Jan 23, 20 Dask Basic

February 19, 2020

- Name: Jikhan Jeong
- Ref: https://docs.dask.org/en/latest/install.html (conda install dask)
- Ref; https://datascienceschool.net/view-notebook/2282b75b2a63448087b77269885c27cb/ (Korean)
-

0.1 Ref: https://towardsdatascience.com/trying-out-dask-dataframes-in-python-for-fast-data-analysis-in-parallel-aa960c18a915 (English)

[2]:
```
import pandas as pd
import numpy as np
import os
print(os.getcwd())
```

```
/data/cahnrs/jikhan.jeong/amazon
```

Part 1: Basic

---
* dask dataframe.compute() -> to pandas dataframe
* Ref; https://datascienceschool.net/view-notebook/2282b75b2a63448087b77269885c27cb/ (Korean)

---

[3]:
```
import dask.dataframe as dd
from dask.diagnostics import ProgressBar
pbar = ProgressBar()
pbar.register()
```

[8]:
```
%%writefile data1.csv
time,temperature,humidity
0,22,58
1,21,57
2,25,57
3,26,55
4,22,53
5,23,59
```

```
Overwriting data1.csv
```

[9]:
```
ls data*.csv
```

```
data1.csv  data2.csv  data3.csv
```

[10]:
```
df = dd.read_csv("data1.csv")
df
```

[10]:
```
Dask DataFrame Structure:
                time temperature humidity
npartitions=1
               int64       int64    int64
                 …           …        …
Dask Name: from-delayed, 3 tasks
```

[11]:
```
type(df)
```

[11]:
```
dask.dataframe.core.DataFrame
```

[12]:
```
df.head(1)
```

```
[########################################] | 100% Completed |  0.1s
[########################################] | 100% Completed |  0.1s
```

[12]:
```
   time  temperature  humidity
0     0           22        58
```

[13]:
```
df.temperature.mean() # task so no results
```

[13]:
```
dd.Scalar<series-…, dtype=float64>
```

[14]:
```
df.temperature.mean().compute()
```

```
[########################################] | 100% Completed |  0.1s
[########################################] | 100% Completed |  0.1s
```

```
[14]: 23.166666666666668
```

```
[15]: df.temperature.compute()
```

```
[########################################] | 100% Completed |  0.1s
[########################################] | 100% Completed |  0.1s
```

```
[15]: 0    22
      1    21
      2    25
      3    26
      4    22
      5    23
      Name: temperature, dtype: int64
```

```
[16]: (df.temperature*10+5).compute()
```

```
[########################################] | 100% Completed |  0.1s
[########################################] | 100% Completed |  0.1s
```

```
[16]: 0    225
      1    215
      2    255
      3    265
      4    225
      5    235
      Name: temperature, dtype: int64
```

```
[12]: df = df.assign(temperature = df.temperature*10)
      df.head(1)
```

```
[12]:    time  temperature  humidity
      0     0          220        58
```

```
[13]: df = df.assign(title = df.temperature.astype(str) + "jikhan is handsome")
      df.head()
```

```
[13]:    time  temperature  humidity                 title
      0     0          220        58  220jikhan is handsome
      1     1          210        57  210jikhan is handsome
      2     2          250        57  250jikhan is handsome
      3     3          260        55  260jikhan is handsome
      4     4          220        53  220jikhan is handsome
```

- read a file with while-card(*)

- count().compute()
- describe().compute()
-

0.2 E.g. df = dd.read_csv('data*.csv') # read data1, data2, data3

[17]:
```
%%writefile data2.csv
time,temperature,humidity
0,22,58
1,21,57
2,25,57
3,26,55
4,22,53
5,23,59
```

```
Overwriting data2.csv
```

[18]:
```
%%writefile data3.csv
time,temperature,humidity
0,22,58
1,21,57
2,25,57
3,26,55
4,22,53
5,23,59
```

```
Overwriting data3.csv
```

[19]:
```
ls data*.csv
```

```
data1.csv  data2.csv  data3.csv
```

[20]:
```
df1 = dd.read_csv('data*.csv')
df1.head(12)
```

```
[########################################] | 100% Completed |  0.1s
[########################################] | 100% Completed |  0.1s
```

```
/opt/apps/anaconda3/19.10.0/lib/python3.7/site-
packages/dask/dataframe/core.py:5738: UserWarning: Insufficient elements for
`head`. 12 elements requested, only 6 elements available. Try passing larger
`npartitions` to `head`.
  warnings.warn(msg.format(n, len(r)))
```

[20]:
```
   time  temperature  humidity
0     0           22        58
1     1           21        57
```

```
2     2           25         57
3     3           26         55
4     4           22         53
5     5           23         59
```

[21]:
```
len(df1)
```

```
[########################################] | 100% Completed |  0.1s
[########################################] | 100% Completed |  0.1s
```

[21]:
```
18
```

[22]:
```
df1.compute()
```

```
[########################################] | 100% Completed |  0.1s
[########################################] | 100% Completed |  0.1s
```

[22]:
```
   time  temperature  humidity
0     0           22        58
1     1           21        57
2     2           25        57
3     3           26        55
4     4           22        53
5     5           23        59
0     0           22        58
1     1           21        57
2     2           25        57
3     3           26        55
4     4           22        53
5     5           23        59
0     0           22        58
1     1           21        57
2     2           25        57
3     3           26        55
4     4           22        53
5     5           23        59
```

[23]:
```
df1.count()
```

[23]:
```
Dask Series Structure:
npartitions=1
humidity    int64
time          ...
dtype: int64
Dask Name: dataframe-count-agg, 13 tasks
```

[24]:
```
df1.count().compute()
```

```
[########################################] | 100% Completed |  0.1s
[########################################] | 100% Completed |  0.1s
```

[24]:
```
time           18
temperature    18
humidity       18
dtype: int64
```

[25]:
```
df1.temperature.describe().compute()
```

```
[########################################] | 100% Completed |  0.1s
[########################################] | 100% Completed |  0.1s
```

[25]:
```
count    18.000000
mean     23.166667
std       1.823055
min      21.000000
25%      22.000000
50%      22.500000
75%      24.500000
max      26.000000
Name: temperature, dtype: float64
```

[26]:
```
type(df1)
```

[26]:
```
dask.dataframe.core.DataFrame
```

[27]:
```
df1_pd = df1.compute()
type(df1_pd)
```

```
[########################################] | 100% Completed |  0.1s
[########################################] | 100% Completed |  0.1s
```

[27]:
```
pandas.core.frame.DataFrame
```

- Dataset: https://catalogtdatatgovudatasetucrimess2001stospresents398a4 (1.3GB)
- (Window) * using **urllib.reques** instead of wget in Linux
- (Linux) !wget -O crime.csv https://data.cityofchicago.org/api/views/ijzp-q8t2/rows.csv?accessType=DOWNLOAD (Linux)
-

0.3 Ref: https://stackoverflow.com/questions/57748687/downloading-files-in-jupyter-wget-on-windows

[24]:
```
import urllib.request
```

```
[25]: url = 'https://data.cityofchicago.org/api/views/ijzp-q8t2/rows.csv?
       ↪accessType=DOWNLOAD'
      filename = 'crime.csv'
      urllib.request.urlretrieve(url, filename)
```

```
[25]: ('crime.csv', <http.client.HTTPMessage at 0x2b13fa0f8c90>)
```

```
[27]: ls crime.csv
```

```
crime.csv
```

```
[4]: df = dd.read_csv("crime.csv", dtype= str, error_bad_lines=False,␣
      ↪warn_bad_lines=False)
     df
```

```
[4]: Dask DataFrame Structure:
                         ID Case Number    Date   Block    IUCR Primary Type
     Description Location Description  Arrest Domestic    Beat District    Ward
     Community Area FBI Code X Coordinate Y Coordinate    Year Updated On Latitude
     Longitude Location
     npartitions=26
                     object      object  object  object  object       object
     object               object  object   object  object   object  object
     object   object       object       object  object     object   object    object
     object
                        …         …     …     …     …          …
     …                  …     …      …     …      …      …
     …      …          …          …     …         …      …        …
     …
     …                …         …     …     …     …          …
     …                  …     …      …     …      …      …
     …      …          …          …     …         …      …        …
     …
                        …         …     …     …     …          …
     …                  …     …      …     …      …      …
     …      …          …          …     …         …      …        …
     …
                        …         …     …     …     …          …
     …                  …     …      …     …      …      …
     …      …          …          …     …         …      …        …
     …
     Dask Name: from-delayed, 78 tasks
```

```
[29]: df.tail()
```

```
[29]:              ID Case Number                    Date                  Block  \
      267230  11700926    JC279725  05/26/2019 05:13:00 PM  036XX W DOUGLAS BLVD
```

```
267231     24560    JC279072  05/26/2019 06:48:00 AM   013XX W HASTINGS ST
267232     24559    JC278908  05/26/2019 02:11:00 AM   013XX W HASTINGS ST
267233  11707734    JC287730  07/01/2014 07:30:00 AM   063XX S NORMAL BLVD
267234  11707239    JC287563  11/30/2017 09:00:00 AM   022XX S KOSTNER AVE

        IUCR        Primary Type                          Description  \
267230  2825        OTHER OFFENSE            HARASSMENT BY TELEPHONE
267231  0110             HOMICIDE                FIRST DEGREE MURDER
267232  0110             HOMICIDE                FIRST DEGREE MURDER
267233  1153  DECEPTIVE PRACTICE  FINANCIAL IDENTITY THEFT OVER $ 300
267234  1153  DECEPTIVE PRACTICE  FINANCIAL IDENTITY THEFT OVER $ 300

       Location Description Arrest Domestic  … Ward Community Area FBI Code  \
267230            APARTMENT  false     true  …   24             29       26
267231       CHA PARKING LOT   true    false  …   25             28      01A
267232               STREET  false    false  …   25             28      01A
267233                  NaN  false    false  …   20             68       11
267234            RESIDENCE  false    false  …   22             29       11

       X Coordinate Y Coordinate  Year             Updated On     Latitude  \
267230      1152126      1893208  2019  06/30/2019 03:56:27 PM  41.862830429
267231      1167752      1893853  2019  07/16/2019 04:17:29 PM  41.864278228
267232      1167746      1893853  2019  06/02/2019 04:09:42 PM  41.864278357
267233          NaN          NaN  2014  06/02/2019 04:09:42 PM           NaN
267234          NaN          NaN  2017  06/02/2019 04:09:42 PM           NaN

           Longitude                      Location
267230  -87.717040084  (41.862830429, -87.717040084)
267231  -87.659660218  (41.864278228, -87.659660218)
267232  -87.659682244  (41.864278357, -87.659682244)
267233            NaN                            NaN
267234            NaN                            NaN

[5 rows x 22 columns]
```

- To know the progress of work in dask task
-

0.4 Show the progress of dask work as a bar

[5]:
```python
from dask.diagnostics import ProgressBar
pbar = ProgressBar()
pbar.register()
```

[7]:
```python
type(df)
```

[7]:
```
dask.dataframe.core.DataFrame
```

```
[6]: %%time
     df.count().compute()
```

```
[########################################] | 100% Completed | 49.2s
[########################################] | 100% Completed | 49.3s
CPU times: user 40.8 s, sys: 4.65 s, total: 45.4 s
Wall time: 49.3 s
```

```
[6]: ID                      7054819
     Case Number             7054815
     Date                    7054819
     Block                   7054819
     IUCR                    7054819
     Primary Type            7054819
     Description             7054819
     Location Description    7048684
     Arrest                  7054819
     Domestic                7054819
     Beat                    7054819
     District                7054772
     Ward                    6439989
     Community Area          6441324
     FBI Code                7054819
     X Coordinate            6987226
     Y Coordinate            6987226
     Year                    7054819
     Updated On              7054819
     Latitude                6987226
     Longitude               6987226
     Location                6987226
     dtype: int64
```

- dask.get: single thread
- dask.threaded.get: multiple thread pool
- dask.multiprocessing.get : multiprocess pool
-

0.5 distributed.Client.get: multiple computer

- 20 CPUs Demo
- 0.6s faster

```
[8]: %%time
     df.count().compute(scheduler='processes', num_workers=20) # 20Cpus
```

```
[########################################] | 100% Completed | 48.2s
[########################################] | 100% Completed | 48.3s
```

```
CPU times: user 841 ms, sys: 335 ms, total: 1.18 s
Wall time: 48.6 s
```

[8]:
```
ID                      7054819
Case Number             7054815
Date                    7054819
Block                   7054819
IUCR                    7054819
Primary Type            7054819
Description             7054819
Location Description    7048684
Arrest                  7054819
Domestic                7054819
Beat                    7054819
District                7054772
Ward                    6439989
Community Area          6441324
FBI Code                7054819
X Coordinate            6987226
Y Coordinate            6987226
Year                    7054819
Updated On              7054819
Latitude                6987226
Longitude               6987226
Location                6987226
dtype: int64
```

- Dask Bags: https://examples.dask.org/bag.html
- Dask Bags: https://docs.dask.org/en/latest/bag.html
- Dask JSON data: https://examples.dask.org/applications/json-data-on-the-web.html
- Dask Tutorial: https://people.duke.edu/~ccc14/sta-663-2017/18A_Dask.html

Dask provides 3 data structures

- dask array ~ numpy array
- dask bag ~ Python dictionary
-

0.6 dask dataframe ~ pandas dataframe

[1]:
```
import dask
import dask.array as da
import dask.bag as db
import dask.dataframe as dd
from dask import delayed
```

dask arrays

- but break a massive job into tasks that are then executed by a scheduler.
- we can tell the dask array how to break the data into chunks for processing.

dask data frames

- treat multiple pandas dataframes that might not simultaneously fit into memory like a single dataframe.

dask bags

- multiset is a set that allows repeats. Unlike lists, order is not preserved.

Conversion from bag to dataframe

- dask_df = dic.to_dataframe(columns=[‘word’, ‘n’])
- dask_df.head(n=5)

Analyze web-hosted JSON data

- Ref: https://examples.dask.org/applications/json-data-on-the-web.html

Read file as a dask bag = db

- import dask.bag as db
- dic= db.read_text(filenames).map(json.loads)
- dic.take(2)

Dask Gabs

- map, filter, groupby
- Ref: https://examples.dask.org/bag.html

Read json as a dask bag

- import dask.bag as db
- import json
- b = db.read_text(’data/*.json’).map(json.loads)
- b.take(2)

Dask bag : Map, Filter, Aggregate

- b.filter(lambda record: record[‘age_variable’] > 30).take(2) # Select only people over 30
- b.map(lambda record: record[‘occupation’]).take(2) # Select the occupation field
- b.count().compute() # Count total number of records

Chain computations

- result = (b.filter(lambda record: record[‘age’] > 30) .map(lambda record: record[‘occupation’]) .frequencies(sort=True) .topk(10, key=1))
- result.compute() # we need to call compute to actually evaluate our result.

Dask Bag Tutorial 2

- https://docs.dask.org/en/latest/bag.html
- da.stack, da.concatenate, and da.block
- Ref: https://docs.dask.org/en/latest/array-stack.html

```
[29]: import dask.array as da
```

```
[30]: arr0 = da.from_array(np.zeros((3, 4)), chunks=(1, 2))
      arr1 = da.from_array(np.ones((3, 4)), chunks=(1, 2))
```

```
[31]: arr0
```

```
[31]: dask.array<array, shape=(3, 4), dtype=float64, chunksize=(1, 2),
      chunktype=numpy.ndarray>
```

```
[33]: data = [arr0, arr1]
      data
```

```
[33]: [dask.array<array, shape=(3, 4), dtype=float64, chunksize=(1, 2),
      chunktype=numpy.ndarray>,
       dask.array<array, shape=(3, 4), dtype=float64, chunksize=(1, 2),
      chunktype=numpy.ndarray>]
```

```
[34]: x = da.stack(data, axis=0)
      x.shape
```

```
[34]: (2, 3, 4)
```

```
[35]: x
```

```
[35]: dask.array<stack, shape=(2, 3, 4), dtype=float64, chunksize=(1, 1, 2),
      chunktype=numpy.ndarray>
```

```
[36]: da.stack(data, axis=1).shape
```

```
[36]: (3, 2, 4)
```

Concatenate

- concatenate existing arrays into a new array, extending them along an existing dimension

```
[38]: import dask.array as da
      import numpy as np

      arr0 = da.from_array(np.zeros((3, 4)))
      arr1 = da.from_array(np.ones((3, 4)))
      data = [arr0, arr1]
```

```
[39]: x = da.concatenate(data, axis=0)
      x.shape
```

```
[39]: (6, 4)
```

```
[40]: da.concatenate(data, axis=1).shape
```

```
[40]: (3, 8)
```

Dataframe concante example

- import dask.dataframe as dd
- df = dd.read_csv([‘data/untermaederbrunnen_station1_xyz_intensity_rgb.txt’], delimiter=’ ’, header=None, names=[‘x’, ‘y’, ‘z’, ‘intensity’, ‘r’, ‘g’, ‘b’])
- df_label = dd.read_csv([‘data/untermaederbrunnen_station1_xyz_intensity_rgb.labels’], header=None, names=[‘label’])
- df = df.repartition(npartitions=200)
- df_label = df_label.repartition(npartitions=200)
- df = dd.concat([df, df_label], axis=1)

Dask Groupby

- Ref: https://examples.dask.org/dataframes/02-groupby.html

```
[41]: import dask
      df = dask.datasets.timeseries()
      df
```

```
[41]: Dask DataFrame Structure:
                         id    name        x        y
      npartitions=30
      2000-01-01      int64  object  float64  float64
      2000-01-02          …       …        …        …
      …                   …       …        …        …
      2000-01-30          …       …        …        …
      2000-01-31          …       …        …        …
      Dask Name: make-timeseries, 30 tasks
```

```
[46]: %%time
      df.groupby('name').x.mean().compute(scheduler='processes', num_workers=20)
```

```
[########################################] | 100% Completed |  1.2s
[########################################] | 100% Completed |  1.3s
CPU times: user 1.05 s, sys: 112 ms, total: 1.17 s
Wall time: 1.34 s
```

[46]:
```
name
Alice        0.001558
Bob          0.001778
Charlie      0.001967
Dan          0.001146
Edith        0.003717
Frank        0.001993
George      -0.000409
Hannah       0.001473
Ingrid       0.000094
Jerry        0.002051
Kevin        0.000493
Laura       -0.002057
Michael     -0.000478
Norbert      0.000516
Oliver      -0.001005
Patricia     0.000314
Quinn       -0.003217
Ray         -0.000248
Sarah       -0.004322
Tim          0.002823
Ursula      -0.001512
Victor       0.002398
Wendy        0.002392
Xavier      -0.002486
Yvonne       0.002242
Zelda        0.002681
Name: x, dtype: float64
```

[ ]: