Dask

Sadržaj

Uvod

Kao što je navedeno u službenoj dokumentaciji, Dask je "fleksibilna knjižnica namijenjena paralelizaciji proračuna u Pythonu". Osim što je usmjerena razvoju paralelnog koda, glavna uloga je omogućiti lagano skaliranje tipičnih data science problema i aplikacija na klaster, koje se tipično razvijaju na osobnim računalima. Ovo postiže kroz kroz imitaciju poznatijih API-ja usmjerenih obradi podataka (poput Numpya i Pandasa), oslanjanje na integriran raspoređivač poslova i činjenicu da je u potpunosti napisana u Pythonu.

Glavna su joj sučelja:

- Array - obrada tenzora s Numpy API-jem
- DataFrame - obrada strukturiranih podataka s Pandas API-jem
- Bag - obrada lista i iteratora namijenjeno tekstualnim podacima, JSON datotekama ili Python objektima
- Delayed - direktna paralelizacija koda funkcijama ili dekoratorima
- Futures - paralelizacija namijenjena izvršavanju poslova koji se mogu vršiti istodobno

Jedna od srodnih knjižnica je i Dask-ML, koja je namijenjena distribuiranom strojnom učenju putem poznatog scikit API-ja i koja omogućava skaliranje na više čvorova putem knjižnice joblib, s kojom scikit paralelizira svoje algoritme. Više o tipičnim problemima koji se rješavaju i primjerima korištenja svakog od sučelja možete naći na online stranicama Daska.

Dostupne verzije

Verzija	Modul
2022.11.1	dask/2022.11.1

Korištenje

Za širenje na Isabelli putem SGE-a, potrebno je koristiti Dask-MPI knjižnicu kojom se stvara Dask klaster i putem kojeg se distribuiraju poslovi korištenjem Client API-ja. Dva su načina na koji se ovo može postignuti:

1. dask-mpi: Kreiranjem dask klastera u SGE skripti prije zvanja python programa
2. initialize: Inicijalizacijom dask klastera unutar python programa

U prvom slučaju (nakon zvanja dask modula u SGE skripti) potrebno je pozvati dask-mpi naredbu prije izvršavanja

python programa, dok se u python skripti treba inicijalizirati klijent s kreiranom scheduler.json datotekom:

dask-mpi SGE primjer

```
...

# aktiviraj dask
module load dask

# pokreni dask klaster putem dask-mpi
mpirun -np $NSLOTS dask-mpi \
    --nthreads 1 \
    --interface ib0 \
    --worker-class distributed.Worker \
    --scheduler-file scheduler.json &

# pokreni python program
python moj_program.py
```

dask-mpi python primjer

```
# pozovi python modul
from dask.distributed import Client

# spoji klijenta
client = Client(scheduler_file='scheduler.json')

# ostatak programa
...
```

U drugom slučaju, dask klaster se inicijalizira unutar python skripte dok se u SGE poziva putem naredbe mpirun:

Pri inicijalizaciji u python skripti putem **dask_mpi.initialize** modula, nužno je izvršiti inicijalizaciju prije pozivanja **dask.distributed.Client** modula kao što je navedeno ispod

initialize SGE primjer

```
...

# aktiviraj dask
module load dask

# pokreni python program
mpirun -np $NSLOTS python moj_program.py
```

initialize python primjer

```
# pozovi i inicijaliziraj klaster
from dask_mpi import initialize
initialize()

# pozovi i definiraj klijenta
from dask.distributed import Client
client = Client()

# izvrši program
...
```

Primjeri

Primjeri obrade tipičnog dataframea, korištenja algoritma K sredina ili izabira najboljeg ML modela podnošenjem

na *mpi paralelnu okolinu se nalaze ispod.

Dataframe

dataframe.sge

```
#$ -cwd
#$ -o output/
#$ -e output/
#$ -pe *mpi 4

# aktiviraj dask
module load dask

# pokreni dask klaster
mpirun -np $NSLOTS dask-mpi \
    --nthreads 1 \
    --interface ib0 \
    --worker-class distributed.Worker \
    --scheduler-file scheduler.json &

# priekaj
sleep 10

# potjeraj python skriptu
python example.py
```

dataframe.py

```
import time
import dask

from dask.distributed import Client

if __name__ == '__main__':

    # spoji klijenta putem datoteke scheduler.json
    client = Client(scheduler_file="scheduler.json")

    # kreiraj dataframe
    df = dask.datasets.timeseries(freq='10ms')

    # izraunaj
    now = time.time()
    computed_df = df.describe().compute()
    df.info(memory_usage=True)
    print('compute elapsed: %f' % (time.time()-now))
```

K-means

kmeans.sge

```
#$ -cwd
#$ -o output/
#$ -e output/
#$ -pe *mpi 4

# aktiviraj modul
module load dask

# pokreni dask klaster
mpirun -np $NSLOTS dask-mpi \
    --nthreads 1 \
    --worker-class distributed.Worker \
    --scheduler-file scheduler.json &

# priekaj
sleep 10

# potjeraj python skriptu
python run_kmeans.py
```

kmeans.py

```
# https://examples.dask.org/machine-learning/training-on-large-datasets.html

import time

from dask_mpi import initialize
from dask.distributed import Client

import dask_ml.datasets
import dask_ml.cluster

import matplotlib.pyplot as plt

if __name__ == '__main__':

    # spoji klijenta putem datoteke scheduler.json
    client = Client(scheduler_file="scheduler.json")

    # kreiraj podatke
    n_clusters = 10
    n_samples = 10**4
    n_chunks = int(os.environ['NSLOTS'])-2
    X, _ = dask_ml.datasets.make_blobs(
        centers = n_clusters,
        n_samples = n_samples,
        chunks = n_samples//n_chunks,
    )

    # izraunaj
    km = dask_ml.cluster.KMeans(n_clusters=n_clusters, oversampling_factor=10)
    now = time.time()
    km.fit(X)
    print('GB: %f' % (int(X.nbytes)/1073741824))
    print('elapsed fit: %f' % (time.time()-now))
```

Joblib

run_joblib.sge

```
#$ -cwd
#$ -o output/
#$ -e output/
#$ -pe *mpi 8

# aktiviraj modul
module load dask

# pokreni dask klaster
mpirun -np $NSLOTS dask-mpi \
    --nthreads 1 \
    --worker-class distributed.Worker \
    --scheduler-file scheduler.json &

# priekaj
sleep 10

# potjeraj python skriptu
python run_joblib.py
```

run_joblib.py

```
# source
# https://ml.dask.org/joblib.html

import time
import numpy as np
from dask.distributed import Client

import joblib
import pandas as pd
from sklearn.datasets import load_digits
from sklearn.model_selection import RandomizedSearchCV
from sklearn.svm import SVC

if __name__ == '__main__':

    # client
    client = Client(scheduler_file="scheduler.json")

    # data
    digits = load_digits()

    # space
    param_space = {
        'C': np.logspace(-6, 6, 20),
        'gamma': np.logspace(-8, 8, 20),
        'tol': np.logspace(-4, -1, 20),
        'class_weight': [None, 'balanced'],
    }

    # fit
    model = SVC(kernel='rbf')
    search = RandomizedSearchCV(model, param_space, cv=10, n_iter=10**3, verbose=1)

    now = time.time()
    with joblib.parallel_backend('dask'):
        search.fit(digits.data, digits.target)
    elapsed = time.time()-now

    # print
    cv_results = pd.DataFrame(search.cv_results_)
    print(cv_results)
    print('elapsed: %is' % elapsed)
```

Performanse

Broj jezgara	Dataframe [s]	K-means [s]	Joblib [s]
4	111	352	1287
8	98	158	1295
16	84	93	317
32	16	137	153
64	17	81	76
128	18.48	89	52