Fare clic per valutare l'elaborazione dei dati di previsione e modellare la formazione

NetApp Solutions

NetApp
April 26, 2024

Sommario

Fare clic per valutare l'elaborazione dei dati di previsione e modellare la formazione

Librerie per l'elaborazione dei dati e la formazione sui modelli

La tabella seguente elenca le librerie e i framework utilizzati per creare questa attività. Tutti questi componenti sono stati completamente integrati con i controlli di sicurezza e accesso basati sui ruoli di Azure.

Librerie/framework	Descrizione
CuML di Dask	Per CONSENTIRE A ML di lavorare su GPU, il "Libreria cuML" Fornisce l'accesso al pacchetto RAPIDS cuML con DAK. RAPIDS cuML implementa i più diffusi algoritmi ML, tra cui clustering, riduzione delle dimensioni e approcci di regressione, con implementazioni basate su GPU ad alte performance, che offrono velocità fino a 100 volte superiori rispetto agli approcci basati su CPU.
Dask cuDF	CuDF include diverse altre funzioni che supportano l'estrazione, la trasformazione, il carico (ETL) con accelerazione GPU, come il sottosetting dei dati, le trasformazioni, la codifica one-hot e molto altro ancora. Il team RAPIDS gestisce un "libreria dask-cudf" Sono inclusi i metodi di supporto per l'utilizzo di Dask e cuDF.
Scikit Impara	Scikit-Learn offre decine di algoritmi e modelli di apprendimento automatico integrati, chiamati stimatori. Ciascuno "stimatore" può essere adattato ad alcuni dati utilizzando its "adatta" metodo.

Abbiamo utilizzato due notebook per costruire LE pipeline ML per il confronto; uno è l'approccio convenzionale Pandas scikit-Learn e l'altro è la formazione distribuita con RAPIDS e Dask. Ciascun notebook può essere testato singolarmente per verificarne le prestazioni in termini di tempo e scalabilità. Copriamo ogni notebook singolarmente per dimostrare i vantaggi della formazione distribuita utilizzando RAPIDS e Dask.

Load Criteo fare clic su Logs giorno 15 in Pandas e formare un modello di foresta casuale scikit-learn

In questa sezione viene descritto come abbiamo utilizzato Pandas e Dask DataFrame per caricare i dati dei registri Click dall'insieme di dati Criteo Terabyte. Il caso d'utilizzo è importante nella pubblicità digitale per gli scambi di annunci per creare i profili degli utenti prevedendo se gli annunci verranno cliccati o se lo scambio non utilizza un modello accurato in una pipeline automatica.

Abbiamo caricato i dati del giorno 15 dal set di dati Click Logs, per un totale di 45 GB. Eseguire la seguente

cella nel notebook Jupyter `CTR-PandasRF-collated.ipynb` Crea un Pandas DataFrame che contiene i primi 50 milioni di righe e genera un modello di foresta casuale scikit-Learn.

```
%%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)
```

Per eseguire la previsione utilizzando un modello di foresta casuale con formazione, eseguire il paragrafo seguente in questo notebook. Abbiamo preso gli ultimi un milione di righe dal giorno 15 come set di test per evitare qualsiasi duplicazione. La cella calcola anche la precisione della previsione, definita come la percentuale di occorrenze che il modello prevede accuratamente se un utente fa clic su un annuncio o meno. Per esaminare eventuali componenti non familiari presenti in questo notebook, consultare la sezione "documentazione ufficiale scikit-learn".

```
# 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))
```

Caricare il giorno 15 in Dask e formare un modello di foresta casuale di Dask cuML

In modo simile alla sezione precedente, caricare Criteo Click Logs Day 15 in Pandas e formare un modello di foresta casuale scikit-learn. In questo esempio, è stato eseguito il caricamento di DataFrame con Dask cuDF e il training di un modello di foresta casuale in Dask cuML. Nella sezione abbiamo confrontato le differenze di tempo e di scala per la formazione ""Confronto tra i tempi di formazione"."

criteo_dask_RF.ipynb

Questo notebook importa `numpy`, `cuml`e il necessario `dask` librerie, come mostrato nell'esempio seguente:

```
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
```

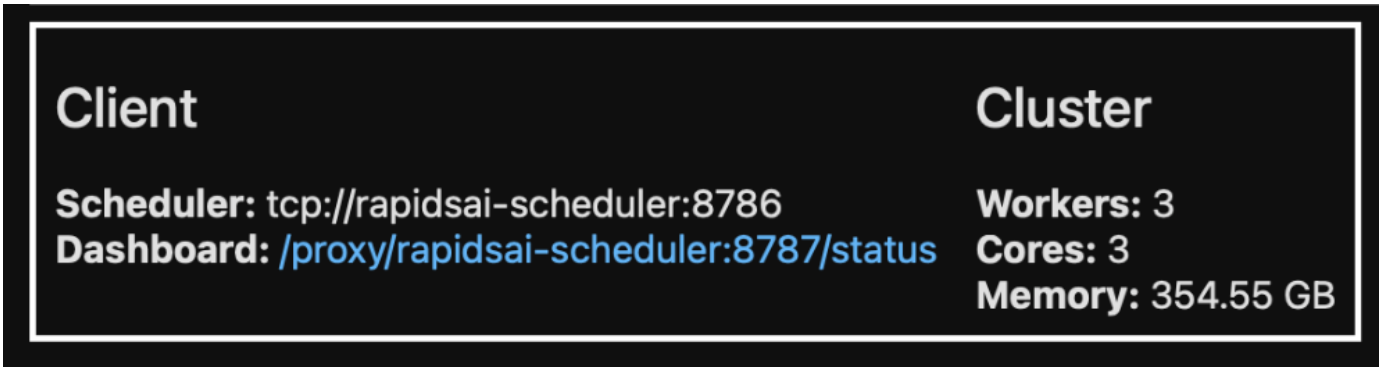
Avviare Dask Client().

```
client = Client()
```

Se il cluster è configurato correttamente, è possibile visualizzare lo stato dei nodi di lavoro.

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

Nel nostro cluster AKS viene visualizzato il seguente stato:



Si noti che Dask utilizza il paradigma di esecuzione pigro: Invece di eseguire il codice di elaborazione istantaneamente, Dask crea invece un DAG (Directed Acyclic Graph) di esecuzione. IL DAG contiene una serie di attività e le relative interazioni che ciascun lavoratore deve eseguire. Questo layout significa che i task non vengono eseguiti finché l'utente non dice a Task di eseguirli in un modo o nell'altro. Con Dask hai tre opzioni principali:

- **Call compute() su un DataFrame.** questa chiamata elabora tutte le partizioni e restituisce i risultati allo scheduler per l'aggregazione finale e la conversione in cuDF DataFrame. Questa opzione deve essere utilizzata con parsimonia e solo in caso di risultati fortemente ridotti, a meno che il nodo dello scheduler non esaurisca la memoria.
- **Call Persistent() su un DataFrame.** questa chiamata esegue il grafico, ma, invece di restituire i risultati al nodo scheduler, li mantiene in memoria nel cluster in modo che l'utente possa riutilizzare questi risultati intermedi lungo la pipeline senza dover eseguire nuovamente la stessa elaborazione.
- **Call head() su un DataFrame.** proprio come con cuDF, questa chiamata restituisce 10 record al nodo Scheduler. Questa opzione consente di verificare rapidamente se il DataFrame contiene il formato di output desiderato o se i record stessi hanno senso, a seconda dell'elaborazione e del calcolo.

Pertanto, a meno che l'utente non chiami una di queste azioni, i lavoratori sono inattivi in attesa che lo scheduler avvii l'elaborazione. Questo paradigma di esecuzione pigro è comune nei moderni framework di calcolo distribuiti e paralleli come Apache Spark.

Il paragrafo seguente forma un modello di foresta casuale utilizzando Dask cuML per il calcolo distribuito con accelerazione GPU e calcola la precisione di previsione del modello.

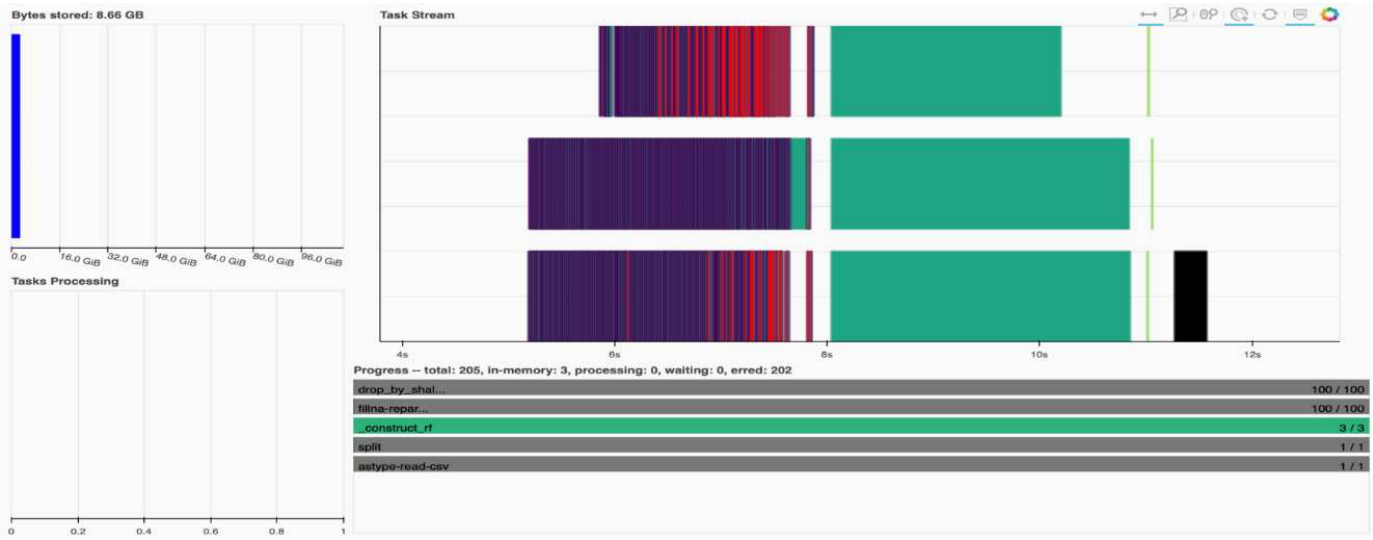
```
Adsf
# Random Forest building parameters
n_streams = 8 # optimization
max_depth = 10
n_bins = 16
n_trees = 10
cuml_model = cumlDaskRF(max_depth=max_depth, n_estimators=n_trees,
n_bins=n_bins, n_streams=n_streams, verbose=True, client=client)
cuml_model.fit(gdf_sliced_small, Y)
# Model prediction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

Monitorate la Task utilizzando la dashboard nativa dei Task Streams

Il "Scheduler distribuito di Dask" fornisce feedback live in due forme:

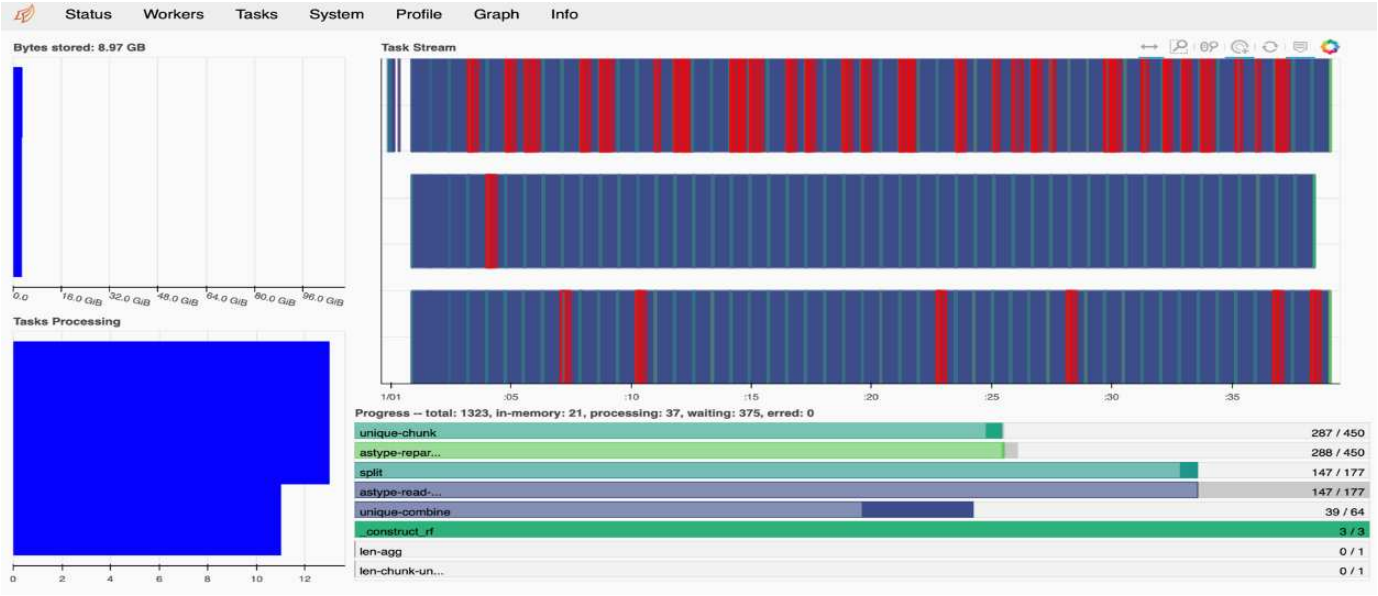
- Una dashboard interattiva contenente numerosi grafici e tabelle con informazioni in tempo reale
- Barra di avanzamento adatta per l'utilizzo interattivo in console o notebook

Nel nostro caso, la figura seguente mostra come è possibile monitorare l'avanzamento del task, inclusi i byte memorizzati, il Task Stream con una dettagliata suddivisione del numero di flussi e l'avanzamento in base ai nomi delle attività con le funzioni associate eseguite. Nel nostro caso, poiché abbiamo tre nodi di lavoro, ci sono tre blocchi principali di flusso e i codici colore indicano attività diverse all'interno di ogni flusso.



È possibile analizzare le singole attività ed esaminare il tempo di esecuzione in millisecondi o identificare eventuali ostacoli o ostacoli. Ad esempio, la figura seguente mostra i flussi di attività per la fase di adattamento del modello di foresta casuale. Le funzioni eseguite sono notevolmente più numerose, tra cui il chunk unico per l'elaborazione di DataFrame, _Construct_rf per l'adattamento della foresta casuale e così via. La maggior parte del tempo è stato dedicato alle operazioni DataFrame a causa delle grandi dimensioni (45 GB) di un

giorno di dati provenienti dai Click Logs di Criteo.



Confronto dei tempi di training

In questa sezione viene confrontato il tempo di training del modello utilizzando i Panda convenzionali rispetto a quello di Dask. Per Pandas, abbiamo caricato una quantità inferiore di dati a causa della natura del tempo di elaborazione più lento per evitare l'overflow della memoria. Pertanto, abbiamo interpolato i risultati per offrire un confronto equo.

La tabella seguente mostra il confronto dei tempi di training raw quando i dati utilizzati per il modello di foresta casuale Pandas sono significativamente inferiori (50 milioni di righe su 20 miliardi al giorno 15 del set di dati). Questo esempio utilizza solo meno del 0.25% di tutti i dati disponibili. Mentre per Dask-cuML abbiamo addestrato il modello di foresta casuale su tutti i 20 miliardi di righe disponibili. I due approcci hanno consentito di ottenere tempi di formazione comparabili.

Approccio	Tempo di training
Scikit-Learn: Utilizzando solo 50M righe nel giorno 15 come dati di training	47 minuti e 21 secondi
RAPIDS-Dask: Utilizzo di tutte le 20B righe del giorno 15 come dati di training	1 ora, 12 minuti e 11 secondi

Se si interpolano i risultati dei tempi di training in modo lineare, come mostrato nella tabella seguente, si ha un vantaggio significativo nell'utilizzo della formazione distribuita con Dask. L'approccio convenzionale Pandas scikit-Learn richiede 13 giorni per elaborare e formare 45 GB di dati per un singolo giorno di log click, mentre L'approccio RAPIDS-Dask elabora la stessa quantità di dati 262.39 volte più velocemente.

Approccio	Tempo di training
Scikit-Learn: Utilizzando tutte le 20B righe del giorno 15 come dati di training	13 giorni, 3 ore, 40 minuti e 11 secondi

Approccio	Tempo di training
RAPIDS-Dask: Utilizzo di tutte le 20B righe del giorno 15 come dati di training	1 ora, 12 minuti e 11 secondi

Nella tabella precedente, è possibile osservare che, utilizzando RAPIDS con Dask per distribuire l'elaborazione dei dati e modellare la formazione su più istanze GPU, il tempo di esecuzione è significativamente più breve rispetto all'elaborazione convenzionale di Pandas DataFrame con il training del modello scikit-Learn. Questo framework consente la scalabilità verticale e orizzontale nel cloud e on-premise in un cluster multi-GPU a più nodi.

Monitoraggio di Dask e RAPIDE con Prometheus e Grafana

Una volta implementato tutto, esegui le inferenze sui nuovi dati. I modelli prevedono se un utente fa clic su un annuncio in base alle attività di navigazione. I risultati della previsione sono memorizzati in un cuDF di Dask. Puoi monitorare i risultati con Prometheus e visualizzarli nelle dashboard Grafana.

Per ulteriori informazioni, consulta questa sezione "RAPIDS ai Medium post".

Versione di set di dati e modelli con NetApp DataOps Toolkit

Il NetApp DataOps Toolkit per Kubernetes astratta le risorse di storage e i carichi di lavoro Kubernetes fino al livello di spazio di lavoro per la scienza dei dati. Queste funzionalità sono integrate in un'interfaccia semplice e facile da usare, progettata per data scientist e data engineer. Utilizzando la forma familiare di un programma Python, il Toolkit consente a data scientist e ingegneri di eseguire il provisioning e la distruzione delle aree di lavoro di JupyterLab in pochi secondi. Queste aree di lavoro possono contenere terabyte, o persino petabyte, di capacità di storage, consentendo agli scienziati dei dati di memorizzare tutti i set di dati di training direttamente nelle aree di lavoro dei progetti. Sono finiti i tempi della gestione separata degli spazi di lavoro e dei volumi di dati.

Per ulteriori informazioni, visitare il Toolkit "Repository di GitHub".

Notebook Jupyter come riferimento

Al report tecnico sono associati due notebook Jupyter:

- **"CTR-PandasRF-collated.ipynb."** Questo notebook carica il giorno 15 dal set di dati Click Logs di Criteo Terabyte, elabora e formatta i dati in un Pandas DataFrame, forma un modello di foresta casuale Scikit-learn, esegue la previsione e calcola la precisione.
- **"criteo_dask_RF.ipynb."** Questo notebook carica il giorno 15 dal set di dati Click Logs di Criteo Terabyte, elabora e formatta i dati in un cuDF Dask, forma un modello di foresta casuale cuML Dask, esegue la previsione e calcola la precisione. Sfruttando nodi di lavoro multipli con GPU, questo approccio di elaborazione e formazione dei dati distribuiti e dei modelli è altamente efficiente. Maggiore è il numero di dati elaborati, maggiore è il risparmio di tempo rispetto a un approccio ML convenzionale. È possibile implementare questo notebook nel cloud, on-premise o in un ambiente ibrido in cui il cluster Kubernetes

contiene calcolo e storage in posizioni diverse, purché la configurazione di rete consenta il libero spostamento dei dati e la distribuzione dei modelli.