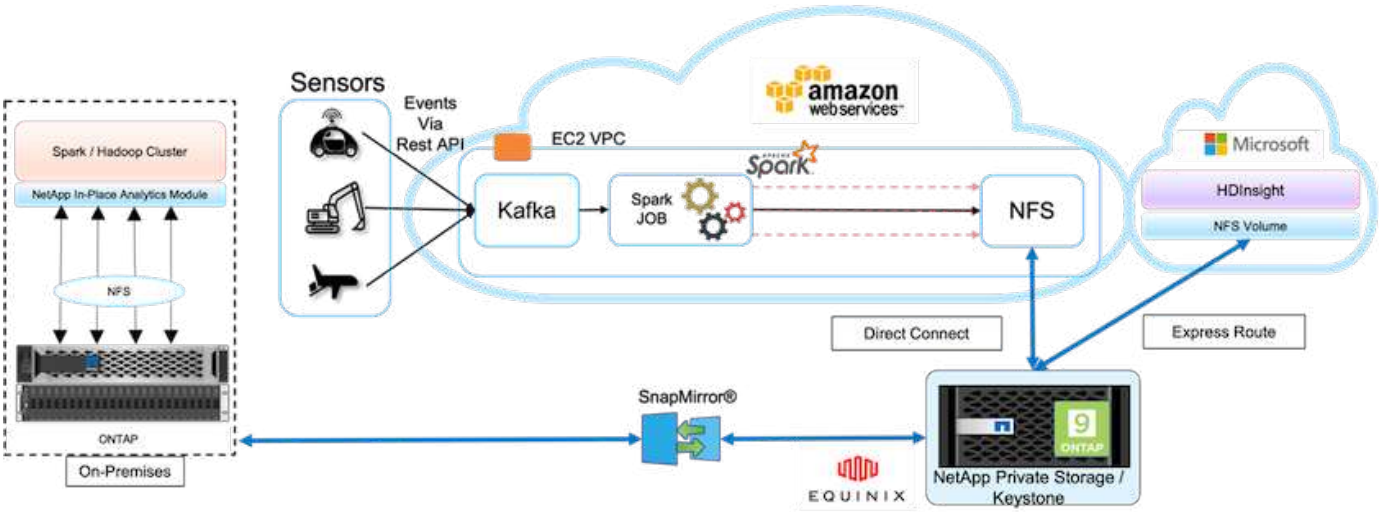


混合云解决方案

现代企业数据中心是一种混合云、它可以在内部和/或多个公有 云中通过具有一致运营模式的持续数据管理平台连接多个分布式基础架构环境。要充分利用混合云、您必须能够在内部环境和多云环境之间无缝移动数据、而无需进行任何数据转换或应用程序重构。

客户表示、他们开始混合云之旅的方式是将二级存储迁移到云以用于数据保护等使用情形、或者将应用程序开发和DevOps等业务关键型工作负载迁移到云。然后、它们会迁移到更关键的工作负载。Web和内容托管、开发运营和应用程序开发、数据库、分析和容器化应用程序是最受欢迎的混合云工作负载。企业人工智能项目的复杂性、成本和风险历来阻碍人工智能从实验阶段到生产阶段的采用。

借助NetApp混合云解决方案 、客户可以通过一个控制面板在分布式环境中管理数据和工作流、从而从集成的安全性、数据监管和合规性工具中受益、同时根据其使用情况优化总拥有成本。下图是一个云服务合作伙伴的示例解决方案 、该合作伙伴负责为客户的大数据分析数据提供多云连接。



在这种情况下、在AWS中从不同来源接收的物联网数据存储在NetApp私有存储(NPS)的中央位置。NPS存储连接到AWS和Azure中的Spark或Hadoop集群、从而支持在多个云中运行的大数据分析应用程序访问相同的数据。此用例的主要要求和挑战包括：

- 客户希望使用多个云对相同数据运行分析作业。
- 数据必须通过不同的传感器和中心从内部和云环境等不同的源接收。
- 解决方案 必须高效且经济高效。
- 主要挑战是构建一个经济高效的解决方案 、以便在不同的内部环境和云环境之间提供混合分析服务。

我们的数据保护和多云连接解决方案 解决了在多个超大规模提供商之间部署云分析应用程序的难题。如上图所示，来自传感器的数据会通过 Kafka 流式传输并输入到 AWS Spark 集群中。数据存储在 NPS 中的 NFS 共享中， NPS 位于 Equinix 数据中心内的云提供商之外。

由于NetApp NPS分别通过Direct Connect和Express Route连接连接到Amazon AWS和Microsoft Azure、因此客户可以利用原位分析模块从Amazon和AWS分析集群访问数据。因此、由于内部和NPS存储均运行ONTAP 软件、 "SnapMirror" 可以将NPS数据镜像到内部集群、从而在内部和多个云之间提供混合云分析。

为了获得最佳性能、NetApp通常建议使用多个网络接口以及直接连接或快速路由从云实例访问数据。我们还有其他数据移动解决方案、包括 "XCP" 和 "BlueXP复制和同步" 帮助客户构建应用程序感知型、安全且经济高效的混合云Spark集群。

适用于每个主要用例的**Python**脚本

以下三个Python脚本对应于所测试的三个主要用例。第一个是`sentiment_analysis _sparknlp.py`。

```
# TR-4570 Refresh NLP testing by Rick Huang
from sys import argv
import os
import sparknlp
import pyspark.sql.functions as F
from sparknlp import Finisher
from pyspark.ml import Pipeline
```

```
from sparknlp.base import *
from sparknlp.annotator import *
from sparknlp.pretrained import PretrainedPipeline
from sparknlp import Finisher
# Start Spark Session with Spark NLP
spark = sparknlp.start()
print("Spark NLP version:")
print(sparknlp.version())
print("Apache Spark version:")
print(spark.version)
spark = sparknlp.SparkSession.builder \
     .master("yarn") \
     .appName("test_hdfs_read_write") \
     .config("spark.executor.cores", "1") \
     .config("spark.jars.packages", "com.johnsnowlabs.nlp:spark-
nlp_2.12:3.4.3")\
     .config('spark.executor.memory', '5gb') \
     .config('spark.executor.memoryOverhead','1000')\
     .config('spark.driver.memoryOverhead','1000')\
     .config("spark.sql.shuffle.partitions", "480")\
     .getOrCreate()
sc = spark.sparkContext
from pyspark.sql import SQLContext
sql = SQLContext(sc)
sqlContext = SQLContext(sc)
# Download pre-trained pipelines & sequence classifier
explain_pipeline_model = PretrainedPipeline('explain_document_dl',
lang='en').model#pipeline_sa =
PretrainedPipeline("classifierdl_bertwiki_finance_sentiment_pipeline",
lang="en")
# pipeline_finbert =
BertForSequenceClassification.loadSavedModel('/sparkusecase/bert_sequence_
classifier_finbert_en_3', spark)
sequenceClassifier = BertForSequenceClassification \
        .pretrained('bert_sequence_classifier_finbert', 'en') \
        .setInputCols(['token', 'document']) \
        .setOutputCol('class') \
        .setCaseSensitive(True) \
        .setMaxSentenceLength(512)
def process_sentence_df(data):
    # Pre-process: begin
    print("1. Begin DataFrame pre-processing...\n")
    print(f"\n\t2. Attaching DocumentAssembler Transformer to the
pipeline")
    documentAssembler = DocumentAssembler() \
        .setInputCol("text") \
```

```
        .setOutputCol("document") \
        .setCleanupMode("inplace_full")
        #.setCleanupMode("shrink", "inplace_full")
    doc_df = documentAssembler.transform(data)
    doc_df.printSchema()
    doc_df.show(truncate=50)
    # Pre-process: get rid of  blank lines
    clean_df = doc_df.withColumn("tmp", F.explode("document")) \
        .select("tmp.result").where("tmp.end !=
-1").withColumnRenamed("result", "text").dropna()
    print("[OK!] DataFrame after initial cleanup:\n")
    clean_df.printSchema()
    clean_df.show(truncate=80)
    # for FinBERT
    tokenizer = Tokenizer() \
        .setInputCols(['document']) \
        .setOutputCol('token')
    print(f"\n\t3. Attaching Tokenizer Annotator to the pipeline")
    pipeline_finbert = Pipeline(stages=[
        documentAssembler,
        tokenizer,
        sequenceClassifier
        ])
    # Use Finisher() & construct PySpark ML pipeline
    finisher = Finisher().setInputCols(["token", "lemma", "pos",
"entities"])
    print(f"\n\t4. Attaching Finisher Transformer to the pipeline")
    pipeline_ex = Pipeline() \
        .setStages([
           explain_pipeline_model,
           finisher
           ])
    print("\n\t\t\t ---- Pipeline Built Successfully ----")
    # Loading pipelines to annotate
    #result_ex_df = pipeline_ex.transform(clean_df)
    ex_model = pipeline_ex.fit(clean_df)
    annotations_finished_ex_df = ex_model.transform(clean_df)
    # result_sa_df = pipeline_sa.transform(clean_df)
    result_finbert_df = pipeline_finbert.fit(clean_df).transform(clean_df)
    print("\n\t\t\t ----Document Explain, Sentiment Analysis & FinBERT
Pipeline Fitted Successfully ----")
    # Check the result entities
    print("[OK!] Simple explain ML pipeline result:\n")
    annotations_finished_ex_df.printSchema()
    annotations_finished_ex_df.select('text',
'finished_entities').show(truncate=False)
```

```
    # Check the result sentiment from FinBERT
    print("[OK!] Sentiment Analysis FinBERT pipeline result:\n")
    result_finbert_df.printSchema()
    result_finbert_df.select('text', 'class.result').show(80, False)
    sentiment_stats(result_finbert_df)
    return
def sentiment_stats(finbert_df):
    result_df = finbert_df.select('text', 'class.result')
    sa_df = result_df.select('result')
    sa_df.groupBy('result').count().show()
    # total_lines = result_clean_df.count()
    # num_neutral = result_clean_df.where(result_clean_df.result ==
['neutral']).count()
    # num_positive = result_clean_df.where(result_clean_df.result ==
['positive']).count()
    # num_negative = result_clean_df.where(result_clean_df.result ==
['negative']).count()
    # print(f"\nRatio of neutral sentiment = {num_neutral/total_lines}")
    # print(f"Ratio of positive sentiment = {num_positive / total_lines}")
    # print(f"Ratio of negative sentiment = {num_negative /
total_lines}\n")
    return
def process_input_file(file_name):
    # Turn input file to Spark DataFrame
    print("START processing input file...")
    data_df = spark.read.text(file_name)
    data_df.show()
    # rename first column 'text' for sparknlp
    output_df = data_df.withColumnRenamed("value", "text").dropna()
    output_df.printSchema()
    return output_dfdef process_local_dir(directory):
    filelist = []
    for subdir, dirs, files in os.walk(directory):
        for filename in files:
            filepath = subdir + os.sep + filename
            print("[OK!] Will process the following files:")
            if filepath.endswith(".txt"):
                print(filepath)
                filelist.append(filepath)
    return filelist
def process_local_dir_or_file(dir_or_file):
    numfiles = 0
    if os.path.isfile(dir_or_file):
        input_df = process_input_file(dir_or_file)
        print("Obtained input_df.")
        process_sentence_df(input_df)
```

```
        print("Processed input_df")
        numfiles += 1
    else:
        filelist = process_local_dir(dir_or_file)
        for file in filelist:
            input_df = process_input_file(file)
            process_sentence_df(input_df)
            numfiles += 1
    return numfiles
def process_hdfs_dir(dir_name):
    # Turn input files to Spark DataFrame
    print("START processing input HDFS directory...")
    data_df = spark.read.option("recursiveFileLookup",
"true").text(dir_name)
    data_df.show()
    print("[DEBUG] total lines in data_df = ", data_df.count())
    # rename first column 'text' for sparknlp
    output_df = data_df.withColumnRenamed("value", "text").dropna()
    print("[DEBUG] output_df looks like: \n")
    output_df.show(40, False)
    print("[DEBUG] HDFS dir resulting data_df schema: \n")
    output_df.printSchema()
    process_sentence_df(output_df)
    print("Processed HDFS directory: ", dir_name)
    returnif __name__ == '__main__':
    try:
        if len(argv) == 2:
            print("Start processing input...\n")
    except:
        print("[ERROR] Please enter input text file or path to
process!\n")
        exit(1)
    # This is for local file, not hdfs:
    numfiles = process_local_dir_or_file(str(argv[1]))
    # For HDFS single file & directory:
    input_df = process_input_file(str(argv[1]))
    print("Obtained input_df.")
    process_sentence_df(input_df)
    print("Processed input_df")
    numfiles += 1
    # For HDFS directory of subdirectories of files:
    input_parse_list = str(argv[1]).split('/')
    print(input_parse_list)
    if input_parse_list[-2:-1] == ['Transcripts']:
        print("Start processing HDFS directory: ", str(argv[1]))
        process_hdfs_dir(str(argv[1]))
```

```
    print(f"[OK!] All done. Number of files processed = {numfiles}")
```

第二个脚本为`keras_sock_horovod_rossmann_estimator.py`。

```
# Copyright 2022 NetApp, Inc.
# Authored by Rick Huang
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#
==============================================================================
====
# The below code was modified from: https://www.kaggle.com/c/rossmann-
store-sales
import argparse
import datetime
import os
import sys
from distutils.version import LooseVersion
import pyspark.sql.types as T
import pyspark.sql.functions as F
from pyspark import SparkConf, Row
from pyspark.sql import SparkSession
import tensorflow as tf
import tensorflow.keras.backend as K
from tensorflow.keras.layers import Input, Embedding, Concatenate, Dense,
Flatten, Reshape, BatchNormalization, Dropout
import horovod.spark.keras as hvd
from horovod.spark.common.backend import SparkBackend
from horovod.spark.common.store import Store
from horovod.tensorflow.keras.callbacks import BestModelCheckpoint
parser = argparse.ArgumentParser(description='Horovod Keras Spark Rossmann
Estimator Example',

formatter_class=argparse.ArgumentDefaultsHelpFormatter)
parser.add_argument('--master',
```

```
                     help='spark cluster to use for training. If set to
None, uses current default cluster. Cluster'
                          'should be set up to provide a Spark task per
multiple CPU cores, or per GPU, e.g. by'
                          'supplying `-c <NUM_GPUS>` in Spark Standalone
mode')
parser.add_argument('--num-proc', type=int,
                     help='number of worker processes for training,
default: `spark.default.parallelism`')
parser.add_argument('--learning_rate', type=float, default=0.0001,
                     help='initial learning rate')
parser.add_argument('--batch-size', type=int, default=100,
                     help='batch size')
parser.add_argument('--epochs', type=int, default=100,
                     help='number of epochs to train')
parser.add_argument('--sample-rate', type=float,
                     help='desired sampling rate. Useful to set to low
number (e.g. 0.01) to make sure that '
                          'end-to-end process works')
parser.add_argument('--data-dir', default='file://' + os.getcwd(),
                     help='location of data on local filesystem (prefixed
with file://) or on HDFS')
parser.add_argument('--local-submission-csv', default='submission.csv',
                     help='output submission predictions CSV')
parser.add_argument('--local-checkpoint-file', default='checkpoint',
                     help='model checkpoint')
parser.add_argument('--work-dir', default='/tmp',
                     help='temporary working directory to write
intermediate files (prefix with hdfs:// to use HDFS)')
if __name__ == '__main__':
    args = parser.parse_args()
    # ================ #
    # DATA PREPARATION #
    # ================ #
    print('================')
    print('Data preparation')
    print('================')
    # Create Spark session for data preparation.
    conf = SparkConf() \
        .setAppName('Keras Spark Rossmann Estimator Example') \
        .set('spark.sql.shuffle.partitions', '480') \
        .set("spark.executor.cores", "1") \
        .set('spark.executor.memory', '5gb') \
        .set('spark.executor.memoryOverhead','1000')\
        .set('spark.driver.memoryOverhead','1000')
    if args.master:
```

```
        conf.setMaster(args.master)
    elif args.num_proc:
        conf.setMaster('local[{}]'.format(args.num_proc))
    spark = SparkSession.builder.config(conf=conf).getOrCreate()
    train_csv = spark.read.csv('%s/train.csv' % args.data_dir,
header=True)
    test_csv = spark.read.csv('%s/test.csv' % args.data_dir, header=True)
    store_csv = spark.read.csv('%s/store.csv' % args.data_dir,
header=True)
    store_states_csv = spark.read.csv('%s/store_states.csv' %
args.data_dir, header=True)
    state_names_csv = spark.read.csv('%s/state_names.csv' % args.data_dir,
header=True)
    google_trend_csv = spark.read.csv('%s/googletrend.csv' %
args.data_dir, header=True)
    weather_csv = spark.read.csv('%s/weather.csv' % args.data_dir,
header=True)
    def expand_date(df):
        df = df.withColumn('Date', df.Date.cast(T.DateType()))
        return df \
            .withColumn('Year', F.year(df.Date)) \
            .withColumn('Month', F.month(df.Date)) \
            .withColumn('Week', F.weekofyear(df.Date)) \
            .withColumn('Day', F.dayofmonth(df.Date))
    def prepare_google_trend():
        # Extract week start date and state.
        google_trend_all = google_trend_csv \
            .withColumn('Date', F.regexp_extract(google_trend_csv.week,
'(.*?) -', 1)) \
            .withColumn('State', F.regexp_extract(google_trend_csv.file,
'Rossmann_DE_(.*)', 1))
        # Map state NI -> HB,NI to align with other data sources.
        google_trend_all = google_trend_all \
            .withColumn('State', F.when(google_trend_all.State == 'NI',
'HB,NI').otherwise(google_trend_all.State))
        # Expand dates.
        return expand_date(google_trend_all)
    def add_elapsed(df, cols):
        def add_elapsed_column(col, asc):
            def fn(rows):
                last_store, last_date = None, None
                for r in rows:
                    if last_store != r.Store:
                        last_store = r.Store
                        last_date = r.Date
                    if r[col]:
```

```
                        last_date = r.Date
                    fields = r.asDict().copy()
                    fields[('After' if asc else 'Before') + col] = (r.Date
- last_date).days
                    yield Row(**fields)
            return fn
        df = df.repartition(df.Store)
        for asc in [False, True]:
            sort_col = df.Date.asc() if asc else df.Date.desc()
            rdd = df.sortWithinPartitions(df.Store.asc(), sort_col).rdd
            for col in cols:
                rdd = rdd.mapPartitions(add_elapsed_column(col, asc))
            df = rdd.toDF()
        return df
    def prepare_df(df):
        num_rows = df.count()
        # Expand dates.
        df = expand_date(df)
        df = df \
            .withColumn('Open', df.Open != '0') \
            .withColumn('Promo', df.Promo != '0') \
            .withColumn('StateHoliday', df.StateHoliday != '0') \
            .withColumn('SchoolHoliday', df.SchoolHoliday != '0')
        # Merge in store information.
        store = store_csv.join(store_states_csv, 'Store')
        df = df.join(store, 'Store')
        # Merge in Google Trend information.
        google_trend_all = prepare_google_trend()
        df = df.join(google_trend_all, ['State', 'Year',
'Week']).select(df['*'], google_trend_all.trend)
        # Merge in Google Trend for whole Germany.
        google_trend_de = google_trend_all[google_trend_all.file ==
'Rossmann_DE'].withColumnRenamed('trend', 'trend_de')
        df = df.join(google_trend_de, ['Year', 'Week']).select(df['*'],
google_trend_de.trend_de)
        # Merge in weather.
        weather = weather_csv.join(state_names_csv, weather_csv.file ==
state_names_csv.StateName)
        df = df.join(weather, ['State', 'Date'])
        # Fix null values.
        df = df \
            .withColumn('CompetitionOpenSinceYear',
F.coalesce(df.CompetitionOpenSinceYear, F.lit(1900))) \
            .withColumn('CompetitionOpenSinceMonth',
F.coalesce(df.CompetitionOpenSinceMonth, F.lit(1))) \
            .withColumn('Promo2SinceYear', F.coalesce(df.Promo2SinceYear,
```

```
F.lit(1900))) \
            .withColumn('Promo2SinceWeek', F.coalesce(df.Promo2SinceWeek,
F.lit(1)))
        # Days & months competition was open, cap to 2 years.
        df = df.withColumn('CompetitionOpenSince',
                           F.to_date(F.format_string('%s-%s-15',
df.CompetitionOpenSinceYear,

df.CompetitionOpenSinceMonth)))
        df = df.withColumn('CompetitionDaysOpen',
                           F.when(df.CompetitionOpenSinceYear > 1900,
                                  F.greatest(F.lit(0), F.least(F.lit(360 *
2), F.datediff(df.Date, df.CompetitionOpenSince))))
                           .otherwise(0))
        df = df.withColumn('CompetitionMonthsOpen',
(df.CompetitionDaysOpen / 30).cast(T.IntegerType()))
        # Days & weeks of promotion, cap to 25 weeks.
        df = df.withColumn('Promo2Since',
                           F.expr('date_add(format_string("%s-01-01",
Promo2SinceYear), (cast(Promo2SinceWeek as int) - 1) * 7)'))
        df = df.withColumn('Promo2Days',
                           F.when(df.Promo2SinceYear > 1900,
                                  F.greatest(F.lit(0), F.least(F.lit(25 *
7), F.datediff(df.Date, df.Promo2Since))))
                           .otherwise(0))
        df = df.withColumn('Promo2Weeks', (df.Promo2Days /
7).cast(T.IntegerType()))
        # Check that we did not lose any rows through inner joins.
        assert num_rows == df.count(), 'lost rows in joins'
        return df
    def build_vocabulary(df, cols):
        vocab = {}
        for col in cols:
            values = [r[0] for r in df.select(col).distinct().collect()]
            col_type = type([x for x in values if x is not None][0])
            default_value = col_type()
            vocab[col] = sorted(values, key=lambda x: x or default_value)
        return vocab
    def cast_columns(df, cols):
        for col in cols:
            df = df.withColumn(col,
F.coalesce(df[col].cast(T.FloatType()), F.lit(0.0)))
        return df
    def lookup_columns(df, vocab):
        def lookup(mapping):
            def fn(v):
```

```
                return mapping.index(v)
            return F.udf(fn, returnType=T.IntegerType())
        for col, mapping in vocab.items():
            df = df.withColumn(col, lookup(mapping)(df[col]))
        return df
    if args.sample_rate:
        train_csv = train_csv.sample(withReplacement=False,
fraction=args.sample_rate)
        test_csv = test_csv.sample(withReplacement=False,
fraction=args.sample_rate)
    # Prepare data frames from CSV files.
    train_df = prepare_df(train_csv).cache()
    test_df = prepare_df(test_csv).cache()
    # Add elapsed times from holidays & promos, the data spanning training
& test datasets.
    elapsed_cols = ['Promo', 'StateHoliday', 'SchoolHoliday']
    elapsed = add_elapsed(train_df.select('Date', 'Store', *elapsed_cols)
                          .unionAll(test_df.select('Date', 'Store',
*elapsed_cols)),
                          elapsed_cols)
    # Join with elapsed times.
    train_df = train_df \
        .join(elapsed, ['Date', 'Store']) \
        .select(train_df['*'], *[prefix + col for prefix in ['Before',
'After'] for col in elapsed_cols])
    test_df = test_df \
        .join(elapsed, ['Date', 'Store']) \
        .select(test_df['*'], *[prefix + col for prefix in ['Before',
'After'] for col in elapsed_cols])
    # Filter out zero sales.
    train_df = train_df.filter(train_df.Sales > 0)
    print('===================')
    print('Prepared data frame')
    print('===================')
    train_df.show()
    categorical_cols = [
        'Store', 'State', 'DayOfWeek', 'Year', 'Month', 'Day', 'Week',
'CompetitionMonthsOpen', 'Promo2Weeks', 'StoreType',
        'Assortment', 'PromoInterval', 'CompetitionOpenSinceYear',
'Promo2SinceYear', 'Events', 'Promo',
        'StateHoliday', 'SchoolHoliday'
    ]
    continuous_cols = [
        'CompetitionDistance', 'Max_TemperatureC', 'Mean_TemperatureC',
'Min_TemperatureC', 'Max_Humidity',
        'Mean_Humidity', 'Min_Humidity', 'Max_Wind_SpeedKm_h',
```

```
'Mean_Wind_SpeedKm_h', 'CloudCover', 'trend', 'trend_de',
        'BeforePromo', 'AfterPromo', 'AfterStateHoliday',
'BeforeStateHoliday', 'BeforeSchoolHoliday', 'AfterSchoolHoliday'
    ]
    all_cols = categorical_cols + continuous_cols
    # Select features.
    train_df = train_df.select(*(all_cols + ['Sales', 'Date'])).cache()
    test_df = test_df.select(*(all_cols + ['Id', 'Date'])).cache()
    # Build vocabulary of categorical columns.
    vocab = build_vocabulary(train_df.select(*categorical_cols)

.unionAll(test_df.select(*categorical_cols)).cache(),
                             categorical_cols)
    # Cast continuous columns to float & lookup categorical columns.
    train_df = cast_columns(train_df, continuous_cols + ['Sales'])
    train_df = lookup_columns(train_df, vocab)
    test_df = cast_columns(test_df, continuous_cols)
    test_df = lookup_columns(test_df, vocab)
    # Split into training & validation.
    # Test set is in 2015, use the same period in 2014 from the training
set as a validation set.
    test_min_date = test_df.agg(F.min(test_df.Date)).collect()[0][0]
    test_max_date = test_df.agg(F.max(test_df.Date)).collect()[0][0]
    one_year = datetime.timedelta(365)
    train_df = train_df.withColumn('Validation',
                                   (train_df.Date > test_min_date -
one_year) & (train_df.Date <= test_max_date - one_year))
    # Determine max Sales number.
    max_sales = train_df.agg(F.max(train_df.Sales)).collect()[0][0]
    # Convert Sales to log domain
    train_df = train_df.withColumn('Sales', F.log(train_df.Sales))
    print('====================================')
    print('Data frame with transformed columns')
    print('====================================')
    train_df.show()
    print('================')
    print('Data frame sizes')
    print('================')
    train_rows = train_df.filter(~train_df.Validation).count()
    val_rows = train_df.filter(train_df.Validation).count()
    test_rows = test_df.count()
    print('Training: %d' % train_rows)
    print('Validation: %d' % val_rows)
    print('Test: %d' % test_rows)
    # ============== #
    # MODEL TRAINING #
```

```
    # ============== #
    print('==============')
    print('Model training')
    print('==============')
    def exp_rmspe(y_true, y_pred):
        """Competition evaluation metric, expects logarithic inputs."""
        pct = tf.square((tf.exp(y_true) - tf.exp(y_pred)) /
tf.exp(y_true))
        # Compute mean excluding stores with zero denominator.
        x = tf.reduce_sum(tf.where(y_true > 0.001, pct,
tf.zeros_like(pct)))
        y = tf.reduce_sum(tf.where(y_true > 0.001, tf.ones_like(pct),
tf.zeros_like(pct)))
        return tf.sqrt(x / y)
    def act_sigmoid_scaled(x):
        """Sigmoid scaled to logarithm of maximum sales scaled by 20%."""
        return tf.nn.sigmoid(x) * tf.math.log(max_sales) * 1.2
    CUSTOM_OBJECTS = {'exp_rmspe': exp_rmspe,
                      'act_sigmoid_scaled': act_sigmoid_scaled}
    # Disable GPUs when building the model to prevent memory leaks
    if LooseVersion(tf.__version__) >= LooseVersion('2.0.0'):
        # See https://github.com/tensorflow/tensorflow/issues/33168
        os.environ['CUDA_VISIBLE_DEVICES'] = '-1'
    else:

K.set_session(tf.Session(config=tf.ConfigProto(device_count={'GPU': 0})))
    # Build the model.
    inputs = {col: Input(shape=(1,), name=col) for col in all_cols}
    embeddings = [Embedding(len(vocab[col]), 10, input_length=1,
name='emb_' + col)(inputs[col])
                  for col in categorical_cols]
    continuous_bn = Concatenate()([Reshape((1, 1), name='reshape_' +
col)(inputs[col])
                                   for col in continuous_cols])
    continuous_bn = BatchNormalization()(continuous_bn)
    x = Concatenate()(embeddings + [continuous_bn])
    x = Flatten()(x)
    x = Dense(1000, activation='relu',
kernel_regularizer=tf.keras.regularizers.l2(0.00005))(x)
    x = Dense(1000, activation='relu',
kernel_regularizer=tf.keras.regularizers.l2(0.00005))(x)
    x = Dense(1000, activation='relu',
kernel_regularizer=tf.keras.regularizers.l2(0.00005))(x)
    x = Dense(500, activation='relu',
kernel_regularizer=tf.keras.regularizers.l2(0.00005))(x)
    x = Dropout(0.5)(x)
```

```
    output = Dense(1, activation=act_sigmoid_scaled)(x)
    model = tf.keras.Model([inputs[f] for f in all_cols], output)
    model.summary()
    opt = tf.keras.optimizers.Adam(lr=args.learning_rate, epsilon=1e-3)
    # Checkpoint callback to specify options for the returned Keras model
    ckpt_callback = BestModelCheckpoint(monitor='val_loss', mode='auto',
save_freq='epoch')
    # Horovod: run training.
    store = Store.create(args.work_dir)
    backend = SparkBackend(num_proc=args.num_proc,
                           stdout=sys.stdout, stderr=sys.stderr,
                           prefix_output_with_timestamp=True)
    keras_estimator = hvd.KerasEstimator(backend=backend,
                                         store=store,
                                         model=model,
                                         optimizer=opt,
                                         loss='mae',
                                         metrics=[exp_rmspe],
                                         custom_objects=CUSTOM_OBJECTS,
                                         feature_cols=all_cols,
                                         label_cols=['Sales'],
                                         validation='Validation',
                                         batch_size=args.batch_size,
                                         epochs=args.epochs,
                                         verbose=2,

checkpoint_callback=ckpt_callback)
    keras_model =
keras_estimator.fit(train_df).setOutputCols(['Sales_output'])
    history = keras_model.getHistory()
    best_val_rmspe = min(history['val_exp_rmspe'])
    print('Best RMSPE: %f' % best_val_rmspe)
    # Save the trained model.
    keras_model.save(args.local_checkpoint_file)
    print('Written checkpoint to %s' % args.local_checkpoint_file)
    # ================ #
    # FINAL PREDICTION #
    # ================ #
    print('================')
    print('Final prediction')
    print('================')
    pred_df=keras_model.transform(test_df)
    pred_df.printSchema()
    pred_df.show(5)
    # Convert from log domain to real Sales numbers
    pred_df=pred_df.withColumn('Sales_pred', F.exp(pred_df.Sales_output))
```

```
    submission_df = pred_df.select(pred_df.Id.cast(T.IntegerType()),
pred_df.Sales_pred).toPandas()
    submission_df.sort_values(by=['Id']).to_csv(args.local_submission_csv,
index=False)
    print('Saved predictions to %s' % args.local_submission_csv)
    spark.stop()
```

第三个脚本为`run_clustery_Criteo_spn.py`。

```
import tempfile, string, random, os, uuid
import argparse, datetime, sys, shutil
import csv
import numpy as np
from sklearn.model_selection import train_test_split
from tensorflow.keras.callbacks import EarlyStopping
from pyspark import SparkContext
from pyspark.sql import SparkSession, SQLContext, Row, DataFrame
from pyspark.mllib import linalg as mllib_linalg
from pyspark.mllib.linalg import SparseVector as mllibSparseVector
from pyspark.mllib.linalg import VectorUDT as mllibVectorUDT
from pyspark.mllib.linalg import Vector as mllibVector, Vectors as
mllibVectors
from pyspark.mllib.regression import LabeledPoint
from pyspark.mllib.classification import LogisticRegressionWithSGD
from pyspark.ml import linalg as ml_linalg
from pyspark.ml.linalg import VectorUDT as mlVectorUDT
from pyspark.ml.linalg import SparseVector as mlSparseVector
from pyspark.ml.linalg import Vector as mlVector, Vectors as mlVectors
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import OneHotEncoder
from math import log
from math import exp  # exp(-t) = e^-t
from operator import add
from pyspark.sql.functions import udf, split, lit
from pyspark.sql.functions import size, sum as sqlsum
import pyspark.sql.functions as F
import pyspark.sql.types as T
from pyspark.sql.types import ArrayType, StructType, StructField,
LongType, StringType, IntegerType, FloatType
from pyspark.sql.functions import explode, col, log, when
from collections import defaultdict
import pandas as pd
import pyspark.pandas as ps
from sklearn.metrics import log_loss, roc_auc_score
from sklearn.model_selection import train_test_split
```

```
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
from deepctr.models import DeepFM
from deepctr.feature_column import SparseFeat, DenseFeat,
get_feature_names
spark = SparkSession.builder \
    .master("yarn") \
    .appName("deep_ctr_classification") \
    .config("spark.jars.packages", "io.github.ravwojdyla:spark-schema-
utils_2.12:0.1.0") \
    .config("spark.executor.cores", "1") \
    .config('spark.executor.memory', '5gb') \
    .config('spark.executor.memoryOverhead', '1500') \
    .config('spark.driver.memoryOverhead', '1500') \
    .config("spark.sql.shuffle.partitions", "480") \
    .config("spark.sql.execution.arrow.enabled", "true") \
    .config("spark.driver.maxResultSize", "50gb") \
    .getOrCreate()
# spark.conf.set("spark.sql.execution.arrow.enabled", "true") # deprecated
print("Apache Spark version:")
print(spark.version)
sc = spark.sparkContext
sqlContext = SQLContext(sc)
parser = argparse.ArgumentParser(description='Spark DCN CTR Prediction
Example',

formatter_class=argparse.ArgumentDefaultsHelpFormatter)
parser.add_argument('--data-dir', default='file://' + os.getcwd(),
                    help='location of data on local filesystem (prefixed
with file://) or on HDFS')
def process_input_file(file_name, sparse_feat, dense_feat):
    # Need this preprocessing to turn Criteo raw file into CSV:
    print("START processing input file...")
    # only convert the file ONCE
    # sample = open(file_name)
    # sample = '\n'.join([str(x.replace('\n', '').replace('\t', ',')) for
x in sample])
    # # Add header in data file and save as CSV
    # header = ','.join(str(x) for x in (['label'] + dense_feat +
sparse_feat))
    # with open('/sparkdemo/tr-4570-data/ctr_train.csv', mode='w',
encoding="utf-8") as f:
    #     f.write(header + '\n' + sample)
    #     f.close()
    # print("Raw training file processed and saved as CSV: ", f.name)
    raw_df = sqlContext.read.option("header", True).csv(file_name)
    raw_df.show(5, False)
```

```
    raw_df.printSchema()
    # convert columns I1 to I13 from string to integers
    conv_df = raw_df.select(col('label').cast("double"),
                            *(col(i).cast("float").alias(i) for i in
raw_df.columns if i in dense_feat),
                            *(col(c) for c in raw_df.columns if c in
sparse_feat))
    print("Schema of raw_df with integer columns type changed:")
    conv_df.printSchema()
    # result_pdf = conv_df.select("*").toPandas()
    tmp_df = conv_df.na.fill(0, dense_feat)
    result_df = tmp_df.na.fill('-1', sparse_feat)
    result_df.show()
    return result_df
if __name__ == "__main__":
    args = parser.parse_args()
    # Pandas read CSV
    # data = pd.read_csv('%s/criteo_sample.txt' % args.data_dir)
    # print("Obtained Pandas df.")
    dense_features = ['I' + str(i) for i in range(1, 14)]
    sparse_features = ['C' + str(i) for i in range(1, 27)]
    # Spark read CSV
    # process_input_file('%s/train.txt' % args.data_dir, sparse_features,
dense_features) # run only ONCE
    spark_df = process_input_file('%s/data.txt' % args.data_dir,
sparse_features, dense_features) # sample data
    # spark_df = process_input_file('%s/ctr_train.csv' % args.data_dir,
sparse_features, dense_features)
    print("Obtained Spark df and filled in missing features.")
    data = spark_df
    # Pandas
    #data[sparse_features] = data[sparse_features].fillna('-1', )
    #data[dense_features] = data[dense_features].fillna(0, )
    target = ['label']
    label_npa = data.select("label").toPandas().to_numpy()
    print("label numPy array has length = ", len(label_npa)) # 45,840,617
w/ 11GB dataset
    label_npa.ravel()
    label_npa.reshape(len(label_npa), )
    # 1.Label Encoding for sparse features,and do simple Transformation
for dense features
    print("Before LabelEncoder():")
    data.printSchema()  # label: float (nullable = true)
    for feat in sparse_features:
        lbe = LabelEncoder()
        tmp_pdf = data.select(feat).toPandas().to_numpy()
```

```
        tmp_ndarray = lbe.fit_transform(tmp_pdf)
        print("After LabelEncoder(), tmp_ndarray[0] =", tmp_ndarray[0])
        # print("Data tmp PDF after lbe transformation, the output ndarray
has length = ", len(tmp_ndarray)) # 45,840,617 for 11GB dataset
        tmp_ndarray.ravel()
        tmp_ndarray.reshape(len(tmp_ndarray), )
        out_ndarray = np.column_stack([label_npa, tmp_ndarray])
        pdf = pd.DataFrame(out_ndarray, columns=['label', feat])
        s_df = spark.createDataFrame(pdf)
        s_df.printSchema() # label: double (nullable = true)
        print("Before joining data df with s_df, s_df example rows:")
        s_df.show(1, False)
        data = data.drop(feat).join(s_df, 'label').drop('label')
        print("After LabelEncoder(), data df example rows:")
        data.show(1, False)
        print("Finished processing sparse_features: ", feat)
    print("Data DF after label encoding: ")
    data.show()
    data.printSchema()
    mms = MinMaxScaler(feature_range=(0, 1))
    # data[dense_features] = mms.fit_transform(data[dense_features]) # for
Pandas df
    tmp_pdf = data.select(dense_features).toPandas().to_numpy()
    tmp_ndarray = mms.fit_transform(tmp_pdf)
    tmp_ndarray.ravel()
    tmp_ndarray.reshape(len(tmp_ndarray), len(tmp_ndarray[0]))
    out_ndarray = np.column_stack([label_npa, tmp_ndarray])
    pdf = pd.DataFrame(out_ndarray, columns=['label'] + dense_features)
    s_df = spark.createDataFrame(pdf)
    s_df.printSchema()
    data.drop(*dense_features).join(s_df, 'label').drop('label')
    print("Finished processing dense_features: ", dense_features)
    print("Data DF after MinMaxScaler: ")
    data.show()

    # 2.count #unique features for each sparse field,and record dense
feature field name
    fixlen_feature_columns = [SparseFeat(feat,
vocabulary_size=data.select(feat).distinct().count() + 1, embedding_dim=4)
                              for i, feat in enumerate(sparse_features)] +
\
                             [DenseFeat(feat, 1, ) for feat in
dense_features]
    dnn_feature_columns = fixlen_feature_columns
    linear_feature_columns = fixlen_feature_columns
    feature_names = get_feature_names(linear_feature_columns +
```

```
dnn_feature_columns)
    # 3.generate input data for model
    # train, test = train_test_split(data.toPandas(), test_size=0.2,
random_state=2020) # Pandas; might hang for 11GB data
    train, test = data.randomSplit(weights=[0.8, 0.2], seed=200)
    print("Training dataset size = ", train.count())
    print("Testing dataset size = ", test.count())
    # Pandas:
    # train_model_input = {name: train[name] for name in feature_names}
    # test_model_input = {name: test[name] for name in feature_names}
    # Spark DF:
    train_model_input = {}
    test_model_input = {}
    for name in feature_names:
        if name.startswith('I'):
            tr_pdf = train.select(name).toPandas()
            train_model_input[name] = pd.to_numeric(tr_pdf[name])
            ts_pdf = test.select(name).toPandas()
            test_model_input[name] = pd.to_numeric(ts_pdf[name])
    # 4.Define Model,train,predict and evaluate
    model = DeepFM(linear_feature_columns, dnn_feature_columns,
task='binary')
    model.compile("adam", "binary_crossentropy",
                  metrics=['binary_crossentropy'], )
    lb_pdf = train.select(target).toPandas()
    history = model.fit(train_model_input,
pd.to_numeric(lb_pdf['label']).values,
                        batch_size=256, epochs=10, verbose=2,
validation_split=0.2, )
    pred_ans = model.predict(test_model_input, batch_size=256)
    print("test LogLoss",
round(log_loss(pd.to_numeric(test.select(target).toPandas()).values,
pred_ans), 4))
    print("test AUC",
round(roc_auc_score(pd.to_numeric(test.select(target).toPandas()).values,
pred_ans), 4))
```

结论

在本文档中、我们将讨论Apache Spark架构、客户用例以及与大数据、现代分析以及AI、ML和DL相关的NetApp存储产品组合。在我们基于行业标准基准测试工具和客户需求进行的性能验证测试中、NetApp Spark解决方案的性能优于原生 Hadoop系统。将本报告中提供的客户用例和性能结果相结合、可以帮助您为您的部署选择合适的Spark解决方案。

从何处查找追加信息

本技术报告使用了以下参考资料：

- Apache Spark架构和组件

  "http://spark.apache.org/docs/latest/cluster-overview.html"

- Apache Spark用例

  "https://www.qubole.com/blog/big-data/apache-spark-use-cases/"

- Apache挑战

  "http://www.infoworld.com/article/2897287/big-data/5-reasons-to-turn-to-spark-for-big-data-analytics.html"

- 激发NLP

  "https://www.johnsnowlabs.com/spark-nlp/"

- Bert

  "https://arxiv.org/abs/1810.04805"

- 深度和跨网络、用于广告点击预测

  "https://arxiv.org/abs/1708.05123"

- FlexGroup

  "http://www.netapp.com/us/media/tr-4557.pdf"

- 流式ETL

  "https://www.infoq.com/articles/apache-spark-streaming"

- 适用于Hadoop的NetApp E系列解决方案

  "https://www.netapp.com/media/16420-tr-3969.pdf"

- 《使用NetApp AI进行客户沟通时的情绪分析》

  "https://docs.netapp.com/us-en/netapp-solutions/pdfs/sidebar/Sentiment_analysis_with_NetApp_AI.pdf"

- NetApp 现代数据分析解决方案

  "https://docs.netapp.com/us-en/netapp-solutions/data-analytics/index.html"

- SnapMirror

  "https://docs.netapp.com/us-en/ontap/data-protection/snapmirror-replication-concept.html"

- XCP

  https://mysupport.netapp.com/documentation/docweb/index.html?productID=63942&language=en-US

- BlueXP复制和同步

  "https://cloud.netapp.com/cloud-sync-service"

- DataOps工具包

  "https://github.com/NetApp/netapp-dataops-toolkit"

大数据分析数据到人工智能

TR-4732： 大数据分析数据到人工智能

NetApp 公司 Karthikeyan Nagalingam

本文档介绍如何将大数据分析数据和 HPC 数据迁移到 AI 。AI 通过 NFS 导出处理 NFS 数据，而客户通常将其 AI 数据存储在 HDFS ， Blob 或 S3 存储等大数据分析平台以及 GPFS 等 HPC 平台中。本白皮书提供了使用 NetApp XCP 和 NIPAM 将大数据分析数据和 HPC 数据迁移到 AI 的准则。我们还讨论了将数据从大数据和 HPC 迁移到 AI 的业务优势。

概念和组件

大数据分析存储

大数据分析是 HDFS 的主要存储提供商。客户通常使用与 Hadoop 兼容的文件系统（ HCFS ），例如 Windows Azure Blob Storage， MapR 文件系统（ MapR-FS ）和 S3 对象存储。

常规并行文件系统

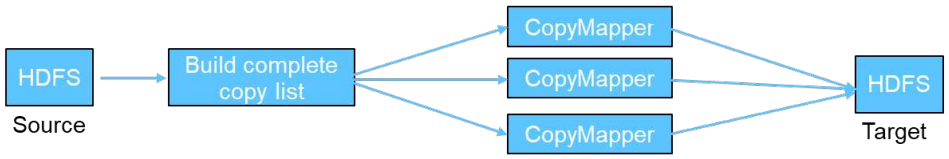
IBM 的 GPFS 是一种企业级文件系统，可替代 HDFS 。利用 GPF ，应用程序可以灵活地确定块大小和复制布局，从而提供良好的性能和效率。

NetApp 原位分析模块

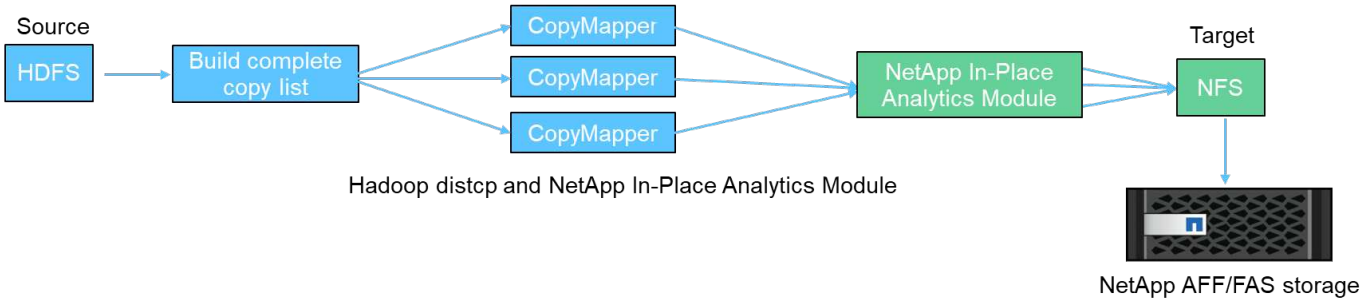
NetApp 原位分析模块（ NIPAM ）可作为 Hadoop 集群访问 NFS 数据的驱动程序。它包含四个组件：连接池，NFS InputStream ，文件句柄缓存和 NFS OutputStream 。有关详细信息，请参见 "TR-4382： NetApp 原位分析模块。"

Hadoop 分布式副本

Hadoop 分布式副本（ DistCp ）是一种分布式副本工具，用于执行大型集群间和集群内应对任务。此工具使用 MapReduce 进行数据分发，错误处理和报告。它会扩展文件和目录列表，并输入这些文件和目录以映射任务，从而从源列表复制数据。下图显示了 HDFS 和非 HDFS 中的 DistCp 操作。



Hadoop distcp basic process



Hadoop distcp and NetApp In-Place Analytics Module

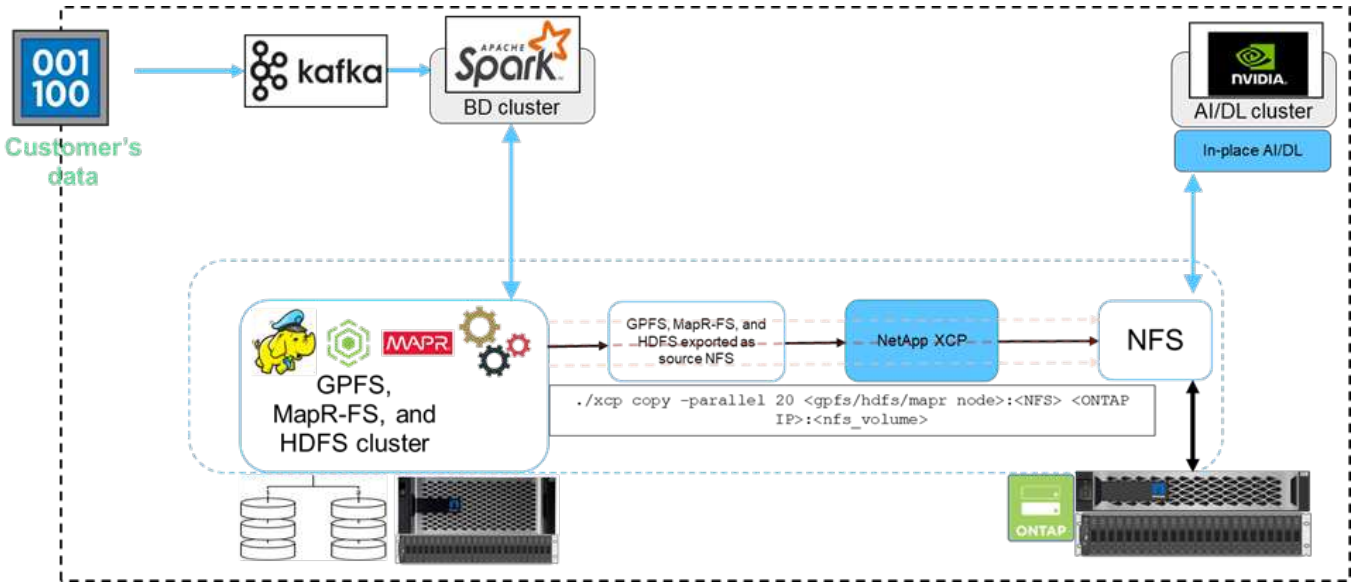
Hadoop DistCp 可在两个 HDFS 系统之间移动数据，而无需使用其他驱动程序。NetApp 为非 HDFS 系统提供了驱动程序。对于 NFS 目标， NIPAM 提供了用于复制数据的驱动程序， Hadoop DistCp 在复制数据时使用这些数据与 NFS 目标进行通信。

NetApp Cloud Volumes Service

NetApp Cloud Volumes Service 是一种性能极高的云原生文件服务。此服务可通过快速启动和关闭资源并使用 NetApp 功能来提高工作效率并减少员工停机时间，帮助客户加快产品上市速度。Cloud Volumes Service 是灾难恢复和备份到云的理想替代方案，因为它可以减少数据中心的整体占用空间，并减少原生公有云存储的使用量。

NetApp XCP

NetApp XCP 是一款客户端软件，可实现快速可靠的任意到 NetApp 和 NetApp 到 NetApp 数据迁移。此工具用于将大量非结构化 NAS 数据从任何 NAS 系统复制到 NetApp 存储控制器。XCP 迁移工具使用多核多通道 I/O 流式引擎，可以并行处理多个请求，例如数据迁移，文件或目录列表以及空间报告。这是默认的 NetApp 数据迁移工具。您可以使用 XCP 将数据从 Hadoop 集群和 HPC 复制到 NetApp NFS 存储。下图显示了使用 XCP 从 Hadoop 和 HPC 集群到 NetApp NFS 卷的数据传输。



NetApp BlueXP复制和同步

NetApp BlueXP复制和同步是一种混合数据复制软件即服务、可在内部存储和云存储之间安全无缝地传输和同步NFS、S3和CIFS数据。此软件用于数据迁移，归档，协作，分析等。传输数据后、BlueXP复制和同步功能会持续同步源和目标之间的数据。接下来，它会传输增量数据。它还可以保护您自己网络，云或内部环境中的数据。此软件采用按需购买模式，可提供经济高效的解决方案并为数据传输提供监控和报告功能。

客户面临的挑战

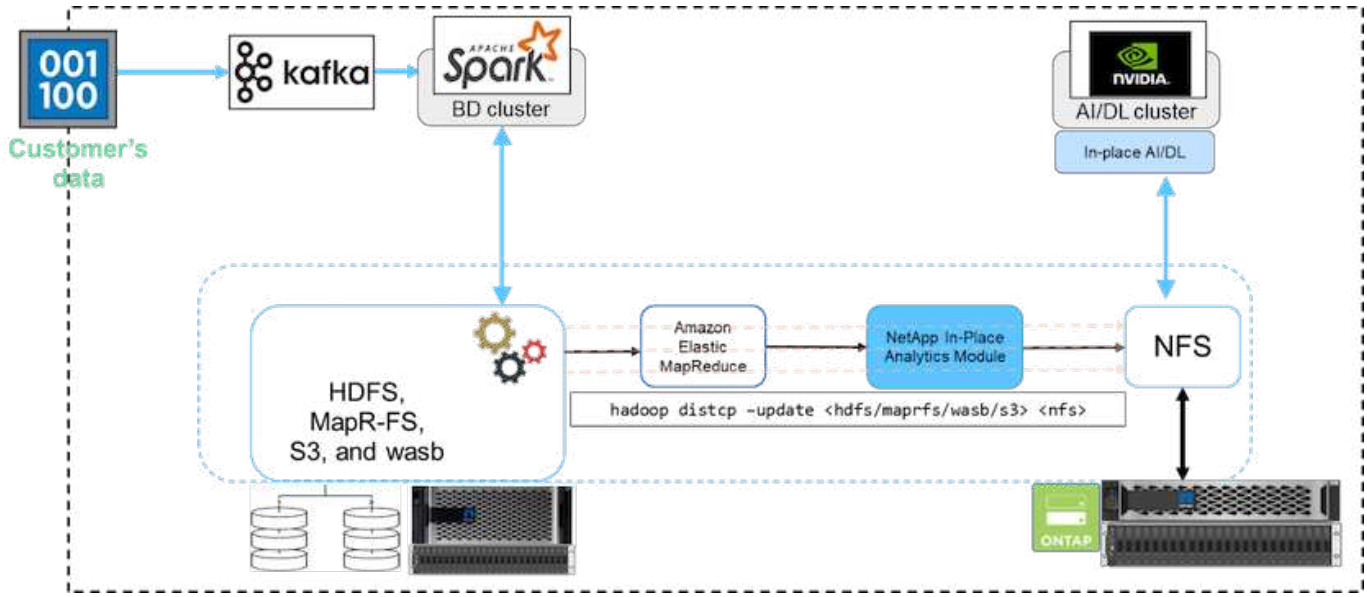
客户在尝试从 AI 运营的大数据分析中访问数据时可能会面临以下挑战：

- 客户数据位于数据湖存储库中。数据湖可以包含不同类型的数据，例如结构化数据，非结构化数据，半结构化数据，日志数据以及计算机到计算机数据。所有这些数据类型都必须在 AI 系统中进行处理。
- AI 与 Hadoop 文件系统不兼容。典型的 AI 架构无法直接访问 HDFS 和 HCFS 数据，必须将这些数据移至 AI 可理解的文件系统（ NFS ）。
- 将数据湖数据迁移到 AI 通常需要专门的流程。数据湖中的数据量可能非常大。客户必须采用高效，高吞吐量且经济高效的方式将数据迁移到 AI 系统中。
- 正在同步数据。如果客户希望在大数据平台和 AI 之间同步数据，有时通过 AI 处理的数据可以与大数据结合使用进行分析处理。

数据移动者解决方案

在大数据集群中，数据存储在 HDFS 或 HCFS 中，例如 MapR-FS ， Windows Azure Storage Blob ， S3 或 Google 文件系统。我们使用 HDFS ， MapR-FS 和 S3 作为源执行了测试，以便在 NIPAM 的帮助下，使用源中的 `Hadoop distcp` 命令将数据复制到 NetApp ONTAP NFS 导出。

下图显示了从运行 HDFS 存储的 Spark 集群到 NetApp ONTAP NFS 卷的典型数据移动，以便 NVIDIA 可以处理 AI 操作。



`hadoop distcp` 命令使用 MapReduce 程序复制数据。NIPAM 可与 MapReduce 结合使用，在复制数据时充当 Hadoop 集群的驱动程序。NIPAM 可以在多个网络接口之间分布负载，以便进行一次导出。在将数据从

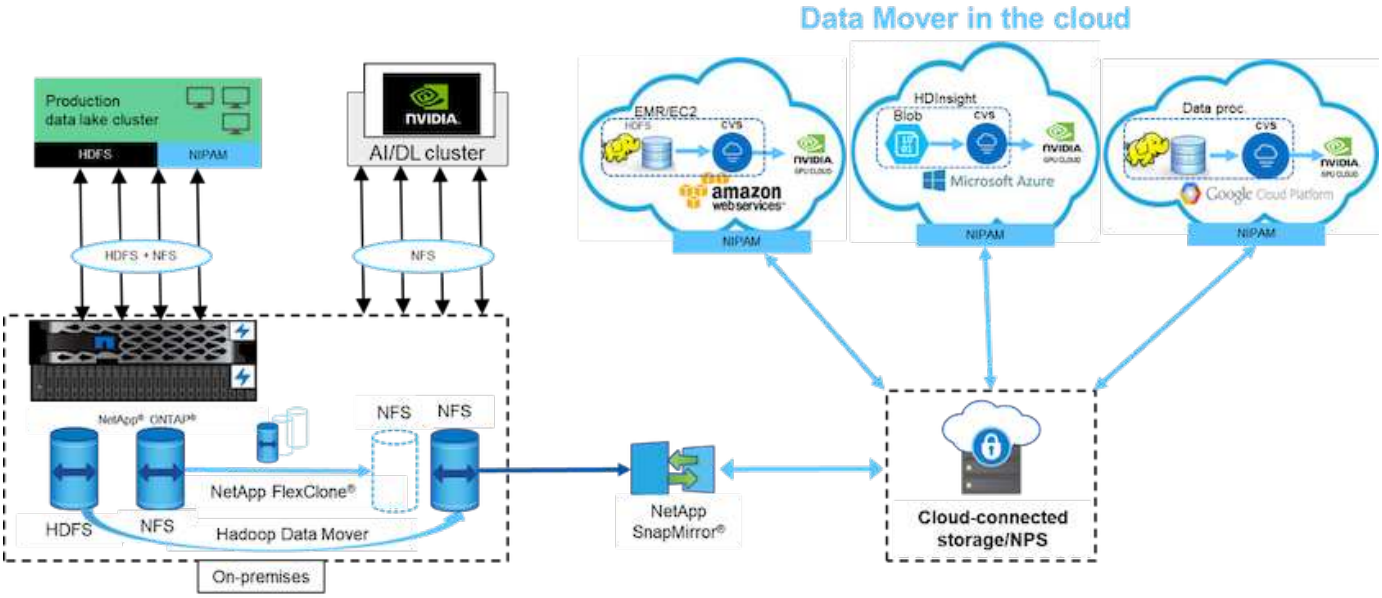
HDFS 或 HCFS 复制到 NFS 时，此过程会通过在多个网络接口之间分布数据来最大程度地提高网络吞吐量。

> 不支持 NIPAM ，也不会通过 MapR 认证。

适用于 AI 的数据移动工具解决方案

适用于 AI 的数据移动工具解决方案基于客户处理 AI 操作中的 Hadoop 数据的需求。NetApp 使用 NIPAM 将数据从 HDFS 移动到 NFS 。在一个使用情形中，客户需要将数据移动到内部的 NFS ，而另一客户则需要将数据从 Windows Azure 存储 Blob 移动到 Cloud Volumes Service ，以便处理云中 GPU 云实例中的数据。

下图显示了数据移动程序解决方案的详细信息。



要构建数据移动程序解决方案，需要执行以下步骤：

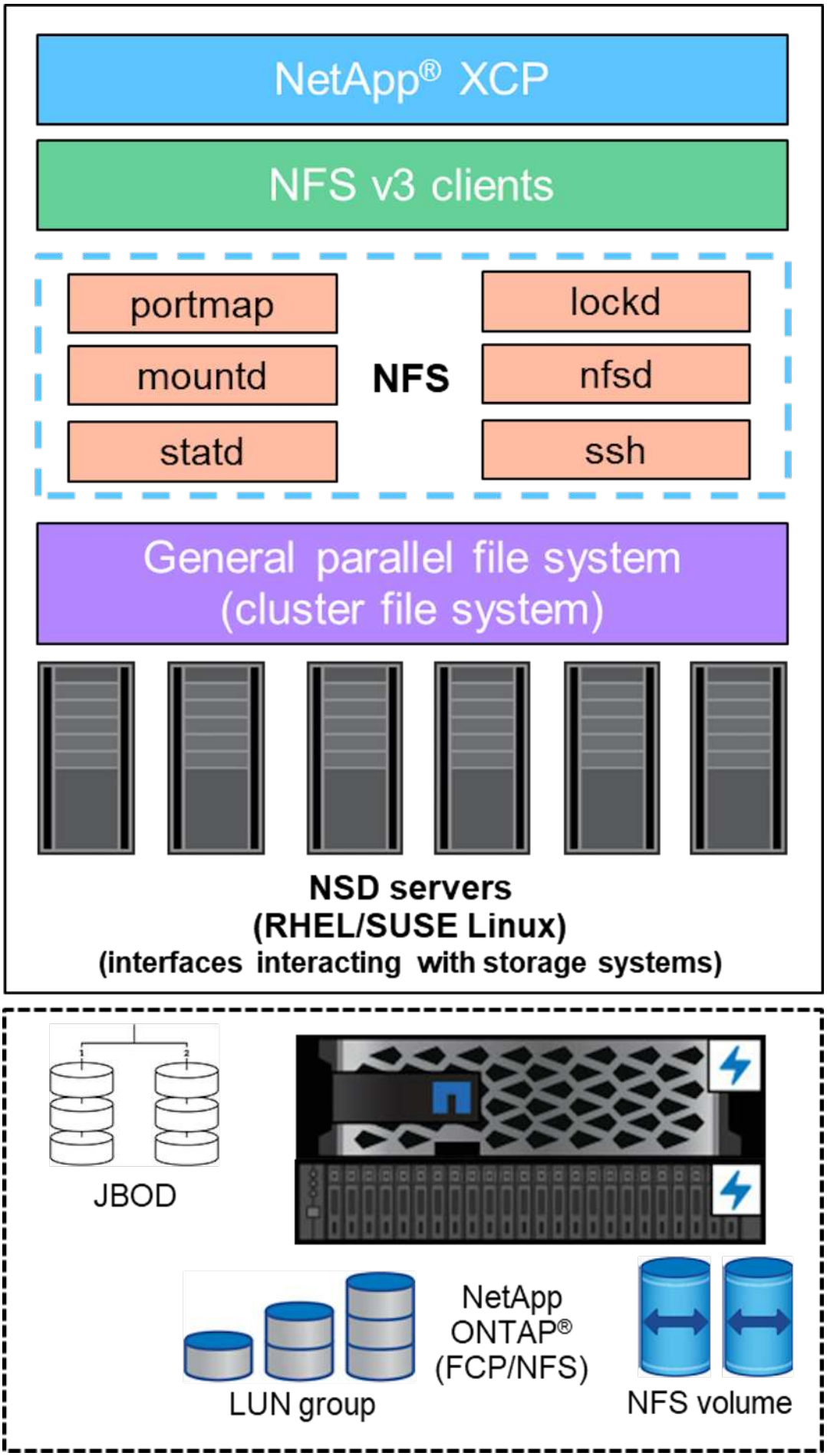
1. ONTAP SAN 可提供 HDFS， NAS 可通过 NIPAM 为生产数据湖集群提供 NFS 卷。
2. 客户的数据位于 HDFS 和 NFS 中。NFS 数据可以是来自其他应用程序的生产数据，用于大数据分析和 AI 操作。
3. NetApp FlexClone 技术可创建生产 NFS 卷的克隆并将其配置到内部的 AI 集群。
4. 使用 NIPAM 和 `Hadoop distcp` 命令将来自 HDFS SAN LUN 的数据复制到 NFS 卷中。NIPAM 使用多个网络接口的带宽传输数据。此过程可缩短数据复制时间，以便传输更多数据。
5. 这两个 NFS 卷都配置到 AI 集群以执行 AI 操作。
6. 要使用云中的 GPU 处理内部 NFS 数据， NFS 卷会采用 NetApp SnapMirror 技术镜像到 NetApp 私有存储（NPS），并挂载到云服务提供商的 GPU 。
7. 客户希望通过云服务提供商提供的 GPU 处理 EC2/EMR， HDInsight 或 DataProc 服务中的数据。Hadoop 数据移动工具可通过 NIPAM 和 `Hadoop distcp` 命令将数据从 Hadoop 服务移动到 Cloud Volumes 服务。
8. Cloud Volumes Service 数据通过 NFS 协议配置到 AI 。通过 AI 处理的数据除了通过 NIPAM， SnapMirror 和 NPS 发送到 NVIDIA 集群之外，还可以发送到内部位置进行大数据分析。

在这种情况下，客户在远程位置的 NAS 系统中有大量文件计数数据，这是在内部 NetApp 存储控制器上进行 AI 处理所需的。在这种情况下，最好使用 XCP 迁移工具以更快的速度迁移数据。

混合用例客户可以使用BlueXP Copy and Sync将内部数据从NFS、CIFS和S3数据迁移到云、反之亦然、以便通过NVIDIA集群中的GPU进行AI处理。BlueXP副本和同步以及XCP迁移工具均用于将NFS数据迁移到NetApp ONTAP NFS。

从 **GPF** 到 **NetApp ONTAP NFS**

在此验证中，我们使用四台服务器作为网络共享磁盘（ Network Shared Disk ， NSD ）服务器来为 GPFS 提供物理磁盘。在 NSD 磁盘上创建了 GPF ，以便将其导出为 NFS 导出，以便 NFS 客户端可以访问它们，如下图所示。我们使用 XCP 将数据从 GPFS 导出的 NFS 复制到 NetApp NFS 卷。


NetApp® XCP
NFS v3 clients
portmap
mountd
statd
NFS
lockd
nfsd
ssh
General parallel file system
(cluster file system)
NSD servers
(RHEL/SUSE Linux)
(interfaces interacting with storage systems)
JBOD
LUN group
NetApp
ONTAP®
(FCP/NFS)
NFS volume

GPF 要点

GPFS 中使用以下节点类型：

- * 管理节点。 * 指定一个可选字段，其中包含管理命令用于在节点之间进行通信的节点名称。例如，管理节点 `mastr-51.netapp.com` 可以将网络检查传递给集群中的所有其他节点。
- * 仲裁节点。 * 确定节点是否包含在从中派生仲裁的节点池中。您至少需要一个节点作为仲裁节点。
- * 管理器节点。 * 表示节点是否属于可从中选择文件系统管理器和令牌管理器的节点池。最好将多个节点定义为管理节点。您指定为管理器的节点数取决于工作负载和您拥有的 GPFS 服务器许可证数。如果您运行的是大型并行作业，则与支持 Web 应用程序的四节点集群相比，您可能需要更多的管理节点。
- * NSD 服务器。 * 用于准备每个物理磁盘以用于 GPFS 的服务器。
- * 协议节点。 * 通过任何安全 Shell （ SSH ）协议与 NFS 直接共享 GPFS 数据的节点。此节点需要 GPFS 服务器许可证。

GPFS ， NFS 和 XCP 的操作列表

本节提供了创建 GPFS ，将 GPFS 导出为 NFS 导出以及使用 XCP 传输数据的操作列表。

创建 GPFS

要创建 GPFS ，请完成以下步骤：

1. 在其中一台服务器上下载并安装 Linux 版本的频谱级数据访问。
2. 在所有节点中安装前提条件包（例如 chef ），并在所有节点中禁用安全增强型 Linux （ SELinux ）。
3. 设置安装节点并将管理节点和 GPFS 节点添加到集群定义文件中。
4. 添加管理器节点，仲裁节点， NSD 服务器和 GPFS 节点。
5. 添加 GUI ，管理和 GPFS 节点，并根据需要添加额外的 GUI 服务器。
6. 添加另一个 GPFS 节点并检查所有节点的列表。
7. 指定要在集群定义文件中的所有 GPFS 节点上设置的集群名称，配置文件，远程 shell 二进制文件，远程文件副本二进制文件和端口范围。
8. 查看 GPFS 配置设置并添加其他管理节点。
9. 禁用数据收集并将数据包上传到 IBM 支持中心。
10. 启用 NTP 并在安装之前预检查配置。
11. 配置，创建和检查 NSD 磁盘。
12. 创建 GPFS 。
13. 挂载 GPFS 。
14. 验证 GPFS 并为其提供所需的权限。
15. 运行 `dd` 命令，验证 GPFS 读写。

将 GPFS 导出到 NFS

要将 GPFS 导出到 NFS ，请完成以下步骤：

1. 通过 ` /etc/exports` 文件将 GPFS 导出为 NFS 。
2. 安装所需的 NFS 服务器软件包。
3. 启动 NFS 服务。
4. 列出 GPFS 中的文件以验证 NFS 客户端。

配置 **NFS** 客户端

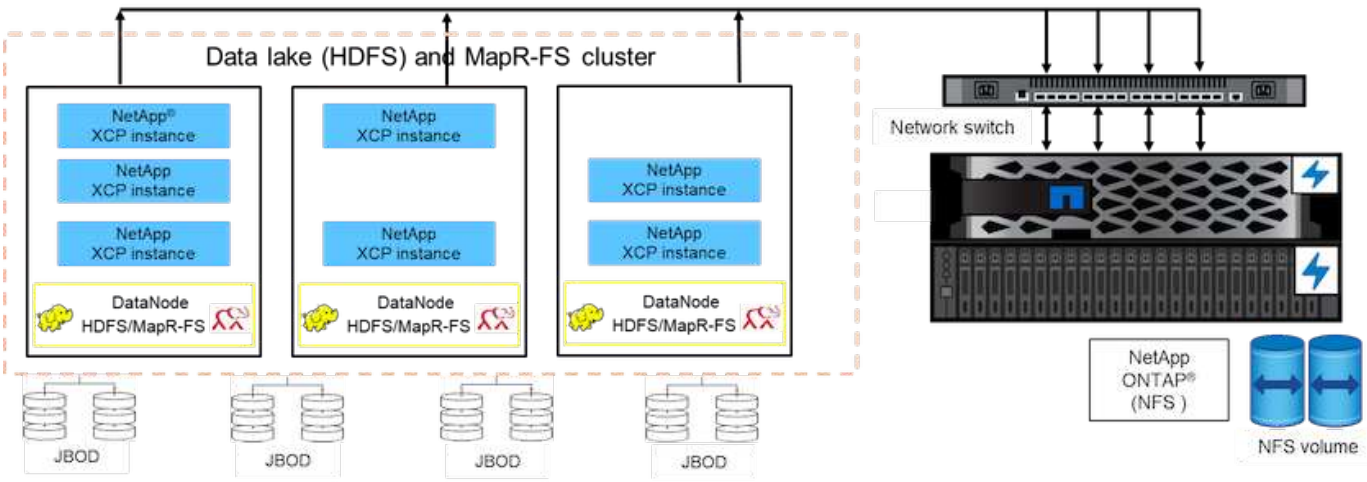
要配置 NFS 客户端，请完成以下步骤：

1. 通过 ` /etc/exports` 文件将 GPFS 导出为 NFS 。
2. 启动 NFS 客户端服务。
3. 通过 NFS 协议在 NFS 客户端上挂载 GPFS 。
4. 验证 NFS 挂载文件夹中的 GPFS 文件列表。
5. 使用 XCP 将数据从 GPFS 导出的 NFS 移动到 NetApp NFS 。
6. 验证 NFS 客户端上的 GPFS 文件。

HDFS 和 MapR-FS 到 ONTAP NFS

对于此解决方案， NetApp 验证了将数据从数据湖（ HDFS ）和 MapR 集群数据迁移到 ONTAP NFS 的过程。数据驻留在 MapR-FS 和 HDFS 中。NetApp XCP 引入了一项新功能，可将数据从分布式文件系统（例如 HDFS 和 MapR-FS ）直接迁移到 ONTAP NFS 。XCP 使用异步线程和 HDFS C API 调用从 MapR- FS 以及 HDFS 进行通信和传输数据。

下图显示了从数据湖（ HDFS ）和 MapR-FS 到 ONTAP NFS 的数据迁移。借助此新功能，您无需将源导出为 NFS 共享。



客户为什么要从 **HDFS** 和 **MapR-FS** 迁移到 **NFS** ？

大多数 Hadoop 分发软件包（例如 Cloudera 和 Hortonworks ）都使用 HDFS ，而 MapR 分发软件包使用自己的文件系统 Mapr-FS 来存储数据。HDFS 和 MapR-FS 数据为数据科学家提供了宝贵的见解，可用于机器学习（ ML ）和深度学习（ DL ）。HDFS 和 MapR-FS 中的数据不会共享，这意味着它不能由其他应用程序使用。

客户正在寻找共享数据，尤其是在银行领域，客户的敏感数据由多个应用程序使用。最新版本的 Hadoop （3.x 或更高版本）支持 NFS 数据源，无需其他第三方软件即可访问该数据源。借助新的 NetApp XCP 功能，可以将数据直接从 HDFS 和 MapR-FS 移动到 NetApp NFS ，以便访问多个应用程序

我们在 Amazon Web Services （AWS）中进行了测试，以便将数据从 MapR-FS 传输到 NFS ，以便对 12 个 MAPR 节点和 4 个 NFS 服务器进行初始性能测试。

	数量	**Size**	**vCPU**	内存	存储	网络
NFS 服务器	4.	i3en.24xlarge	96	488 GiB	8 个 7500 NVMe SSD	100
MapR 节点	12	I3en.12 个大型	48	384 GiB	4 个 7500 NVMe SSD	50

根据初始测试，我们获得了 20 Gbps 的吞吐量，并且能够每天传输 2 PB 的数据。

有关在不将 HDFS 导出到 NFS 的情况下进行 HDFS 数据迁移的详细信息，请参见中的 " 部署步骤 - NAS" 一节 "TR-4863： TR-4863：《 NetApp XCP 最佳实践指南—数据移动，文件迁移和分析》"。

业务优势

将数据从大数据分析迁移到 AI 具有以下优势：

- 能够从不同的 Hadoop 文件系统和 GPFS 中提取数据到统一的 NFS 存储系统中
- 一种与 Hadoop 集成的自动化数据传输方式
- 降低从 Hadoop 文件系统移动数据的库开发成本
- 通过使用 NIPAM 从单个数据源聚合多个网络接口的吞吐量实现最高性能
- 按计划和按需传输数据的方法
- 使用 ONTAP 数据管理软件为统一 NFS 数据提供存储效率和企业管理功能
- 使用 Hadoop 数据传输方法实现数据移动零成本

从 **GPF** 到 **NFS** 详细步骤

本节详细介绍了使用 NetApp XCP 配置 GPFS 并将数据移动到 NFS 所需的步骤。

配置 **GPFS**

1. 在其中一台服务器上下载并安装适用于 Linux 的 Spectrum Scale Data Access 。

```
[root@mastr-51 Spectrum_Scale_Data_Access-5.0.3.1-x86_64-Linux-
install_folder]# ls
Spectrum_Scale_Data_Access-5.0.3.1-x86_64-Linux-install
[root@mastr-51 Spectrum_Scale_Data_Access-5.0.3.1-x86_64-Linux-
install_folder]# chmod +x Spectrum_Scale_Data_Access-5.0.3.1-x86_64-
Linux-install
[root@mastr-51 Spectrum_Scale_Data_Access-5.0.3.1-x86_64-Linux-
install_folder]# ./Spectrum_Scale_Data_Access-5.0.3.1-x86_64-Linux-
install --manifest
manifest
…
<contents removes to save page space>
…
```

2. 在所有节点上安装前提条件包（包括 chef 和内核标头）。

```
[root@mastr-51 5.0.3.1]# for i in 51 53 136 138 140  ; do ssh
10.63.150.$i "hostname; rpm -ivh  /gpfs_install/chef* "; done
mastr-51.netapp.com
warning: /gpfs_install/chef-13.6.4-1.el7.x86_64.rpm: Header V4 DSA/SHA1
Signature, key ID 83ef826a: NOKEY
Preparing...
########################################
package chef-13.6.4-1.el7.x86_64 is already installed
mastr-53.netapp.com
warning: /gpfs_install/chef-13.6.4-1.el7.x86_64.rpm: Header V4 DSA/SHA1
Signature, key ID 83ef826a: NOKEY
Preparing...
########################################
Updating / installing...
chef-13.6.4-1.el7
########################################
Thank you for installing Chef!
workr-136.netapp.com
warning: /gpfs_install/chef-13.6.4-1.el7.x86_64.rpm: Header V4 DSA/SHA1
Signature, key ID 83ef826a: NOKEY
Preparing...
########################################
Updating / installing...
chef-13.6.4-1.el7
########################################
Thank you for installing Chef!
workr-138.netapp.com
warning: /gpfs_install/chef-13.6.4-1.el7.x86_64.rpm: Header V4 DSA/SHA1
```

```
Signature, key ID 83ef826a: NOKEY
Preparing...
########################################
Updating / installing...
chef-13.6.4-1.el7
########################################
Thank you for installing Chef!
workr-140.netapp.com
warning: /gpfs_install/chef-13.6.4-1.el7.x86_64.rpm: Header V4 DSA/SHA1
Signature, key ID 83ef826a: NOKEY
Preparing...
########################################
Updating / installing...
chef-13.6.4-1.el7
########################################
Thank you for installing Chef!
[root@mastr-51 5.0.3.1]#
[root@mastr-51 installer]# for i in 51 53 136 138 140  ; do ssh
10.63.150.$i "hostname; yumdownloader kernel-headers-3.10.0-
862.3.2.el7.x86_64 ; rpm -Uvh --oldpackage kernel-headers-3.10.0-
862.3.2.el7.x86_64.rpm"; done
mastr-51.netapp.com
Loaded plugins: priorities, product-id, subscription-manager
Preparing...
########################################
Updating / installing...
kernel-headers-3.10.0-862.3.2.el7
########################################
Cleaning up / removing...
kernel-headers-3.10.0-957.21.2.el7
########################################
mastr-53.netapp.com
Loaded plugins: product-id, subscription-manager
Preparing...
########################################
Updating / installing...
kernel-headers-3.10.0-862.3.2.el7
########################################
Cleaning up / removing...
kernel-headers-3.10.0-862.11.6.el7
########################################
workr-136.netapp.com
Loaded plugins: product-id, subscription-manager
Repository ambari-2.7.3.0 is listed more than once in the configuration
Preparing...
########################################
```

```
Updating / installing...
kernel-headers-3.10.0-862.3.2.el7
##########################################
Cleaning up / removing...
kernel-headers-3.10.0-862.11.6.el7
##########################################
workr-138.netapp.com
Loaded plugins: product-id, subscription-manager
Preparing...
##########################################
package kernel-headers-3.10.0-862.3.2.el7.x86_64 is already installed
workr-140.netapp.com
Loaded plugins: product-id, subscription-manager
Preparing...
##########################################
Updating / installing...
kernel-headers-3.10.0-862.3.2.el7
##########################################
Cleaning up / removing...
kernel-headers-3.10.0-862.11.6.el7
##########################################
[root@mastr-51 installer]#
```

3. 在所有节点中禁用 SELinux 。

```
[root@mastr-51 5.0.3.1]# for i in 51 53 136 138 140  ; do ssh
10.63.150.$i "hostname; sudo setenforce 0"; done
mastr-51.netapp.com
setenforce: SELinux is disabled
mastr-53.netapp.com
setenforce: SELinux is disabled
workr-136.netapp.com
setenforce: SELinux is disabled
workr-138.netapp.com
setenforce: SELinux is disabled
workr-140.netapp.com
setenforce: SELinux is disabled
[root@mastr-51 5.0.3.1]#
```

4. 设置安装节点。

```
[root@mastr-51 installer]# ./spectrumscale setup -s 10.63.150.51
[ INFO  ] Installing prerequisites for install node
[ INFO  ] Existing Chef installation detected. Ensure the PATH is
configured so that chef-client and knife commands can be run.
[ INFO  ] Your control node has been configured to use the IP
10.63.150.51 to communicate with other nodes.
[ INFO  ] Port 8889 will be used for chef communication.
[ INFO  ] Port 10080 will be used for package distribution.
[ INFO  ] Install Toolkit setup type is set to Spectrum Scale (default).
If an ESS is in the cluster, run this command to set ESS mode:
./spectrumscale setup -s server_ip -st ess
[ INFO  ] SUCCESS
[ INFO  ] Tip : Designate protocol, nsd and admin nodes in your
environment to use during install:./spectrumscale -v node add <node> -p
-a -n
[root@mastr-51 installer]#
```

5. 将管理节点和 GPFS 节点添加到集群定义文件中。

```
[root@mastr-51 installer]# ./spectrumscale node add mastr-51 -a
[ INFO  ] Adding node mastr-51.netapp.com as a GPFS node.
[ INFO  ] Setting mastr-51.netapp.com as an admin node.
[ INFO  ] Configuration updated.
[ INFO  ] Tip : Designate protocol or nsd nodes in your environment to
use during install:./spectrumscale node add <node> -p -n
[root@mastr-51 installer]#
```

6. 添加管理器节点和 GPFS 节点。

```
[root@mastr-51 installer]# ./spectrumscale node add mastr-53 -m
[ INFO  ] Adding node mastr-53.netapp.com as a GPFS node.
[ INFO  ] Adding node mastr-53.netapp.com as a manager node.
[root@mastr-51 installer]#
```

7. 添加仲裁节点和 GPFS 节点。

```
[root@mastr-51 installer]# ./spectrumscale node add workr-136 -q
[ INFO  ] Adding node workr-136.netapp.com as a GPFS node.
[ INFO  ] Adding node workr-136.netapp.com as a quorum node.
[root@mastr-51 installer]#
```

8. 添加 NSD 服务器和 GPFS 节点。

```
[root@mastr-51 installer]# ./spectrumscale node add workr-138 -n
[ INFO  ] Adding node workr-138.netapp.com as a GPFS node.
[ INFO  ] Adding node workr-138.netapp.com as an NSD server.
[ INFO  ] Configuration updated.
[ INFO  ] Tip :If all node designations are complete, add NSDs to your
cluster definition and define required filessytems:./spectrumscale nsd
add <device> -p <primary node> -s <secondary node> -fs <file system>
[root@mastr-51 installer]#
```

9. 添加 GUI ，管理和 GPFS 节点。

```
[root@mastr-51 installer]# ./spectrumscale node add workr-136 -g
[ INFO  ] Setting workr-136.netapp.com as a GUI server.
[root@mastr-51 installer]# ./spectrumscale node add workr-136 -a
[ INFO  ] Setting workr-136.netapp.com as an admin node.
[ INFO  ] Configuration updated.
[ INFO  ] Tip : Designate protocol or nsd nodes in your environment to
use during install:./spectrumscale node add <node> -p -n
[root@mastr-51 installer]#
```

10. 添加另一个 GUI 服务器。

```
[root@mastr-51 installer]# ./spectrumscale node add mastr-53 -g
[ INFO  ] Setting mastr-53.netapp.com as a GUI server.
[root@mastr-51 installer]#
```

11. 添加另一个 GPFS 节点。

```
[root@mastr-51 installer]# ./spectrumscale node add workr-140
[ INFO  ] Adding node workr-140.netapp.com as a GPFS node.
[root@mastr-51 installer]#
```

12. 验证并列出所有节点。

```
[root@mastr-51 installer]# ./spectrumscale node list
[ INFO  ] List of nodes in current configuration:
[ INFO  ] [Installer Node]
[ INFO  ] 10.63.150.51
[ INFO  ]
[ INFO  ] [Cluster Details]
[ INFO  ] No cluster name configured
[ INFO  ] Setup Type: Spectrum Scale
[ INFO  ]
[ INFO  ] [Extended Features]
[ INFO  ] File Audit logging     : Disabled
[ INFO  ] Watch folder           : Disabled
[ INFO  ] Management GUI         : Enabled
[ INFO  ] Performance Monitoring : Disabled
[ INFO  ] Callhome               : Enabled
[ INFO  ]
[ INFO  ] GPFS                 Admin  Quorum  Manager   NSD   Protocol
GUI   Callhome   OS   Arch
[ INFO  ] Node                  Node   Node     Node   Server   Node
Server  Server
[ INFO  ] mastr-51.netapp.com    X
rhel7  x86_64
[ INFO  ] mastr-53.netapp.com                    X
X             rhel7  x86_64
[ INFO  ] workr-136.netapp.com   X       X
X             rhel7  x86_64
[ INFO  ] workr-138.netapp.com                          X
rhel7  x86_64
[ INFO  ] workr-140.netapp.com
rhel7  x86_64
[ INFO  ]
[ INFO  ] [Export IP address]
[ INFO  ] No export IP addresses configured
[root@mastr-51 installer]#
```

13. 在集群定义文件中指定集群名称。

```
[root@mastr-51 installer]# ./spectrumscale config gpfs -c mastr-
51.netapp.com
[ INFO  ] Setting GPFS cluster name to mastr-51.netapp.com
[root@mastr-51 installer]#
```

14. 指定配置文件。

```
[root@mastr-51 installer]# ./spectrumscale config gpfs -p default
[ INFO  ] Setting GPFS profile to default
[root@mastr-51 installer]#
Profiles options: default [gpfsProtocolDefaults], random I/O
[gpfsProtocolsRandomIO], sequential I/O [gpfsProtocolDefaults], random
I/O [gpfsProtocolRandomIO]
```

15. 指定 GPFS 要使用的远程 shell 二进制文件；使用 ` -r 参数` 。

```
[root@mastr-51 installer]# ./spectrumscale config gpfs -r /usr/bin/ssh
[ INFO  ] Setting Remote shell command to /usr/bin/ssh
[root@mastr-51 installer]#
```

16. 指定 GPFS 要使用的远程文件副本二进制文件；使用 ` -rc 参数` 。

```
[root@mastr-51 installer]# ./spectrumscale config gpfs -rc /usr/bin/scp
[ INFO  ] Setting Remote file copy command to /usr/bin/scp
[root@mastr-51 installer]#
```

17. 指定要在所有 GPFS 节点上设置的端口范围；使用 ` -e 参数` 。

```
[root@mastr-51 installer]# ./spectrumscale config gpfs -e 60000-65000
[ INFO  ] Setting GPFS Daemon communication port range to 60000-65000
[root@mastr-51 installer]#
```

18. 查看 GPFS 配置设置。

```
[root@mastr-51 installer]# ./spectrumscale config gpfs --list
[ INFO  ] Current settings are as follows:
[ INFO  ] GPFS cluster name is mastr-51.netapp.com.
[ INFO  ] GPFS profile is default.
[ INFO  ] Remote shell command is /usr/bin/ssh.
[ INFO  ] Remote file copy command is /usr/bin/scp.
[ INFO  ] GPFS Daemon communication port range is 60000-65000.
[root@mastr-51 installer]#
```

19. 添加管理节点。

```
[root@mastr-51 installer]# ./spectrumscale node add 10.63.150.53 -a
[ INFO  ] Setting mastr-53.netapp.com as an admin node.
[ INFO  ] Configuration updated.
[ INFO  ] Tip : Designate protocol or nsd nodes in your environment to
use during install:./spectrumscale node add <node> -p -n
[root@mastr-51 installer]#
```

20. 禁用数据收集并将数据包上传到 IBM 支持中心。

```
[root@mastr-51 installer]# ./spectrumscale callhome disable
[ INFO  ] Disabling the callhome.
[ INFO  ] Configuration updated.
[root@mastr-51 installer]#
```

21. 启用 NTP 。

```
[root@mastr-51 installer]# ./spectrumscale config ntp -e on
[root@mastr-51 installer]# ./spectrumscale config ntp -l
[ INFO  ] Current settings are as follows:
[ WARN  ] No value for Upstream NTP Servers(comma separated IP's with NO
space between multiple IPs) in clusterdefinition file.
[root@mastr-51 installer]# ./spectrumscale config ntp -s 10.63.150.51
[ WARN  ] The NTP package must already be installed and full
bidirectional access to the UDP port 123 must be allowed.
[ WARN  ] If NTP is already running on any of your nodes, NTP setup will
be skipped. To stop NTP run 'service ntpd stop'.
[ WARN  ] NTP is already on
[ INFO  ] Setting Upstream NTP Servers(comma separated IP's with NO
space between multiple IPs) to 10.63.150.51
[root@mastr-51 installer]# ./spectrumscale config ntp -e on
[ WARN  ] NTP is already on
[root@mastr-51 installer]# ./spectrumscale config ntp -l
[ INFO  ] Current settings are as follows:
[ INFO  ] Upstream NTP Servers(comma separated IP's with NO space
between multiple IPs) is 10.63.150.51.
[root@mastr-51 installer]#

[root@mastr-51 installer]# service ntpd start
Redirecting to /bin/systemctl start ntpd.service
[root@mastr-51 installer]# service ntpd status
Redirecting to /bin/systemctl status ntpd.service
● ntpd.service - Network Time Service
   Loaded: loaded (/usr/lib/systemd/system/ntpd.service; enabled; vendor
```

```
preset: disabled)
   Active: active (running) since Tue 2019-09-10 14:20:34 UTC; 1s ago
  Process: 2964 ExecStart=/usr/sbin/ntpd -u ntp:ntp $OPTIONS
(code=exited, status=0/SUCCESS)
 Main PID: 2965 (ntpd)
   CGroup: /system.slice/ntpd.service
           └─2965 /usr/sbin/ntpd -u ntp:ntp -g

Sep 10 14:20:34 mastr-51.netapp.com ntpd[2965]: ntp_io: estimated max
descriptors: 1024, initial socket boundary: 16
Sep 10 14:20:34 mastr-51.netapp.com ntpd[2965]: Listen and drop on 0
v4wildcard 0.0.0.0 UDP 123
Sep 10 14:20:34 mastr-51.netapp.com ntpd[2965]: Listen and drop on 1
v6wildcard :: UDP 123
Sep 10 14:20:34 mastr-51.netapp.com ntpd[2965]: Listen normally on 2 lo
127.0.0.1 UDP 123
Sep 10 14:20:34 mastr-51.netapp.com ntpd[2965]: Listen normally on 3
enp4s0f0 10.63.150.51 UDP 123
Sep 10 14:20:34 mastr-51.netapp.com ntpd[2965]: Listen normally on 4 lo
::1 UDP 123
Sep 10 14:20:34 mastr-51.netapp.com ntpd[2965]: Listen normally on 5
enp4s0f0 fe80::219:99ff:feef:99fa UDP 123
Sep 10 14:20:34 mastr-51.netapp.com ntpd[2965]: Listening on routing
socket on fd #22 for interface updates
Sep 10 14:20:34 mastr-51.netapp.com ntpd[2965]: 0.0.0.0 c016 06 restart
Sep 10 14:20:34 mastr-51.netapp.com ntpd[2965]: 0.0.0.0 c012 02 freq_set
kernel 11.890 PPM
[root@mastr-51 installer]#
```

22. 安装前预检查配置。

```
[root@mastr-51 installer]# ./spectrumscale install -pr
[ INFO  ] Logging to file: /usr/lpp/mmfs/5.0.3.1/installer/logs/INSTALL-
PRECHECK-10-09-2019_14:51:43.log
[ INFO  ] Validating configuration
[ INFO  ] Performing Chef (deploy tool) checks.
[ WARN  ] NTP is already running on: mastr-51.netapp.com. The install
toolkit will no longer setup NTP.
[ INFO  ] Node(s): ['workr-138.netapp.com'] were defined as NSD node(s)
but the toolkit has not been told about any NSDs served by these node(s)
nor has the toolkit been told to create new NSDs on these node(s). The
install will continue and these nodes will be assigned server licenses.
If NSDs are desired, either add them to the toolkit with
<./spectrumscale nsd add> followed by a <./spectrumscale install> or add
them manually afterwards using mmcrnsd.
[ INFO  ] Install toolkit will not configure file audit logging as it
has been disabled.
[ INFO  ] Install toolkit will not configure watch folder as it has been
disabled.
[ INFO  ] Checking for knife bootstrap configuration...
[ INFO  ] Performing GPFS checks.
[ INFO  ] Running environment checks
[ INFO  ] Skipping license validation as no existing GPFS cluster
detected.
[ INFO  ] Checking pre-requisites for portability layer.
[ INFO  ] GPFS precheck OK
[ INFO  ] Performing Performance Monitoring checks.
[ INFO  ] Running environment checks for Performance Monitoring
[ INFO  ] Performing GUI checks.
[ INFO  ] Performing FILE AUDIT LOGGING checks.
[ INFO  ] Running environment checks for file  Audit logging
[ INFO  ] Network check from admin node workr-136.netapp.com to all
other nodes in the cluster passed
[ INFO  ] Network check from admin node mastr-51.netapp.com to all other
nodes in the cluster passed
[ INFO  ] Network check from admin node mastr-53.netapp.com to all other
nodes in the cluster passed
[ INFO  ] The install toolkit will not configure call home as it is
disabled. To enable call home, use the following CLI command:
./spectrumscale callhome enable
[ INFO  ] Pre-check successful for install.
[ INFO  ] Tip : ./spectrumscale install
[root@mastr-51 installer]#
```

23. 配置 NSD 磁盘。

```
[root@mastr-51 cluster-test]# cat disk.1st
%nsd: device=/dev/sdf
nsd=nsd1
servers=workr-136
usage=dataAndMetadata
failureGroup=1

%nsd: device=/dev/sdf
nsd=nsd2
servers=workr-138
usage=dataAndMetadata
failureGroup=1
```

24. 创建 NSD 磁盘。

```
[root@mastr-51 cluster-test]# mmcrnsd -F disk.1st -v no
mmcrnsd: Processing disk sdf
mmcrnsd: Processing disk sdf
mmcrnsd: Propagating the cluster configuration data to all
  affected nodes.  This is an asynchronous process.
[root@mastr-51 cluster-test]#
```

25. 检查 NSD 磁盘状态。

```
[root@mastr-51 cluster-test]# mmlsnsd

 File system   Disk name    NSD servers
---------------------------------------------------------------------------
---
 (free disk)   nsd1         workr-136.netapp.com
 (free disk)   nsd2         workr-138.netapp.com

[root@mastr-51 cluster-test]#
```

26. 创建 GPFS 。

```
[root@mastr-51 cluster-test]# mmcrfs gpfs1 -F disk.1st -B 1M -T /gpfs1

The following disks of gpfs1 will be formatted on node workr-
136.netapp.com:
    nsd1: size 3814912 MB
    nsd2: size 3814912 MB
Formatting file system ...
Disks up to size 33.12 TB can be added to storage pool system.
Creating Inode File
Creating Allocation Maps
Creating Log Files
Clearing Inode Allocation Map
Clearing Block Allocation Map
Formatting Allocation Map for storage pool system
Completed creation of file system /dev/gpfs1.
mmcrfs: Propagating the cluster configuration data to all
  affected nodes.  This is an asynchronous process.
[root@mastr-51 cluster-test]#
```

27. 挂载 GPFS 。

```
[root@mastr-51 cluster-test]# mmmount all -a
Tue Oct  8 18:05:34 UTC 2019: mmmount: Mounting file systems ...
[root@mastr-51 cluster-test]#
```

28. 检查 GPFS 并为其提供所需的权限。

```
[root@mastr-51 cluster-test]# mmlsdisk gpfs1
disk         driver   sector     failure holds    holds
storage
name         type       size       group metadata data  status
availability pool
------------ -------- ------ ----------- -------- ----- -------------
------------ ------------
nsd1         nsd         512           1 Yes      Yes   ready         up
system
nsd2         nsd         512           1 Yes      Yes   ready         up
system
[root@mastr-51 cluster-test]#

[root@mastr-51 cluster-test]# for i in 51 53 136 138  ; do ssh
10.63.150.$i "hostname; chmod 777 /gpfs1" ; done;
mastr-51.netapp.com
mastr-53.netapp.com
workr-136.netapp.com
workr-138.netapp.com
[root@mastr-51 cluster-test]#
```

29. 运行 dd 命令检查 GPFS 读写。

```
[root@mastr-51 cluster-test]# dd if=/dev/zero of=/gpfs1/testfile
bs=1024M count=5
5+0 records in
5+0 records out
5368709120 bytes (5.4 GB) copied, 8.3981 s, 639 MB/s
[root@mastr-51 cluster-test]# for i in 51 53 136 138  ; do ssh
10.63.150.$i "hostname; ls -ltrh /gpfs1" ; done;
mastr-51.netapp.com
total 5.0G
-rw-r--r-- 1 root root 5.0G Oct  8 18:10 testfile
mastr-53.netapp.com
total 5.0G
-rw-r--r-- 1 root root 5.0G Oct  8 18:10 testfile
workr-136.netapp.com
total 5.0G
-rw-r--r-- 1 root root 5.0G Oct  8 18:10 testfile
workr-138.netapp.com
total 5.0G
-rw-r--r-- 1 root root 5.0G Oct  8 18:10 testfile
[root@mastr-51 cluster-test]#
```

将 GPFS 导出到 NFS

要将 GPFS 导出到 NFS ，请完成以下步骤：

1. 通过 ` /etc/exports` 文件将 GPFS 导出为 NFS 。

```
[root@mastr-51 gpfs1]# cat /etc/exports
/gpfs1        *(rw,fsid=745)
[root@mastr-51 gpfs1]
```

2. 安装所需的 NFS 服务器软件包。

```
[root@mastr-51 ~]# yum install rpcbind
Loaded plugins: priorities, product-id, search-disabled-repos,
subscription-manager
Resolving Dependencies
--> Running transaction check
---> Package rpcbind.x86_64 0:0.2.0-47.el7 will be updated
---> Package rpcbind.x86_64 0:0.2.0-48.el7 will be an update
--> Finished Dependency Resolution

Dependencies Resolved

=================================================================
=================================================================
=================================================================
======================
 Package                                                Arch
Version                                                     Repository
Size
=================================================================
=================================================================
=================================================================
======================
Updating:
 rpcbind                                                x86_64
0.2.0-48.el7                                                rhel-7-
server-rpms                                                 60 k

Transaction Summary
=================================================================
=================================================================
=================================================================
======================
Upgrade  1 Package
```

```
Total download size: 60 k
Is this ok [y/d/N]: y
Downloading packages:
No Presto metadata available for rhel-7-server-rpms
rpcbind-0.2.0-48.el7.x86_64.rpm
|  60 kB  00:00:00
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
  Updating   : rpcbind-0.2.0-48.el7.x86_64
1/2
  Cleanup    : rpcbind-0.2.0-47.el7.x86_64
2/2
  Verifying  : rpcbind-0.2.0-48.el7.x86_64
1/2
  Verifying  : rpcbind-0.2.0-47.el7.x86_64
2/2

Updated:
  rpcbind.x86_64 0:0.2.0-48.el7

Complete!
[root@mastr-51 ~]#
```

3. 启动 NFS 服务。

```
[root@mastr-51 ~]# service nfs status
Redirecting to /bin/systemctl status nfs.service
● nfs-server.service - NFS server and services
   Loaded: loaded (/usr/lib/systemd/system/nfs-server.service; disabled;
vendor preset: disabled)
  Drop-In: /run/systemd/generator/nfs-server.service.d
           └─order-with-mounts.conf
   Active: inactive (dead)
[root@mastr-51 ~]# service rpcbind start
Redirecting to /bin/systemctl start rpcbind.service
[root@mastr-51 ~]# service nfs start
Redirecting to /bin/systemctl start nfs.service
[root@mastr-51 ~]# service nfs status
Redirecting to /bin/systemctl status nfs.service
● nfs-server.service - NFS server and services
   Loaded: loaded (/usr/lib/systemd/system/nfs-server.service; disabled;
vendor preset: disabled)
  Drop-In: /run/systemd/generator/nfs-server.service.d
           └─order-with-mounts.conf
   Active: active (exited) since Wed 2019-11-06 16:34:50 UTC; 2s ago
  Process: 24402 ExecStartPost=/bin/sh -c if systemctl -q is-active
gssproxy; then systemctl reload gssproxy ; fi (code=exited,
status=0/SUCCESS)
  Process: 24383 ExecStart=/usr/sbin/rpc.nfsd $RPCNFSDARGS (code=exited,
status=0/SUCCESS)
  Process: 24379 ExecStartPre=/usr/sbin/exportfs -r (code=exited,
status=0/SUCCESS)
 Main PID: 24383 (code=exited, status=0/SUCCESS)
   CGroup: /system.slice/nfs-server.service

Nov 06 16:34:50 mastr-51.netapp.com systemd[1]: Starting NFS server and
services...
Nov 06 16:34:50 mastr-51.netapp.com systemd[1]: Started NFS server and
services.
[root@mastr-51 ~]#
```

4. 列出 GPFS 中的文件以验证 NFS 客户端。

```
[root@mastr-51 gpfs1]# df -Th
Filesystem                                  Type      Size  Used Avail
Use% Mounted on
/dev/mapper/rhel_stlrx300s6--22--irmc-root xfs        94G   55G   39G
59% /
devtmpfs                                    devtmpfs   32G     0   32G
0% /dev
tmpfs                                       tmpfs      32G     0   32G
0% /dev/shm
tmpfs                                       tmpfs      32G  3.3G   29G
11% /run
tmpfs                                       tmpfs      32G     0   32G
0% /sys/fs/cgroup
/dev/sda7                                   xfs       9.4G  210M  9.1G
3% /boot
tmpfs                                       tmpfs     6.3G     0  6.3G
0% /run/user/10065
tmpfs                                       tmpfs     6.3G     0  6.3G
0% /run/user/10068
tmpfs                                       tmpfs     6.3G     0  6.3G
0% /run/user/10069
10.63.150.213:/nc_volume3                   nfs4      380G  8.0M  380G
1% /mnt
tmpfs                                       tmpfs     6.3G     0  6.3G
0% /run/user/0
gpfs1                                       gpfs      7.3T  9.1G  7.3T
1% /gpfs1
[root@mastr-51 gpfs1]#
[root@mastr-51 ~]# cd /gpfs1
[root@mastr-51 gpfs1]# ls
catalog  ces  gpfs-ces  ha  testfile
[root@mastr-51 gpfs1]#
[root@mastr-51 ~]# cd /gpfs1
[root@mastr-51 gpfs1]# ls
ces  gpfs-ces  ha  testfile
[root@mastr-51 gpfs1]# ls -ltrha
total 5.1G
dr-xr-xr-x   2 root root 8.0K Jan  1  1970 .snapshots
-rw-r--r--   1 root root 5.0G Oct  8 18:10 testfile
dr-xr-xr-x. 30 root root 4.0K Oct  8 18:19 ..
drwxr-xr-x   2 root root 4.0K Nov  5 20:02 gpfs-ces
drwxr-xr-x   2 root root 4.0K Nov  5 20:04 ha
drwxrwxrwx   5 root root 256K Nov  5 20:04 .
drwxr-xr-x   4 root root 4.0K Nov  5 20:35 ces
[root@mastr-51 gpfs1]#
```

配置 **NFS** 客户端

要配置 NFS 客户端，请完成以下步骤：

1. 在 NFS 客户端中安装软件包。

```
[root@hdp2 ~]# yum install nfs-utils rpcbind
Loaded plugins: product-id, search-disabled-repos, subscription-manager
HDP-2.6-GPL-repo-4
| 2.9 kB  00:00:00
HDP-2.6-repo-4
| 2.9 kB  00:00:00
HDP-3.0-GPL-repo-2
| 2.9 kB  00:00:00
HDP-3.0-repo-2
| 2.9 kB  00:00:00
HDP-3.0-repo-3
| 2.9 kB  00:00:00
HDP-3.1-repo-1
| 2.9 kB  00:00:00
HDP-3.1-repo-51
| 2.9 kB  00:00:00
HDP-UTILS-1.1.0.22-repo-1
| 2.9 kB  00:00:00
HDP-UTILS-1.1.0.22-repo-2
| 2.9 kB  00:00:00
HDP-UTILS-1.1.0.22-repo-3
| 2.9 kB  00:00:00
HDP-UTILS-1.1.0.22-repo-4
| 2.9 kB  00:00:00
HDP-UTILS-1.1.0.22-repo-51
| 2.9 kB  00:00:00
ambari-2.7.3.0
| 2.9 kB  00:00:00
epel/x86_64/metalink
|  13 kB  00:00:00
epel
| 5.3 kB  00:00:00
mysql-connectors-community
| 2.5 kB  00:00:00
mysql-tools-community
| 2.5 kB  00:00:00
mysql56-community
| 2.5 kB  00:00:00
rhel-7-server-optional-rpms
| 3.2 kB  00:00:00
```

```
rhel-7-server-rpms
| 3.5 kB  00:00:00
(1/10): mysql-connectors-community/x86_64/primary_db
|  49 kB  00:00:00
(2/10): mysql-tools-community/x86_64/primary_db
|  66 kB  00:00:00
(3/10): epel/x86_64/group_gz
|  90 kB  00:00:00
(4/10): mysql56-community/x86_64/primary_db
| 241 kB  00:00:00
(5/10): rhel-7-server-optional-rpms/7Server/x86_64/updateinfo
| 2.5 MB  00:00:00
(6/10): rhel-7-server-rpms/7Server/x86_64/updateinfo
| 3.4 MB  00:00:00
(7/10): rhel-7-server-optional-rpms/7Server/x86_64/primary_db
| 8.3 MB  00:00:00
(8/10): rhel-7-server-rpms/7Server/x86_64/primary_db
|  62 MB  00:00:01
(9/10): epel/x86_64/primary_db
| 6.9 MB  00:00:08
(10/10): epel/x86_64/updateinfo
| 1.0 MB  00:00:13
Resolving Dependencies
--> Running transaction check
---> Package nfs-utils.x86_64 1:1.3.0-0.61.el7 will be updated
---> Package nfs-utils.x86_64 1:1.3.0-0.65.el7 will be an update
---> Package rpcbind.x86_64 0:0.2.0-47.el7 will be updated
---> Package rpcbind.x86_64 0:0.2.0-48.el7 will be an update
--> Finished Dependency Resolution

Dependencies Resolved

=========================================================================
===============================================
 Package                 Arch                 Version
Repository                         Size
=========================================================================
===============================================
Updating:
 nfs-utils               x86_64               1:1.3.0-0.65.el7
rhel-7-server-rpms               412 k
 rpcbind                 x86_64               0.2.0-48.el7
rhel-7-server-rpms                60 k

Transaction Summary
=========================================================================
```

```
================================================
Upgrade  2 Packages

Total download size: 472 k
Is this ok [y/d/N]: y
Downloading packages:
No Presto metadata available for rhel-7-server-rpms
(1/2): rpcbind-0.2.0-48.el7.x86_64.rpm
|  60 kB  00:00:00
(2/2): nfs-utils-1.3.0-0.65.el7.x86_64.rpm
| 412 kB  00:00:00
--------------------------------------------------------------------------
-----------------------------------------------
Total
1.2 MB/s | 472 kB  00:00:00
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
  Updating   : rpcbind-0.2.0-48.el7.x86_64
1/4
service rpcbind start

  Updating   : 1:nfs-utils-1.3.0-0.65.el7.x86_64
2/4
  Cleanup    : 1:nfs-utils-1.3.0-0.61.el7.x86_64
3/4
  Cleanup    : rpcbind-0.2.0-47.el7.x86_64
4/4
  Verifying  : 1:nfs-utils-1.3.0-0.65.el7.x86_64
1/4
  Verifying  : rpcbind-0.2.0-48.el7.x86_64
2/4
  Verifying  : rpcbind-0.2.0-47.el7.x86_64
3/4
  Verifying  : 1:nfs-utils-1.3.0-0.61.el7.x86_64
4/4

Updated:
  nfs-utils.x86_64 1:1.3.0-0.65.el7
rpcbind.x86_64 0:0.2.0-48.el7

Complete!
[root@hdp2 ~]#
```

2. 启动 NFS 客户端服务。

```
[root@hdp2 ~]# service rpcbind start
Redirecting to /bin/systemctl start rpcbind.service
 [root@hdp2 ~]#
```

3. 通过 NFS 协议在 NFS 客户端上挂载 GPFS 。

```
[root@hdp2 ~]# mkdir /gpfstest
[root@hdp2 ~]# mount 10.63.150.51:/gpfs1 /gpfstest
[root@hdp2 ~]# df -h
Filesystem                              Size  Used Avail Use% Mounted on
/dev/mapper/rhel_stlrx300s6--22-root  1.1T  113G  981G  11% /
devtmpfs                                126G     0  126G   0% /dev
tmpfs                                   126G   16K  126G   1% /dev/shm
tmpfs                                   126G  510M  126G   1% /run
tmpfs                                   126G     0  126G   0%
/sys/fs/cgroup
/dev/sdd2                               197M  191M  6.6M  97% /boot
tmpfs                                    26G     0   26G   0% /run/user/0
10.63.150.213:/nc_volume2                95G  5.4G   90G   6% /mnt
10.63.150.51:/gpfs1                     7.3T  9.1G  7.3T   1% /gpfstest
[root@hdp2 ~]#
```

4. 验证 NFS 挂载文件夹中的 GPFS 文件列表。

```
[root@hdp2 ~]# cd /gpfstest/
[root@hdp2 gpfstest]# ls
ces  gpfs-ces  ha  testfile
[root@hdp2 gpfstest]# ls -l
total 5242882
drwxr-xr-x 4 root root       4096 Nov  5 15:35 ces
drwxr-xr-x 2 root root       4096 Nov  5 15:02 gpfs-ces
drwxr-xr-x 2 root root       4096 Nov  5 15:04 ha
-rw-r--r-- 1 root root 5368709120 Oct  8 14:10 testfile
[root@hdp2 gpfstest]#
```

5. 使用 XCP 将数据从 GPFS 导出的 NFS 移动到 NetApp NFS 。

```
[root@hdp2 linux]# ./xcp copy -parallel 20 10.63.150.51:/gpfs1
10.63.150.213:/nc_volume2/
XCP 1.4-17914d6; (c) 2019 NetApp, Inc.; Licensed to Karthikeyan
Nagalingam [NetApp Inc] until Tue Nov  5 12:39:36 2019

xcp: WARNING: your license will expire in less than one week! You can
renew your license at https://xcp.netapp.com
xcp: open or create catalog 'xcp': Creating new catalog in
'10.63.150.51:/gpfs1/catalog'
xcp: WARNING: No index name has been specified, creating one with name:
autoname_copy_2019-11-11_12.14.07.805223
xcp: mount '10.63.150.51:/gpfs1': WARNING: This NFS server only supports
1-second timestamp granularity. This may cause sync to fail because
changes will often be undetectable.
 34 scanned, 32 copied, 32 indexed, 1 giant, 301 MiB in (59.5 MiB/s),
784 KiB out (155 KiB/s), 6s
 34 scanned, 32 copied, 32 indexed, 1 giant, 725 MiB in (84.6 MiB/s),
1.77 MiB out (206 KiB/s), 11s
 34 scanned, 32 copied, 32 indexed, 1 giant, 1.17 GiB in (94.2 MiB/s),
2.90 MiB out (229 KiB/s), 16s
 34 scanned, 32 copied, 32 indexed, 1 giant, 1.56 GiB in (79.8 MiB/s),
3.85 MiB out (194 KiB/s), 21s
 34 scanned, 32 copied, 32 indexed, 1 giant, 1.95 GiB in (78.4 MiB/s),
4.80 MiB out (191 KiB/s), 26s
 34 scanned, 32 copied, 32 indexed, 1 giant, 2.35 GiB in (80.4 MiB/s),
5.77 MiB out (196 KiB/s), 31s
 34 scanned, 32 copied, 32 indexed, 1 giant, 2.79 GiB in (89.6 MiB/s),
6.84 MiB out (218 KiB/s), 36s
 34 scanned, 32 copied, 32 indexed, 1 giant, 3.16 GiB in (75.3 MiB/s),
7.73 MiB out (183 KiB/s), 41s
 34 scanned, 32 copied, 32 indexed, 1 giant, 3.53 GiB in (75.4 MiB/s),
8.64 MiB out (183 KiB/s), 46s
 34 scanned, 32 copied, 32 indexed, 1 giant, 4.00 GiB in (94.4 MiB/s),
9.77 MiB out (230 KiB/s), 51s
 34 scanned, 32 copied, 32 indexed, 1 giant, 4.46 GiB in (94.3 MiB/s),
10.9 MiB out (229 KiB/s), 56s
 34 scanned, 32 copied, 32 indexed, 1 giant, 4.86 GiB in (80.2 MiB/s),
11.9 MiB out (195 KiB/s), 1m1s
Sending statistics...
34 scanned, 33 copied, 34 indexed, 1 giant, 5.01 GiB in (81.8 MiB/s),
12.3 MiB out (201 KiB/s), 1m2s.
[root@hdp2 linux]#
```

6. 验证 NFS 客户端上的 GPFS 文件。

```
[root@hdp2 mnt]# df -Th
Filesystem                          Type      Size  Used Avail Use%
Mounted on
/dev/mapper/rhel_stlrx300s6--22-root xfs       1.1T  113G  981G  11% /
devtmpfs                            devtmpfs  126G     0  126G   0%
/dev
tmpfs                               tmpfs     126G   16K  126G   1%
/dev/shm
tmpfs                               tmpfs     126G  518M  126G   1%
/run
tmpfs                               tmpfs     126G     0  126G   0%
/sys/fs/cgroup
/dev/sdd2                           xfs       197M  191M  6.6M  97%
/boot
tmpfs                               tmpfs      26G     0   26G   0%
/run/user/0
10.63.150.213:/nc_volume2           nfs4       95G  5.4G   90G   6%
/mnt
10.63.150.51:/gpfs1                 nfs4      7.3T  9.1G  7.3T   1%
/gpfstest
[root@hdp2 mnt]#
[root@hdp2 mnt]# ls -ltrha
total 128K
dr-xr-xr-x   2 root        root               4.0K Dec 31  1969
.snapshots
drwxrwxrwx   2 root        root               4.0K Feb 14  2018 data
drwxrwxrwx   3 root        root               4.0K Feb 14  2018
wcresult
drwxrwxrwx   3 root        root               4.0K Feb 14  2018
wcresult1
drwxrwxrwx   2 root        root               4.0K Feb 14  2018
wcresult2
drwxrwxrwx   2 root        root               4.0K Feb 16  2018
wcresult3
-rw-r--r--   1 root        root               2.8K Feb 20  2018
README demo
drwxrwxrwx   3 root        root               4.0K Jun 28 13:38 scantg
drwxrwxrwx   3 root        root               4.0K Jun 28 13:39
scancopyFromLocal
-rw-r--r--   1 hdfs        hadoop             1.2K Jul  3 19:28 f3
-rw-r--r--   1 hdfs        hadoop             1.2K Jul  3 19:28 README
-rw-r--r--   1 hdfs        hadoop             1.2K Jul  3 19:28 f9
-rw-r--r--   1 hdfs        hadoop             1.2K Jul  3 19:28 f6
-rw-r--r--   1 hdfs        hadoop             1.2K Jul  3 19:28 f5
-rw-r--r--   1 hdfs        hadoop             1.2K Jul  3 19:30 f4
-rw-r--r--   1 hdfs        hadoop             1.2K Jul  3 19:30 f8
```

```
-rw-r--r--   1 hdfs        hadoop              1.2K Jul  3 19:30 f2
-rw-r--r--   1 hdfs        hadoop              1.2K Jul  3 19:30 f7
drwxrwxrwx   2 root        root                4.0K Jul  9 11:14 test
drwxrwxrwx   3 root        root                4.0K Jul 10 16:35
warehouse
drwxr-xr-x   3       10061 tester1             4.0K Jul 15 14:40 sdd1
drwxrwxrwx   3 testeruser1 hadoopkerberosgroup 4.0K Aug 20 17:00
kermkdir
-rw-r--r--   1 testeruser1 hadoopkerberosgroup    0 Aug 21 14:20 newfile
drwxrwxrwx   2 testeruser1 hadoopkerberosgroup 4.0K Aug 22 10:13
teragen1copy_3
drwxrwxrwx   2 testeruser1 hadoopkerberosgroup 4.0K Aug 22 10:33
teragen2copy_1
-rw-rwxr--   1 root        hdfs                1.2K Sep 19 16:38 R1
drwx------   3 root        root                4.0K Sep 20 17:28 user
-rw-r--r--   1 root        root                5.0G Oct  8 14:10
testfile
drwxr-xr-x   2 root        root                4.0K Nov  5 15:02 gpfs-
ces
drwxr-xr-x   2 root        root                4.0K Nov  5 15:04 ha
drwxr-xr-x   4 root        root                4.0K Nov  5 15:35 ces
dr-xr-xr-x. 26 root        root                4.0K Nov  6 11:40 ..
drwxrwxrwx  21 root        root                4.0K Nov 11 12:14 .
drwxrwxrwx   7 nobody      nobody              4.0K Nov 11 12:14 catalog
[root@hdp2 mnt]#
```

MapR-FS 到 ONTAP NFS

本节详细介绍了使用 NetApp XCP 将 MapR-FS 数据移动到 ONTAP NFS 中所需的步骤。

1. 为每个 MapR 节点配置三个 LUN ，并为所有 MapR 节点授予 LUN 所有权。
2. 在安装期间，为 MapR 集群磁盘选择新添加的 LUN 以用于 MapR-FS 。
3. 根据安装 MapR 集群 "MapR 6.1 文档"。
4. 使用 `Hadoop JAR xxx` 等 MapReduce 命令检查基本 Hadoop 操作。
5. 将客户数据保留在 MapR-FS 中。例如，我们使用 Teragen 在 MapR-FS 中生成了大约 1 TB 的样本数据。
6. 将 MapR-FS 配置为 NFS 导出。
   a. 在所有 MapR 节点上禁用 nlockmgr 服务。

```
root@workr-138: ~$ rpcinfo -p
   program vers proto   port  service
    100000    4   tcp    111  portmapper
    100000    3   tcp    111  portmapper
    100000    2   tcp    111  portmapper
    100000    4   udp    111  portmapper
    100000    3   udp    111  portmapper
    100000    2   udp    111  portmapper
    100003    4   tcp   2049  nfs
    100227    3   tcp   2049  nfs_acl
    100003    4   udp   2049  nfs
    100227    3   udp   2049  nfs_acl
    100021    3   udp  55270  nlockmgr
    100021    4   udp  55270  nlockmgr
    100021    3   tcp  35025  nlockmgr
    100021    4   tcp  35025  nlockmgr
    100003    3   tcp   2049  nfs
    100005    3   tcp   2049  mountd
    100005    1   tcp   2049  mountd
    100005    3   udp   2049  mountd
    100005    1   udp   2049  mountd
root@workr-138: ~$

root@workr-138: ~$ rpcinfo -d 100021 3
root@workr-138: ~$ rpcinfo -d 100021 4
```

b. 从 ` /opt/mapr/conf/exports` 文件中所有 MapR 节点上的 MapR-FS 导出特定文件夹。导出子文件夹时，请勿使用不同的权限导出父文件夹。

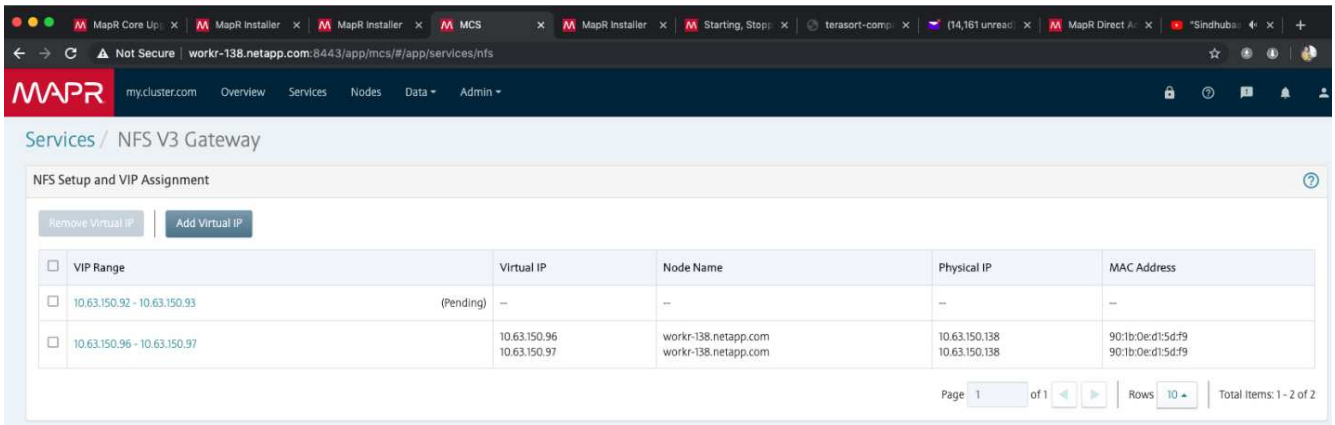
```
[mapr@workr-138 ~]$ cat /opt/mapr/conf/exports
# Sample Exports file
# for /mapr exports
# <Path> <exports_control>
#access_control -> order is specific to default
# list the hosts before specifying a default for all
#  a.b.c.d,1.2.3.4(ro) d.e.f.g(ro) (rw)
#  enforces ro for a.b.c.d & 1.2.3.4 and everybody else is rw
# special path to export clusters in mapr-clusters.conf. To disable
exporting,
# comment it out. to restrict access use the exports_control
#
#/mapr (rw)
#karthik
/mapr/my.cluster.com/tmp/testnfs /maprnfs3 (rw)
#to export only certain clusters, comment out the /mapr & uncomment.
#/mapr/clustername (rw)
#to export /mapr only to certain hosts (using exports_control)
#/mapr a.b.c.d(rw),e.f.g.h(ro)
# export /mapr/cluster1 rw to a.b.c.d & ro to e.f.g.h (denied for
others)
#/mapr/cluster1 a.b.c.d(rw),e.f.g.h(ro)
# export /mapr/cluster2 only to e.f.g.h (denied for others)
#/mapr/cluster2 e.f.g.h(rw)
# export /mapr/cluster3 rw to e.f.g.h & ro to others
#/mapr/cluster2 e.f.g.h(rw) (ro)
#to export a certain cluster, volume or a subdirectory as an alias,
#comment out  /mapr & uncomment
#/mapr/clustername         /alias1 (rw)
#/mapr/clustername/vol     /alias2 (rw)
#/mapr/clustername/vol/dir /alias3 (rw)
#only the alias will be visible/exposed to the nfs client not the
mapr path, host options as before
[mapr@workr-138 ~]$
```

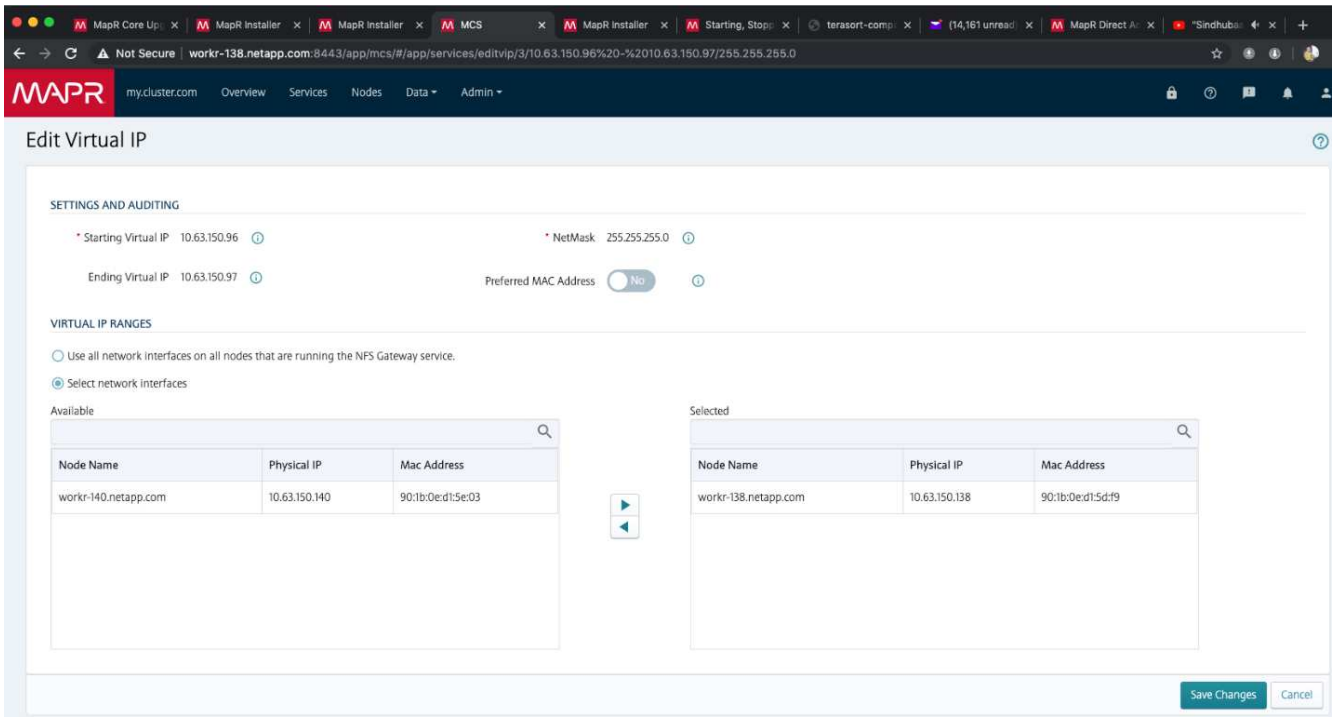
7. 刷新 MapR-FS NFS 服务。

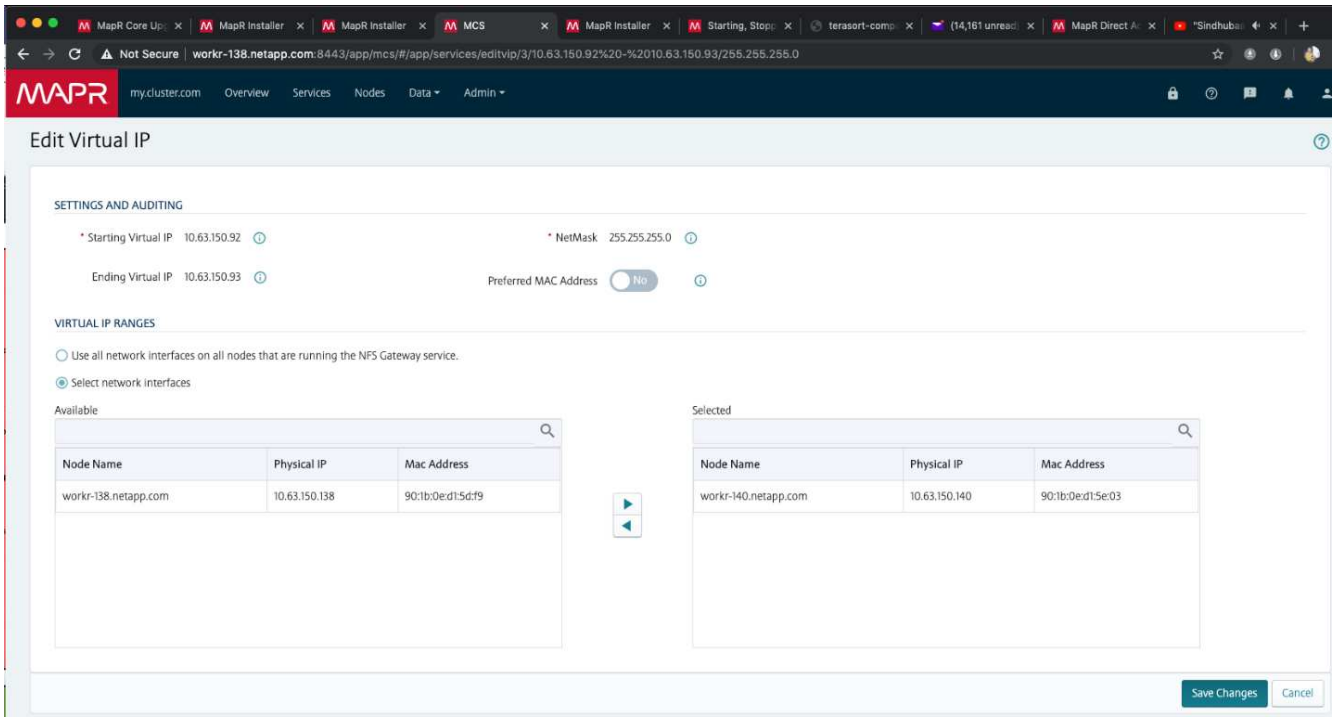
```
root@workr-138: tmp$ maprcli nfsmgmt refreshexports
ERROR (22) -  You do not have a ticket to communicate with
127.0.0.1:9998. Retry after obtaining a new ticket using maprlogin
root@workr-138: tmp$ su - mapr
[mapr@workr-138 ~]$ maprlogin password -cluster my.cluster.com
[Password for user 'mapr' at cluster 'my.cluster.com': ]
MapR credentials of user 'mapr' for cluster 'my.cluster.com' are written
to '/tmp/maprticket_5000'
[mapr@workr-138 ~]$ maprcli nfsmgmt refreshexports
```

8. 将虚拟 IP 范围分配给 MapR 集群中的特定服务器或一组服务器。然后， MapR 集群会为特定服务器分配一个 IP 以进行 NFS 数据访问。这些 IP 可实现高可用性，这意味着，如果具有特定 IP 的服务器或网络出现故障，则可以使用 IP 范围内的下一个 IP 进行 NFS 访问。

> 如果要从所有 MapR 节点提供 NFS 访问，则可以为每个服务器分配一组虚拟 IP ，并使用每个 MapR 节点的资源进行 NFS 数据访问。







9. 检查在每个 MapR 节点上分配的虚拟 IP ，并使用它们进行 NFS 数据访问。

```
root@workr-138: ~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
```

```
       valid_lft forever preferred_lft forever
2: ens3f0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9000 qdisc mq state UP
group default qlen 1000
    link/ether 90:1b:0e:d1:5d:f9 brd ff:ff:ff:ff:ff:ff
    inet 10.63.150.138/24 brd 10.63.150.255 scope global noprefixroute
ens3f0
       valid_lft forever preferred_lft forever
    inet 10.63.150.96/24 scope global secondary ens3f0:~m0
       valid_lft forever preferred_lft forever
    inet 10.63.150.97/24 scope global secondary ens3f0:~m1
       valid_lft forever preferred_lft forever
    inet6 fe80::921b:eff:fed1:5df9/64 scope link
       valid_lft forever preferred_lft forever
3: eno1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP
group default qlen 1000
    link/ether 90:1b:0e:d1:af:b4 brd ff:ff:ff:ff:ff:ff
4: ens3f1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP
group default qlen 1000
    link/ether 90:1b:0e:d1:5d:fa brd ff:ff:ff:ff:ff:ff
5: eno2: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc mq state
DOWN group default qlen 1000
    link/ether 90:1b:0e:d1:af:b5 brd ff:ff:ff:ff:ff:ff
[root@workr-138: ~$
[root@workr-140 ~]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens3f0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9000 qdisc mq state UP
group default qlen 1000
    link/ether 90:1b:0e:d1:5e:03 brd ff:ff:ff:ff:ff:ff
    inet 10.63.150.140/24 brd 10.63.150.255 scope global noprefixroute
ens3f0
       valid_lft forever preferred_lft forever
    inet 10.63.150.92/24 scope global secondary ens3f0:~m0
       valid_lft forever preferred_lft forever
    inet6 fe80::921b:eff:fed1:5e03/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
3: eno1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP
group default qlen 1000
    link/ether 90:1b:0e:d1:af:9a brd ff:ff:ff:ff:ff:ff
4: ens3f1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP
group default qlen 1000
```

```
    link/ether 90:1b:0e:d1:5e:04 brd ff:ff:ff:ff:ff:ff
5: eno2: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc mq state
DOWN group default qlen 1000
    link/ether 90:1b:0e:d1:af:9b brd ff:ff:ff:ff:ff:ff
[root@workr-140 ~]#
```

10. 使用分配的虚拟 IP 挂载 NFS 导出的 MapR-FS 以检查 NFS 操作。但是，使用 NetApp XCP 进行数据传输时，不需要执行此步骤。

```
root@workr-138: tmp$ mount -v -t nfs 10.63.150.92:/maprnfs3
/tmp/testmount/
mount.nfs: timeout set for Thu Dec  5 15:31:32 2019
mount.nfs: trying text-based options
'vers=4.1,addr=10.63.150.92,clientaddr=10.63.150.138'
mount.nfs: mount(2): Protocol not supported
mount.nfs: trying text-based options
'vers=4.0,addr=10.63.150.92,clientaddr=10.63.150.138'
mount.nfs: mount(2): Protocol not supported
mount.nfs: trying text-based options 'addr=10.63.150.92'
mount.nfs: prog 100003, trying vers=3, prot=6
mount.nfs: trying 10.63.150.92 prog 100003 vers 3 prot TCP port 2049
mount.nfs: prog 100005, trying vers=3, prot=17
mount.nfs: trying 10.63.150.92 prog 100005 vers 3 prot UDP port 2049
mount.nfs: portmap query retrying: RPC: Timed out
mount.nfs: prog 100005, trying vers=3, prot=6
mount.nfs: trying 10.63.150.92 prog 100005 vers 3 prot TCP port 2049
root@workr-138: tmp$ df -h
Filesystem              Size  Used Avail Use% Mounted on
/dev/sda7                84G   48G   37G  57% /
devtmpfs                126G     0  126G   0% /dev
tmpfs                   126G     0  126G   0% /dev/shm
tmpfs                   126G   19M  126G   1% /run
tmpfs                   126G     0  126G   0% /sys/fs/cgroup
/dev/sdd1               3.7T  201G  3.5T   6% /mnt/sdd1
/dev/sda6               946M  220M  726M  24% /boot
tmpfs                    26G     0   26G   0% /run/user/5000
gpfs1                   7.3T  9.1G  7.3T   1% /gpfs1
tmpfs                    26G     0   26G   0% /run/user/0
localhost:/mapr         100G     0  100G   0% /mapr
10.63.150.92:/maprnfs3   53T  8.4G   53T   1% /tmp/testmount
root@workr-138: tmp$
```

11. 配置 NetApp XCP 以将数据从 MapR-FS NFS 网关传输到 ONTAP NFS 。
    a. 配置 XCP 的目录位置。

```
[root@hdp2 linux]# cat /opt/NetApp/xFiles/xcp/xcp.ini
# Sample xcp config
[xcp]
#catalog =  10.63.150.51:/gpfs1
catalog =  10.63.150.213:/nc_volume1
```

b. 将此许可证文件复制到 ` /opt/netapp/xFiles/XCP/` 。

```
root@workr-138: src$ cd /opt/NetApp/xFiles/xcp/
root@workr-138: xcp$ ls -ltrha
total 252K
drwxr-xr-x 3 root   root     16 Apr  4  2019 ..
-rw-r--r-- 1 root   root    105 Dec  5 19:04 xcp.ini
drwxr-xr-x 2 root   root     59 Dec  5 19:04 .
-rw-r--r-- 1 faiz89 faiz89  336 Dec  6 21:12 license
-rw-r--r-- 1 root   root    192 Dec  6 21:13 host
-rw-r--r-- 1 root   root   236K Dec 17 14:12 xcp.log
root@workr-138: xcp$
```

c. 使用 `XCP activate` 命令激活 XCP 。

d. 检查 NFS 导出的源。

```
[root@hdp2 linux]# ./xcp show 10.63.150.92
XCP 1.4-17914d6; (c) 2019 NetApp, Inc.; Licensed to Karthikeyan
Nagalingam [NetApp Inc] until Wed Feb  5 11:07:27 2020
getting pmap dump from 10.63.150.92 port 111...
getting export list from 10.63.150.92...
sending 1 mount and 4 nfs requests to 10.63.150.92...
== RPC Services ==
'10.63.150.92': TCP rpc services: MNT v1/3, NFS v3/4, NFSACL v3, NLM
v1/3/4, PMAP v2/3/4, STATUS v1
'10.63.150.92': UDP rpc services: MNT v1/3, NFS v4, NFSACL v3, NLM
v1/3/4, PMAP v2/3/4, STATUS v1
== NFS Exports ==
 Mounts  Errors  Server
      1       0  10.63.150.92
     Space    Files      Space    Files
      Free     Free       Used     Used Export
  52.3 TiB    53.7B   8.36 GiB    53.7B 10.63.150.92:/maprnfs3
== Attributes of NFS Exports ==
drwxr-xr-x --- root root 2 2 10m51s 10.63.150.92:/maprnfs3
1.77 KiB in (8.68 KiB/s), 3.16 KiB out (15.5 KiB/s), 0s.
[root@hdp2 linux]#
```

e. 使用 XCP 从多个 MapR 节点从多个源 IP 和多个目标 IP （ONTAP LIF）传输数据。

```
root@workr-138: linux$ ./xcp_yatin copy --parallel 20
10.63.150.96,10.63.150.97:/maprnfs3/tg4
10.63.150.85,10.63.150.86:/datapipeline_dataset/tg4_dest
XCP 1.6-dev; (c) 2019 NetApp, Inc.; Licensed to Karthikeyan
Nagalingam [NetApp Inc] until Wed Feb  5 11:07:27 2020
xcp: WARNING: No index name has been specified, creating one with
name: autoname_copy_2019-12-06_21.14.38.652652
xcp: mount '10.63.150.96,10.63.150.97:/maprnfs3/tg4': WARNING: This
NFS server only supports 1-second timestamp granularity. This may
cause sync to fail because changes will often be undetectable.
 130 scanned, 128 giants, 3.59 GiB in (723 MiB/s), 3.60 GiB out (724
MiB/s), 5s
 130 scanned, 128 giants, 8.01 GiB in (889 MiB/s), 8.02 GiB out (890
MiB/s), 11s
 130 scanned, 128 giants, 12.6 GiB in (933 MiB/s), 12.6 GiB out (934
MiB/s), 16s
 130 scanned, 128 giants, 16.7 GiB in (830 MiB/s), 16.7 GiB out (831
MiB/s), 21s
 130 scanned, 128 giants, 21.1 GiB in (907 MiB/s), 21.1 GiB out (908
MiB/s), 26s
```

```
 130 scanned, 128 giants, 25.5 GiB in (893 MiB/s), 25.5 GiB out (894
MiB/s), 31s
 130 scanned, 128 giants, 29.6 GiB in (842 MiB/s), 29.6 GiB out (843
MiB/s), 36s
….
[root@workr-140 linux]# ./xcp_yatin copy  --parallel 20
10.63.150.92:/maprnfs3/tg4_2
10.63.150.85,10.63.150.86:/datapipeline_dataset/tg4_2_dest
XCP 1.6-dev; (c) 2019 NetApp, Inc.; Licensed to Karthikeyan
Nagalingam [NetApp Inc] until Wed Feb  5 11:07:27 2020
xcp: WARNING: No index name has been specified, creating one with
name: autoname_copy_2019-12-06_21.14.24.637773
xcp: mount '10.63.150.92:/maprnfs3/tg4_2': WARNING: This NFS server
only supports 1-second timestamp granularity. This may cause sync to
fail because changes will often be undetectable.
 130 scanned, 128 giants, 4.39 GiB in (896 MiB/s), 4.39 GiB out (897
MiB/s), 5s
 130 scanned, 128 giants, 9.94 GiB in (1.10 GiB/s), 9.96 GiB out
(1.10 GiB/s), 10s
 130 scanned, 128 giants, 15.4 GiB in (1.09 GiB/s), 15.4 GiB out
(1.09 GiB/s), 15s
 130 scanned, 128 giants, 20.1 GiB in (953 MiB/s), 20.1 GiB out (954
MiB/s), 20s
 130 scanned, 128 giants, 24.6 GiB in (928 MiB/s), 24.7 GiB out (929
MiB/s), 25s
 130 scanned, 128 giants, 29.0 GiB in (877 MiB/s), 29.0 GiB out (878
MiB/s), 31s
 130 scanned, 128 giants, 33.2 GiB in (852 MiB/s), 33.2 GiB out (853
MiB/s), 36s
 130 scanned, 128 giants, 37.8 GiB in (941 MiB/s), 37.8 GiB out (942
MiB/s), 41s
 130 scanned, 128 giants, 42.0 GiB in (860 MiB/s), 42.0 GiB out (861
MiB/s), 46s
 130 scanned, 128 giants, 46.1 GiB in (852 MiB/s), 46.2 GiB out (853
MiB/s), 51s
 130 scanned, 128 giants, 50.1 GiB in (816 MiB/s), 50.2 GiB out (817
MiB/s), 56s
 130 scanned, 128 giants, 54.1 GiB in (819 MiB/s), 54.2 GiB out (820
MiB/s), 1m1s
 130 scanned, 128 giants, 58.5 GiB in (897 MiB/s), 58.6 GiB out (898
MiB/s), 1m6s
 130 scanned, 128 giants, 62.9 GiB in (900 MiB/s), 63.0 GiB out (901
MiB/s), 1m11s
 130 scanned, 128 giants, 67.2 GiB in (876 MiB/s), 67.2 GiB out (877
MiB/s), 1m16s
```

f. 检查存储控制器上的负载分布。

```
Hadoop-AFF8080::*> statistics show-periodic -interval 2 -iterations 0
-summary true -object nic_common -counter rx_bytes|tx_bytes -node
Hadoop-AFF8080-01 -instance e3b
Hadoop-AFF8080: nic_common.e3b: 12/6/2019 15:55:04
 rx_bytes tx_bytes
 -------- --------
    879MB   4.67MB
    856MB   4.46MB
    973MB   5.66MB
    986MB   5.88MB
    945MB   5.30MB
    920MB   4.92MB
    894MB   4.76MB
    902MB   4.79MB
    886MB   4.68MB
    892MB   4.78MB
    908MB   4.96MB
    905MB   4.85MB
    899MB   4.83MB
Hadoop-AFF8080::*> statistics show-periodic -interval 2 -iterations 0
-summary true -object nic_common -counter rx_bytes|tx_bytes -node
Hadoop-AFF8080-01 -instance e9b
Hadoop-AFF8080: nic_common.e9b: 12/6/2019 15:55:07
 rx_bytes tx_bytes
 -------- --------
    950MB   4.93MB
    991MB   5.84MB
    959MB   5.63MB
    914MB   5.06MB
    903MB   4.81MB
    899MB   4.73MB
    892MB   4.71MB
    890MB   4.72MB
    905MB   4.86MB
    902MB   4.90MB
```

从何处查找追加信息

要了解有关本文档中所述信息的更多信息，请查看以下文档和 / 或网站：

- NetApp 原位分析模块最佳实践

  "https://www.netapp.com/us/media/tr-4382.pdf"

- 《 NetApp FlexGroup 卷最佳实践和实施指南》

  "https://www.netapp.com/us/media/tr-4571.pdf"

- NetApp 产品文档

  https://www.netapp.com/us/documentation/index.aspx

Confluent Kafka 的最佳实践

TR-4912： 《采用 NetApp 的 Confluent Kafka 分层存储最佳实践指南》

Karthikeyan Nagalingam， Joseph Kandatilparambil， NetApp Rankesh Kumar， Confluent

Apache Kafka 是一个社区分布的事件流式平台，能够每天处理数万亿次的事件。Kafka 最初是一个消息队列，它基于分布式提交日志的抽象化。自从 2011 年由 LinkedIn 创建和开源以来， Kafka 已从消息队列发展为成熟的事件流式平台。Confluent 将 Apache Kafka 与 Confluent Platform 一起分发。Confluent 平台为 Kafka 提供了更多的社区和商业功能，旨在增强大规模生产中操作员和开发人员的流式体验。

本文档通过提供以下内容介绍在 NetApp 的对象存储产品上使用 Confluent 分层存储的最佳实践准则：

- 与 NetApp 对象存储— NetApp StorageGRID 相结合的验证
- 分层存储性能测试
- 在 NetApp 存储系统上使用 Confluent 的最佳实践准则

为什么选择 **Confluent** 分层存储?

对于许多应用程序来说， Confuent 已成为默认的实时流式平台，尤其是对于大数据，分析和流式工作负载。通过分层存储，用户可以将计算与 Confluent 平台中的存储分开。它可以提高数据存储的成本效益，使您能够存储几乎无限数量的数据并按需扩展工作负载，并使数据和租户重新平衡等管理任务变得更加轻松。与 S3 兼容的存储系统可以利用所有这些功能在一个位置实现数据的民主化，从而无需复杂的数据工程。有关为什么应为 Kafka 使用分层存储的详细信息，请查看 "本篇文章由 Confluent 提供"。

为什么要使用 **NetApp StorageGRID** 进行分层存储?

StorageGRID 是 NetApp 行业领先的对象存储平台。StorageGRID 是一款基于对象的软件定义存储解决方案，支持行业标准对象 API ，包括 Amazon Simple Storage Service （ S3 ） API 。StorageGRID 可大规模存储和管理非结构化数据，以提供安全，持久的对象存储。内容放置在合适的位置，合适的时间和合适的存储层上，从而优化工作流并降低全球分布式富媒体的成本。

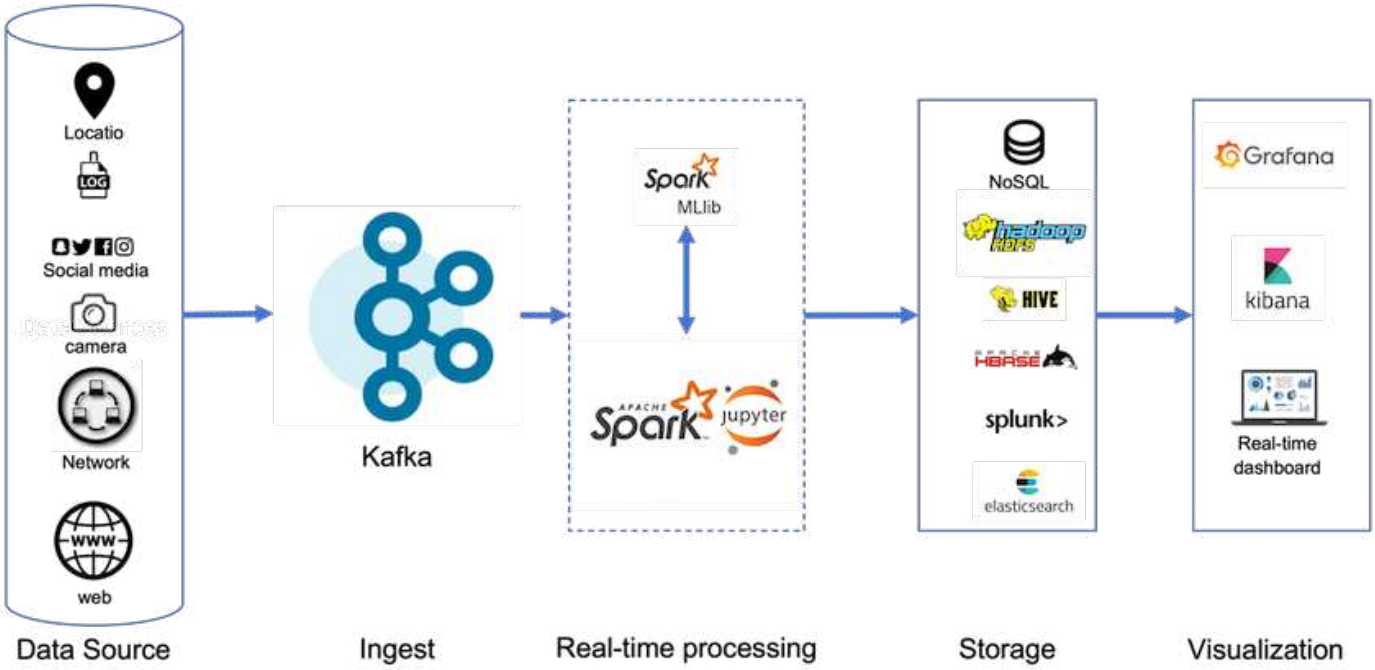
StorageGRID 最大的竞争优势是其信息生命周期管理（ ILM ）策略引擎，该引擎支持策略驱动型数据生命周期管理。策略引擎可以使用元数据管理数据在其生命周期内的存储方式，以便在数据老化时对性能进行初始优化并自动优化成本和持久性。

启用 **Confluent** 分层存储

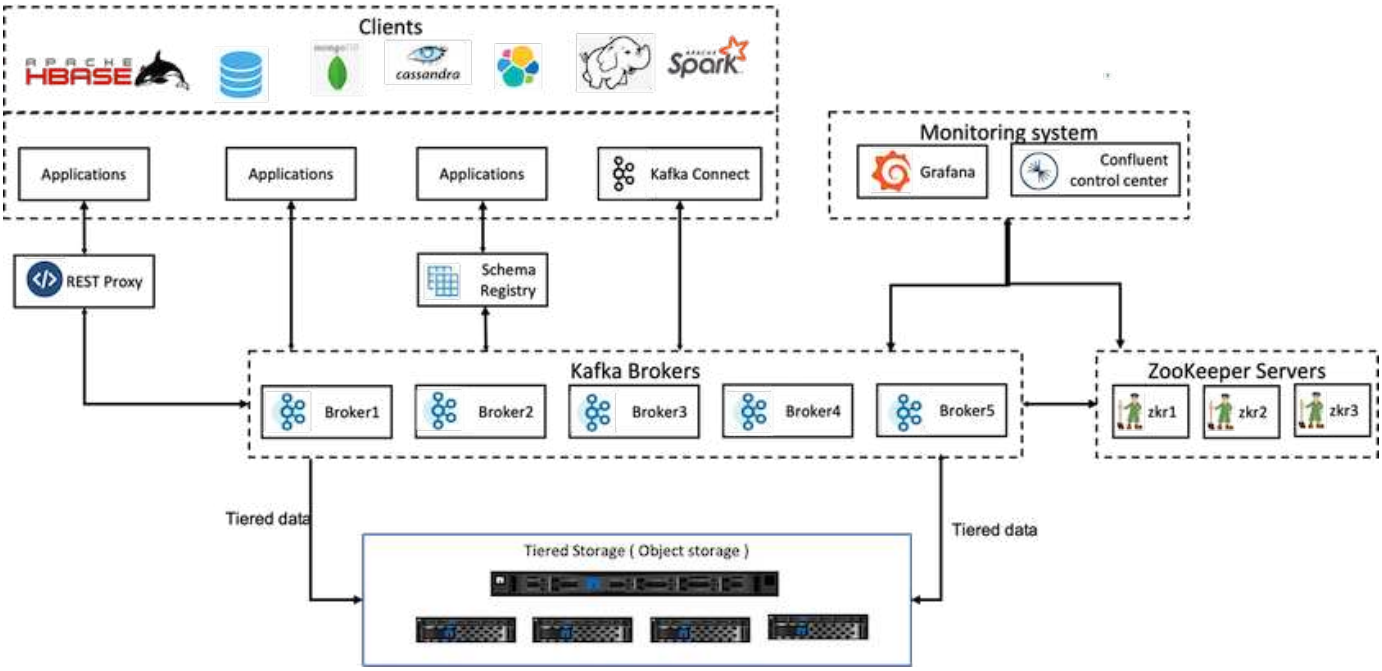
分层存储的基本理念是，将数据存储任务与数据处理任务分开。通过这种分离，数据存储层和数据处理层独立扩展变得更加容易。

适用于 Confluent 的分层存储解决方案必须与两个因素相抗衡。首先， IT 必须解决或避免常见的对象存储一致性和可用性属性，例如列表操作不一致以及偶尔出现的对象不可用性。其次， IT 必须正确处理分层存储与 Kafka 的复制和容错模型之间的交互，包括 zombie 领导者继续分层偏移范围的可能性。NetApp 对象存储可提供一致的对象可用性和 HA 模式，从而使陈旧的存储可用于分层偏移范围。NetApp 对象存储可提供一致的对象可用性和 HA 模式，使陈旧的存储可用于分层偏移范围。

借助分层存储，您可以使用高性能平台在流式数据末尾附近进行低延迟读写，还可以使用 NetApp StorageGRID 等更便宜且可扩展的对象存储来进行高吞吐量历史读取。我们还提供了适用于采用 NetApp 存储控制器的 Spark 的技术解决方案，详细信息请参见此处。下图显示了 Kafka 如何融入实时分析管道。



下图展示了 NetApp StorageGRID 如何成为 Confluent Kafka 的对象存储层。



解决方案架构详细信息

本节介绍了用于 Confluent 验证的硬件和软件。此信息适用于使用 NetApp 存储部署 Confluent Platform 。下表介绍了经过测试的解决方案架构和基本组件。

解决方案组件	详细信息
Confluent Kafka 6.2 版	• 三个 Zookepers • 五个代理服务器 • 五个工具服务器 • 一个 Grafana • 一个控制中心
Linux （ Ubuntu 18.04 ）	所有服务器
适用于分层存储的 NetApp StorageGRID	• StorageGRID 软件 • 1 个 SG1000 （负载平衡器） • 4 个 SGF6024 • 4 个 24 x 800 SSD • S3 协议 • 4 个 100GbE （代理和 StorageGRID 实例之间的网络连接）
15 台 Fujitsu PRIMERGY RX2540 服务器	每个均配备： * 2 个 CPU ，总共 16 个物理核心 * Intel Xeon * 256 GB 物理内存 * 100GbE 双端口

技术概述

本节介绍此解决方案中使用的技术。

NetApp StorageGRID

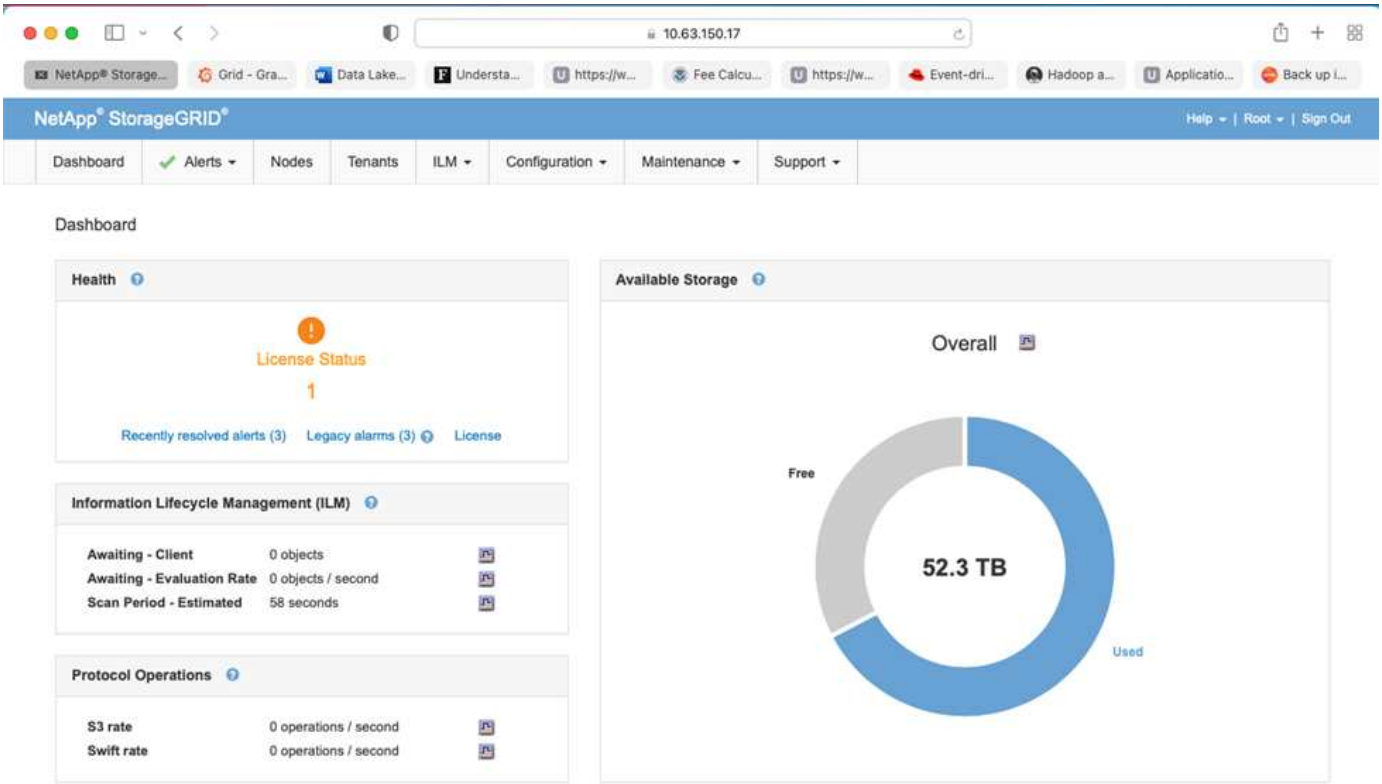
NetApp StorageGRID 是一款高性能，经济高效的对象存储平台。通过使用分层存储，存储在代理的本地存储或 SAN 存储中的 Confluent Kafka 上的大部分数据将卸载到远程对象存储。此配置可通过减少重新平衡，扩展或缩减集群或更换故障代理所需的时间和成本，显著改善运营。对象存储在管理对象存储层上的数据方面发挥着重要作用，因此选择合适的对象存储非常重要。

StorageGRID 采用基于节点的分布式网格架构，提供智能的策略驱动型全局数据管理。它通过其无处不在的全局对象命名空间以及复杂的数据管理功能，简化了对数 PB 的非结构化数据和数十亿个对象的管理。单次调用对象访问可扩展到各个站点，并简化高可用性架构，同时确保无论站点或基础架构是否中断，都能持续访问对象。

多租户支持在同一网格中安全地处理多个非结构化云和企业数据应用程序，从而提高 NetApp StorageGRID 的 ROI 并增加其用例。您可以使用元数据驱动型对象生命周期策略创建多个服务级别，以优化多个地理位置的持久性，保护，性能和位置。用户可以调整数据管理策略并监控和应用流量限制，以便在不断变化的 IT 环境中需求发生变化时无中断地与数据环境重新对齐。

使用 **Grid Manager** 进行简单管理

StorageGRID 网格管理器是一个基于浏览器的图形界面，可用于在一个管理平台中跨全球分布位置配置，管理和监控 StorageGRID 系统。



您可以使用 StorageGRID 网格管理器界面执行以下任务：

- 管理全局分布的 PB 级对象存储库，例如图像，视频和记录。
- 监控网格节点和服务以确保对象可用性。
- 使用信息生命周期管理（ ILM ）规则管理对象数据随时间的放置。这些规则用于控制在载入对象数据后该数据会发生什么情况，如何防止其丢失，对象数据的存储位置以及数据的存储时间。
- 监控系统内的事务，性能和操作。

信息生命周期管理策略

StorageGRID 具有灵活的数据管理策略，其中包括保留对象的副本以及使用 2+1 和 4+2 等 EC （纠删编码）方案来存储对象，具体取决于特定的性能和数据保护要求。随着工作负载和要求随时间的变化， ILM 策略也往往会随时间的变化而变化。修改 ILM 策略是一项核心功能，可使 StorageGRID 客户快速轻松地适应不断变化的环境。请检查 "ILM 策略" 和 "ILM 规则" 在 StorageGRID 中设置。

性能

StorageGRID 可通过添加更多存储节点来扩展性能，这些节点可以是 VM ，裸机或专门构建的设备，如 "SG5712 ， SG5760 ， SG6060 或 SGF6024"。在我们的测试中，使用 SGF6024 设备时，使用最小大小的三节点网格，超出了 Apache Kafka 的主要性能要求。随着客户通过更多代理扩展 Kafka 集群，他们可以添加更多存储节点以提高性能和容量。

负载平衡器和端点配置

StorageGRID 中的管理节点提供了网格管理器 UI （用户界面）和 REST API 端点，用于查看，配置和管理 StorageGRID 系统，并可通过审核日志来跟踪系统活动。为了为 Confluent Kafka 分层存储提供高可用性的 S3 端点，我们实施了 StorageGRID 负载平衡器，该平衡器作为一项服务在管理节点和网关节点上运行。此外，负载平衡器还管理本地流量并与 GSLB （全局服务器负载平衡）进行通信，以帮助进行灾难恢复。

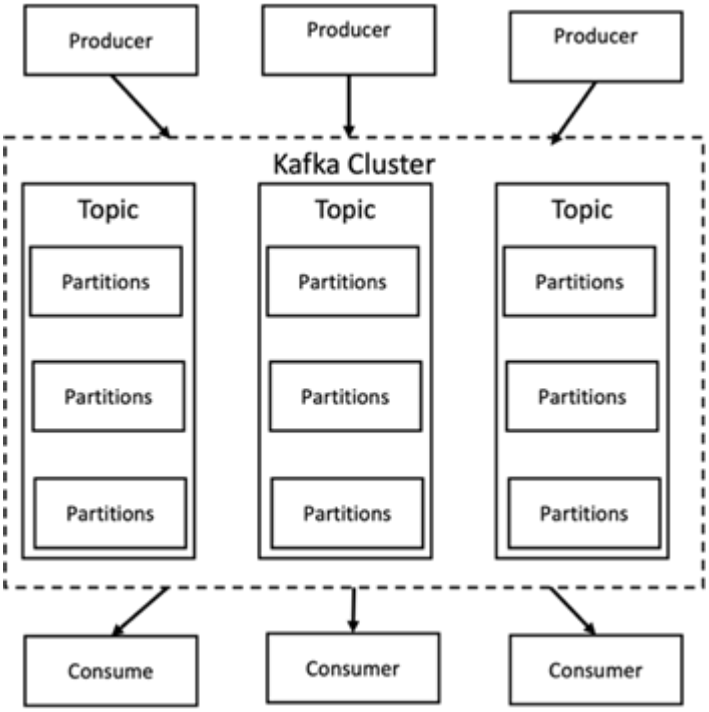
为了进一步增强端点配置， StorageGRID 提供了内置于管理节点中的流量分类策略，可用于监控工作负载流量，并对工作负载应用各种服务质量（ QoS ）限制。流量分类策略会应用于网关节点和管理节点的 StorageGRID 负载平衡器服务上的端点。这些策略可帮助您调整和监控流量。

StorageGRID 中的流量分类

StorageGRID 具有内置的 QoS 功能。流量分类策略有助于监控来自客户端应用程序的不同类型的 S3 流量。然后，您可以创建并应用策略，根据传入 / 输出带宽，读 / 写并发请求数或读 / 写请求率对此流量施加限制。

Apache Kafka

Apache Kafka 是一种使用 Java 和 Scala 编写的流处理的软件总线框架实施。它旨在提供一个统一的高吞吐量，低延迟平台来处理实时数据馈送。Kafka 可以连接到外部系统，以便通过 Kafka Connect 导出和导入数据，并提供 Kafka 流，这是一个 Java 流处理库。Kafka 使用基于 TCP 的二进制协议，该协议针对效率进行了优化，并依赖于 " 消息集 " 抽象，该抽象概念可将消息自然分组在一起，从而降低网络往返开销。这样可以实现更大的顺序磁盘操作，更大的网络数据包和连续的内存块，从而使 Kafka 能够将突发的随机消息写入流转变为线性写入。下图显示了 Apache Kafka 的基本数据流。



Kafka 存储来自任意数量的进程的关键价值消息，这些进程称为 " 生产者 " 。可以将数据分区为不同主题中的不同分区。在分区中，消息严格按其偏移量（消息在分区中的位置）排序，并编制索引并随时间戳一起存储。其他称为使用者的进程可以从分区读取消息。对于流处理， Kafka 提供了流 API ，可用于写入使用 Kafka 数据的 Java 应用程序，并将结果写入 Kafka 。Apache Kafka 还可与外部流处理系统配合使用，例如 Apache Apex ， Apache Flink ， Apache Spark ， Apache Storm 和 Apache NiFi 。

Kafka 在一个或多个服务器（称为代理）组成的集群上运行，所有主题的分区分布在各个集群节点上。此外，分

区会复制到多个代理。这种架构使 Kafka 能够以容错方式提供大量消息流，并使其能够取代一些传统消息传送系统，例如 Java 消息服务（ Java Message Service ， JMS ），高级消息队列协议（ Advanced Message Queuing Protocol ， AMQP ）等。自 0.11.0.0 版本以来， Kafka 提供了事务写入功能，可使用流 API 精确处理一次流。

Kafka 支持两种类型的主题：常规和压缩。可以为常规主题配置保留时间或空间限制。如果存在早于指定保留时间的记录，或者超出分区的绑定空间，则允许 Kafka 删除旧数据以释放存储空间。默认情况下，主题的保留时间为 7 天，但也可以无限期地存储数据。对于压缩主题，记录不会根据时间或空间限制过期。相反， Kafka 会将后续消息视为对具有相同密钥的旧消息的更新，并保证不会删除每个密钥的最新消息。用户可以通过使用特定密钥的空值编写所谓的 tombstone 消息来完全删除消息。

Kafka 中有五个主要 API：

- * 生成程序 API-* 允许应用程序发布记录流。
- * 使用者 API-* 允许应用程序订阅主题并处理记录流。
- * 连接器 API" 。 * 执行可重复使用的生产者和使用者 API ，以便将这些主题链接到现有应用程序。
- * 流 API 。 * 此 API 将输入流转换为输出并生成结果。
- * 管理 API-* 用于管理 Kafka 主题，代理和其他 Kafka 对象。

消费者和生产者 API 基于 Kafka 消息传送协议构建，并在 Java 中为 Kafka 消费者和生产者客户端提供了参考实施。底层消息传送协议是一种二进制协议，开发人员可以使用该协议以任何编程语言编写自己的使用者或生产者客户端。这样便可解除 Kafka 对 Java 虚拟机（ JVM ）生态系统的锁定。可用的非 Java 客户端列表会保存在 Apache Kafka wiki 中。

Apache Kafka 用例

Apache Kafka 在消息传送，网站活动跟踪，指标，日志聚合，流处理， 事件源和提交日志记录。

- Kafka 提高了吞吐量，内置分区，复制和容错功能，使其成为大规模消息处理应用程序的良好解决方案。
- Kafka 可以在跟踪管道中重建用户的活动（页面视图，搜索），并将其作为一组实时发布订阅源。
- Kafka 经常用于运行监控数据。其中包括汇总分布式应用程序的统计信息，以生成集中式运营数据源。
- 许多人使用 Kafka 代替日志聚合解决方案。日志聚合通常从服务器中收集物理日志文件，并将其置于中央位置（例如文件服务器或 HDFS ）进行处理。Kafka 可对文件详细信息进行抽象，并将日志或事件数据更清晰地抽象为一个消息流。这样可以降低延迟处理，并更轻松地支持多个数据源和分布式数据使用。
- Kafka 的许多用户会在由多个阶段组成的处理管道中处理数据，在这些阶段中，原始输入数据会从 Kafka 主题中使用，然后进行聚合，丰富或转换为新主题，以供进一步使用或进行后续处理。例如，用于推荐新闻文章的处理管道可能会从 rss 源中搜寻文章内容并将其发布到 " 文章 " 主题。进一步处理可能会使此内容规范化或进行重复数据删除，并将经过清理的文章内容发布到新主题中，最终处理阶段可能会尝试向用户推荐此内容。此类处理管道会根据各个主题创建实时数据流图形。
- 事件源化是一种应用程序设计模式，其状态更改将记录为按时间顺序排列的记录序列。Kafka 支持存储的非常大的日志数据，因此它是以这种模式构建的应用程序的理想后端。
- Kafka 可以用作分布式系统的一种外部提交日志。此日志有助于在节点之间复制数据，并充当故障节点恢复数据的重新同步机制。Kafka 中的日志缩减功能有助于支持此用例。

两者结合

Confluent Platform 是一款企业就绪平台，为 Kafka 提供了高级功能，旨在帮助加快应用程序开发和连接速度，通过流处理实现转型，大规模简化企业运营并满足严格的架构要求。Confluent 由 Apache Kafka 的原始创建者

构建，通过企业级功能扩展了 Kafka 的优势，同时消除了 Kafka 的管理或监控负担。如今，《财富》 100 强企业中有 80% 以上的企业都采用数据流技术，其中大多数企业都采用了流畅技术。

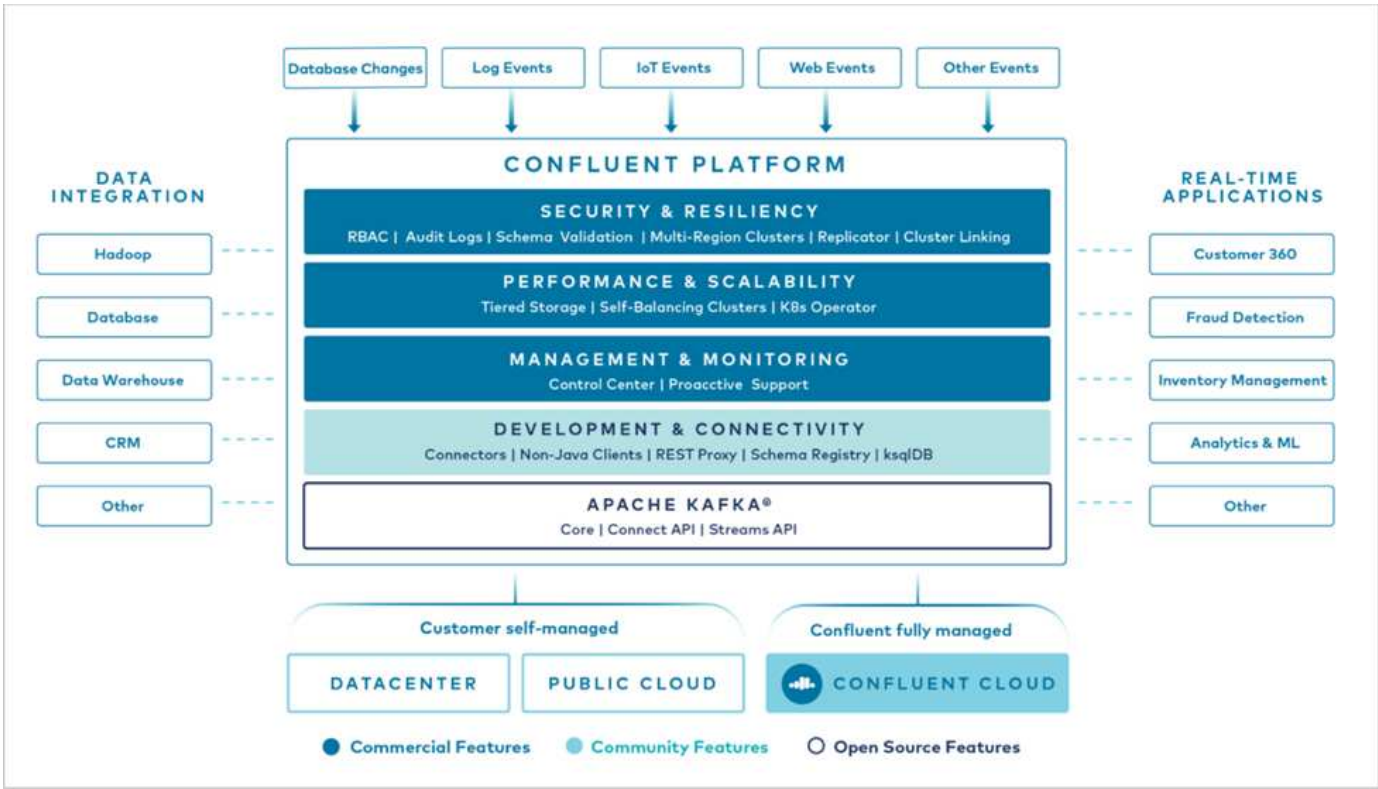
为什么选择 **Confluent**？

通过将历史数据和实时数据集成到一个统一的中央真相来源中， Confluent 可以轻松构建一个全新的现代化事件驱动型应用程序类别，获得通用数据管道，并充分扩展性，性能和可靠性，释放出强大的新用例。

Confluent 的用途是什么?

借助整合平台，您可以专注于如何从数据中获得业务价值，而不是担心底层机制，例如如何在不同系统之间传输或集成数据。具体而言， Confluent Platform 可简化将数据源连接到 Kafka 的过程，构建流式应用程序，以及保护，监控和管理 Kafka 基础架构。如今， Consfluent Platform 已广泛用于各行各业的各种用例，从金融服务，全渠道零售和自动驾驶汽车到欺诈检测， 微服务和物联网。

下图显示了 Confluent Kafka 平台的组件。



Confluent 事件流技术概述

Confluent Platform 的核心是 "Apache Kafka"一种最受欢迎的开源分布式流式平台。Kafka 的主要功能如下：

- 发布并订阅记录流。
- 以容错方式存储记录流。
- 处理记录流。

即装即用的 Confluent 平台还包括架构注册表， REST 代理，总共 100 多个预构建的 Kafka 连接器和 ksqlDB 。

- * 流畅控制中心 * 。一种基于 GUI 的系统，用于管理和监控 Kafka 。您可以通过它轻松管理 Kafka Connect ，以及创建，编辑和管理与其他系统的连接。
- * Kubernetes 的 Confluent 。 * Kubernetes 的 Confluent 是 Kubernetes 的操作员。Kubernetes 操作员通过为特定平台应用程序提供独特的功能和要求，扩展了 Kubernetes 的业务流程功能。对于 Confluent Platform ，这包括大幅简化 Kubernetes 上 Kafka 的部署流程，并自动执行典型的基础架构生命周期任务。
- * 连接 Kafka 的流畅连接器。 * 连接器使用 Kafka Connect API 将 Kafka 连接到数据库，密钥值存储，搜索索引和文件系统等其他系统。Confluent Hub 提供可下载的连接器，用于最常用的数据源和数据池，包括这些连接器经过全面测试且受支持的版本以及 Confluent 平台。有关更多详细信息，请参见 "此处"。
- * 自平衡集群。 * 提供自动化负载平衡，故障检测和自我修复功能。它支持根据需要添加或停用代理，无需手动调整。
- * 流畅集群链接。 * 直接将集群连接在一起，并通过链路网桥将主题从一个集群镜像到另一个集群。集群链接可简化多数据中心，多集群和混合云部署的设置。
- * 流畅自动数据平衡器。 * 监控集群中的代理数量，分区大小，分区数量和导数。它允许您在集群中移动数据以创建均匀的工作负载，同时限制重新平衡流量，以便在重新平衡的同时最大限度地减少对生产工作负载的影响。
- * 流畅复制器。 * 使在多个数据中心维护多个 Kafka 集群变得比以往任何时候都更轻松。
- * 分层存储。 * 提供了使用您喜欢的云提供商存储大量 Kafka 数据的选项，从而减轻了运营负担并降低了成本。借助分层存储，您只能在需要更多计算资源时，才可以将数据保存在经济高效的对象存储和扩展代理上。
- * 流畅的 jms 客户端。 * 流畅平台包括适用于 Kafka 的与 jms 兼容的客户端。此 Kafka 客户端使用 Kafka 代理作为后端，实施了 Jms 1.1 标准 API 。如果旧版应用程序使用的是 jms ，并且您希望将现有的 jms 消息代理替换为 Kafka ，则此功能非常有用。
- * 流畅的 MQT 代理。 * 提供了一种从 MQT 设备和网关直接向 Kafka 发布数据的方法，而无需在中间使用 MQT 代理。
- * 流畅安全插件。 * 流畅安全插件用于为各种流畅平台工具和产品添加安全功能。目前，可以为 Confluent REST 代理提供一个插件，用于对传入请求进行身份验证，并将经过身份验证的主体传播到 Kafka 请求。这样， Confluent REST 代理客户端便可利用 Kafka 代理的多租户安全功能。

冲突验证

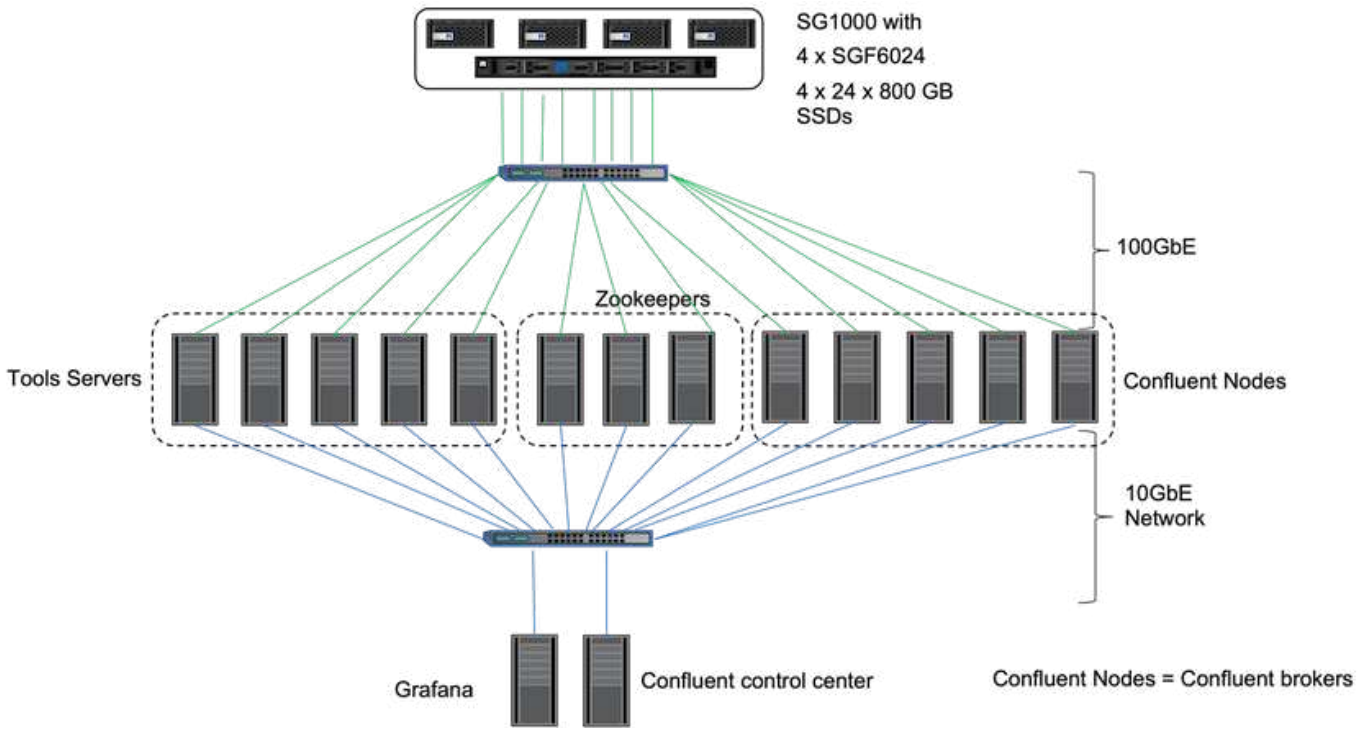
我们使用 NetApp StorageGRID 中的 Confluent Platform 6.2 分层存储执行了验证。NetApp 和 Confluent 团队共同执行了此验证，并运行了验证所需的测试用例。

Confuent Platform 设置

我们使用以下设置进行验证。

为了进行验证，我们使用了三个 Zookeepers ，五个代理，五个测试脚本执行服务器，命名为 Tools 服务器，该服务器具有 256 GB RAM 和 16 个 CPU 。对于 NetApp 存储，我们使用的是带有四个 SGF6024 的 SG1000 负载平衡器的 StorageGRID 。存储和代理通过 100GbE 连接进行连接。

下图显示了用于 Confluent 验证的配置的网络拓扑。



工具服务器充当向 Confluent 节点发送请求的应用程序客户端。

融合的分层存储配置

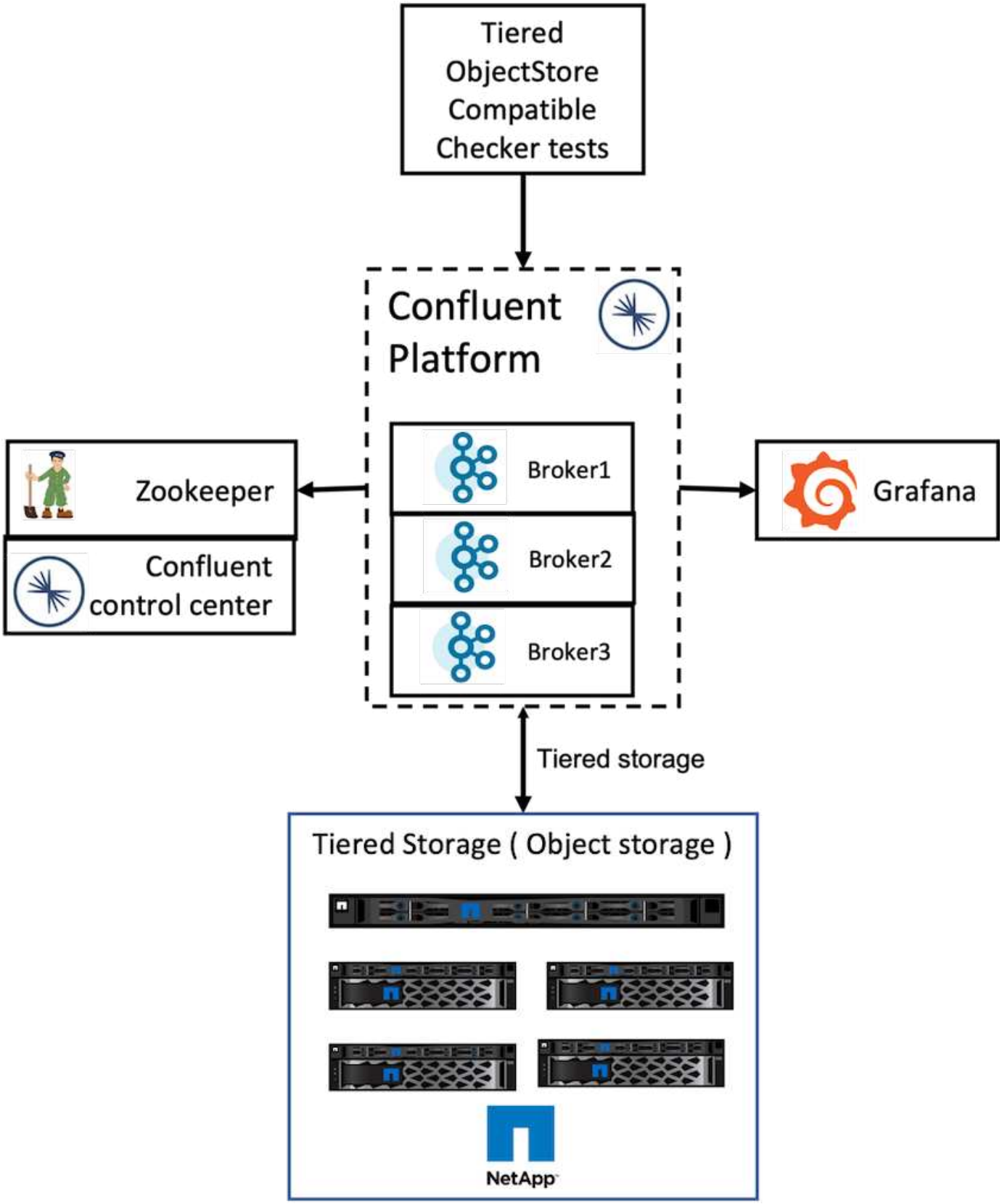
分层存储配置在 Kafka 中需要以下参数：

```
Confluent.tier.archiver.num.threads=16
confluent.tier.fetcher.num.threads=32
confluent.tier.enable=true
confluent.tier.feature=true
confluent.tier.backend=S3
confluent.tier.s3.bucket=kafkasgdbucket1-2
confluent.tier.s3.region=us-west-2
confluent.tier.s3.cred.file.path=/data/kafka/.ssh/credentials
confluent.tier.s3.aws.endpoint.override=http://kafkasgd.rtpppe.netapp.com:
10444/
confluent.tier.s3.force.path.style.access=true
```

为了进行验证，我们将 StorageGRID 与 HTTP 协议结合使用，但 HTTPS 也有效。访问密钥和机密密钥存储在 `confuent.tier.s3.cred.file.path` 参数中提供的文件名中。

NetApp 对象存储— StorageGRID

我们在 StorageGRID 中配置了单站点配置以进行分层。



验证测试

我们已完成以下五个测试案例以进行验证。这些测试将在 Trogdor 框架上执行。前两项是功能测试，其余三项是性能测试。

对象存储正确性测试

此测试将根据分层存储的需求确定对象存储 API 上的所有基本操作（例如 GET ， PUT 或 DELETE ）是否运行良好。这是一项基本测试，每个对象存储服务都应在以下测试之前通过。这是一项自信的测试，无论通过还是失败。

分层功能正确性测试

此测试可确定端到端分层存储功能是否运行良好，并通过或失败的自信测试。此测试将创建一个测试主题，默认情况下，此主题会配置为启用分层并大幅减小热设置大小。它会为新创建的测试主题生成一个事件流，并等待代理将这些分段归档到对象存储，然后使用事件流并验证已使用的流是否与生成的流匹配。生成给事件流的消息数量是可配置的，这样用户可以根据测试需求生成足够大的工作负载。减小的热集大小可确保活动分段之外的使用者提取仅从对象存储提供；这有助于测试对象存储的读取是否正确。我们执行此测试时，无论是否注入了对象存储故障。我们通过在 StorageGRID 中的一个节点中停止服务管理器服务并验证端到端功能是否适用于对象存储来模拟节点故障。

层提取基准测试

此测试验证了分层对象存储的读取性能，并检查了基准测试生成的区块在负载过重时的范围提取读取请求。在此基准测试中， Confluent 开发了自定义客户端来处理层提取请求。

生产 - 使用工作负载基准测试

此测试会通过归档区块在对象存储上间接生成写入工作负载。读取工作负载（区块读取）是在使用者组提取区块时从对象存储生成的。此工作负载由测试脚本生成。此测试检查了并行线程中对象存储上的读写性能。与分层功能正确性测试一样，我们测试了是否存在对象存储故障注入。

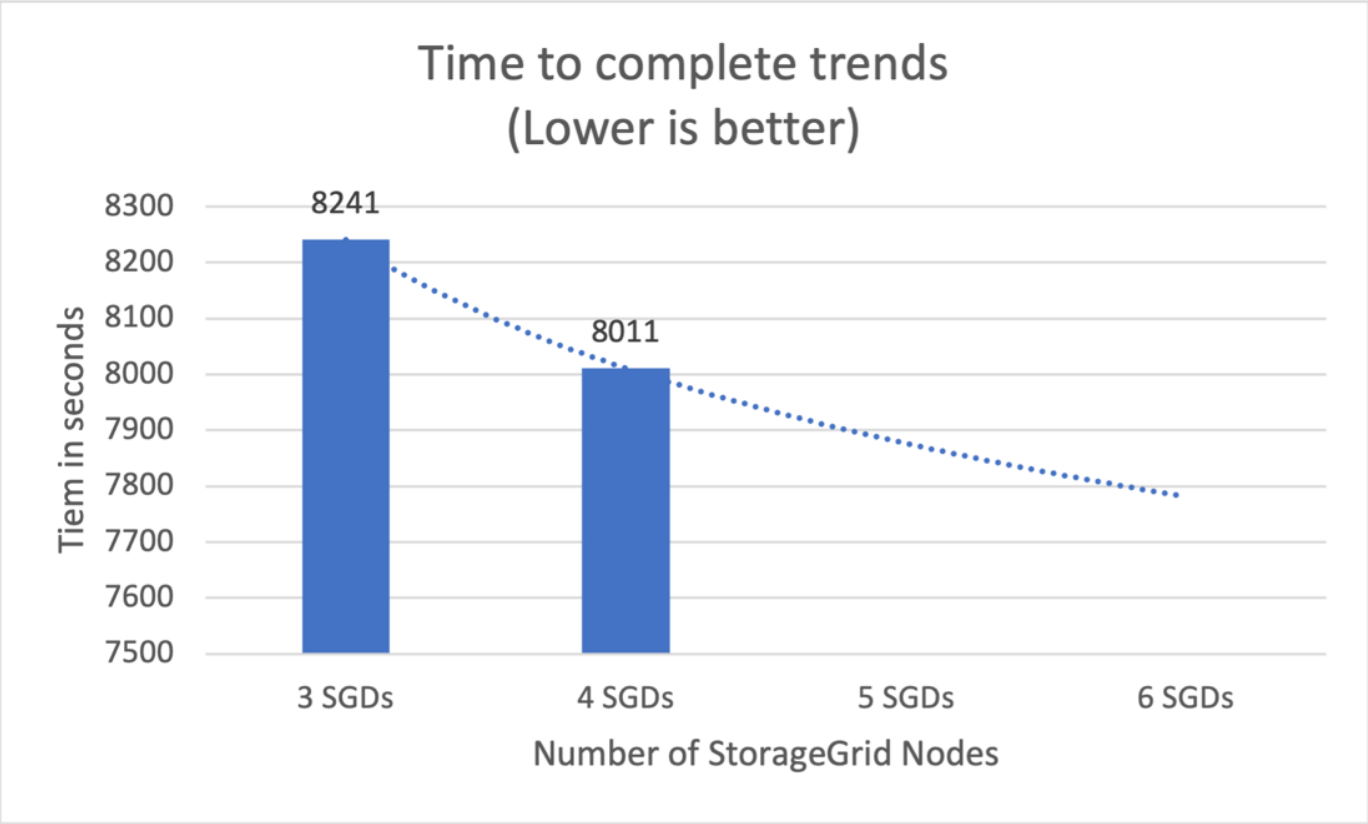
保留工作负载基准测试

此测试检查了在主题保留工作负载繁重的情况下对象存储的删除性能。保留工作负载是使用测试脚本生成的，该脚本会与测试主题并行生成许多消息。本测试主题使用主动式基于大小和基于时间的保留设置进行配置，此设置会导致从对象存储中持续清除事件流。然后，这些区块会归档。这导致代理在对象存储中删除了大量内容，并收集了对象存储删除操作的性能。

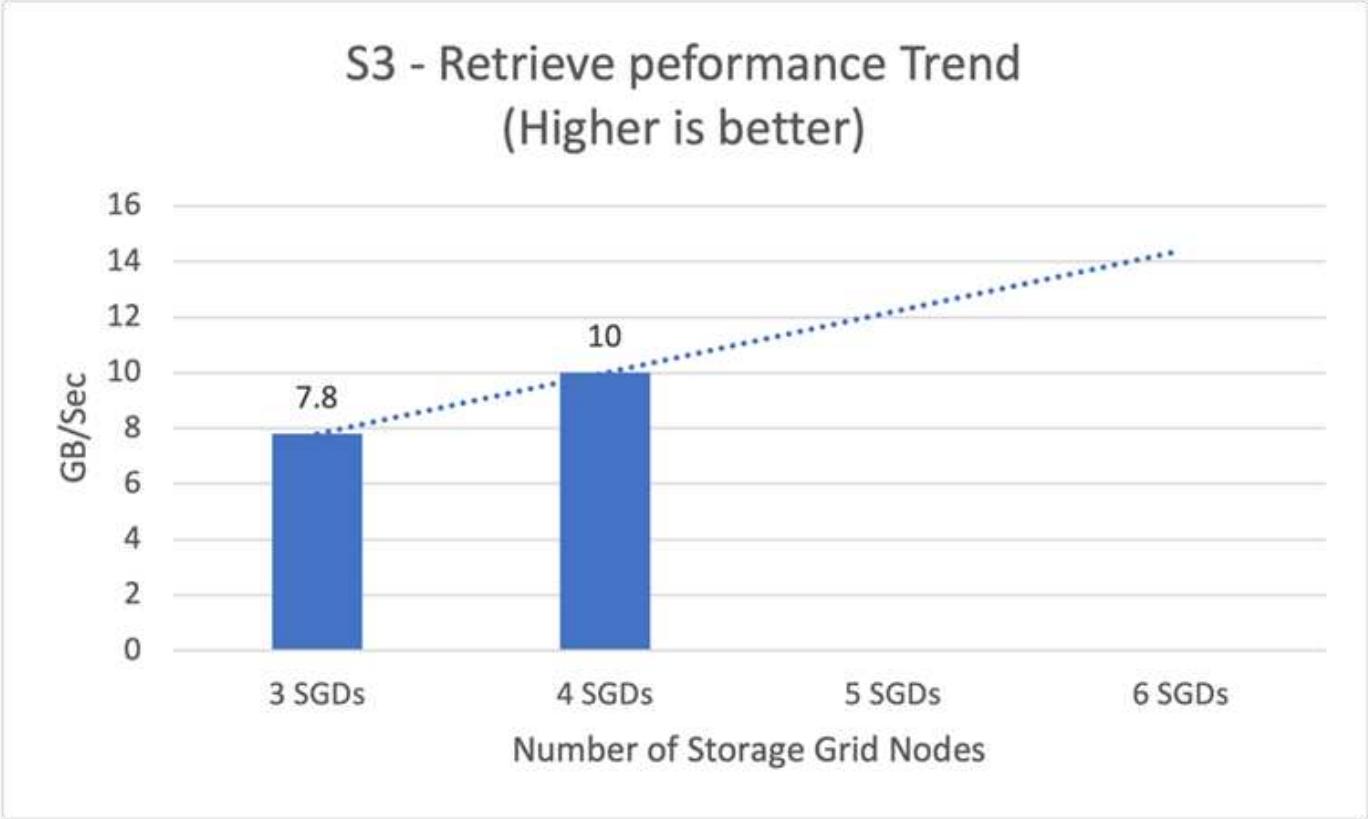
性能测试与可扩展性

我们使用 NetApp StorageGRID 设置对生产者和使用者工作负载执行了三到四个节点的分层存储测试。根据我们的测试，完成时间和性能结果与 StorageGRID 节点数成正比。StorageGRID 设置至少需要三个节点。

- 当存储节点数量增加时，完成生产和消费模式操作所需的时间呈线性下降趋势。



- 根据 StorageGRID 节点的数量， S3 检索操作的性能呈线性增长。StorageGRID 最多支持 200 个 StorgeGRID 节点。



融合 S3 连接器

Amazon S3 Sink Connector 以 Avro ， JSON 或字节格式将数据从 Apache Kafka 主题导出到 S3 对象。Amazon S3 Sink Connector 会定期轮询 Kafka 中的数据，然后将其上传到 S3 。分区程序用于将每个 Kafka 分区的数据拆分为多个区块。每个数据区块都表示为 S3 对象。密钥名称会对主题， Kafka 分区以及此数据块的起始偏移进行编码。

在此设置中，我们将向您展示如何使用 Kafka S3 接收器连接器直接从 Kafka 读取和写入对象存储中的主题。在此测试中，我们使用了独立的 Consfluent 集群，但此设置适用于分布式集群。

1. 从 Confluent 网站下载 Confluent Kafka 。
2. 将软件包解压缩到服务器上的文件夹。
3. 导出两个变量。

```
Export CONFLUENT_HOME=/data/confluent/confluent-6.2.0
export PATH=$PATH:/data/confluent/confluent-6.2.0/bin
```

4. 对于独立的 Confluent Kafka 设置，集群会在 ` /tmp` 中创建一个临时根文件夹。它还会创建 Zookeeper ， Kafka ，模式注册表， connect ， ksql-server ， 和控制中心文件夹，并从 ` $confuent_home` 复制其各自的配置文件。请参见以下示例：

```
root@stlrx2540m1-108:~# ls -ltr /tmp/confluent.406980/
total 28
drwxr-xr-x 4 root root 4096 Oct 29 19:01 zookeeper
drwxr-xr-x 4 root root 4096 Oct 29 19:37 kafka
drwxr-xr-x 4 root root 4096 Oct 29 19:40 schema-registry
drwxr-xr-x 4 root root 4096 Oct 29 19:45 kafka-rest
drwxr-xr-x 4 root root 4096 Oct 29 19:47 connect
drwxr-xr-x 4 root root 4096 Oct 29 19:48 ksql-server
drwxr-xr-x 4 root root 4096 Oct 29 19:53 control-center
root@stlrx2540m1-108:~#
```

5. 配置 Zookeeper 。如果使用默认参数，则无需更改任何内容。

```
root@stlrx2540m1-108:~# cat
/tmp/confluent.406980/zookeeper/zookeeper.properties  | grep -iv ^#
dataDir=/tmp/confluent.406980/zookeeper/data
clientPort=2181
maxClientCnxns=0
admin.enableServer=false
tickTime=2000
initLimit=5
syncLimit=2
server.179=controlcenter:2888:3888
root@stlrx2540m1-108:~#
```

在上述配置中，我们更新了 s服务器。xxx 属性。默认情况下，您需要三个 zookepers 来选择 Kafka 领导者。

6. 我们在 ` /tmp/confuent.406980/zookeeper /data` 中创建了一个 myid 文件，其唯一 ID 为：

```
root@stlrx2540m1-108:~# cat /tmp/confluent.406980/zookeeper/data/myid
179
root@stlrx2540m1-108:~#
```

我们使用 myid 文件的最后一个 IP 地址数。我们使用了 Kafka ， connect ， control-center ， Kafka ， Kafka-REST ， ksql-server 和模式注册表配置。

7. 启动 Kafka 服务。

```
root@stlrx2540m1-108:/data/confluent/confluent-6.2.0/bin# confluent
local services  start
The local commands are intended for a single-node development
environment only,
NOT for production usage.

Using CONFLUENT_CURRENT: /tmp/confluent.406980
ZooKeeper is [UP]
Kafka is [UP]
Schema Registry is [UP]
Kafka REST is [UP]
Connect is [UP]
ksqlDB Server is [UP]
Control Center is [UP]
root@stlrx2540m1-108:/data/confluent/confluent-6.2.0/bin#
```

每个配置都有一个日志文件夹，可帮助您解决问题。在某些情况下，服务需要较长时间才能启动。确保所有服务均已启动且正在运行。

8. 使用 `Confluent-hub` 安装 Kafka 连接。

```
root@stlrx2540m1-108:/data/confluent/confluent-6.2.0/bin# ./confluent-
hub install confluentinc/kafka-connect-s3:latest
The component can be installed in any of the following Confluent
Platform installations:
  1. /data/confluent/confluent-6.2.0 (based on $CONFLUENT_HOME)
  2. /data/confluent/confluent-6.2.0 (where this tool is installed)
Choose one of these to continue the installation (1-2): 1
Do you want to install this into /data/confluent/confluent-
6.2.0/share/confluent-hub-components? (yN) y

Component's license:
Confluent Community License
http://www.confluent.io/confluent-community-license
I agree to the software license agreement (yN) y
Downloading component Kafka Connect S3 10.0.3, provided by Confluent,
Inc. from Confluent Hub and installing into /data/confluent/confluent-
6.2.0/share/confluent-hub-components
Do you want to uninstall existing version 10.0.3? (yN) y
Detected Worker's configs:
  1. Standard: /data/confluent/confluent-6.2.0/etc/kafka/connect-
distributed.properties
  2. Standard: /data/confluent/confluent-6.2.0/etc/kafka/connect-
standalone.properties
  3. Standard: /data/confluent/confluent-6.2.0/etc/schema-
registry/connect-avro-distributed.properties
  4. Standard: /data/confluent/confluent-6.2.0/etc/schema-
registry/connect-avro-standalone.properties
  5. Based on CONFLUENT_CURRENT:
/tmp/confluent.406980/connect/connect.properties
  6. Used by Connect process with PID 15904:
/tmp/confluent.406980/connect/connect.properties
Do you want to update all detected configs? (yN) y
Adding installation directory to plugin path in the following files:
  /data/confluent/confluent-6.2.0/etc/kafka/connect-
distributed.properties
  /data/confluent/confluent-6.2.0/etc/kafka/connect-
standalone.properties
  /data/confluent/confluent-6.2.0/etc/schema-registry/connect-avro-
distributed.properties
  /data/confluent/confluent-6.2.0/etc/schema-registry/connect-avro-
standalone.properties
  /tmp/confluent.406980/connect/connect.properties
  /tmp/confluent.406980/connect/connect.properties
```

```
Completed
root@stlrx2540m1-108:/data/confluent/confluent-6.2.0/bin#
```

您也可以使用 `Confluent-hub install conflientint/Kafka-connect-S3 ： 10.0.3` 来安装特定版本。

9. 默认情况下， `confuentine-Kafka-connect-S3` 安装在 ` /data/confuent/confuent-6.2.0/share/confuent-hub-components/confuentine-Kafka-connect-S3` 中。
10. 使用新的 `Confluentine-Kafka-connect-S3` 更新插件路径。

```
root@stlrx2540m1-108:~# cat /data/confluent/confluent-
6.2.0/etc/kafka/connect-distributed.properties | grep plugin.path
#
plugin.path=/usr/local/share/java,/usr/local/share/kafka/plugins,/opt/co
nnectors,
plugin.path=/usr/share/java,/data/zookeeper/confluent/confluent-
6.2.0/share/confluent-hub-components,/data/confluent/confluent-
6.2.0/share/confluent-hub-components,/data/confluent/confluent-
6.2.0/share/confluent-hub-components/confluentinc-kafka-connect-s3
root@stlrx2540m1-108:~#
```

11. 停止并重新启动 Consfluent 服务。

```
confluent local services  stop
confluent local services  start
root@stlrx2540m1-108:/data/confluent/confluent-6.2.0/bin# confluent
local services  status
The local commands are intended for a single-node development
environment only,
NOT for production usage.

Using CONFLUENT_CURRENT: /tmp/confluent.406980
Connect is [UP]
Control Center is [UP]
Kafka is [UP]
Kafka REST is [UP]
ksqlDB Server is [UP]
Schema Registry is [UP]
ZooKeeper is [UP]
root@stlrx2540m1-108:/data/confluent/confluent-6.2.0/bin#
```

12. 在 ` /root/.AWS/credentials` 文件中配置访问 ID 和机密密钥。

```
root@stlrx2540m1-108:~# cat /root/.aws/credentials
[default]
aws_access_key_id = xxxxxxxxxxxx
aws_secret_access_key = xxxxxxxxxxxxxxxxxxxxxxxxx
root@stlrx2540m1-108:~#
```

13. 验证存储分段是否可访问。

```
root@stlrx2540m4-01:~# aws s3 -endpoint-url
http://kafkasgd.rtpppe.netapp.com:10444 ls kafkasgdbucket1-2
2021-10-29 21:04:18       1388 1
2021-10-29 21:04:20       1388 2
2021-10-29 21:04:22       1388 3
root@stlrx2540m4-01:~#
```

14. 为 S3 和存储分段配置 S3-sink 属性文件。

```
root@stlrx2540m1-108:~# cat /data/confluent/confluent-
6.2.0/share/confluent-hub-components/confluentinc-kafka-connect-
s3/etc/quickstart-s3.properties | grep -v ^#
name=s3-sink
connector.class=io.confluent.connect.s3.S3SinkConnector
tasks.max=1
topics=s3_testtopic
s3.region=us-west-2
s3.bucket.name=kafkasgdbucket1-2
store.url=http://kafkasgd.rtpppe.netapp.com:10444/
s3.part.size=5242880
flush.size=3
storage.class=io.confluent.connect.s3.storage.S3Storage
format.class=io.confluent.connect.s3.format.avro.AvroFormat
partitioner.class=io.confluent.connect.storage.partitioner.DefaultPartit
ioner
schema.compatibility=NONE
root@stlrx2540m1-108:~#
```

15. 将一些记录导入到 S3 存储分段中。

```
kafka-avro-console-producer --broker-list localhost:9092 --topic
s3_topic \
--property
value.schema='{"type":"record","name":"myrecord","fields":[{"name":"f1",
"type":"string"}]}'
{"f1": "value1"}
{"f1": "value2"}
{"f1": "value3"}
{"f1": "value4"}
{"f1": "value5"}
{"f1": "value6"}
{"f1": "value7"}
{"f1": "value8"}
{"f1": "value9"}
```

16. 加载 S3-sink 连接器。

```
root@stlrx2540m1-108:~# confluent local services connect connector load
s3-sink  --config /data/confluent/confluent-6.2.0/share/confluent-hub-
components/confluentinc-kafka-connect-s3/etc/quickstart-s3.properties
The local commands are intended for a single-node development
environment only,
NOT for production usage.
https://docs.confluent.io/current/cli/index.html
{
  "name": "s3-sink",
  "config": {
    "connector.class": "io.confluent.connect.s3.S3SinkConnector",
    "flush.size": "3",
    "format.class": "io.confluent.connect.s3.format.avro.AvroFormat",
    "partitioner.class":
"io.confluent.connect.storage.partitioner.DefaultPartitioner",
    "s3.bucket.name": "kafkasgdbucket1-2",
    "s3.part.size": "5242880",
    "s3.region": "us-west-2",
    "schema.compatibility": "NONE",
    "storage.class": "io.confluent.connect.s3.storage.S3Storage",
    "store.url": "http://kafkasgd.rtpppe.netapp.com:10444/",
    "tasks.max": "1",
    "topics": "s3_testtopic",
    "name": "s3-sink"
  },
  "tasks": [],
  "type": "sink"
}
root@stlrx2540m1-108:~#
```

17. 检查 S3-sink 状态。

```
root@stlrx2540m1-108:~# confluent local services connect connector
status s3-sink
The local commands are intended for a single-node development
environment only,
NOT for production usage.
https://docs.confluent.io/current/cli/index.html
{
  "name": "s3-sink",
  "connector": {
    "state": "RUNNING",
    "worker_id": "10.63.150.185:8083"
  },
  "tasks": [
    {
      "id": 0,
      "state": "RUNNING",
      "worker_id": "10.63.150.185:8083"
    }
  ],
  "type": "sink"
}
root@stlrx2540m1-108:~#
```

18. 检查日志以确保 S3-sink 已准备好接受主题。

```
root@stlrx2540m1-108:~# confluent local services connect log
```

19. 查看 Kafka 中的主题。

```
kafka-topics --list --bootstrap-server localhost:9092
…
connect-configs
connect-offsets
connect-statuses
default_ksql_processing_log
s3_testtopic
s3_topic
s3_topic_new
root@stlrx2540m1-108:~#
```

20. 检查 S3 存储分段中的对象。

```
root@stlrx2540m1-108:~# aws s3 --endpoint-url
http://kafkasgd.rtpppe.netapp.com:10444 ls --recursive kafkasgdbucket1-
2/topics/
2021-10-29 21:24:00        213
topics/s3_testtopic/partition=0/s3_testtopic+0+0000000000.avro
2021-10-29 21:24:00        213
topics/s3_testtopic/partition=0/s3_testtopic+0+0000000003.avro
2021-10-29 21:24:00        213
topics/s3_testtopic/partition=0/s3_testtopic+0+0000000006.avro
2021-10-29 21:24:08        213
topics/s3_testtopic/partition=0/s3_testtopic+0+0000000009.avro
2021-10-29 21:24:08        213
topics/s3_testtopic/partition=0/s3_testtopic+0+0000000012.avro
2021-10-29 21:24:09        213
topics/s3_testtopic/partition=0/s3_testtopic+0+0000000015.avro
root@stlrx2540m1-108:~#
```

21. 要验证内容，请运行以下命令将每个文件从 S3 复制到本地文件系统：

```
root@stlrx2540m1-108:~# aws s3 --endpoint-url
http://kafkasgd.rtpppe.netapp.com:10444 cp s3://kafkasgdbucket1-
2/topics/s3_testtopic/partition=0/s3_testtopic+0+0000000000.avro
tes.avro
download: s3://kafkasgdbucket1-
2/topics/s3_testtopic/partition=0/s3_testtopic+0+0000000000.avro to
./tes.avro
root@stlrx2540m1-108:~#
```

22. 要打印记录，请使用 avro-tools-1.11.0.1.jar （可在中找到 "Apache 归档"）。

```
root@stlrx2540m1-108:~# java -jar /usr/src/avro-tools-1.11.0.1.jar
tojson tes.avro
21/10/30 00:20:24 WARN util.NativeCodeLoader: Unable to load native-
hadoop library for your platform... using builtin-java classes where
applicable
{"f1":"value1"}
{"f1":"value2"}
{"f1":"value3"}
root@stlrx2540m1-108:~#
```

融合的自平衡集群

如果您以前管理过 Kafka 集群，则可能会熟悉手动将分区重新分配给不同代理所带来的挑战，以确保在集群中平衡工作负载。对于部署了大型 Kafka 的组织来说，重新配置大量数据可能会令人望而生畏，繁琐且存在风险，尤其是在集群上构建任务关键型应用程序时。但是，即使对于最小型的 Kafka 使用情形，该过程也非常耗时，并且容易出现人为错误。

在我们的实验室中，我们测试了 Confluent 自平衡集群功能，该功能可根据集群拓扑变化或负载不平衡自动重新平衡。当节点故障或扩展节点需要在代理之间重新平衡数据时， Confluent 重新平衡测试有助于测量添加新代理的时间。在典型的 Kafka 配置中，要重新平衡的数据量会随着集群的增长而增加，但在分层存储中，重新平衡仅限于少量数据。根据我们的验证，在典型的 Kafka 架构中，分层存储的重新平衡需要几秒或几分钟的时间，并且随着集群的增长而线性增长。

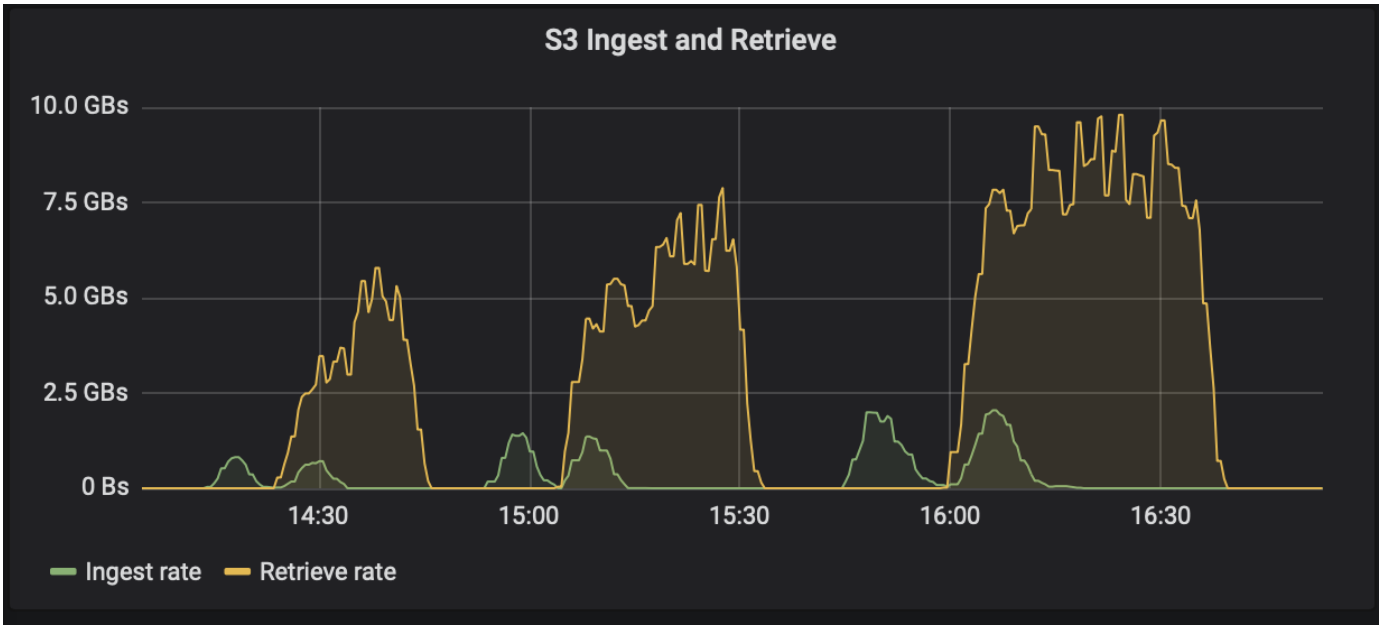
在自平衡集群中，分区重新平衡完全自动化，可优化 Kafka 的吞吐量，加快代理扩展速度并减少运行大型集群的操作负担。在稳定状态下，自平衡集群会监控代理之间的数据偏差，并持续重新分配分区以优化集群性能。在纵向或横向扩展平台时，自平衡集群会自动识别是否存在新代理或删除旧代理，并触发后续分区重新分配。这样，您就可以轻松添加和停用代理，从而从根本上提高 Kafka 集群的弹性。这些优势不需要任何人工干预，复杂的数学运算，也不需要分区重新分配通常会带来人为错误的风险。因此，完成数据重新平衡所需的时间远远少于完成时间，您可以自由地专注于价值更高的事件流式项目，而无需持续监控集群。

最佳实践准则

本节将介绍从此认证中获得的经验教训。

- 根据我们的验证， S3 对象存储对于 Confluent 来说是保留数据的最佳选择。
- 我们可以使用高吞吐量 SAN （尤其是 FC ）来保留代理热数据或本地磁盘，因为在 Confluent 分层存储配置中， 代理数据目录中保留的数据大小取决于数据移动到对象存储时的区块大小和保留时间。
- 当 seg.bytes 较高时，对象存储可提供更好的性能；我们测试了 512 MB 。
- 在 Kafka 中，为主题生成的每个记录的密钥或值的长度（以字节为单位）由 `lage.key.value` 参数控制。对于 StorageGRID ， S3 对象载入和检索性能已提高到更高值。例如， 512 字节提供了 5.8 GBps 检索，1024 字节提供了 7.5 GBps S3 检索， 2048 字节提供了接近 10 Gbps 的值。

下图显示了基于 `length` 。 `key.value` 的 S3 对象载入和检索。



- * Kafka 调整。 * 要提高分层存储的性能，您可以增加 TierFetcherNumThreads 和 TierArchiverNumThreads 。一般情况下，您需要增加 TierFetcherNumThreads 以匹配物理 CPU 核数，并将 TierArchiverNumThreads 增加到 CPU 核数的一半。例如，在服务器属性中，如果您的计算机具有八个物理核心，请将 confuent.tier.fetcher.num.threads = 8 ，而将 confuent.tier.archiver.num.threads = 4 。
- * 主题删除的时间间隔。 * 删除主题后，不会立即开始删除对象存储中的日志段文件。而是在删除这些文件之前，有一个默认值为 3 小时的时间间隔。您可以修改配置 confluent.tier.topic.delete.check.interval.ms 以更改此间隔的值。如果删除某个主题或集群，也可以手动删除相应存储分段中的对象。
- * 分层存储内部主题上的 ACL 。 * 建议内部部署的最佳实践是，在分层存储使用的内部主题上启用 ACL 授权者。设置 ACL 规则，以便仅允许代理用户访问此数据。这样可以保护内部主题的安全，并防止对分层存储数据和元数据进行未经授权的访问。

```
kafka-acls --bootstrap-server localhost:9092 --command-config adminclient-
configs.conf \
--add --allow-principal User:<kafka> --operation All --topic "_confluent-
tier-state"
```

> 将用户 ` <Kafka>` 替换为部署中的实际代理主体。

例如，命令 `confuent-tier-state` 会在内部主题上为分层存储设置 ACL 。目前，只有一个内部主题与分层存储相关。此示例将创建一个 ACL ，为内部主题上的所有操作提供主体 Kafka 权限。

规模估算

Kafka 规模估算可通过四种配置模式来执行：简单，精细，反向和分区。

简单

简单模式适用于首次使用 Apache Kafka 的用户或早期使用情形。对于此模式，您可以提供吞吐量 MBps ，读取扇出，保留和资源利用率百分比等要求（默认为 60% ）。您还可以进入内部环境（裸机， VMware ，Kubernetes 或 OpenStack ）或云。根据这些信息， Kafka 集群的规模估算可提供代理， zookeeper ， Apache

Kafka Connect 工作人员，架构注册表， REST 代理， ksqlDB 和 Confluent 控制中心所需的服务器数量。

对于分层存储，请考虑采用粒度配置模式来估算 Kafka 集群的规模。粒度模式适用于经验丰富的 Apache Kafka 用户或定义明确的用例。本节介绍了针对生产者，流处理器和使用者的规模估算。

生产者

要描述 Apache Kafka 的生成方（例如原生客户端， REST 代理或 Kafka 连接器），请提供以下信息：

- * 名称 * Spark 。
- * 生成方类型。 * 应用程序或服务，代理（ REST ， MQT ，其他）和现有数据库（ RDBMS ， NoSQL ，其他）。您也可以选择 " 我不知道 " 。
- * 平均吞吐量 * ，以每秒事件数（例如 1 ， 000 ， 000 ）表示。
- * 峰值吞吐量 * ，以每秒事件数（例如 4 ， 000 ， 000 ）表示。
- * 平均消息大小 * ，以字节为单位，未压缩（最大 1 MB ；例如 1000 ）。
- * 消息格式。 * 选项包括 Avro ， JSON ，协议缓冲区，二进制文件，文本， " 我不知道 " 等。
- * 复制因子。 * 选项包括 1 ， 2 ， 3 （ Confluent 建议）， 4 ， 5 ， 或 6.
- * 保留时间 * 一天（例如）。您希望将数据存储在 Apache Kafka 中多长时间？输入 -1 并输入任意单位，持续时间不受限制。计算器假定保留时间为 10 年，以实现无限保留。
- 选中 " 启用分层存储以减少代理数量并允许无限存储？ " 复选框
- 启用分层存储后，保留字段将控制存储在代理本地的热数据集。归档保留字段用于控制数据在归档对象存储中的存储时间。
- * 归档存储保留。 * 一年（例如）。您希望将数据存储在归档存储中多长时间？输入 -1 并输入任意单位，持续时间不受限制。计算器假定保留 10 年以实现无限保留。
- * 增长乘数 * 1 （例如）。如果此参数的值基于当前吞吐量，请将其设置为 1 。要根据其他增长调整大小，请将此参数设置为增长乘数。
- * 生产商实例数。 * 10 （例如）。将运行多少个生产商实例？要将 CPU 负载纳入规模估算计算，需要使用此输入。如果值为空，则表示 CPU 负载未计入计算中。

根据此示例输入，规模估算对生成方具有以下影响：

- 未压缩字节的平均吞吐量： 1 Gbps 。未压缩字节的峰值吞吐量： 4 Gbps 。以压缩字节为单位的平均吞吐量： 400 Mbps 。以压缩字节为单位的峰值吞吐量： 1.6 GBps 。此值基于默认 60% 的压缩率（您可以更改此值）。
  - 所需的总代理热设置存储： 31 ， 104 TB ，包括复制，已压缩。所需的代理外归档存储总量： 378 ， 432 TB ，已压缩。使用 … "https://fusion.netapp.com" 用于 StorageGRID 规模估算。

流处理器必须描述其应用程序或服务，这些应用程序或服务会使用 Apache Kafka 中的数据并生成回 Apache Kafka 。大多数情况下，这些数据存储库内置在 ksqlDB 或 Kafka 流中。

- * 名称 * 。 * Spark 流式传输器。
- * 处理时间。 * 此处理器处理单条消息需要多长时间？
  - 1 毫秒（简单，无状态转换） [示例] ， 10 毫秒（有状态内存操作）。
  - 100 毫秒（有状态网络或磁盘操作）， 1000 毫秒（第三方 REST 调用）。

    - 我已对该参数进行了基准测试，并确切了解它需要多长时间。
- * 输出保留 * 1 天（示例）。流处理器会将其输出返回到 Apache Kafka 。您希望将此输出数据存储在 Apache Kafka 中多长时间？输入 -1 并输入任意单位，持续时间不受限制。
- 选中 " 启用分层存储以减少代理数量并允许无限存储？ " 复选框
- * 归档存储保留。 * 1 年（例如）。您希望将数据存储在归档存储中多长时间？输入 -1 并输入任意单位，持续时间不受限制。计算器假定保留 10 年以实现无限保留。
- * 输出直通百分比 * 100 （例如）。流处理器会将其输出返回到 Apache Kafka 。多少百分比的入站吞吐量将输出回 Apache Kafka ？ 例如，如果入站吞吐量为 20Mbps ，而此值为 10 ，则输出吞吐量将为 2Mbps 。
- 从哪些应用程序读取此数据？选择 "Spark " ，即在基于生产商类型的规模估算中使用的名称。根据上述输入，估算流处理程序实例和主题分区估计值时可能会产生以下影响：
- 此流处理器应用程序需要以下数量的实例。传入的主题可能也需要这么多分区。请联系 Confluent 以确认此参数。
    - 1 ， 000 表示平均吞吐量，无增长乘数
    - 4 ， 000 表示峰值吞吐量，无增长乘数
    - 1 ， 000 表示使用增长乘数的平均吞吐量
    - 4 ， 000 表示峰值吞吐量，并使用增长乘数

使用者

请描述您使用 Apache Kafka 中的数据而不生成回 Apache Kafka 的应用程序或服务，例如原生客户端或 Kafka 连接器。

- * 名称 * 。 * Spark 使用者。
- * 处理时间。 * 此使用者需要多长时间处理一条消息？
    - 1 毫秒（例如，日志记录等简单无状态任务）
    - 10 毫秒（快速写入数据存储库）
    - 100 毫秒（写入数据存储库的速度较慢）
    - 1000 毫秒（第三方 REST 调用）
    - 已知持续时间的其他一些基准流程。
- * 使用者类型。 * 应用程序，代理或接收到现有数据存储库（ RDBMS ， NoSQL ，其他）。
- 从哪些应用程序读取此数据？将此参数与先前确定的生成方和流规模估算相连接。

根据上述输入，您必须确定使用者实例的规模估算和主题分区估算。使用者应用程序需要以下数量的实例。

- 2 ， 000 表示平均吞吐量，无增长乘数
- 峰值吞吐量为 8 ， 000 ，无增长乘数
- 2 ， 000 表示平均吞吐量，包括增长乘数
- 8 ， 000 表示峰值吞吐量，包括增长乘数

传入主题可能也需要此数量的分区。请联系 Confluent 进行确认。

除了对生产者，流处理器和使用者的要求之外，您还必须满足以下附加要求：

- * 重建时间。 * 例如， 4 小时。如果 Apache Kafka 代理主机发生故障，其数据丢失，并且配置了新主机来更换发生故障的主机，则新主机必须自行重建多快？如果此值未知，请将此参数留空。
- * 资源利用率目标（百分比）。 * 例如 60 。您希望主机在平均吞吐量期间的利用率如何？除非您使用的是 Confluent 自平衡集群，否则 Confuent 建议使用 60% 的利用率，在这种情况下，利用率可能会更高。

描述您的环境

- * 您的集群将在什么环境中运行？ * Amazon Web Services ， Microsoft Azure ， Google 云平台，内部裸机，内部部署 VMware ， 内部使用 OpenStack 还是内部使用 Kubernates ？
- * 主机详细信息。 * 核心数： 48 （例如），网卡类型（ 10GbE ， 40GbE ， 16GbE ， 1GbE 或其他类型）。
- * 存储卷。 * 主机： 12 （例如）。每个主机支持多少个硬盘驱动器或 SSD ？ 因此，建议每个主机配置 12 个硬盘驱动器。
- * 存储容量 / 卷（以 GB 为单位）。 * 1000 （例如）。单个卷可以存储多少 GB 的存储？这两者建议使用 1 TB 磁盘。
- * 存储配置。 * 如何配置存储卷？Confuent 建议使用 RAID10 来利用所有 Confluent 功能。JBOD ， SAN ， RAID 1 ， RAID 0 ， RAID 5 ， 此外，还支持其他类型。
- * 单卷吞吐量（ MBps ）。 * 125 （例如）。单个存储卷的每秒读取或写入速度（以 MB/ 秒为单位）有多快？ Confuent 建议使用标准硬盘驱动器，这些驱动器的吞吐量通常为 125 MBps 。
- * 内存容量（ GB ） .* 64 （例如）。

确定环境变量后，选择调整集群大小。根据上述示例参数，我们确定了以下 Confluent Kafka 规模估算：

- * Apache Kafka.* 代理计数： 22 。您的集群受存储限制。请考虑启用分层存储以减少主机数量并允许无限存储。
- * Apache zookeeper 。 * 计数： 5 ； Apache Kafka Connect Worker ：计数： 2 ；架构注册表：计数： 2 ； REST 代理：计数： 2 ； ksqlDB ：计数： 2 ； Confluent Control Center ：计数： 1 。

对平台团队使用反向模式，而不考虑使用情形。使用分区模式计算单个主题所需的分区数。请参见 https://eventsizer.io 用于根据反向和分区模式进行规模估算。

结论

本文档提供了将 Confluent 分层存储与 NetApp 存储结合使用的最佳实践准则，其中包括验证测试，分层存储性能结果，调整， Confluent S3 连接器以及自平衡功能。考虑到 ILM 策略，具有多个验证性能测试的流畅性能以及行业标准 S3 API ， NetApp StorageGRID 对象存储是流畅分层存储的最佳选择。

从何处查找追加信息

要了解有关本文档中所述信息的更多信息，请查看以下文档和 / 或网站：

- 什么是 Apache Kafka

  "https://www.confluent.io/what-is-apache-kafka/"

- NetApp 产品文档

"https://www.netapp.com/support-and-training/documentation/"

- S3-sink 参数详细信息

  "https://docs.confluent.io/kafka-connect-s3-sink/current/configuration_options.html#s3-configuration-options"

- Apache Kafka

  "https://en.wikipedia.org/wiki/Apache_Kafka"

- 在 Confluent 平台中提供无限存储

  "https://www.confluent.io/blog/infinite-kafka-storage-in-confluent-platform/"

- 融合分层存储—最佳实践和规模估算

  "https://docs.confluent.io/platform/current/kafka/tiered-storage.html#best-practices-and-recommendations"

- 适用于 Confluent Platform 的 Amazon S3 Sink Connector

  "https://docs.confluent.io/kafka-connect-s3-sink/current/overview.html"

- Kafka 规模估算

  "https://eventsizer.io"

- StorageGRID 规模估算

  "https://fusion.netapp.com/"

- Kafka 用例

  "https://kafka.apache.org/uses"

- 在融合平台 6.0 中实现自我平衡的 Kafka 集群

  "https://www.confluent.io/blog/self-balancing-kafka-clusters-in-confluent-platform-6-0/"

  "https://www.confluent.io/blog/confluent-platform-6-0-delivers-the-most-powerful-event-streaming-platform-to-date/"

NetApp 混合云数据解决方案—基于客户用例的 Spark 和 Hadoop

TR-4657： NetApp 混合云数据解决方案—基于客户用例的 Spark 和 Hadoop

NetApp 公司 Karthikeyan Nagalingam 和 Sathish Thyagarajan

本文档介绍使用 NetApp AFF 和 FAS 存储系统， NetApp Cloud Volumes ONTAP ，NetApp 互联存储以及适用于 Spark 和 Hadoop 的 NetApp FlexClone 技术的混合云数据解决方案。客户可以通过这些解决方案架构为其环境选择合适的数据保护解决方案。NetApp

在与客户及其业务用例进行交互的基础上设计了这些解决方案。本文档提供了以下详细信息：

- 为什么我们需要针对 Spark 和 Hadoop 环境以及客户面临的挑战提供数据保护。
- 由 NetApp 愿景提供支持的 Data Fabric 及其组件和服务。
- 如何使用这些组件构建灵活的数据保护工作流。
- 根据实际客户使用情形确定的多个架构的优缺点。每个用例都包含以下组件：
  - 客户情形
  - 要求和挑战
  - 解决方案
  - 解决方案摘要

为什么选择 **Hadoop** 数据保护?

在 Hadoop 和 Spark 环境中，必须解决以下问题：

- * 软件或人为故障。 * 执行 Hadoop 数据操作时，软件更新中出现人为错误，可能导致出现错误行为，发生原因从而可能导致作业产生意外结果。在这种情况下，我们需要保护数据，以避免出现故障或出现不合理的结果。例如，由于对流量信号分析应用程序进行的软件更新执行不当，这是一项新功能，无法以纯文本的形式正确分析流量信号数据。该软件仍会分析 JSON 和其他非文本文件格式，从而导致实时流量控制分析系统生成缺少数据点的预测结果。这种情况可能会导致发生原因输出出现故障，从而可能导致交通信号发生意外。通过提供快速回滚到先前工作应用程序版本的功能，数据保护可以解决此问题描述问题。
- * 大小和规模。 * 由于数据源和卷的数量不断增加，分析数据的大小每天都在增长。社交媒体，移动应用程序，数据分析和云计算平台是当前大数据市场中的主要数据源，该市场的增长速度非常快，因此需要对数据进行保护，以确保准确的数据运行。
- * Hadoop 的原生数据保护。 * Hadoop 具有一个原生命令来保护数据，但此命令不会在备份期间提供数据的一致性。它仅支持目录级备份。Hadoop 创建的快照为只读快照，不能用于直接重复使用备份数据。

Hadoop 和 **Spark** 客户面临的数据保护挑战

Hadoop 和 Spark 客户面临的一个常见挑战是，在数据保护期间，在不对生产集群性能产生负面影响的情况下，缩短备份时间并提高备份可靠性。

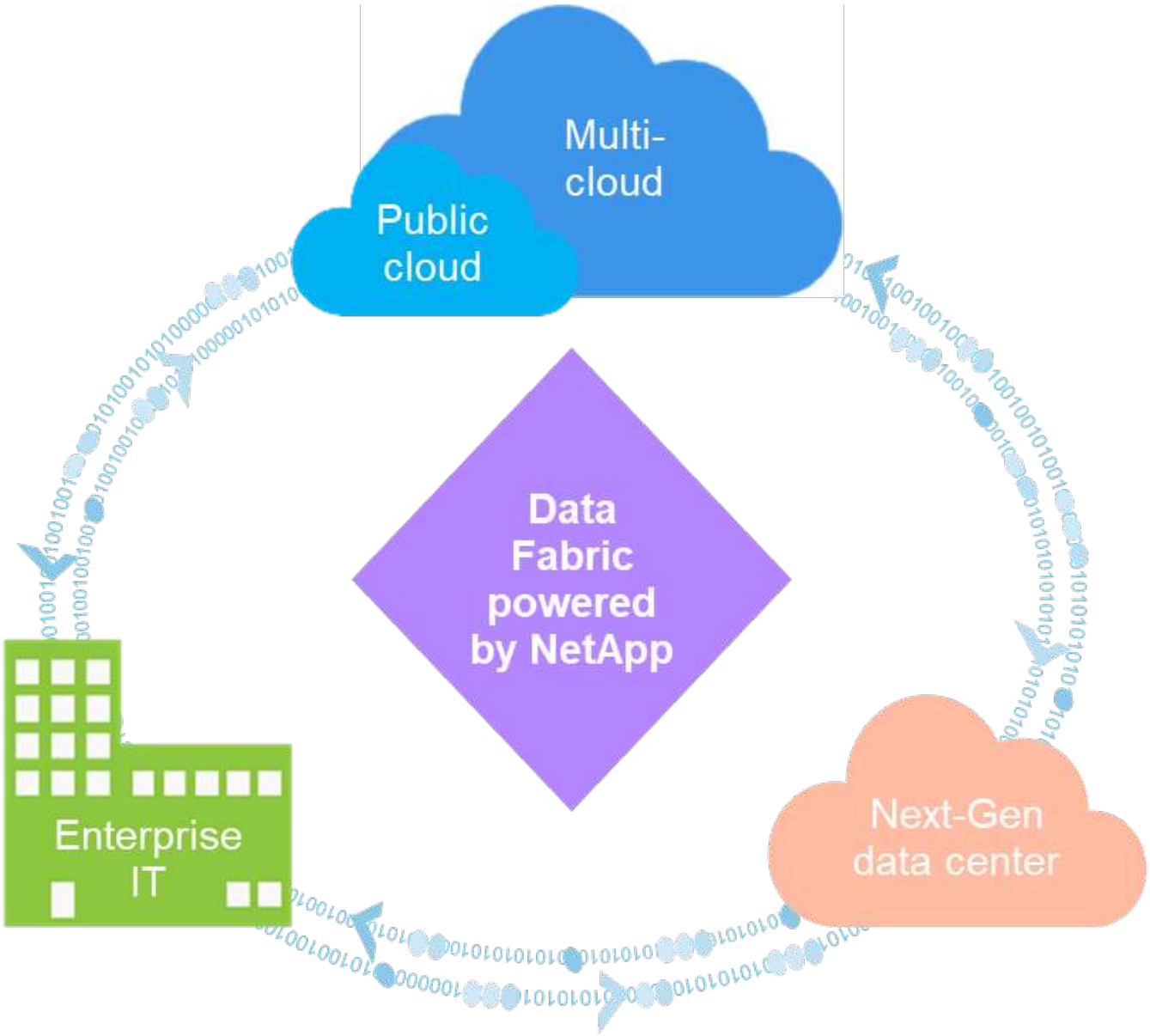
客户还需要最大限度地减少恢复点目标（ RPO ）和恢复时间目标（ RTO ）停机时间，并控制其内部和基于云的灾难恢复站点，以实现最佳业务连续性。这种控制通常来自企业级管理工具。

Hadoop 和 Spark 环境非常复杂，因为不仅数据量庞大且不断增长，而且数据的到达速度也在不断提高。在这种情况下，很难从源数据快速创建高效，最新的 DevTest 和 QA 环境。NetApp 认识到这些挑战，并提供了本白皮书中介绍的解决方案。

NetApp 为大数据架构提供支持的 **Data Fabric**

由 NetApp 提供支持的 Data Fabric 可简化并集成云环境和内部环境中的数据管理，以加速数字化转型。

由 NetApp 提供支持的 Data Fabric 可提供一致且集成的数据管理服务和应用程序（组件），以实现数据可见性和洞察力，数据访问和控制以及数据保护和安全，如下图所示。



经验证的 **Data Fabric** 客户用例

由 NetApp 提供支持的 Data Fabric 为客户提供了以下九个经验证的使用情形：

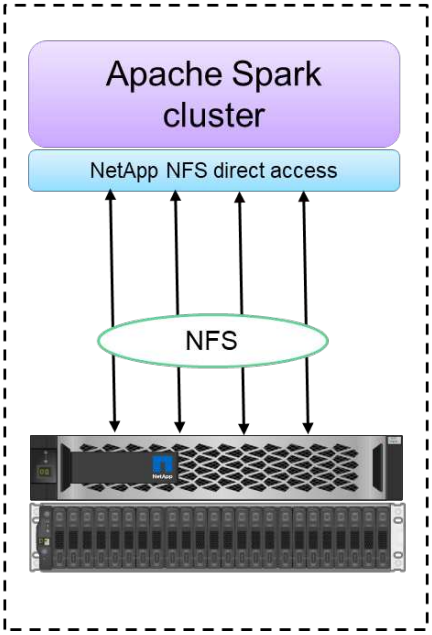
- 加快分析工作负载的速度
- 加快开发运营转型
- 构建云托管基础架构
- 集成云数据服务
- 保护和保护数据
- 优化非结构化数据
- 提高数据中心效率
- 提供数据洞察力和控制力
- 简化和自动化

本文档涵盖九个使用情形中的两个（及其解决方案）：

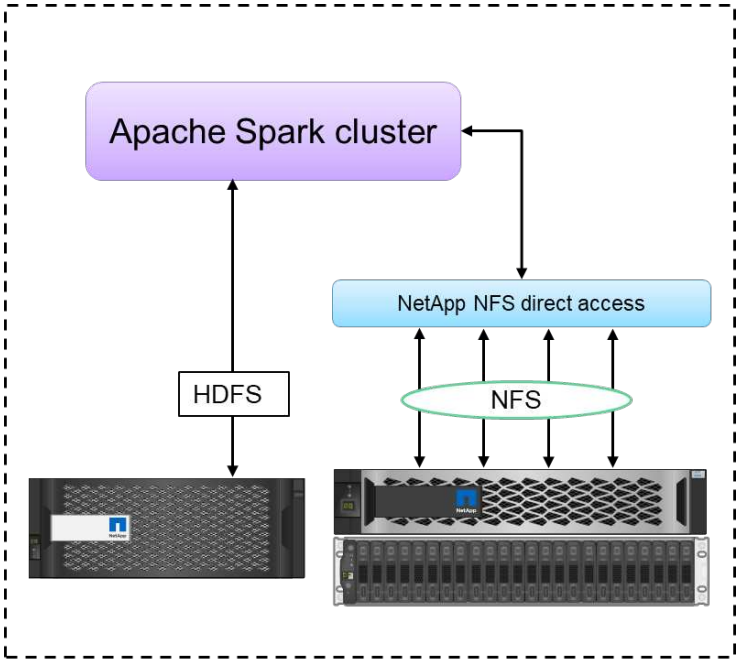
- 加快分析工作负载的速度
- 保护和保护数据

NetApp NFS 直接访问

借助NetApp NFS、客户可以对现有或新的NFSv3或NFSv4数据运行大数据分析作业、而无需移动或复制数据。它可以防止多个数据副本，并且无需将数据与源进行同步。例如，在金融领域，将数据从一个位置移动到另一个位置必须履行法律义务，这不是一项容易的任务。在这种情况下， NetApp NFS 直接访问会分析其原始位置的财务数据。另一个主要优势是，使用 NetApp NFS 直接访问可通过使用原生 Hadoop 命令简化对 Hadoop 数据的保护，并利用 NetApp 丰富的数据管理产品组合支持数据保护工作流。



Configuration 1: NFS as primary storage



Configuration 2: HDFS and NFS in single Spark cluster

NetApp NFS 直接访问为 Hadoop/Spark 集群提供了两种部署选项：

- 默认情况下， Hadoop/Spark 集群使用 Hadoop 分布式文件系统（ HDFS ）进行数据存储和默认文件系统。NetApp NFS 直接访问可以将默认 HDFS 替换为 NFS 存储作为默认文件系统，从而对 NFS 数据执行直接分析操作。
- 在另一种部署选项中， NetApp NFS 直接访问支持在一个 Hadoop/Spark 集群中将 NFS 与 HDFS 配置为额外的存储。在这种情况下，客户可以通过 NFS 导出共享数据，并从同一集群访问数据以及 HDFS 数据。

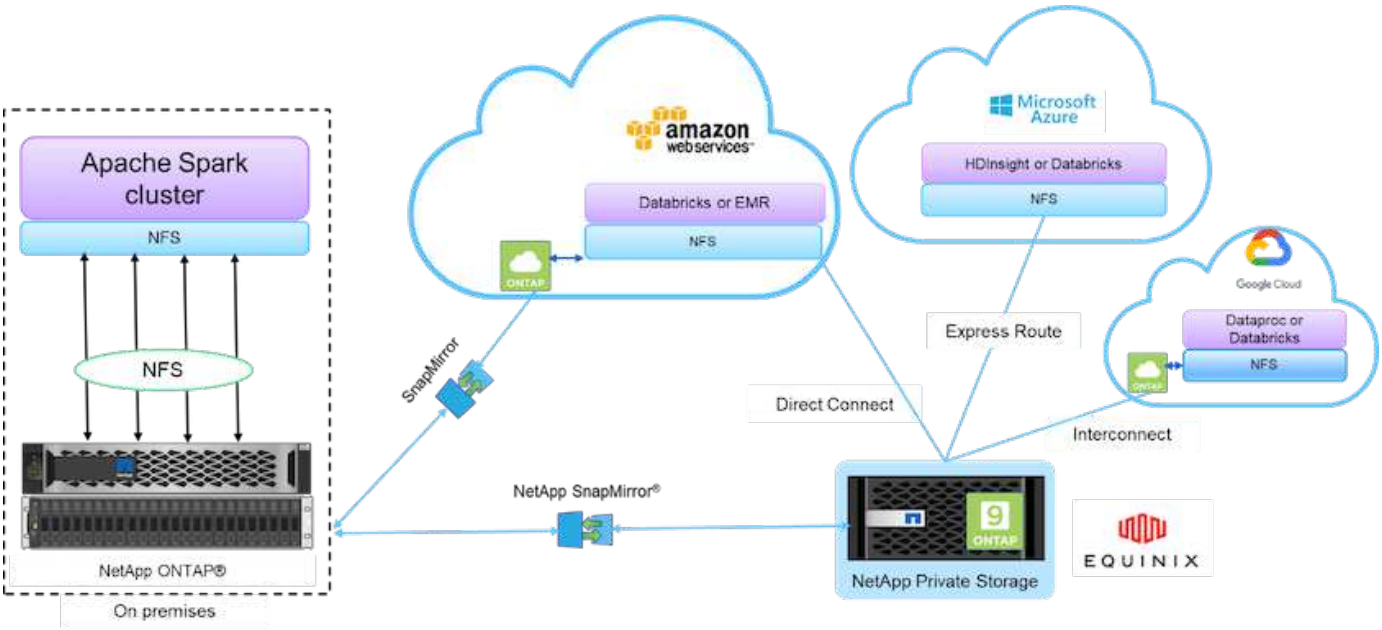
使用 NetApp NFS 直接访问的主要优势包括：

- 分析当前位置的数据，防止执行将分析数据移动到 HDFS 等 Hadoop 基础架构这一耗时且性能极高的任务。
- 将副本数量从三个减少为一个。
- 支持用户分离计算和存储，以便独立扩展。
- 利用 ONTAP 丰富的数据管理功能提供企业数据保护。
- 已通过 Hortonworks 数据平台认证。

- 支持混合数据分析部署。
- 利用动态多线程功能缩短备份时间。

大数据的组件

由 NetApp 提供支持的 Data Fabric 集成了数据管理服务和应用程序（组件），用于实现数据访问，控制，保护和安全性，如下图所示。



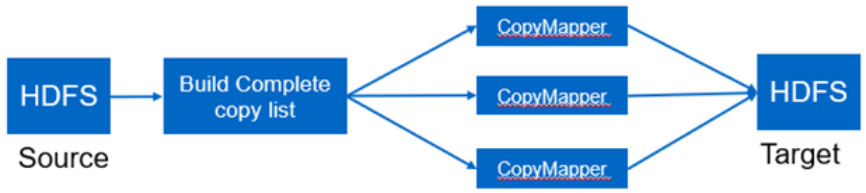
上图中的组件包括：

- * NetApp NFS 直接访问。 * 为最新的 Hadoop 和 Spark 集群提供对 NetApp NFS 卷的直接访问，而无需额外的软件或驱动程序要求。
- * NetApp Cloud Volumes ONTAP 和云卷服务。 * 基于在 Microsoft Azure 云服务的 Amazon Web Services （ AWS ） 或 Azure NetApp Files （ ANF ） 中运行的 ONTAP 的软件定义的连接存储。
- * NetApp SnapMirror 技术 * 。在内部部署和 ONTAP 云或 NPS 实例之间提供数据保护功能。
- * 云服务提供商。 * 这些提供商包括 AWS ， Microsoft Azure ， Google Cloud 和 IBM Cloud 。
- * PaaaS 。 * 基于云的分析服务，例如 AWS 中的 Amazon Elastic MapReduce （ EMR ）和 Databricks 以及 Microsoft Azure HDInsight 和 Azure Databricks 。

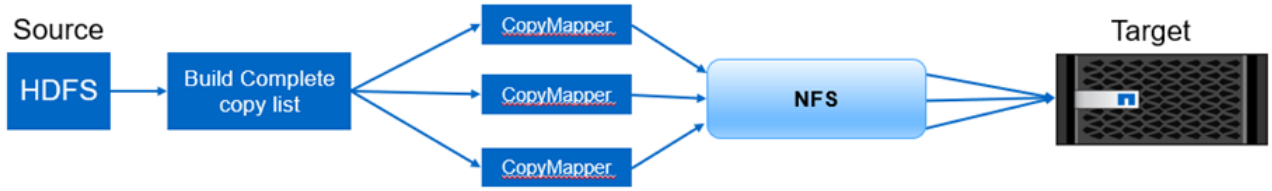
Hadoop 数据保护和 NetApp

Hadoop DistCp 是一款原生工具，用于大型集群间和集群内复制。下图所示的 Hadoop DistCp 基本过程是一个典型的备份工作流，它使用诸如 MapReduce 等 Hadoop 原生工具将 Hadoop 数据从 HDFS 源复制到相应目标。

通过 NetApp NFS 直接访问，客户可以将 NFS 设置为 Hadoop DistCp 工具的目标，以便通过 MapReduce 将数据从 HDFS 源复制到 NFS 共享。NetApp NFS 直接访问可用作 DistCp 工具的 NFS 驱动程序。



Hadoop Distcp Basic Process



Hadoop Distcp and NetApp

Hadoop 数据保护用例概述

本节将简要介绍数据保护使用情形的问题描述，这是本白皮书的重点内容。其余部分将详细介绍每个使用情形，例如客户问题（场景），要求和挑战以及解决方案。

用例 **1**： 备份 **Hadoop** 数据

在此使用情形中、NetApp NFS卷帮助一家大型金融机构将备份时间从超过24小时缩短到几小时以内。

用例 **2**： 从云到内部环境的备份和灾难恢复

一家大型广播公司利用 NetApp 提供支持的 Data Fabric 作为组件，根据不同的数据传输模式（例如按需，即时， 或基于 Hadoop/Spark 集群负载。

用例 **3**： 对现有 **Hadoop** 数据启用 **DevTest**

NetApp 解决方案帮助在线音乐分销商在不同的分支中快速构建多个节省空间的 Hadoop 集群，以便使用计划的策略创建报告并运行每日 DevTest 任务。

用例 **4**： 数据保护和多云连接

一家大型服务提供商使用由 NetApp 提供支持的 Data Fabric ，从不同的云实例为其客户提供多云分析。

用例 **5**： 加快分析工作负载的速度

最大的金融服务和投资银行之一使用 NetApp 网络连接存储解决方案来缩短 I/O 等待时间并加快其量化金融分析平台的运行速度。

用例 1： 备份 Hadoop 数据

场景

在这种情况下，客户拥有一个大型内部 Hadoop 存储库，并希望对其进行备份以实现灾难恢复。但是，客户当前的备份解决方案成本高昂，并且备份时间过长，超过 24 小时。

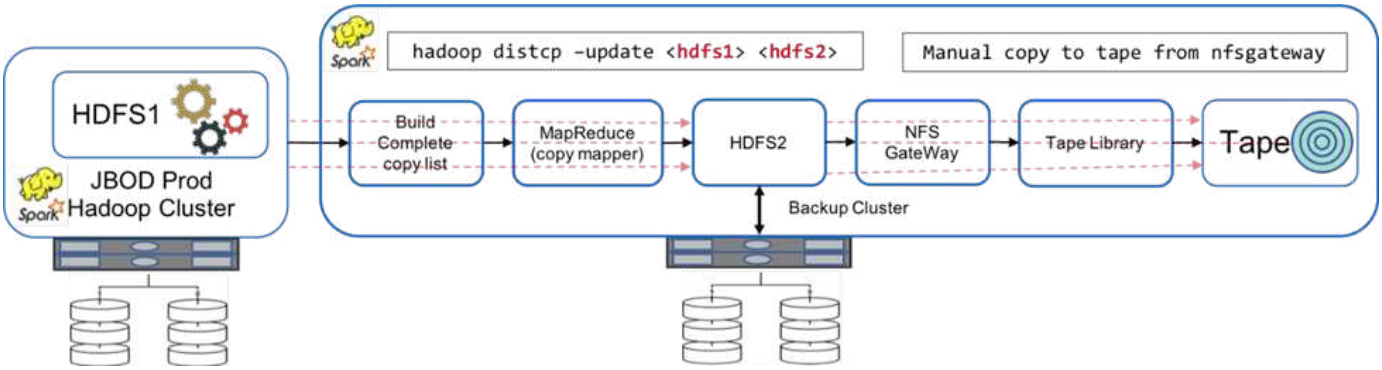
要求和挑战

此用例的主要要求和挑战包括：

- 软件向后兼容性：
  - 建议的备用备份解决方案应与生产 Hadoop 集群中当前正在运行的软件版本兼容。
- 为了满足承诺的 SLA ，建议的替代解决方案应实现极低的 RPO 和 RTO 。
- NetApp 备份解决方案创建的备份可以在本地构建于数据中心的 Hadoop 集群中使用，也可以在远程站点的灾难恢复位置运行的 Hadoop 集群中使用。
- 建议的解决方案必须经济高效。
- 建议的解决方案必须减少备份期间对当前正在运行的生产分析作业的性能影响。

客户的现有备份解决方案

下图显示了原始 Hadoop 原生备份解决方案。



生产数据通过中间备份集群保护到磁带：

- 通过运行 `hadoop distcp -update <hdfs1> <hdfs2>` 命令，可以将 HDFS1 数据复制到 HDFS2 。
- 备份集群充当 NFS 网关，数据通过 Linux `CP` 命令通过磁带库手动复制到磁带。

原始 Hadoop 原生备份解决方案的优势包括：

- 解决方案基于 Hadoop 原生命令，用户无需学习新过程。
- 解决方案采用行业标准架构和硬件。

原始 Hadoop 原生备份解决方案的缺点包括：

- 备份窗口时间过长超过 24 小时，因此生产数据容易受到影响。
- 备份期间集群性能显著下降。
- 复制到磁带是一个手动过程。

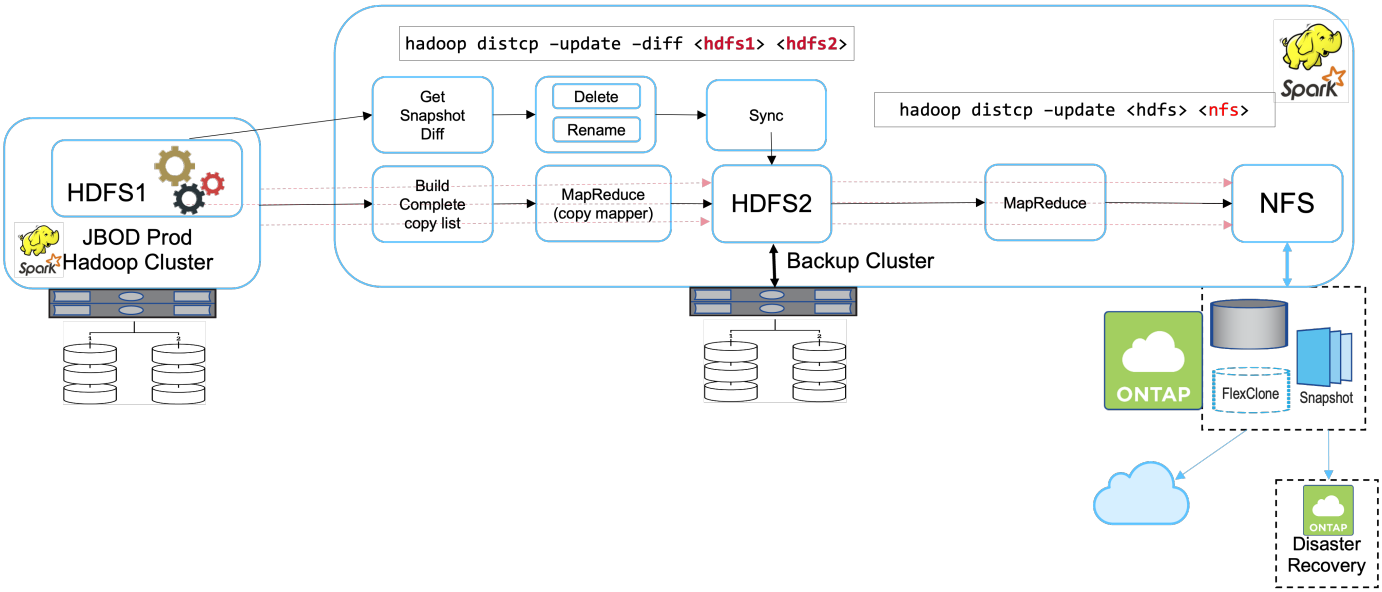
- 备份解决方案在所需的硬件和手动过程所需的工时方面成本高昂。

备份解决方案

根据这些挑战和要求，并考虑到现有备份系统，建议了三种可能的备份解决方案。以下各小节分别介绍了这三种不同的备份解决方案，分别标记为解决方案 A 到解决方案 C

解决方案 A

在解决方案A中、备份Hadoop集群会将二级备份发送到NetApp NFS存储系统、从而无需使用磁带、如下图所示。



解决方案 A 的详细任务包括：

- 生产 Hadoop 集群的 HDFS 中包含需要保护的客户分析数据。
- 具有 HDFS 的备份 Hadoop 集群充当数据的中间位置。在生产和备份 Hadoop 集群中，只需一组磁盘（JBOD）即可为 HDFS 提供存储。
- 运行 `hadoop distcp - update - diff <hdfs1> <hdfs2>` 命令，保护 Hadoop 生产数据，使其从生产集群 HDFS 传输到备份集群 HDFS 。

> Hadoop 快照用于保护数据从生产环境传输到备份 Hadoop 集群。

- NetApp ONTAP 存储控制器可提供一个 NFS 导出卷，该卷会配置到备份 Hadoop 集群。
- 运行 `Hadoop distcp` 命令利用Mapreduce"和多个映射程序、可以保护分析数据从Hadoop集群备份到NFS。

  将数据存储在 NetApp 存储系统的 NFS 中后，将使用 NetApp Snapshot ， SnapRestore 和 FlexClone 技术根据需要备份，还原和复制 Hadoop 数据。

> 使用 SnapMirror 技术，可以将 Hadoop 数据保护到云端以及灾难恢复位置。

解决方案 A 的优势包括：

- Hadoop 生产数据受备份集群保护。
- HDFS 数据通过 NFS 进行保护，从而可以保护云和灾难恢复位置。
- 通过将备份操作卸载到备份集群来提高性能。
- 无需手动执行磁带操作
- 支持通过 NetApp 工具实现企业管理功能。
- 只需对现有环境进行极少的更改。
- 是一种经济高效的解决方案。

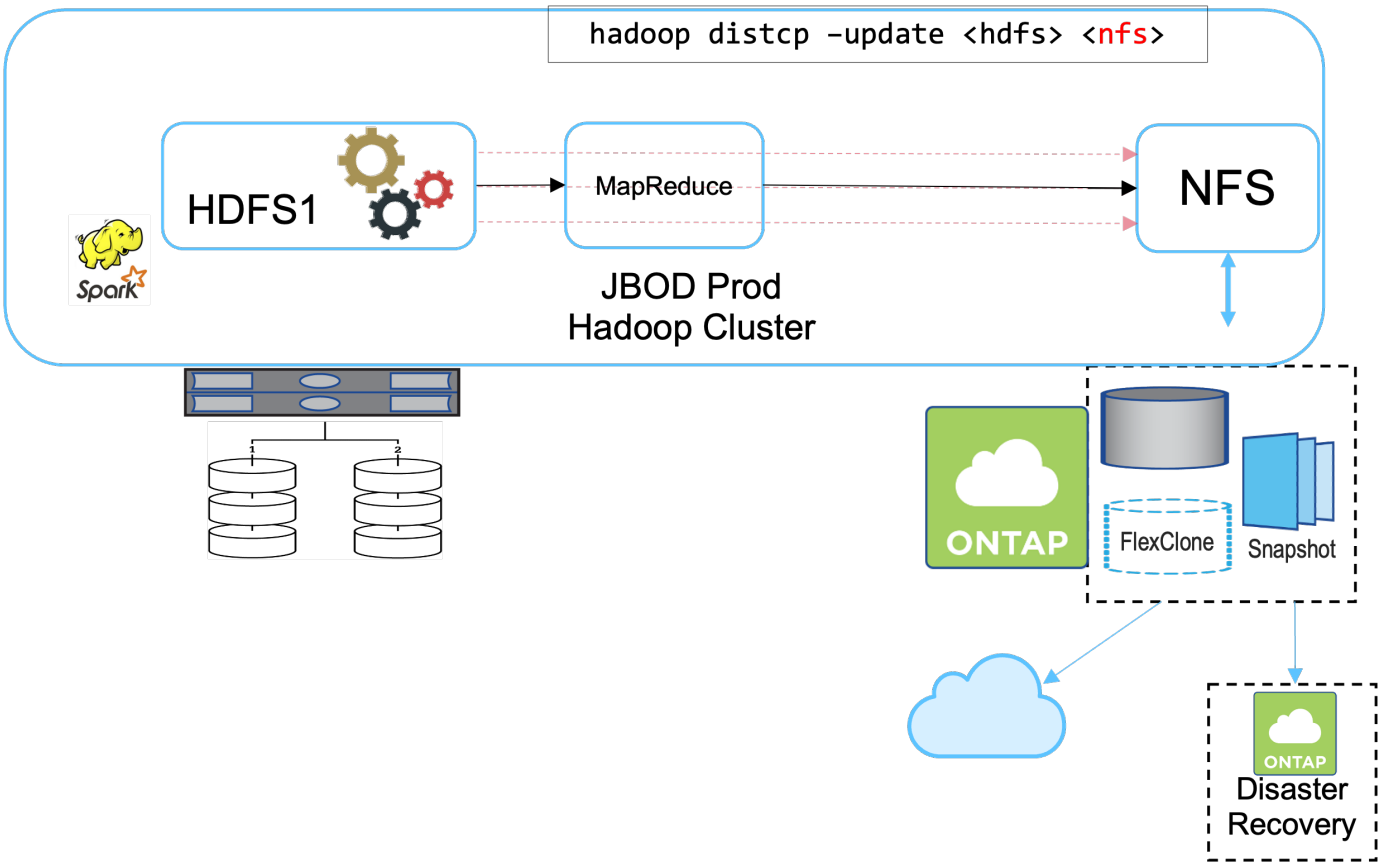
此解决方案的缺点是，它需要一个备份集群和额外的映射器来提高性能。

由于解决方案 A 的简单性，成本和整体性能，客户最近部署了它。

在此解决方案中，可以使用 ONTAP 中的 SAN 磁盘，而不是 JBOD 。此选项会将备份集群存储负载卸载到 ONTAP ；但是，缺点是需要 SAN 网络结构交换机。

解决方案 B

解决方案B会将NFS卷添加到生产Hadoop集群中、从而无需备份Hadoop集群、如下图所示。



解决方案 B 的详细任务包括：

- NetApp ONTAP 存储控制器可为生产 Hadoop 集群配置 NFS 导出。

  Hadoop本机 `hadoop distcp` 命令可保护Hadoop数据从生产集群HDFS到NFS。

- 将数据存储在 NetApp 存储系统的 NFS 中后，将使用 Snapshot ， SnapRestore 和 FlexClone 技术根据需要备份，还原和复制 Hadoop 数据。

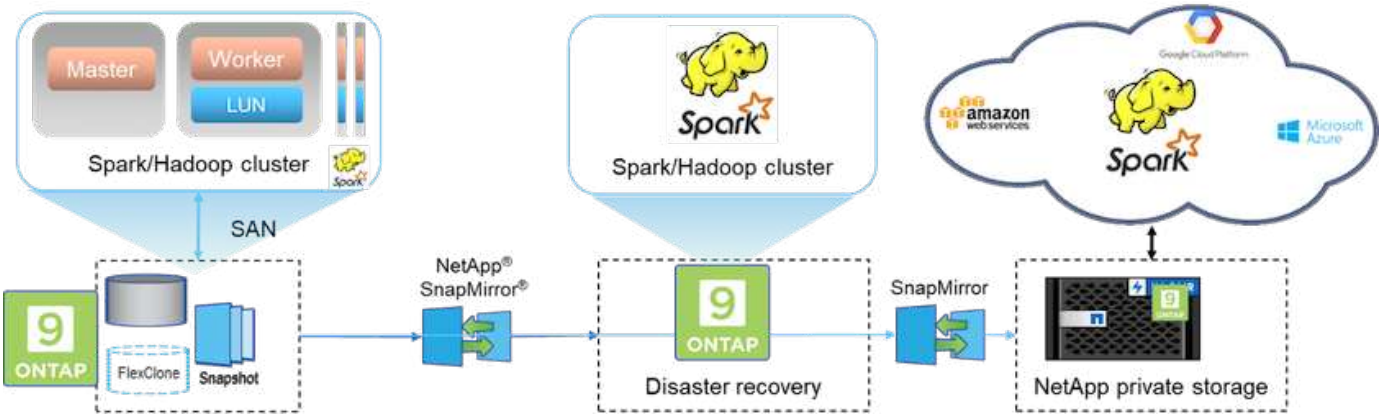
解决方案 B 的优势包括：

- 生产集群针对备份解决方案进行了少许修改，从而简化了实施并降低了额外的基础架构成本。
- 备份操作不需要备份集群。
- HDFS 生产数据在转换为 NFS 数据时会受到保护。
- 解决方案支持通过 NetApp 工具执行企业管理功能。

此解决方案的缺点是它在生产集群中实施，这可能会在生产集群中添加其他管理员任务。

解决方案 C

在解决方案 C 中， NetApp SAN 卷会直接配置到 Hadoop 生产集群中以用于 HDFS 存储，如下图所示。



解决方案 C 的详细步骤包括：

- NetApp ONTAP SAN 存储在生产 Hadoop 集群上配置为用于 HDFS 数据存储。
- NetApp Snapshot 和 SnapMirror 技术用于备份生产 Hadoop 集群中的 HDFS 数据。
- 在 Snapshot 副本备份过程中， Hadoop/Spark 集群的生产不会对性能产生影响，因为备份位于存储层。

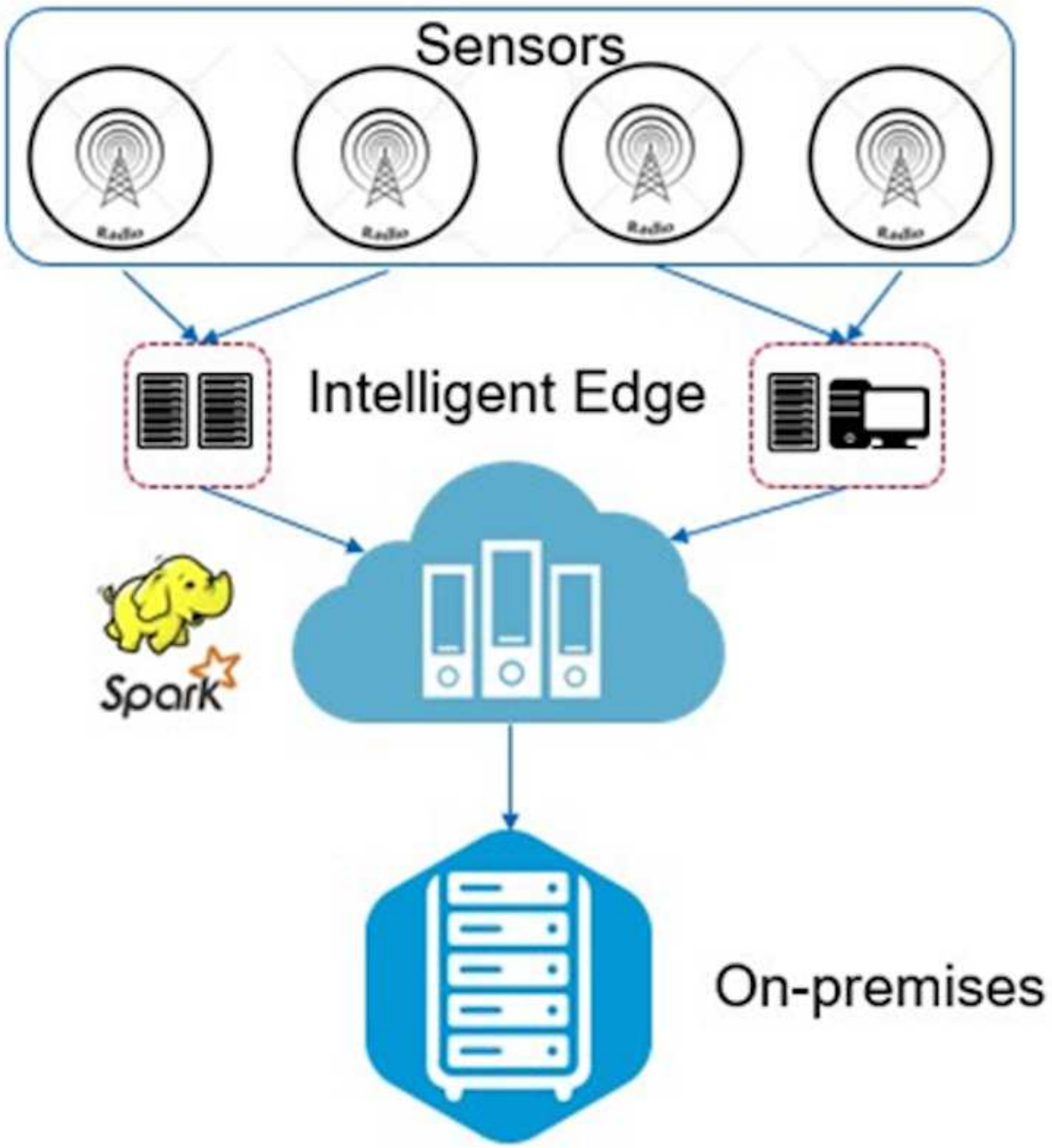
> 无论数据大小如何， Snapshot 技术均可在数秒内完成备份。

解决方案 C 的优势包括：

- 可以使用 Snapshot 技术创建节省空间的备份。
- 支持通过 NetApp 工具实现企业管理功能。

用例 2： 从云到内部环境的备份和灾难恢复

此使用情形基于广播客户，该客户需要将基于云的分析数据备份到其内部数据中心，如下图所示。



场景

在这种情况下， IoT 传感器数据会载入云中，并使用 AWS 中的开源 Apache Spark 集群进行分析。需要将处理后的数据从云备份到内部环境。

要求和挑战

此用例的主要要求和挑战包括：

- 启用数据保护不会发生原因对云中的生产 Spark 或 Hadoop 集群产生任何性能影响。
- 云传感器数据需要以高效安全的方式移动和保护到内部。
- 可以在不同条件下灵活地将数据从云传输到内部环境，例如按需，瞬时以及在集群负载较低的情况下。

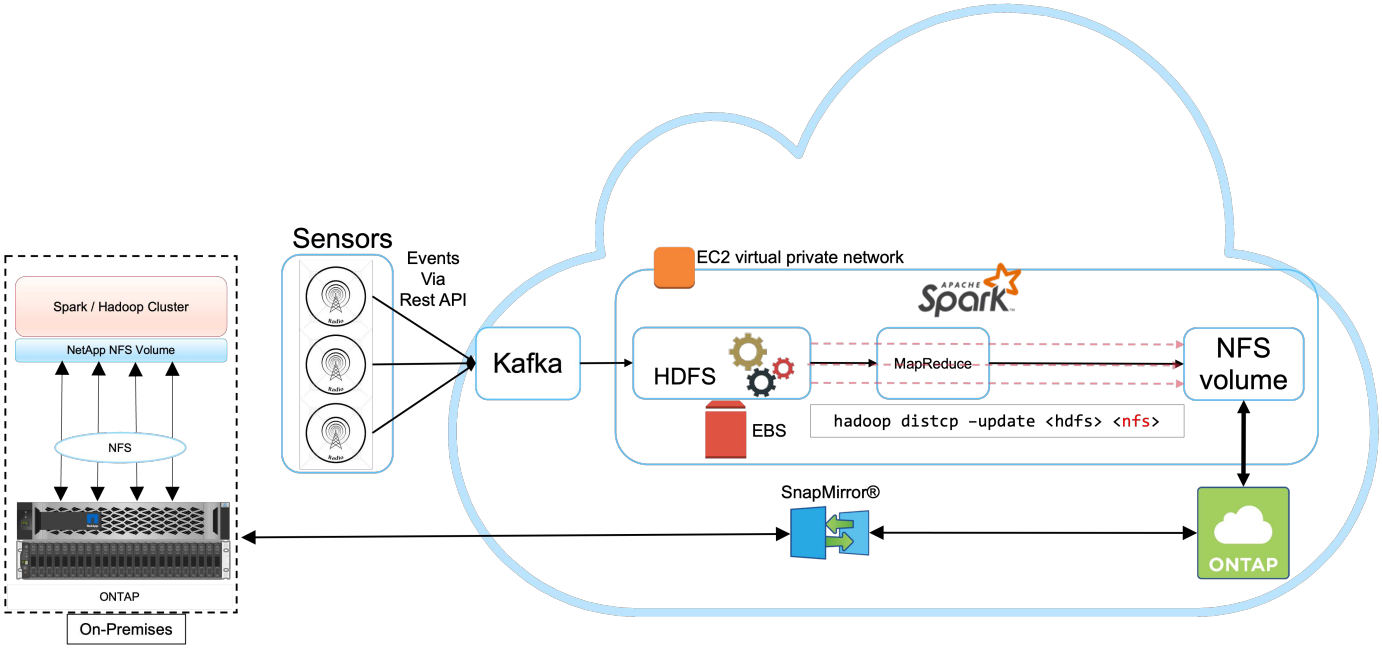
解决方案

客户使用 AWS Elastic Block Store （ EBS ）作为其 Spark 集群 HDFS 存储，通过 Kafka 从远程传感器接收和载入数据。因此， HDFS 存储充当备份数据的源。

为了满足这些要求， NetApp ONTAP 云部署在 AWS 中，并创建一个 NFS 共享作为 Spark 或 Hadoop 集群的备份目标。

创建NFS共享后、将数据从HDFS EBS存储复制到ONTAP NFS共享中。数据驻留在 ONTAP 云的 NFS 中后，可以根据需要使用 SnapMirror 技术将数据从云镜像到内部存储，从而实现安全高效的方式。

此图显示了从云到内部解决方案的备份和灾难恢复。



用例 3： 对现有 Hadoop 数据启用 DevTest

在此使用情形中，客户要求在同一数据中心和远程位置快速高效地基于包含大量分析数据以用于 DevTest 和报告目的的现有 Hadoop 集群构建新的 Hadoop/Spark 集群。

场景

在这种情况下，多个 Spark 或 Hadoop 集群是通过在内部以及灾难恢复位置实施大型 Hadoop 数据湖而构建的。

要求和挑战

此用例的主要要求和挑战包括：

- 创建多个 Hadoop 集群以实现 DevTest ， QA 或任何其他需要访问相同生产数据的目的。此处的挑战是，以节省空间的方式瞬时克隆多个非常大的 Hadoop 集群。
- 将 Hadoop 数据同步到 DevTest 和报告团队，以提高运营效率。
- 使用相同的凭据在生产集群和新集群之间分发 Hadoop 数据。

- 使用计划策略高效创建 QA 集群，而不会影响生产集群。

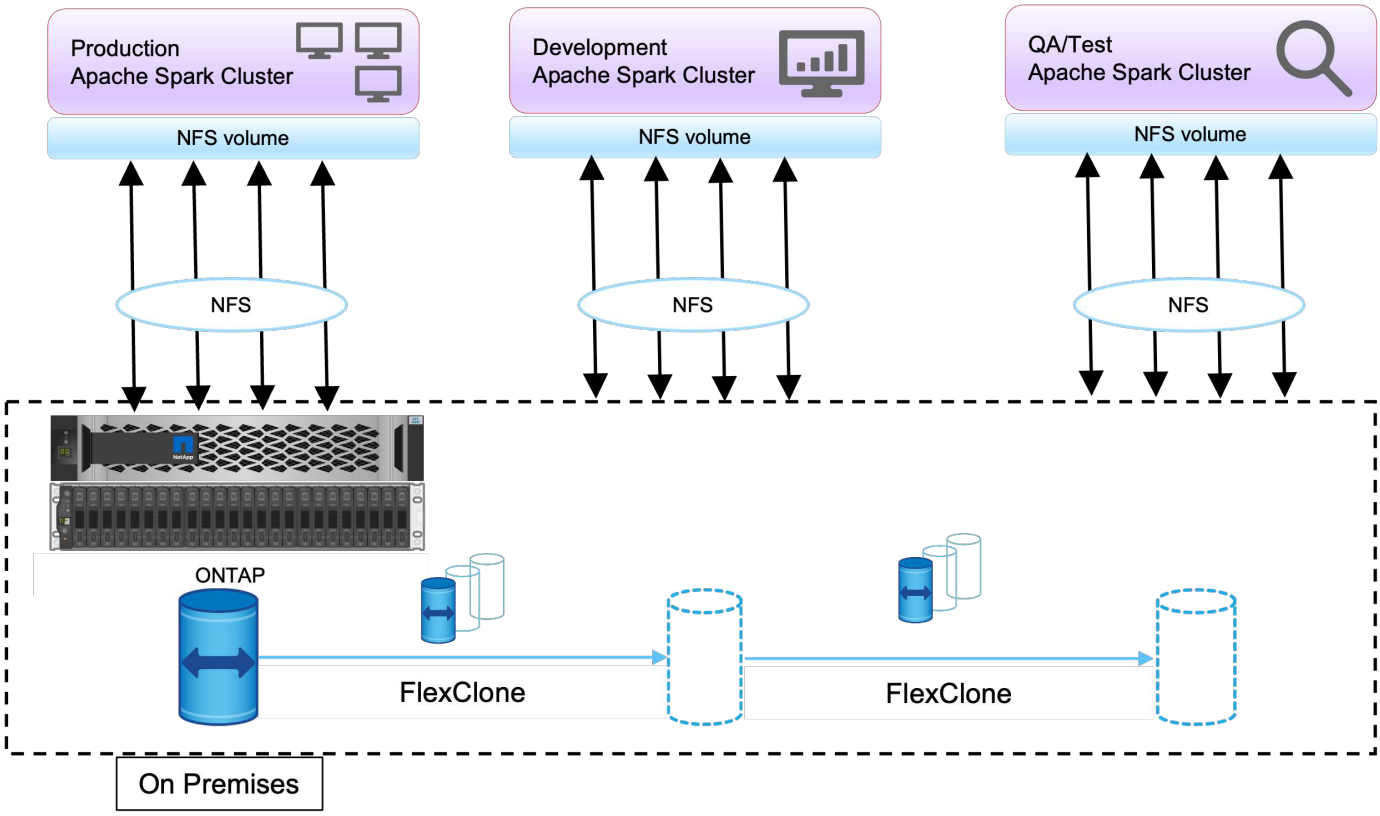
解决方案

FlexClone 技术用于问题解答上述要求。FlexClone 技术是 Snapshot 副本的读 / 写副本。它会从父 Snapshot 副本数据读取数据，并且只会为新的 / 修改的块占用额外空间。速度快，节省空间。

首先，使用 NetApp 一致性组创建现有集群的 Snapshot 副本。

NetApp System Manager 或存储管理员提示符中的 Snapshot 副本。一致性组 Snapshot 副本是应用程序一致的组 Snapshot 副本， FlexClone 卷是根据一致性组 Snapshot 副本创建的。值得一提的是， FlexClone 卷会继承父卷的 NFS 导出策略。创建 Snapshot 副本后，必须安装一个新的 Hadoop 集群以用于 DevTest 和报告目的，如下图所示。从新Hadoop集群克隆的NFS卷可访问NFS数据。

此图显示了用于 DevTest 的 Hadoop 集群。



用例 4： 数据保护和多云连接

此使用情形适用于负责为客户的大数据分析数据提供多云连接的云服务合作伙伴。

场景

在这种情况下，在 AWS 中从不同来源接收的物联网数据存储在 NPS 的中央位置。NPS 存储连接到 AWS 和 Azure 中的 Spark 或 Hadoop 集群，从而支持在多个云中运行的大数据分析应用程序访问相同的数据。

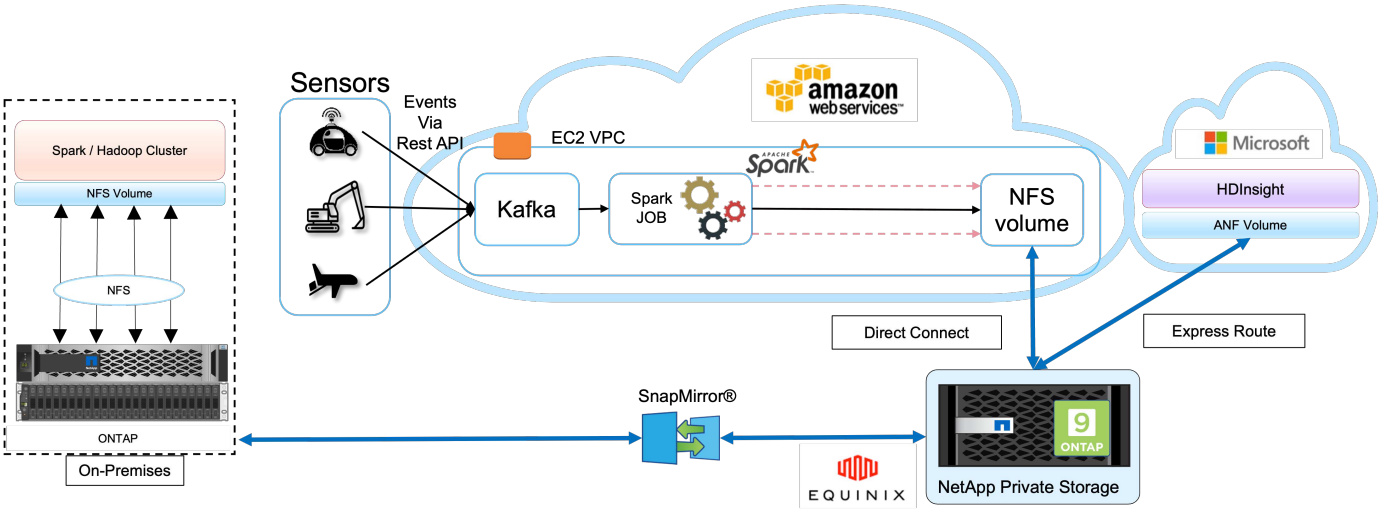
要求和挑战

此用例的主要要求和挑战包括：

- 客户希望使用多个云对相同数据运行分析作业。
- 必须通过不同的传感器和中心从内部和云等不同来源接收数据。
- 解决方案必须高效且经济高效。
- 主要挑战是构建一个经济高效的解决方案，以便在内部和不同云之间提供混合分析服务。

解决方案

此图显示了数据保护和多云连接解决方案。



如上图所示，来自传感器的数据会通过 Kafka 流式传输并输入到 AWS Spark 集群中。数据存储在 NPS 中的 NFS 共享中， NPS 位于 Equinix 数据中心内的云提供商之外。由于NetApp NPS分别通过Direct Connect和Express Route连接到Amazon AWS和Microsoft Azure、因此客户可以从Amazon和AWS分析集群访问NFS数据。这种方法解决了跨多个超大规模云提供商进行云分析的问题。

因此，由于内部和 NPS 存储均运行 ONTAP 软件，因此 SnapMirror 可以将 NPS 数据镜像到内部集群，从而在内部和多个云之间提供混合云分析。

为了获得最佳性能， NetApp 通常建议使用多个网络接口和直接连接 / 快速路由从云实例访问数据。

用例 5： 加快分析工作负载的速度

在这种情况下，一家大型金融服务和投资银行的分析平台使用 NetApp NFS 存储解决方案进行了现代化改造，从而显著改进了资产管理和定量业务部门的投资风险和衍生产品分析。

场景

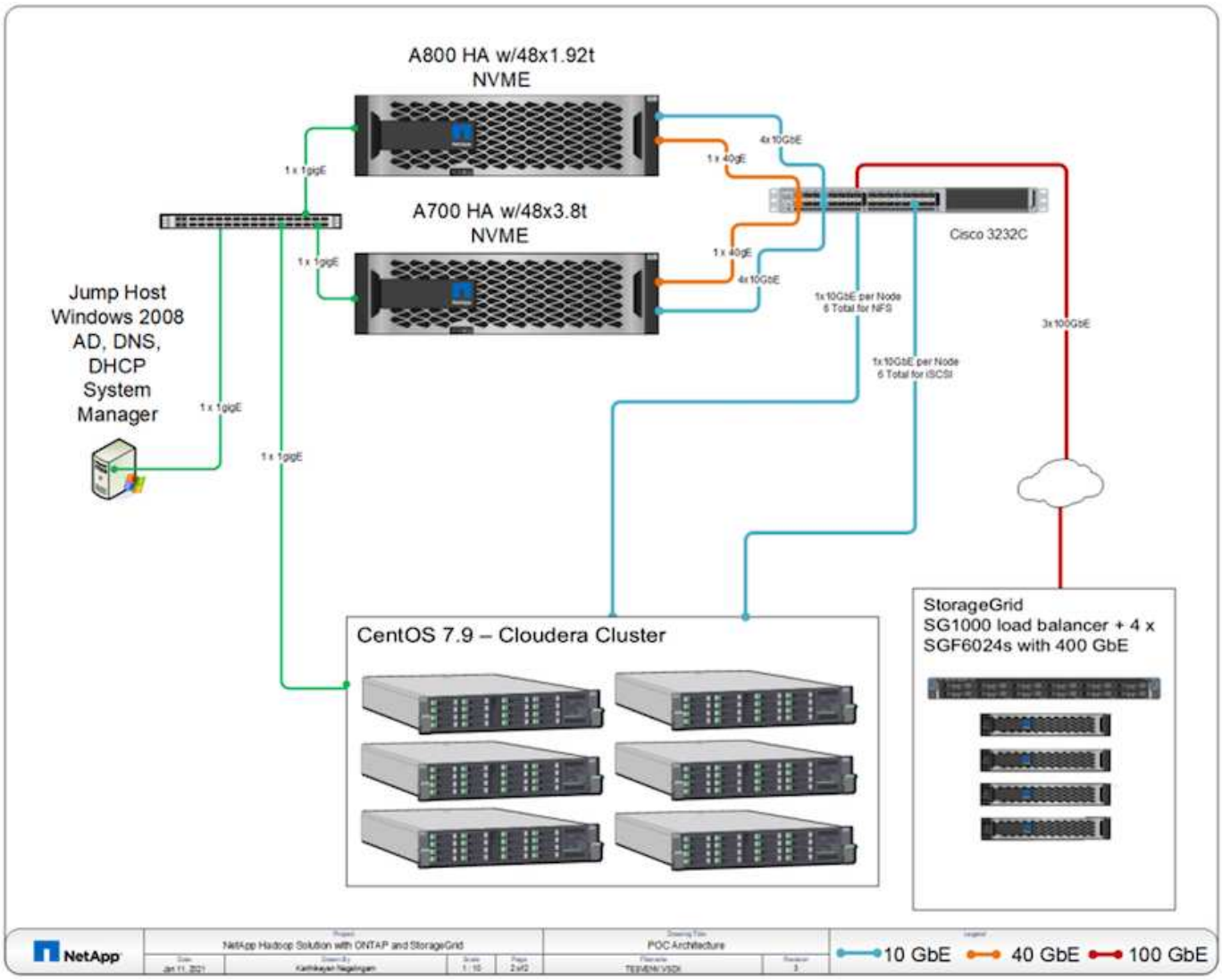
在客户的现有环境中，用于分析平台的 Hadoop 基础架构利用了 Hadoop 服务器的内部存储。由于 JBOD 环境的专有性，组织内的许多内部客户无法利用其蒙特卡罗量化模型，这种模拟依赖于重复的实时数据样本。无法以最佳方式了解市场波动的不确定性影响，这对量化资产管理业务单位不利。

要求和挑战

该银行的量化业务部门需要一种高效的预测方法来实现准确，及时的预测。为此，该团队认识到需要对基础架构进行现代化改造，减少现有 I/O 等待时间并提高 Hadoop 和 Spark 等分析应用程序的性能，以便高效模拟投资模型，衡量潜在收益并分析风险。

解决方案

客户的现有 Spark 解决方案具有 JBOD 。然后，利用 NetApp ONTAP ， NetApp StorageGRID 和从 MinIO 网关到 NFS ，减少了银行的量化财务团队对评估潜在收益和风险的投资模型进行模拟和分析的 I/O 等待时间。此图显示了采用 NetApp 存储的 Spark 解决方案。



如上图所示，我们部署了 AFF A800 ， A700 系统和 StorageGRID ，以便在具有 Spark 的六节点 Hadoop 集群中通过 NFS 和 S3 协议访问 parquet 文件，并为数据分析操作提供了 YARN 和 Hive 元数据服务。

客户旧环境中的直连存储（ DAS ）解决方案在独立扩展计算和存储方面存在缺点。借助 NetApp ONTAP 解决方案 for Spark ，该银行的财务分析业务部门能够将存储与计算分离，并根据需要无缝地更有效地提供基础架构资源。

通过将 ONTAP 与 NFS 结合使用，计算服务器 CPU 几乎可以完全用于 Spark SQL 作业， I/O 等待时间缩短了近 70% ，从而为 Spark 工作负载提供了更好的计算能力和性能提升。随后，随着 CPU 利用率的提高，客户还

可以利用 GPUDirect 等 GPU 进一步实现平台现代化。此外， StorageGRID 还为 Spark 工作负载提供了低成本存储选项，而 MinIO 网关则可通过 S3 协议安全访问 NFS 数据。对于云中的数据， NetApp 建议使用 Cloud Volumes ONTAP ， Azure NetApp Files 和 NetApp Cloud Volumes Service 。

结论

本节概述了 NetApp 为满足各种 Hadoop 数据保护要求而提供的使用情形和解决方案。通过使用 NetApp 提供支持的 Data Fabric ，客户可以：

- 利用 NetApp 丰富的数据管理功能并与 Hadoop 原生工作流集成，灵活选择合适的数据保护解决方案。
- 将 Hadoop 集群备份窗口时间缩短近 70% 。
- 消除 Hadoop 集群备份所带来的任何性能影响。
- 同时提供从不同云提供商到单个分析数据源的多云数据保护和数据访问。
- 使用 FlexClone 技术快速创建节省空间的 Hadoop 集群副本。

从何处查找追加信息

要了解有关本文档中所述信息的更多信息，请参见以下文档和 / 或网站：

- NetApp 大数据分析解决方案

  "https://www.netapp.com/us/solutions/applications/big-data-analytics/index.aspx"

- 采用 NetApp 存储的 Apache Spark 工作负载

  https://www.netapp.com/pdf.html?item=/media/26877-nva-1157-deploy.pdf

- 适用于 Apache Spark 的 NetApp 存储解决方案

  "https://www.netapp.com/media/16864-tr-4570.pdf"

- NetApp 支持的 Data Fabric 上的 Apache Hadoop

  "https://www.netapp.com/media/16877-tr-4529.pdf"

致谢

- NetApp ANZ 维多利亚地区销售代表 Paul Burland
- NetApp 业务开发经理 Hoseb Dermanilian
- NetApp MPSG 总监 Lee Dorrier
- NetApp ANZ 维多利亚区 SE 系统工程师 David Thiessen

版本历史记录

version	Date	文档版本历史记录
版本 1.0	2018 年 1 月	初始版本。

version	Date	文档版本历史记录
版本 2.0	2021年10月	更新了用例 5： 加快分析工作负载的速度
版本 3.0	2023年11月	已删除NIPAM详细信息

现代数据分析—适用于不同分析策略的不同解决方案

本白皮书介绍了NetApp的现代数据分析解决方案 战略。其中包括有关业务成果、客户挑战、技术趋势、竞争传统架构、现代工作流、用例、行业、云、技术合作伙伴、数据移动者、NetApp Active IQ 、NetApp DataOps工具包、Hadoop to Spark、采用NetApp Astra Control的软件定义存储、容器、企业数据管理、归档和分层、以实现AI和分析的目标、以及NetApp和客户如何携手打造现代化的数据架构。

"现代数据分析—适用于不同分析策略的不同解决方案"

TR-4623：NetApp E系列E5700和Splunk Enterprise

NetApp公司Mitch Blackburn

TR-4623介绍了NetApp E系列和Splunk设计的集成架构。针对节点存储平衡、可靠性、性能、存储容量和密度进行了优化、此设计采用Splunk集群索引节点模式、可扩展性更高、TCO更低。通过将存储与计算分离、可以单独扩展每个存储、从而节省过度配置一个或另一个存储的成本。此外、本文档还总结了从Splunk计算机日志事件模拟工具获得的性能测试结果。

"TR-4623：NetApp E系列E5700和Splunk Enterprise"

NVA-1157-Deploy：采用NetApp Storage解决方案 的Apache Spark工作负载

NetApp 公司 Karthikeyan Nagalingam

NVA-1157-Deploy介绍了在NetApp NFS AFF 存储系统上验证Apache Spark SQL的性能和功能。本文档将根据各种场景审查配置、架构和性能测试、并提出将Spark与NetApp ONTAP 数据管理软件结合使用的建议。此外、本指南还介绍了仅基于一组磁盘(JBOD)与NetApp AFF A800存储控制器的测试结果。

"NVA-1157-Deploy：采用NetApp Storage解决方案 的Apache Spark工作负载"