Click through rate prediction data processing and model training

NetApp Solutions

NetApp
January 18, 2023

Table of Contents

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
```

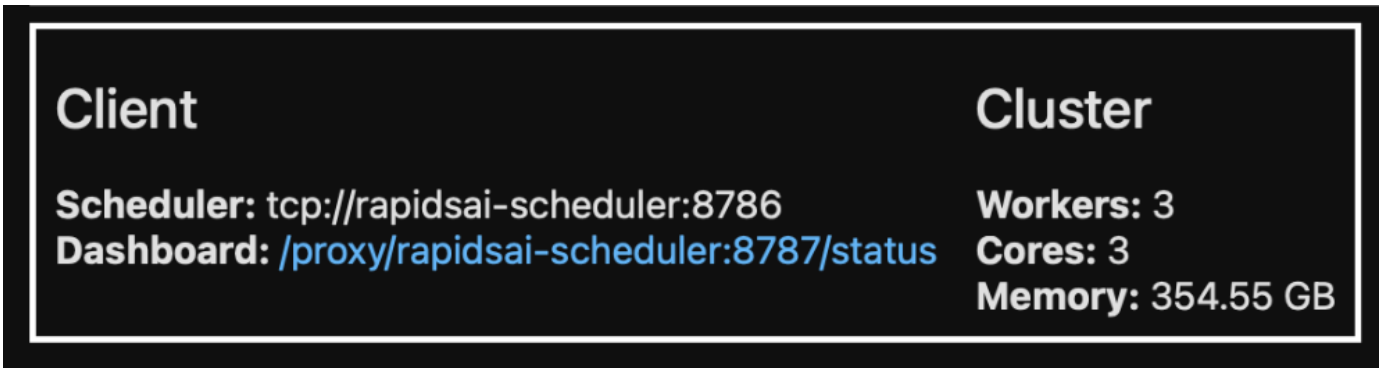
Initiate Dask Client().

```
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.

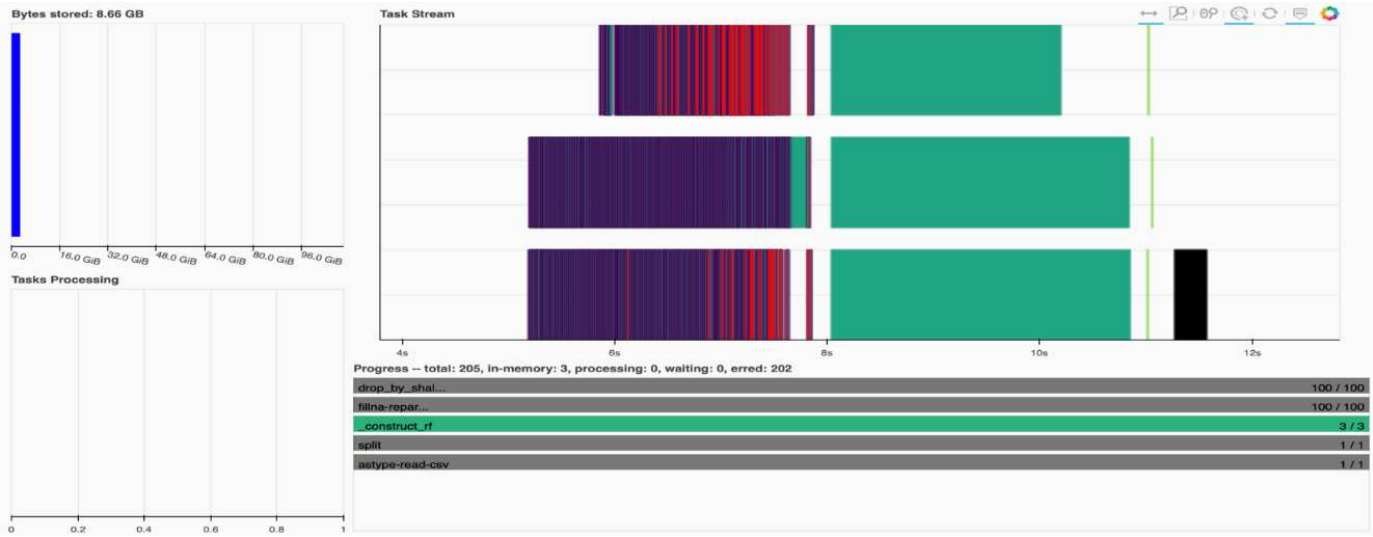
Monitor Dask using native Task Streams dashboard

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

The Dask distributed scheduler provides live feedback in two forms:

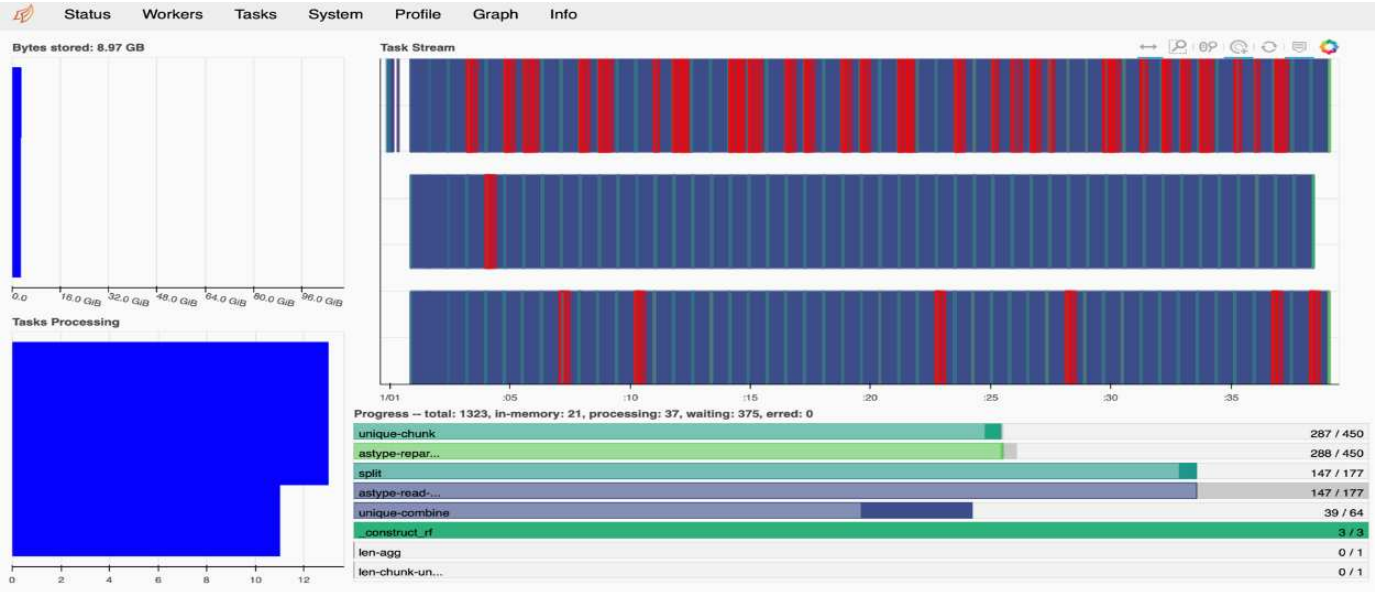
- An interactive dashboard containing many plots and tables with live information
- A progress bar suitable for interactive use in consoles or notebooks

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.

Copyright information

Trademark information