Distributed training in Azure - Click-Through Rate Prediction

NetApp Solutions

NetApp
November 15, 2022

Table of Contents

Distributed training in Azure - Click-Through Rate Prediction

TR-4904: Distributed training in Azure - Click-Through Rate Prediction

Rick Huang, Verron Martina, Muneer Ahmad, NetApp

The work of a data scientist should be focused on the training and tuning of machine learning (ML) and artificial intelligence (AI) models. However, according to research by Google, data scientists spend approximately 80% of their time figuring out how to make their models work with enterprise applications and run at scale.

To manage end-to-end AI/ML projects, a wider understanding of enterprise components is needed. Although DevOps have taken over the definition, integration, and deployment, these types of components, ML operations target a similar flow that includes AI/ML projects. To get an idea of what an end-to-end AI/ML pipeline touches in the enterprise, see the following list of required components:

- Storage
- Networking
- Databases
- File systems
- Containers
- Continuous integration and continuous deployment (CI/CD) pipeline
- Integrated development environment (IDE)
- Security
- Data access policies
- Hardware
- Cloud
- Virtualization
- Data science toolsets and libraries

Target audience

The world of data science touches multiple disciplines in IT and business:

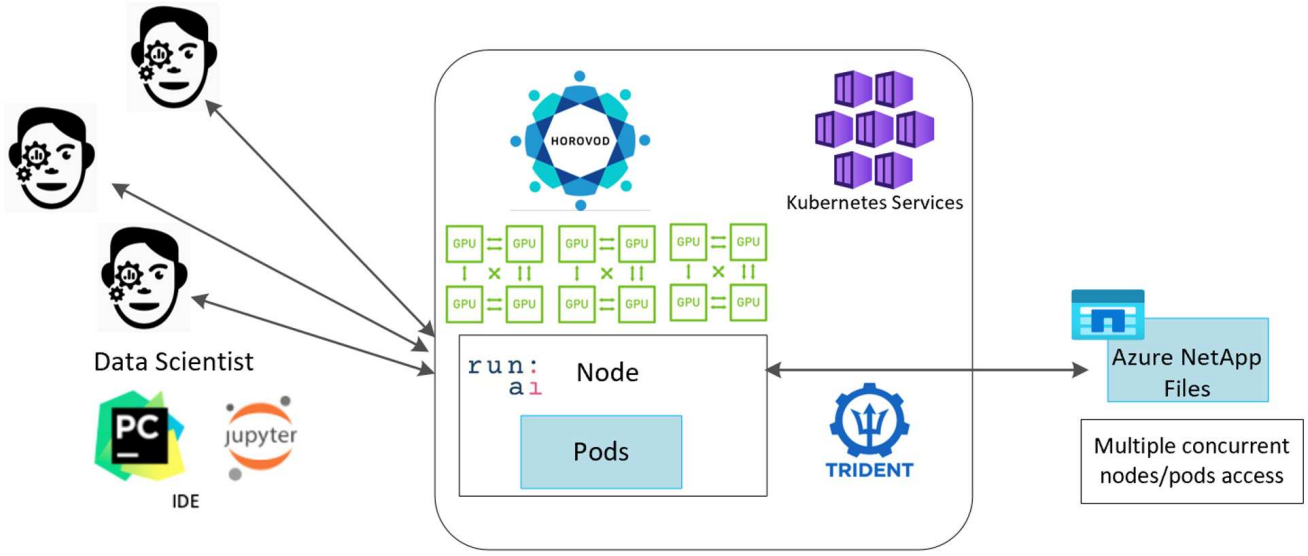
- The data scientist needs the flexibility to use their tools and libraries of choice.
- The data engineer needs to know how the data flows and where it resides.
- A DevOps engineer needs the tools to integrate new AI/ML applications into their CI/CD pipelines.
- Cloud administrators and architects need to be able to set up and manage Azure resources.
- Business users want to have access to AI/ML applications.

In this technical report, we describe how Azure NetApp Files, RAPIDS AI, Dask, and Azure help each of these roles bring value to business.

Solution overview

This solution follows the lifecycle of an AI/ML application. We start with the work of data scientists to define the different steps needed to prepare data and train models. By leveraging RAPIDS on Dask, we perform distributed training across the Azure Kubernetes Service (AKS) cluster to drastically reduce the training time when compared to the conventional Python scikit-learn approach. To complete the full cycle, we integrate the pipeline with Azure NetApp Files.

Azure NetApp Files provides various performance tiers. Customers can start with a Standard tier and scale out and scale up to a high-performance tier nondisruptively without moving any data. This capability enables data scientists to train models at scale without any performance issues, avoiding any data silos across the cluster, as shown in figure below.



Next: Technology overview.

Technology overview

Previous: Introduction.

Microsoft and NetApp

Since May 2019, Microsoft has delivered an Azure native, first-party portal service for enterprise NFS and SMB file services based on NetApp ONTAP technology. This development is driven by a strategic partnership between Microsoft and NetApp and further extends the reach of world-class ONTAP data services to Azure.

Azure NetApp Files

The Azure NetApp Files service is an enterprise-class, high-performance, metered file storage service. Azure NetApp Files supports any workload type and is highly available by default. You can select service and performance levels and set up Snapshot copies through the service. Azure NetApp Files is an Azure first-party service for migrating and running the most demanding enterprise-file workloads in the cloud, including databases, SAP, and high-performance computing applications with no code changes.

This reference architecture gives IT organizations the following advantages:

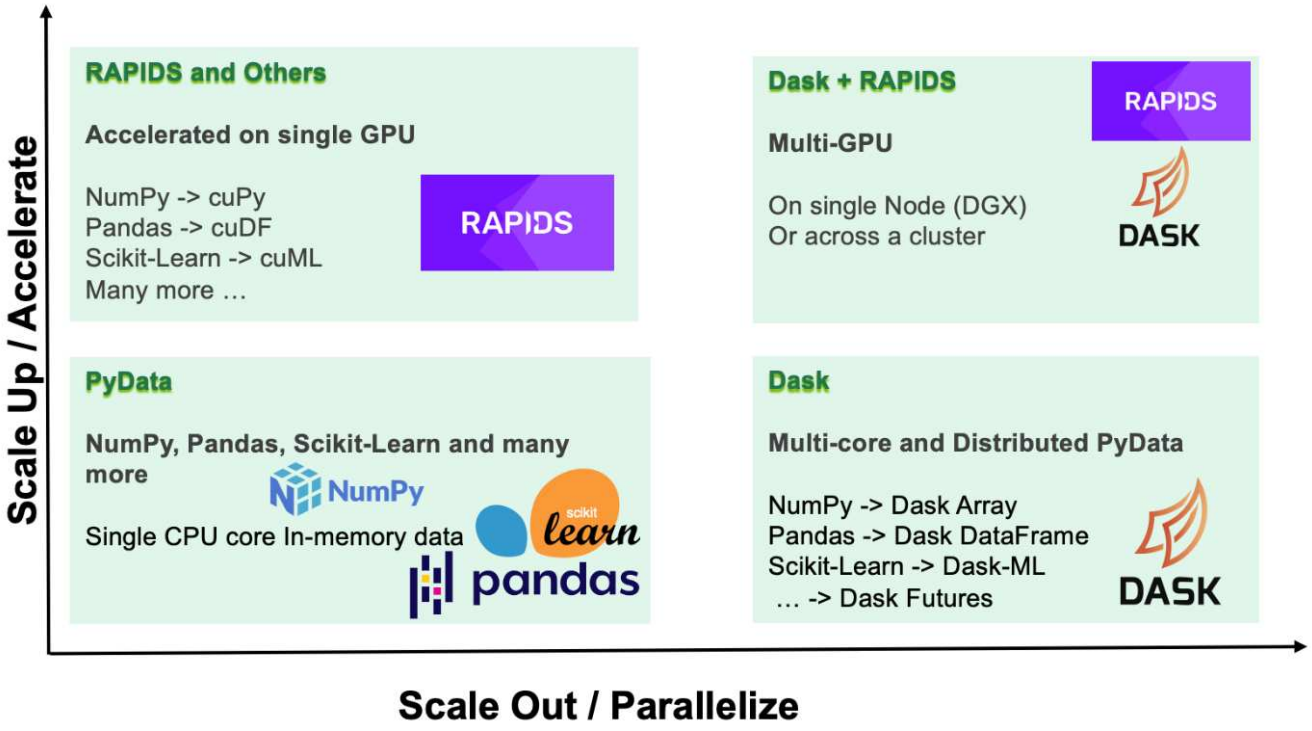
- Eliminates design complexities
- Enables independent scaling of compute and storage
- Enables customers to start small and scale seamlessly
- Offers a range of storage tiers for various performance and cost points

Dask and NVIDIA RAPIDS overview

Dask is an open-source, parallel computing tool that scales Python libraries on multiple machines and provides faster processing of large amounts of data. It provides an API similar to single-threaded conventional Python libraries, such as Pandas, Numpy, and scikit-learn. As a result, native Python users are not forced to change much in their existing code to use resources across the cluster.

NVIDIA RAPIDS is a suite of open-source libraries that makes it possible to run end-to-end ML and data analytics workflows entirely on GPUs. Together with Dask, it enables you to easily scale from GPU workstation (scale up) to multinode, multi-GPU clusters (scale out).

For deploying Dask on a cluster, you could use Kubernetes for resource orchestration. You could also scale up or scale down the worker nodes as per the process requirement, which in-turn can help to optimize the cluster resource consumption, as shown in the following figure.



Next: Software requirements.

Software requirements

Previous: Technology overview.

The following table lists the software requirements needed for this solution.

Software	Version
Azure Kubernetes Service	1.18.14
RAPIDS and Dask container image	Repository: "rapidsai/rapidsai" Tag: 0.17-cuda11.0-runtime-ubuntu18.04
NetApp Trident	20.01.1
Helm	3.0.0

Next: Cloud resource requirements.

Cloud resource requirements

Previous: Software requirements.

Configure Azure NetApp Files

Configure Azure NetApp Files as described in QuickStart: Set up Azure NetApp Files and create an NFS volume.

You can proceed past the section “Create NFS volume for Azure NetApp Files” because you are going to create volumes through Trident. Before continuing, complete the following steps:

1. Register for Azure NetApp Files and NetApp Resource Provider (through the Azure Shell) (link).
2. Create an account in Azure NetApp Files (link).
3. Set up a capacity pool (a minimum 4TB Standard or Premium, depending on your need) (link).The following table lists the network configuration requirements for setting up in the cloud. The Dask cluster and Azure NetApp Files must be in the same Azure Virtual Network (VNet) or a peered VNet.

Resources	Type/version
Azure Kubernetes Service	1.18.14
Agent node	3x Standard_DS2_v2
GPU node	3x Standard_NC6s_v3
Azure NetApp Files	Standard capacity pool
Capacity in TB	4

Next: Click-through rate prediction use case summary.

Click-through rate prediction use case summary

Previous: Cloud resource requirements.

This use case is based on the publicly available Terabyte Click Logs dataset from Criteo AI Lab. With the recent advances in ML platforms and applications, a lot of attention is now on learning at scale. The click-through rate (CTR) is defined as the average number of click-throughs per hundred online ad impressions (expressed as a percentage). It is widely adopted as a key metric in various industry verticals and use cases, including digital marketing, retail, e-commerce, and service providers. Examples of using CTR as an important metric for potential customer traffic include the following:

- **Digital marketing:** In Google Analytics, CTR can be used to gauge how well an advertiser or merchant's keywords, ads, and free listings are performing. A high CTR is a good indication that users find your ads and listings helpful and relevant. CTR also contributes to your keyword's expected CTR, which is a component of Ad Rank.
- **E-commerce:** In addition to leveraging Google Analytics, there are at least some visitor statistics in an e-commerce backend. Although these statistics might not seem useful at first glance, they are typically easy to read and might be more accurate than other information. First-party datasets composed of such statistics are proprietary and are therefore the most relevant to e-commerce sellers, buyers, and platforms. These datasets can be used for setting benchmarks, comparing results to last year and yesterday by constructing a time-series for further analysis.
- **Retail:** Brick-and-mortar retailers can correlate the number of visitors and the number of customers to the CTR. The number of customers can be seen from their point-of-sale history. The CTR from retailers' websites or ad traffic might result in the aforementioned sales. Loyalty programs are another use case, because customers redirected from online ads or other websites might join to earn rewards. Retailers can acquire customers via loyalty programs and record behaviors from sales histories to build a recommendation system that not only predicts consumer buying behaviors in different categories but also personalizes coupons and decreases churn.
- **Service providers:** Telecommunication companies and internet service providers have an abundance of first-party user telemetry data for insightful AI, ML, and analytics use cases. For example, a telecom can leverage its mobile subscribers' web browsing top level domain history logs daily to fine-tune existing models to produce up-to-date audience segmentation, predict customer behavior, and collaborate with advertisers to place real-time ads for better online experience. In such data-driven marketing workflow, CTR is an important metric to reflect conversions.

In the context of digital marketing, Criteo Terabyte Click Logs are now the dataset of reference in assessing the scalability of ML platforms and algorithms. By predicting the click-through rate, an advertiser can select the visitors who are most likely to respond to the ads, analyze their browsing history, and show the most relevant ads based on the interests of the user.

The solution provided in this technical report highlights the following benefits:

- Azure NetApp Files advantages in distributed or large-scale training
- RAPIDS CUDA-enabled data processing (cuDF, cuPy, and so on) and ML algorithms (cuML)
- The Dask parallel computing framework for distributed training

An end-to-end workflow built on RAPIDS AI and Azure NetApp Files demonstrates the drastic improvement in random forest model training time by two orders of magnitude. This improvement is significant comparing to the conventional Pandas approach when dealing with real-world click logs with 45GB of structured tabular data (on average) each day. This is equivalent to a DataFrame containing roughly twenty billion rows. We will demonstrate cluster environment setup, framework and library installation, data loading and processing, conventional versus distributed training, visualization and monitoring, and compare critical end-to-end runtime results in this technical report.

Next: Install and set up the aks cluster.

Setup

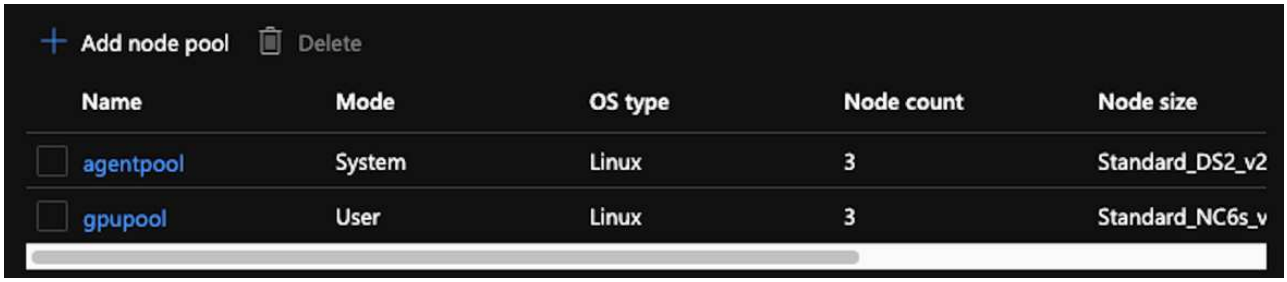
Install and set up the AKS cluster

Previous: Click-through rate prediction use case summary.

To install and set up the AKS cluster, see the webpage Create an AKS Cluster and then complete the following

steps:

1. When selecting the type of node (system [CPU] or worker [GPU] nodes), select the following:
   a. Primary system nodes should be Standard DS2v2 (`agentpool` default three nodes).
   b. Then add the worker node Standard_NC6s_v3 pool (three nodes minimum) for the user group (for GPU nodes) named `gpupool`.



2. Deployment takes 5 to 10 minutes. After it is complete, click Connect to Cluster.
3. To connect to the newly created AKS cluster, install the following from your local environment (laptop/pc):
   a. The Kubernetes command-line tool using the instructions provided for your specific OS
   b. The Azure CLI as described in the document, Install the Azure CLI
4. To access the AKS cluster from the terminal, enter `az login` and enter the credentials.
5. Run the following two commands:

```
az account set --subscription xxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxx
aks get-credentials --resource-group resourcegroup --name aksclustername
```

6. Enter `Azure CLI: kubectl get nodes`.
7. If all six nodes are up and running, as shown in the following example, your AKS cluster is ready and connected to your local environment

```
verronmartina@verron-mac-0 ~ % kubectl get nodes
NAME                                STATUS   ROLES   AGE   VERSION
aks-agentpool-34613062-vmss000000   Ready    agent   22m   v1.18.14
aks-agentpool-34613062-vmss000001   Ready    agent   22m   v1.18.14
aks-agentpool-34613062-vmss000002   Ready    agent   22m   v1.18.14
aks-gpupool-34613062-vmss000000     Ready    agent   20m   v1.18.14
aks-gpupool-34613062-vmss000001     Ready    agent   20m   v1.18.14
aks-gpupool-34613062-vmss000002     Ready    agent   20m   v1.18.14
verronmartina@verron-mac-0 ~ %
```

Next: Create a delegated subnet for Azure NetApp Files.

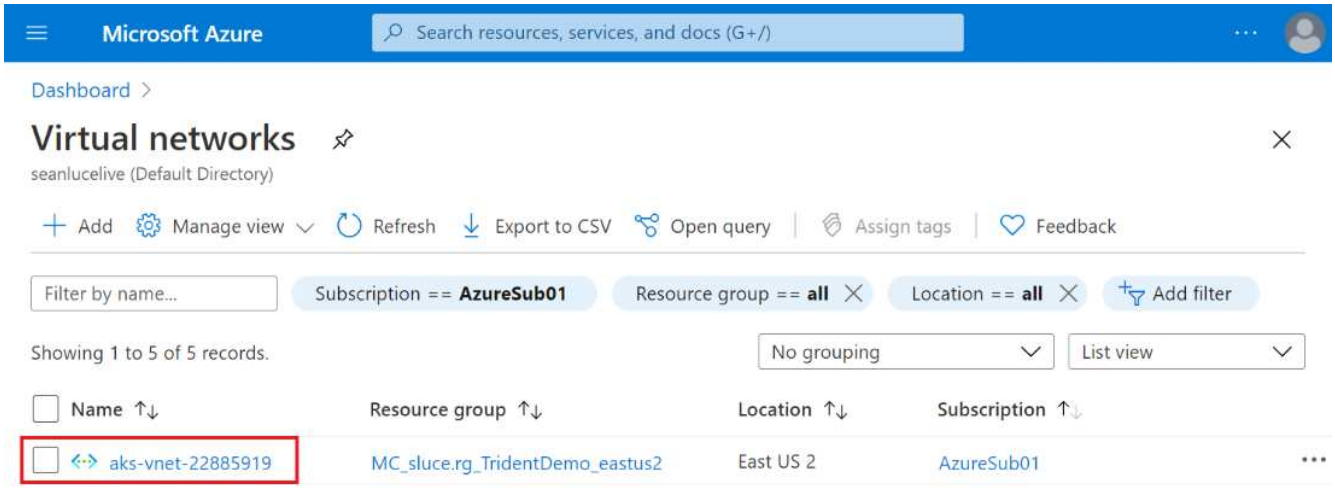
Create a delegated subnet for Azure NetApp Files

Previous: Install and set up the AKS cluster.

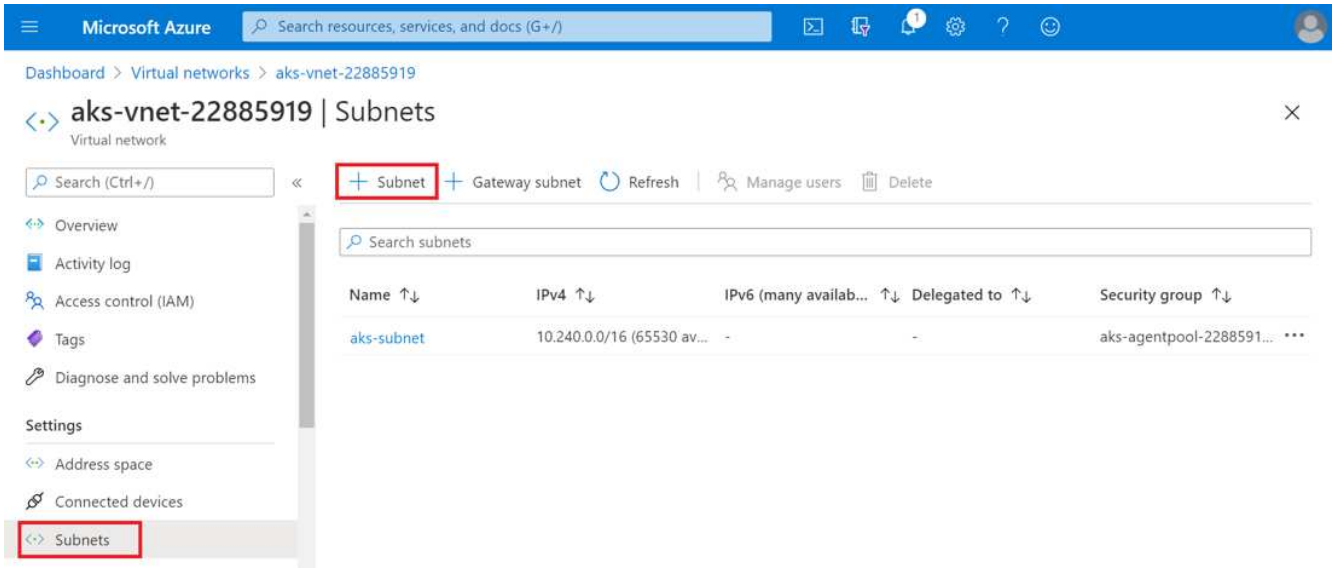
To create a delegated subnet for Azure NetApp Files, complete the following steps:

1. Navigate to Virtual Networks within the Azure portal. Find your newly created virtual network. It should have a prefix such as `aks-vnet`.

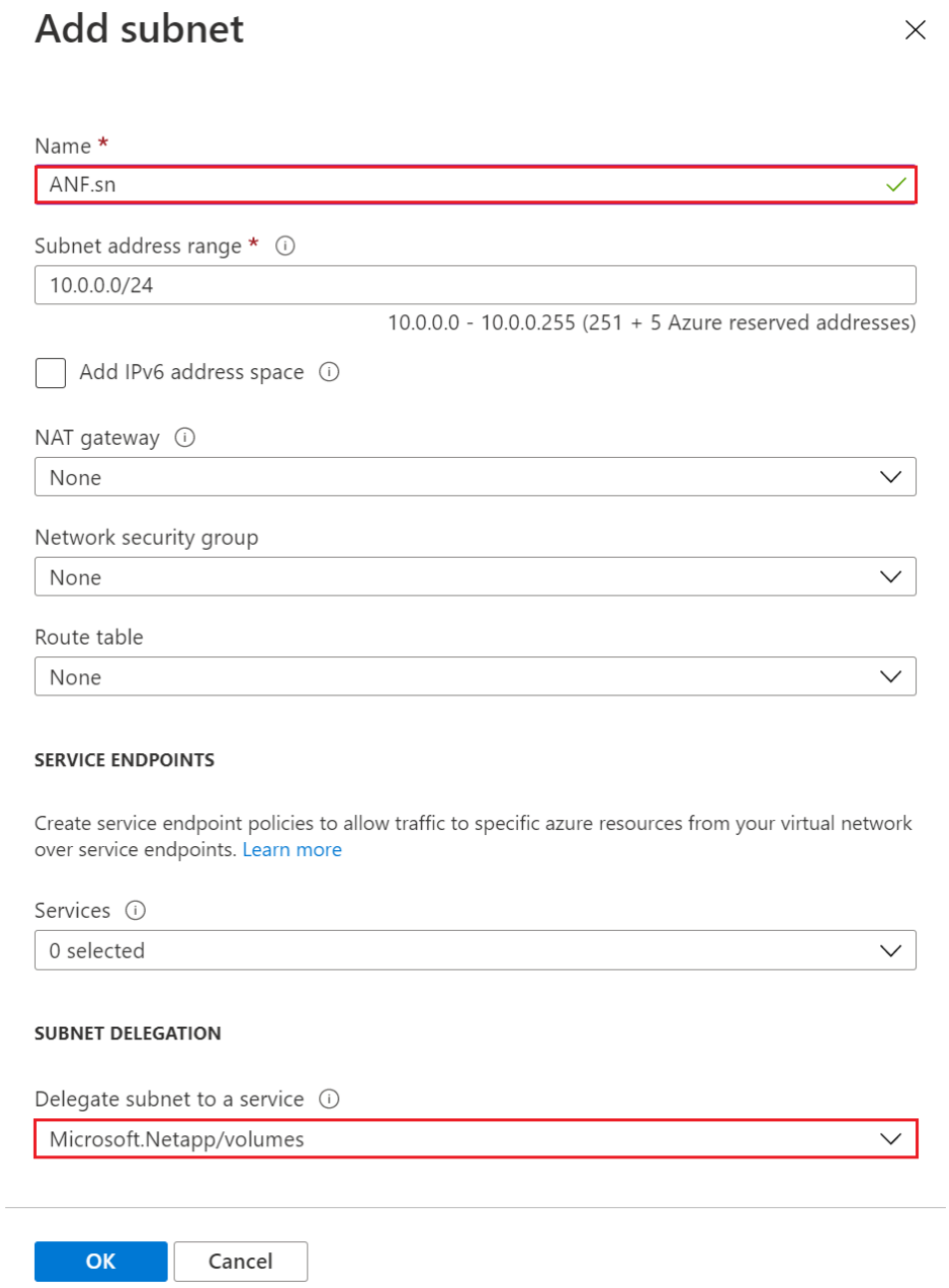
2. Click the name of the VNet.



3. Click Subnets and click +Subnet from the top toolbar.



4. Provide the subnet with a name such as `ANF.sn` and, under the Subnet Delegation heading, select `Microsoft.Netapp/volumes`. Do not change anything else. Click OK.



Azure NetApp Files volumes are allocated to the application cluster and are consumed as persistent volume claims (PVCs) in Kubernetes. In turn, this process provides you the flexibility to map them to different services, such as Jupyter notebooks, serverless functions, and so on.

Users of services can consume storage from the platform in many ways. As this technical report discusses NFSs, the main benefits of Azure NetApp Files are:

- Providing users with the ability to use Snapshot copies.
- Enabling users to store large quantities of data on Azure NetApp Files volumes.
- Using the performance benefits of Azure NetApp Files volumes when running their models on large sets of files.

Next: Peer AKS vnet and Azure NetApp Files vnet.

Peer AKS VNet and Azure NetApp Files VNet

Previous: Create a delegated subnet for Azure NetApp Files.

To peer the AKS VNet to the Azure NetApp Files VNet, complete the following steps:

1. Enter Virtual Networks in the search field.
2. Select `vnet aks-vnet-name.` Click it and enter Peerings in the search field.
3. Click +Add.
4. Enter the following descriptors:
   a. The peering link name is `aks-vnet-name_to_anf`.
   b. subscriptionID and Azure NetApp Files VNet as the VNet peering partner.
   c. Leave all the nonasterisk sections with the default values.
5. Click Add.

For more information, see Create, change, or delete a virtual network peering.

Next: Install Trident.

Install Trident

Previous: Peer AKS VNet and Azure NetApp Files VNet.

To install Trident using Helm, complete the following steps:

1. Install Helm (for installation instructions, visit the source).
2. Download and extract the Trident 20.01.1 installer.

```
$wget
$tar -xf trident-installer-21.01.1.tar.gz
```

3. Change the directory to `trident-installer`.

```
$cd trident-installer
```

4. Copy `tridentctl` to a directory in your system `$PATH`.

```
$sudo cp ./tridentctl /usr/local/bin
```

5. Install Trident on the Kubernetes (K8s) cluster with Helm (source):
   a. Change the directory to the `helm` directory.

```
$cd helm
```

b. Install Trident.

```
$helm install trident trident-operator-21.01.1.tgz --namespace
trident --create-namespace
```

c. Check the status of Trident pods.

```
$kubectl -n trident get pods
```

If all the pods are up and running, then Trident is installed and you can move forward.

6. Set up the Azure NetApp Files backend and storage class for AKS.

a. Create an Azure Service Principle.

The service principal is how Trident communicates with Azure to manipulate your Azure NetApp Files resources.

```
$az ad sp create-for-rbac --name ""
```

The output should look like the following example:

```
{
"appId": "xxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
"displayName": "netapptrident",
"name": "",
"password": "xxxxxxxxxxxxxx.xxxxxxxxxxxxx",
"tenant": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx"
}
```

7. Create a Trident backend json file, example name `anf-backend.json`.

8. Using your preferred text editor, complete the following fields inside the `anf-backend.json` file:

```
{
    "version": 1,
    "storageDriverName": "azure-netapp-files",
    "subscriptionID": "fakec765-4774-fake-ae98-a721add4fake",
    "tenantID": "fakef836-edc1-fake-bff9-b2d865eefake",
    "clientID": "fake0f63-bf8e-fake-8076-8de91e57fake",
    "clientSecret": "SECRET",
    "location": "westeurope",
    "serviceLevel": "Standard",
    "virtualNetwork": "anf-vnet",
    "subnet": "default",
    "nfsMountOptions": "vers=3,proto=tcp",
    "limitVolumeSize": "500Gi",
    "defaults": {
    "exportRule": "0.0.0.0/0",
    "size": "200Gi"
}
```

9. Substitute the following fields:
   - `subscriptionID`. Your Azure subscription ID.
   - `tenantID`. Your Azure Tenant ID from the output of `az ad sp` in the previous step.
   - `clientID`. Your appID from the output of `az ad sp` in the previous step.
   - `clientSecret`. Your password from the output of `az ad sp` in the previous step.
10. Instruct Trident to create the Azure NetApp Files backend in the `trident` namespace using `anf-backend.json` as the configuration file:

```
$tridentctl create backend -f anf-backend.json -n trident
```

```
+------------------------+-------------------+--------------------------------------+--------+---------+
|          NAME          |   STORAGE DRIVER  |                 UUID                 | STATE  | VOLUMES |
+------------------------+-------------------+--------------------------------------+--------+---------+
| azurenetappfiles_86181 | azure-netapp-files | 2ca85462-59ac-4946-be05-c03f5575a2ad | online |       0 |
+------------------------+-------------------+--------------------------------------+--------+---------+
```

11. Create a storage class. Kubernetes users provision volumes by using PVCs that specify a storage class by name. Instruct K8s to create a storage class `azurenetappfiles` that references the Trident backend created in the previous step.
12. Create a YAML (`anf-storage-class.yaml`) file for storage class and copy.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
name: azurenetappfiles
provisioner: netapp.io/trident
parameters:
backendType: "azure-netapp-files"
$kubectl create -f anf-storage-class.yaml
```

13. Verify that the storage class was created.

```
kubectl get sc azurenetappfiles
```

```
NAME              PROVISIONER             RECLAIMPOLICY   VOLUMEBINDINGMODE   ALLOWVOLUMEEXPANSION   AGE
azurenetappfiles  csi.trident.netapp.io   Delete          Immediate           false                  98s
```

Next: Set up Dask with RAPIDS deployment on AKS using Helm.

Set up Dask with RAPIDS deployment on AKS using Helm

Previous: Install Trident.

To set up Dask with RAPIDS deployment on AKS using Helm, complete the following steps:

1. Create a namespace for installing Dask with RAPIDS.

```
kubectl create namespace rapids-dask
```

2. Create a PVC to store the click-through rate dataset:
   a. Save the following YAML content to a file to create a PVC.

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-criteo-data
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 1000Gi
  storageClassName: azurenetappfiles
```

b. Apply the YAML file to your Kubernetes cluster.

```
kubectl -n rapids-dask apply -f <your yaml file>
```

3. Clone the `rapidsai git` repository (https://github.com/rapidsai/helm-chart).

```
git clone https://github.com/rapidsai/helm-chart helm-chart
```

4. Modify `values.yaml` and include the PVC created earlier for workers and Jupyter workspace.

a. Go to the `rapidsai` directory of the repository.

```
cd helm-chart/rapidsai
```

b. Update the `values.yaml` file and mount the volume using PVC.

```
dask:
  …
  worker:
    name: worker
    …
    mounts:
      volumes:
        - name: data
          persistentVolumeClaim:
            claimName: pvc-criteo-data
      volumeMounts:
        - name: data
          mountPath: /data
    …
  jupyter:
    name: jupyter
    …
    mounts:
      volumes:
        - name: data
          persistentVolumeClaim:
            claimName: pvc-criteo-data
      volumeMounts:
        - name: data
          mountPath: /data
    …
```

5. Go to the repository's home directory and deploy Dask with three worker nodes on AKS using Helm.

```
cd ..
helm dep update rapidsai
helm install rapids-dask --namespace rapids-dask rapidsai
```

Next: Azure NetApp Files performance tiers.

Azure NetApp Files performance tiers

Previous: Set up Dask with RAPIDS deployment on AKS using Helm.

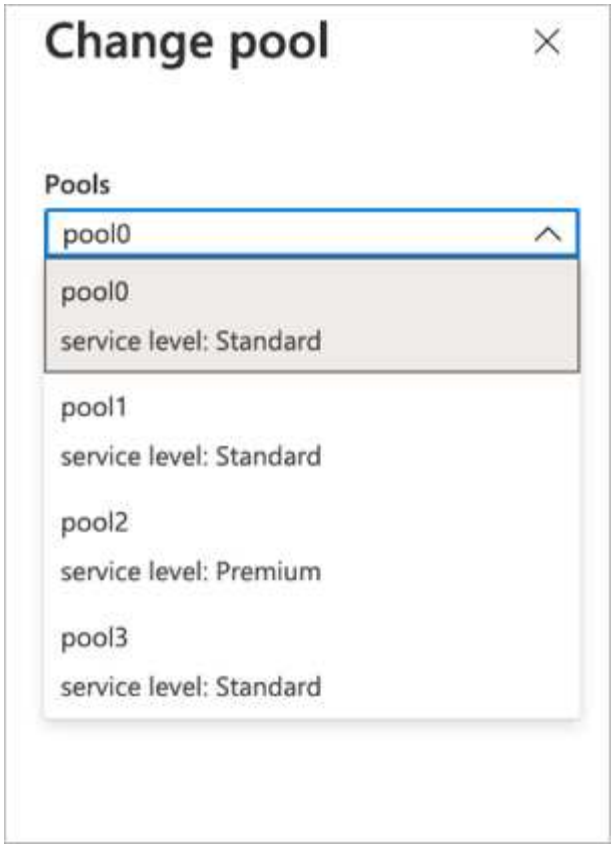
You can change the service level of an existing volume by moving the volume to another capacity pool that uses the service level you want for the volume. This solution enables customers to start with a small dataset and small number of GPUs in Standard Tier and scale out or scale up to Premium Tier as the amount of data and GPUs increase. The Premium Tier offers four times the throughput per terabyte as the Standard Tier, and scale up is performed without having to move any data to change the service level of a volume.

Dynamically change the service level of a volume

To dynamically change the service level of a volume, complete the following steps:

1. On the Volumes page, right-click the volume whose service level you want to change. Select Change Pool.

2. In the Change Pool window, select the capacity pool to which you want to move the volume.



3. Click OK.

Automate performance tier change

The following options are available to automate performance tier changes:

- Dynamic Service Level change is still in Public Preview at this time and not enabled by default. To enable this feature on the Azure Subscription, see this documentation about how to Dynamically change the service level of a volume.
- Azure CLI volume pool change commands are provided in volume pool change documentation and in the following example:

```
az netappfiles volume pool-change -g mygroup --account-name myaccname
--pool-name mypoolname --name myvolname --new-pool-resource-id
mynewresourceid
```

- PowerShell: The Set-AzNetAppFilesVolumePool cmdlet changes the pool of an Azure NetApp Files volume and is shown in the following example:

```
Set-AzNetAppFilesVolumePool
-ResourceGroupName "MyRG"
-AccountName "MyAnfAccount"
-PoolName "MyAnfPool"
-Name "MyAnfVolume"
-NewPoolResourceId 7d6e4069-6c78-6c61-7bf6-c60968e45fbf
```

Next: Libraries for data processing and model training.

Click through rate prediction data processing and model training

Libraries for data processing and model training

Previous: Azure NetApp Files performance tiers.

The following table lists the libraries and frameworks that were used to build this task. All these components have been fully integrated with Azure's role-based access and security controls.

Libraries/framework	Description
Dask cuML	For ML to work on GPU, the cuML library provides access to the RAPIDS cuML package with Dask. RAPIDS cuML implements popular ML algorithms, including clustering, dimensionality reduction, and regression approaches, with high-performance GPU-based implementations, offering speed-ups of up to 100x over CPU-based approaches.
Dask cuDF	cuDF includes various other functions supporting GPU-accelerated extract, transform, load (ETL), such as data subsetting, transformations, one-hot encoding, and more. The RAPIDS team maintains a dask-cudf library that includes helper methods to use Dask and cuDF.
Scikit Learn	Scikit-learn provides dozens of built-in machine learning algorithms and models, called estimators. Each estimator can be fitted to some data using its fit method.

We used two notebooks to construct the ML pipelines for comparison; one is the conventional Pandas scikit-learn approach, and the other is distributed training with RAPIDS and Dask. Each notebook can be tested individually to see the performance in terms of time and scale. We cover each notebook individually to demonstrate the benefits of distributed training using RAPIDS and Dask.

Next: Load Criteo Click Logs day 15 in Pandas and train a scikit-learn random forest model.

Load Criteo Click Logs day 15 in Pandas and train a scikit-learn random forest model

Previous: Libraries for data processing and model training.

This section describes how we used Pandas and Dask DataFrames to load Click Logs data from the Criteo Terabyte dataset. The use case is relevant in digital advertising for ad exchanges to build users' profiles by predicting whether ads will be clicked or if the exchange isn't using an accurate model in an automated pipeline.

We loaded day 15 data from the Click Logs dataset, totaling 45GB. Running the following cell in Jupyter notebook `CTR-PandasRF-collated.ipynb` creates a Pandas DataFrame that contains the first 50 million rows and generates a scikit-learn random forest model.

```
%%time
import pandas as pd
import numpy as np
header = ['col'+str(i) for i in range (1,41)] #note that according to
criteo, the first column in the dataset is Click Through (CT). Consist of
40 columns
first_row_taken = 50_000_000 # use this in pd.read_csv() if your compute
resource is limited.
# total number of rows in day15 is 20B
# take 50M rows
"""
Read data & display the following metrics:
1. Total number of rows per day
2. df loading time in the cluster
3. Train a random forest model
"""
df = pd.read_csv(file, nrows=first_row_taken, delimiter='\t',
names=header)
# take numerical columns
df_sliced = df.iloc[:, 0:14]
# split data into training and Y
Y = df_sliced.pop('col1') # first column is binary (click or not)
# change df_sliced data types & fillna
df_sliced = df_sliced.astype(np.float32).fillna(0)
from sklearn.ensemble import RandomForestClassifier
# Random Forest building parameters
# n_streams = 8 # optimization
max_depth = 10
n_bins = 16
n_trees = 10
rf_model = RandomForestClassifier(max_depth=max_depth,
n_estimators=n_trees)
rf_model.fit(df_sliced, Y)
```

To perform prediction by using a trained random forest model, run the following paragraph in this notebook. We took the last one million rows from day 15 as the test set to avoid any duplication. The cell also calculates accuracy of prediction, defined as the percentage of occurrences the model accurately predicts whether a user clicks an ad or not. To review any unfamiliar components in this notebook, see the official scikit-learn documentation.

```
# testing data, last 1M rows in day15
test_file = '/data/day_15_test'
with open(test_file) as g:
    print(g.readline())

# dataFrame processing for test data
test_df = pd.read_csv(test_file, delimiter='\t', names=header)
test_df_sliced = test_df.iloc[:, 0:14]
test_Y = test_df_sliced.pop('col1')
test_df_sliced = test_df_sliced.astype(np.float32).fillna(0)
# prediction & calculating error
pred_df = rf_model.predict(test_df_sliced)
from sklearn import metrics
# Model Accuracy
print("Accuracy:",metrics.accuracy_score(test_Y, pred_df))
```

Next: Load Day 15 in Dask and train a Dask cuML random forest model.

Load Day 15 in Dask and train a Dask cuML random forest model

Previous: Load Criteo Click Logs day 15 in Pandas and train a scikit-learn random forest model.

In a manner similar to the previous section, load Criteo Click Logs day 15 in Pandas and train a scikit-learn random forest model. In this example, we performed DataFrame loading with Dask cuDF and trained a random forest model in Dask cuML. We compared the differences in training time and scale in the section "Training time comparison."

criteo_dask_RF.ipynb

This notebook imports `numpy`, `cuml`, and the necessary `dask` libraries, as shown in the following example:

```
import cuml
from dask.distributed import Client, progress, wait
import dask_cudf
import numpy as np
import cudf
from cuml.dask.ensemble import RandomForestClassifier as cumlDaskRF
from cuml.dask.common import utils as dask_utils
```

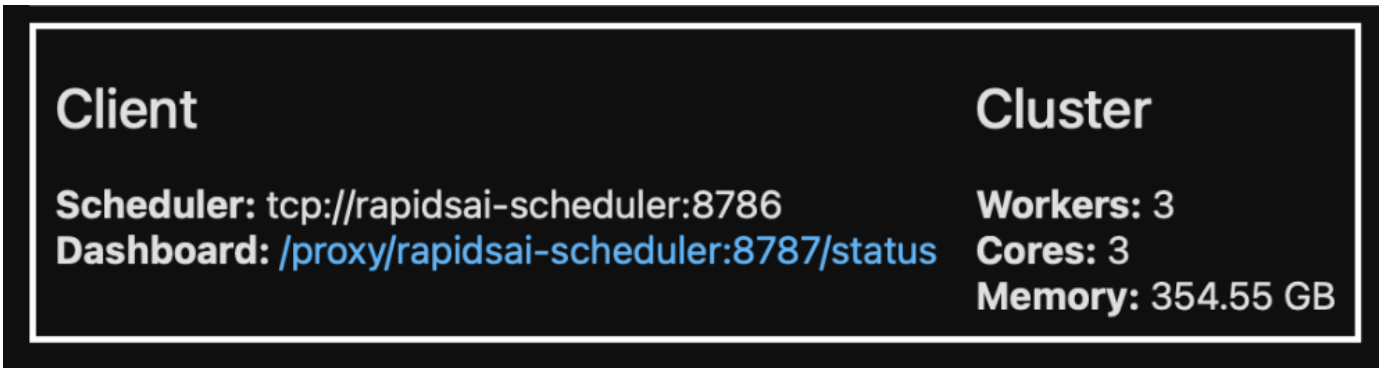
Initiate Dask Client().

```
client = Client()
```

If your cluster is configured correctly, you can see the status of worker nodes.

```
client
workers = client.has_what().keys()
n_workers = len(workers)
n_streams = 8 # Performance optimization
```

In our AKS cluster, the following status is displayed:



Note that Dask employs the lazy execution paradigm: rather than executing the processing code instantly, Dask builds a Directed Acyclic Graph (DAG) of execution instead. DAG contains a set of tasks and their interactions that each worker needs to run. This layout means the tasks do not run until the user tells Dask to execute them in one way or another. With Dask you have three main options:

- **Call compute() on a DataFrame.** This call processes all the partitions and then returns results to the scheduler for final aggregation and conversion to cuDF DataFrame. This option should be used sparingly and only on heavily reduced results unless your scheduler node runs out of memory.
- **Call persist() on a DataFrame.** This call executes the graph, but, instead of returning the results to the scheduler node, it maintains them across the cluster in memory so the user can reuse these intermediate results down the pipeline without the need for rerunning the same processing.
- **Call head() on a DataFrame.** Just like with cuDF, this call returns 10 records back to the scheduler node. This option can be used to quickly check if your DataFrame contains the desired output format, or if the records themselves make sense, depending on your processing and calculation.

Therefore, unless the user calls either of these actions, the workers sit idle waiting for the scheduler to initiate the processing. This lazy execution paradigm is common in modern parallel and distributed computing frameworks such as Apache Spark.

The following paragraph trains a random forest model by using Dask cuML for distributed GPU-accelerated computing and calculates model prediction accuracy.

```
Adsf
# Random Forest building parameters
n_streams = 8 # optimization
max_depth = 10
n_bins = 16
n_trees = 10
cuml_model = cumlDaskRF(max_depth=max_depth, n_estimators=n_trees,
n_bins=n_bins, n_streams=n_streams, verbose=True, client=client)
cuml_model.fit(gdf_sliced_small, Y)
# Model prediction
pred_df = cuml_model.predict(gdf_test)
# calculate accuracy
cu_score = cuml.metrics.accuracy_score( test_y, pred_df )
```

Next: Monitor Dask using native Task Streams dashboard.

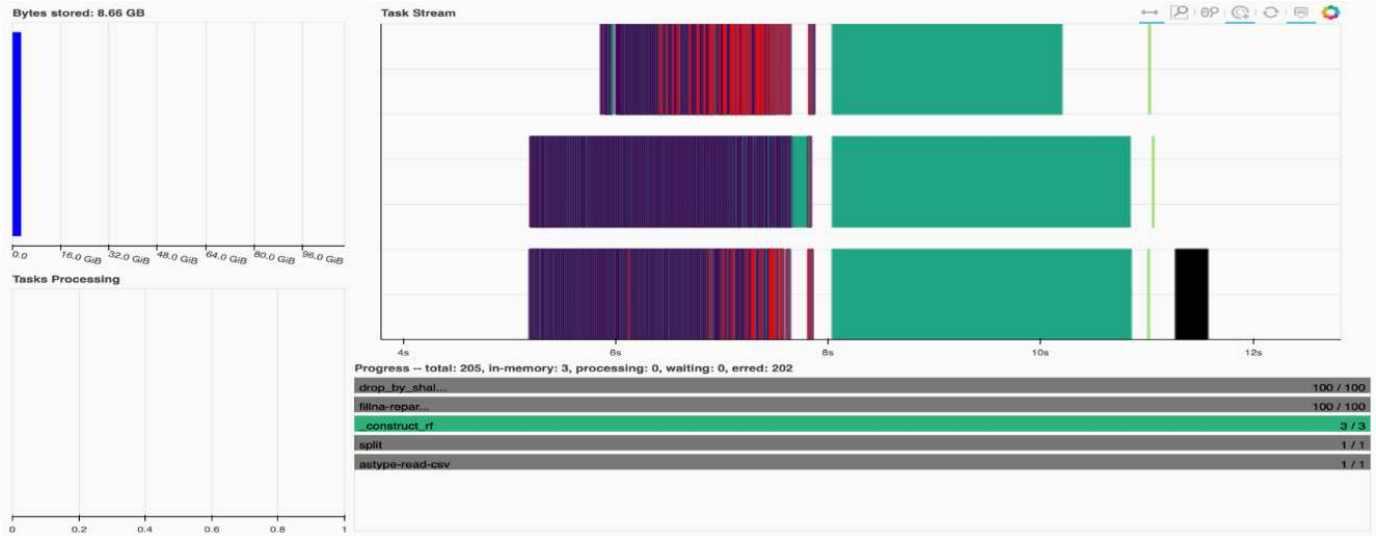
Monitor Dask using native Task Streams dashboard

Previous: Load Day 15 in Dask and train a Dask cuML random forest model.

The Dask distributed scheduler provides live feedback in two forms:

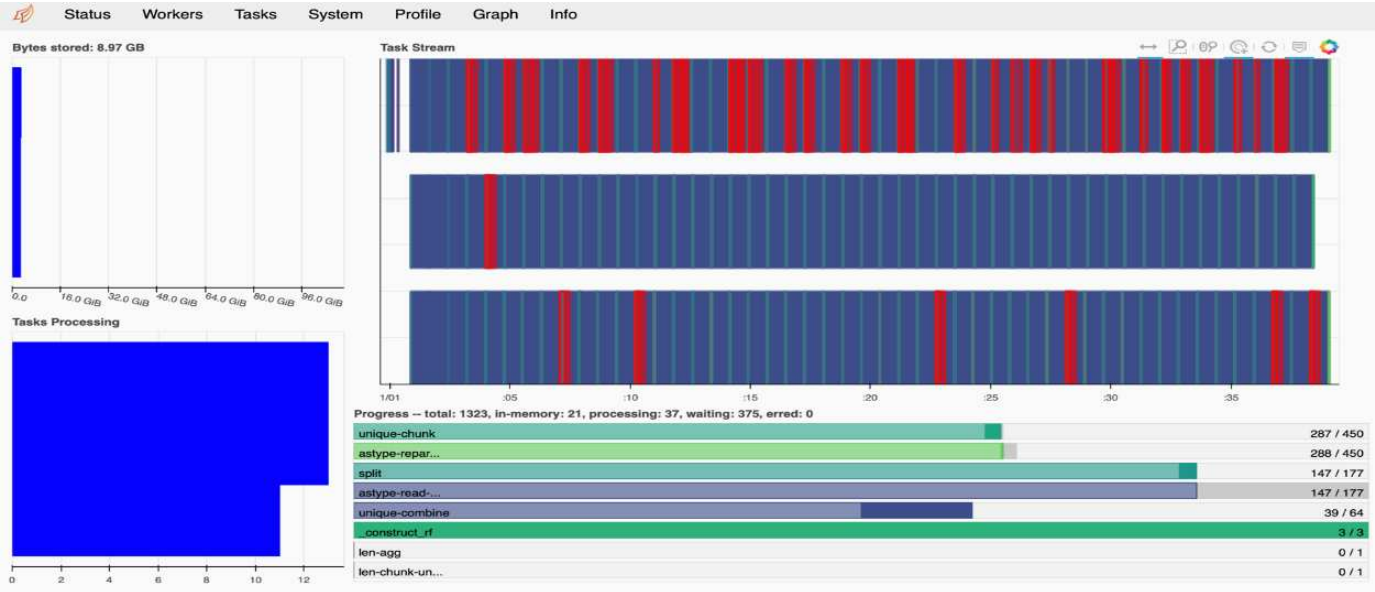
- An interactive dashboard containing many plots and tables with live information
- A progress bar suitable for interactive use in consoles or notebooks

In our case, the following figure shows how you can monitor the task progress, including Bytes Stored, the Task Stream with a detailed breakdown of the number of streams, and Progress by task names with associated functions executed. In our case, because we have three worker nodes, there are three main chunks of stream and the color codes denote different tasks within each stream.



You have the option to analyze individual tasks and examine the execution time in milliseconds or identify any obstacles or hindrances. For example, the following figure shows the Task Streams for the random forest model fitting stage. There are considerably more functions being executed, including unique chunk for

DataFrame processing, _construct_rf for fitting the random forest, and so on. Most of the time was spent on DataFrame operations due to the large size (45GB) of one day of data from the Criteo Click Logs.



Next: Training time comparison.

Training time comparison

Previous: Monitor Dask using native Task Streams dashboard.

This section compares the model training time using conventional Pandas compared to Dask. For Pandas, we loaded a smaller amount of data due to the nature of slower processing time to avoid memory overflow. Therefore, we interpolated the results to offer a fair comparison.

The following table shows the raw training time comparison when there is significantly less data used for the Pandas random forest model (50 million rows out of 20 billion per day15 of the dataset). This sample is only using less than 0.25% of all available data. Whereas for Dask-cuML we trained the random forest model on all 20 billion available rows. The two approaches yielded comparable training time.

Approach	Training time
Scikit-learn: Using only 50M rows in day15 as the training data	47 minutes and 21 seconds
RAPIDS-Dask: Using all 20B rows in day15 as the training data	1 hour, 12 minutes, and 11 seconds

If we interpolate the training time results linearly, as shown in the following table, there is a significant advantage to using distributed training with Dask. It would take the conventional Pandas scikit-learn approach 13 days to process and train 45GB of data for a single day of click logs, whereas the RAPIDS-Dask approach processes the same amount of data 262.39 times faster.

Approach	Training time
Scikit-learn: Using all 20B rows in day15 as the training data	13 days, 3 hours, 40 minutes, and 11 seconds

Approach	Training time
RAPIDS-Dask: Using all 20B rows in day15 as the training data	1 hour, 12 minutes, and 11 seconds

In the previous table, you can see that by using RAPIDS with Dask to distribute the data processing and model training across multiple GPU instances, the run time is significantly shorter compared to conventional Pandas DataFrame processing with scikit-learn model training. This framework enables scaling up and out in the cloud as well as on-premises in a multinode, multi-GPU cluster.

Next: Monitor Dask and RAPIDS with Prometheus and Grafana.

Monitor Dask and RAPIDS with Prometheus and Grafana

Previous: Training time comparison.

After everything is deployed, run inferences on new data. The models predict whether a user clicks an ad based on browsing activities. The results of the prediction are stored in a Dask cuDF. You can monitor the results with Prometheus and visualize in Grafana dashboards.

For more information, see this RAPIDS AI Medium post.

Next: Dataset and Model Versioning using NetApp DataOps Toolkit.

Dataset and model versioning using NetApp DataOps Toolkit

Previous: Monitor Dask and RAPIDS with Prometheus and Grafana.

The NetApp DataOps Toolkit for Kubernetes abstracts storage resources and Kubernetes workloads up to the data-science workspace level. These capabilities are packaged in a simple, easy-to-use interface that is designed for data scientists and data engineers. Using the familiar form of a Python program, the Toolkit enables data scientists and engineers to provision and destroy JupyterLab workspaces in just seconds. These workspaces can contain terabytes, or even petabytes, of storage capacity, enabling data scientists to store all their training datasets directly in their project workspaces. Gone are the days of separately managing workspaces and data volumes.

For more information, visit the Toolkit's GitHub repository.

Next: Conclusion.

Jupyter notebooks for reference

Previous: Dataset and Model Versioning using NetApp DataOps Toolkit.

There are two Jupyter notebooks associated with this technical report:

- **CTR-PandasRF-collated.ipynb.** This notebook loads Day 15 from the Criteo Terabyte Click Logs dataset, processes and formats data into a Pandas DataFrame, trains a Scikit-learn random forest model, performs prediction, and calculates accuracy.
- **criteo_dask_RF.ipynb.** This notebook loads Day 15 from the Criteo Terabyte Click Logs dataset, processes and formats data into a Dask cuDF, trains a Dask cuML random forest model, performs prediction, and calculates accuracy. By leveraging multiple worker nodes with GPUs, this distributed data and model processing and training approach is highly efficient. The more data you process, the greater the time savings versus a conventional ML approach. You can deploy this notebook in the cloud, on-premises,

or in a hybrid environment where your Kubernetes cluster contains compute and storage in different locations, as long as your networking setup enables the free movement of data and model distribution.

Next: Conclusion.

Conclusion

Previous: Dataset and Model Versioning using NetApp DataOps Toolkit.

Azure NetApp Files, RAPIDS, and Dask speed up and simplify the deployment of large-scale ML processing and training by integrating with orchestration tools such as Docker and Kubernetes. By unifying the end-to-end data pipeline, this solution reduces the latency and complexity inherent in many advanced computing workloads, effectively bridging the gap between development and operations. Data scientists can run queries on large datasets and securely share data and algorithmic models with other users during the training phase.

When building your own AI/ML pipelines, configuring the integration, management, security, and accessibility of the components in an architecture is a challenging task. Giving developers access and control of their environment presents another set of challenges.

By building an end-to-end distributed training model and data pipeline in the cloud, we demonstrated two orders of magnitude improvement in total workflow completion time versus a conventional, open-source approach that did not leverage GPU-accelerated data processing and compute frameworks.

The combination of NetApp, Microsoft, opens-source orchestration frameworks, and NVIDIA brings the latest technologies together as managed services with great flexibility to accelerate technology adoption and improve the time to market for new AI/ML applications. These advanced services are delivered in a cloud-native environment that can be easily ported for on-premises as well as hybrid deployment architectures.

Next: Where to find additional information.

Where to find additional information

Previous: Conclusion.

To learn more about the information that is described in this document, see the following resources:

- Azure NetApp Files:
  - Solutions architecture page for Azure NetApp Files

    https://docs.microsoft.com/azure/azure-netapp-files/azure-netapp-files-solution-architectures
- Trident persistent storage for containers:
  - Azure NetApp Files and Trident

    https://netapptrident.readthedocs.io/en/stablev20.07/kubernetes/operations/tasks/backends/anf.html
- Dask and RAPIDS:
  - Dask

    https://docs.dask.org/en/latest/
  - Install Dask

https://docs.dask.org/en/latest/install.html

- Dask API

  https://docs.dask.org/en/latest/api.html

- Dask Machine Learning

  https://examples.dask.org/machine-learning.html

- Dask Distributed Diagnostics

  https://docs.dask.org/en/latest/diagnostics-distributed.html

- ML framework and tools:
  - TensorFlow: An Open-Source Machine Learning Framework for Everyone

    https://www.tensorflow.org/

  - Docker

    https://docs.docker.com

  - Kubernetes

    https://kubernetes.io/docs/home/

  - Kubeflow

    http://www.kubeflow.org/

  - Jupyter Notebook Server

    http://www.jupyter.org/

Next: Version history.

Version history

Previous: Where to find additional information.

Version	Date	Document version history
Version 1.0	August 2021	Initial release.

Copyright information

Trademark information