Pandas with Dask, For an Ultra-Fast Notebook

If you are into the data world then the need of using pandas might occur very frequently. Well, we all are also aware of one thing that Pandas is amazing and is more like a boon to the data world. Most of us spend at least a few minutes or even hours using pandas for data manipulation and day to day analysis. Pandas does not need much introduction but here are a few things that you should be aware of:

pandas is well suited for many different kinds of data:

- Tabular data with heterogeneously-typed columns, as in an SQL table or Excel spreadsheet
- Ordered and unordered (not necessarily fixed-frequency) time-series data.
- Arbitrary matrix data (homogeneously typed or heterogeneous) with row and column labels
- Any other form of observational/statistical data sets. The data need not be labeled at all to be placed into a pandas data structure.

Using pandas with gigabytes or terabytes of data is more like a pain in the notebook :D. I frequently use files that are into GBs and the notebook goes like *“wait a minute, let me freeze now!”*. By the way, my system has some amazing configuration. Dealing with a humongous amount

of data cannot be done using pandas, Hence that's where we have to start using **dask**. In this article I'll demonstrate how you can use pandas with dask and speed up your notebook. In dask, reading GBs of files takes only a few seconds. Before jumping onto the demo, let me give you a brief introduction about dask.

Dask:

Dask has 3 parallel collections namely Dataframes, Bags, and Arrays. Which enables it to store data that is larger than RAM. Each of these can use data partitioned between RAM and a hard disk as well distributed across multiple nodes in a cluster. A Dask DataFrame is partitioned row-wise, grouping rows by index value for efficiency. These Pandas objects may live on disk or other machines. Dask DataFrames coordinate many Pandas DataFrames or Series arranged along the index

Dask can enable efficient parallel computations on single machines by leveraging their multi-core CPUs and streaming data efficiently from disk. It can run on a distributed cluster. Dask also allows the user to replace clusters with a single-machine scheduler which would bring down the overhead. These schedulers require no setup and can run entirely within the same process as the user's session. Few other things about dask:

- The ability to work in parallel with NumPy array and Pandas DataFrame objects

- integration with other projects.
- Distributed computing
- Faster operation because of its low overhead and minimum serialization
- Runs resiliently on clusters with thousands of cores
- Real-time feedback and diagnostics

Demonstration:

The entire code used for this demonstration can be found in my Github repo and you can find the link at the end of this article. Now, let's jump on to the demo:

Importing the libraries:

```
In [1]: # Importing pandas and numpy
        import numpy as np
        import pandas as pd
```

```
In [35]: # Importing dask dataframe
         import dask
         import dask.dataframe as dd
```

Code to calculate elapsed time and file size:

```
In [ ]: # code to measure the time taken to do an operation
        start_time = datetime.now()
        # insert th code here
        time_elapsed = datetime.now() - start_time
        print('Time elapsed (hh:mm:ss.ms) {}'.format(time_elapsed))
```

```
In [15]: # Get the file size
         import os
         file = os.path.getsize("/Users/Kunal/Downloads/Data/demofile.csv")
         print ('File size in bytes is {}'.format(file))

File size in bytes is 4179614585
```

The first snippet is to calculate the elapsed time for an operation that we perform in here, for example: reading a file. In the second snippet we are displaying the file size that we'll be using for this demo. The file size is around 4GB.

> *First, we'll see a few things Dask*
>
> *is better at and then we'll skip to the things that pandas does better. This will help you to combine these two libraries and perform your analysis.*

Dask over Pandas:

Reading a file — Pandas & Dask:

```
In [20]: # Reading a file with pandas, size over 4GB
         start_time = datetime.now()
         pandasfile = pd.read_csv('/Users/Kunal/Downloads/Data/demofile.csv')
         time_elapsed = datetime.now() - start_time
         print('Time elapsed (hh:mm:ss.ms) {}'.format(time_elapsed))

         Time elapsed (hh:mm:ss.ms) 0:04:55.012559
```

Pandas took around 5 minutes to read a file of size 4gb. Wait, the size is not everything, the number of columns and rows present in a data set plays a major role in the time consumption. Let's see how much time Dask takes for the same file.

```
In [21]: # Reading a file with dask, size over 4GB
         start_time = datetime.now()
         daskfile = dd.read_csv('/Users/Kunal/Downloads/Data/demofile.csv')
         time_elapsed = datetime.now() - start_time
         print('Time elapsed (hh:mm:ss.ms) {}'.format(time_elapsed))

         Time elapsed (hh:mm:ss.ms) 0:00:00.248462
```

Holy moly, It just took around 2 milliseconds to read the

same file whereas pandas took around 5 minutes. Isn't it just amazing? Let's perform some more operations on both the pandas data frame and the dask data frame.

Appending two files — Pandas & Dask:

To perform this operation, we'll be reading another file and then append it to the previous one.

```
In [22]: # let's read another file for appending pandas dataframe
         start_time = datetime.now()
         pandasAppend= pd.read_csv('/Users/Kunal/Downloads/Data/demofile2.csv')
         time_elapsed = datetime.now() - start_time
         print('Time elapsed (hh:mm:ss.ms) {}'.format(time_elapsed))

         Time elapsed (hh:mm:ss.ms) 0:04:48.535269

In [23]: # Appending pandas dataframes
         start_time = datetime.now()
         finalFile= pandasfile.append(pandasAppend)
         time_elapsed = datetime.now() - start_time
         print('Time elapsed (hh:mm:ss.ms) {}'.format(time_elapsed))

         Time elapsed (hh:mm:ss.ms) 0:03:41.327846
```

It took around 9 minutes to perform the above-mentioned operations. Now let's see how we can optimize it by using Dask.

```
In [25]: # let's read another file for appending dask dataframe
         start_time = datetime.now()
         daskAppend = dd.read_csv('/Users/Kunal/Downloads/Data/demofile2.csv')
         time_elapsed = datetime.now() - start_time
         print('Time elapsed (hh:mm:ss.ms) {}'.format(time_elapsed))

         Time elapsed (hh:mm:ss.ms) 0:00:00.343578

In [26]: # Appending dask dataframes
         start_time = datetime.now()
         daskFinalFile= daskfile.append(daskAppend)
         time_elapsed = datetime.now() - start_time
         print('Time elapsed (hh:mm:ss.ms) {}'.format(time_elapsed))

         Time elapsed (hh:mm:ss.ms) 0:00:00.068017
```

Well, Well, you just saved around another 9 minutes there. Let us see now other frequently used this we do with

pandas.

Grouping the data — Pandas & Dask:

```
In [29]: # Grouping with pandas
         start_time = datetime.now()
         finalFile.groupby('status')['id']
         time_elapsed = datetime.now() - start_time
         print('Time elapsed (hh:mm:ss.ms) {}'.format(time_elapsed))

         Time elapsed (hh:mm:ss.ms) 0:06:07.065004
```

Pandas took over 6 minutes to do a simple grouping. Let's see now how much more time we can save with dask.

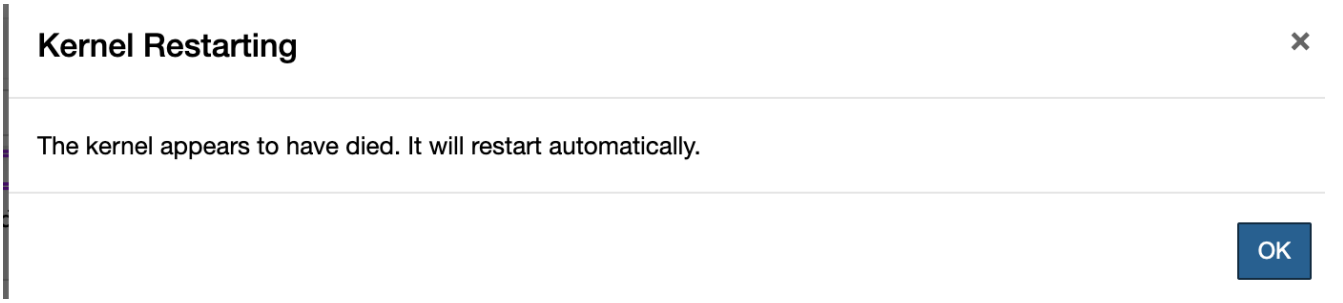
```
In [32]: # Grouping with dash
         start_time = datetime.now()
         daskFinalFile.groupby('status')['sid']
         time_elapsed = datetime.now() - start_time
         print('Time elapsed (hh:mm:ss.ms) {}'.format(time_elapsed))

         Time elapsed (hh:mm:ss.ms) 0:00:00.005593
```

Woah, you just saved whooping 6 minutes again. There numerous other things where you can save much more time if you are using dask.

Merging the datasets — Pandas & Dask:

```
In [ ]: # Merging the datasets with pandas
        start_time = datetime.now()
        mergedPandas= pd.merge(pandasfile, pandasAppend, on="sid")
        time_elapsed = datetime.now() - start_time
        print('Time elapsed (hh:mm:ss.ms) {}'.format(time_elapsed))
```



So this is what happened. It tried for like 30 minutes and

still couldn't merge those two files with pandas. Let's see if we can do it by using dask.

```
In [8]: # Merging the datasets with dask
        start_time = datetime.now()
        mergedDask= dd.merge(daskFile, daskAppend, on="sid")
        time_elapsed = datetime.now() - start_time
        print('Time elapsed (hh:mm:ss.ms) {}'.format(time_elapsed))

        Time elapsed (hh:mm:ss.ms) 0:00:00.048704
```

Well, It barely took me a second to do it using dask. What just happened is, Unlike `pandas.read_csv` which reads in the entire file before inferring data types, `dask.dataframe.read_csv` only reads in a sample from the beginning of the file (or first file if using a glob). These inferred data types are then enforced when reading all partitions.

Pandas over Dask:

Sorting — Pandas & Dask:

```
In [32]: # Sorting the data using pandas
         start_time = datetime.now()
         sortedPandas=pandasfile.sort_values(by='skippedQ')
         time_elapsed = datetime.now() - start_time
         print('Time elapsed (hh:mm:ss.ms) {}'.format(time_elapsed))

         Time elapsed (hh:mm:ss.ms) 0:01:28.420653
```

I tried to sort the data frame based on the values in a column and it took me around a minute and a half, that's pretty decent. Let's see how dask can help us with this.

```
In [29]: # Sorting the data using dask
start_time = datetime.now()
sortedDask=daskFinalFile.sort_values(by='skippedQ')
time_elapsed = datetime.now() - start_time
print('Time elapsed (hh:mm:ss.ms) {}'.format(time_elapsed))
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-29-6528d2c90530> in <module>
----> 1 daskFinalFile.sort_values(by='skippedQ')

~/opt/anaconda3/envs/notebook/lib/python3.7/site-packages/dask/dataframe/core.py in __getattr__(self, key)
   3412             return self[key]
   3413         else:
-> 3414             raise AttributeError("'DataFrame' object has no attribute %r" % key)
   3415
   3416     def __dir__(self):

AttributeError: 'DataFrame' object has no attribute 'sort_values'
```

Unfortunately, dask cannot even start this task because dask has no functionality of sorting although it uses pandas API. All hail pandas!

Unique & notNA — Pandas & Dask:

```
In [41]: # Getting not NA values using pandas
start_time = datetime.now()
notnaPandas=FinalFile.notna()
time_elapsed = datetime.now() - start_time
print('Time elapsed (hh:mm:ss.ms) {}'.format(time_elapsed))
```

```
Time elapsed (hh:mm:ss.ms) 0:00:48.975447
```

```
In [47]: # Getting unique values using pandas
start_time = datetime.now()
uniquePandas=pd.unique(pandasfile['id'])
time_elapsed = datetime.now() - start_time
print('Time elapsed (hh:mm:ss.ms) {}'.format(time_elapsed))
```

```
Time elapsed (hh:mm:ss.ms) 0:00:11.269945
```

It took around only 1 minute to get these tasks done. Pandas does most of the things pretty well but screws in quite a few.

```
In [49]: # Getting not NA values using dask
start_time = datetime.now()
notnaDask=daskFinalFile.notna()
time_elapsed = datetime.now() - start_time
print('Time elapsed (hh:mm:ss.ms) {}'.format(time_elapsed))
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-49-dfae8a7c4654> in <module>
      1 # Getting not NA values using dask
      2 start_time = datetime.now()
----> 3 notnaDask=daskFinalFile.notna()
      4 time_elapsed = datetime.now() - start_time
      5 print('Time elapsed (hh:mm:ss.ms) {}'.format(time_elapsed))

~/opt/anaconda3/envs/notebook/lib/python3.7/site-packages/dask/dataframe/core.py in __getattr__(self, key)
   3412             return self[key]
   3413         else:
-> 3414             raise AttributeError("'DataFrame' object has no attribute %r" % key)
   3415
   3416     def __dir__(self):

AttributeError: 'DataFrame' object has no attribute 'notna'
```

```
In [50]: # Getting unique values using dask
start_time = datetime.now()
notnaDask=dd.unique(daskFinalFile['id'])
time_elapsed = datetime.now() - start_time
print('Time elapsed (hh:mm:ss.ms) {}'.format(time_elapsed))
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-50-8d4717c1a0f5> in <module>
      1 # Getting not NA values using dask
      2 start_time = datetime.now()
----> 3 notnaDask=dd.unique(daskFinalFile['id'])
      4 time_elapsed = datetime.now() - start_time
      5 print('Time elapsed (hh:mm:ss.ms) {}'.format(time_elapsed))

AttributeError: module 'dask.dataframe' has no attribute 'unique'
```

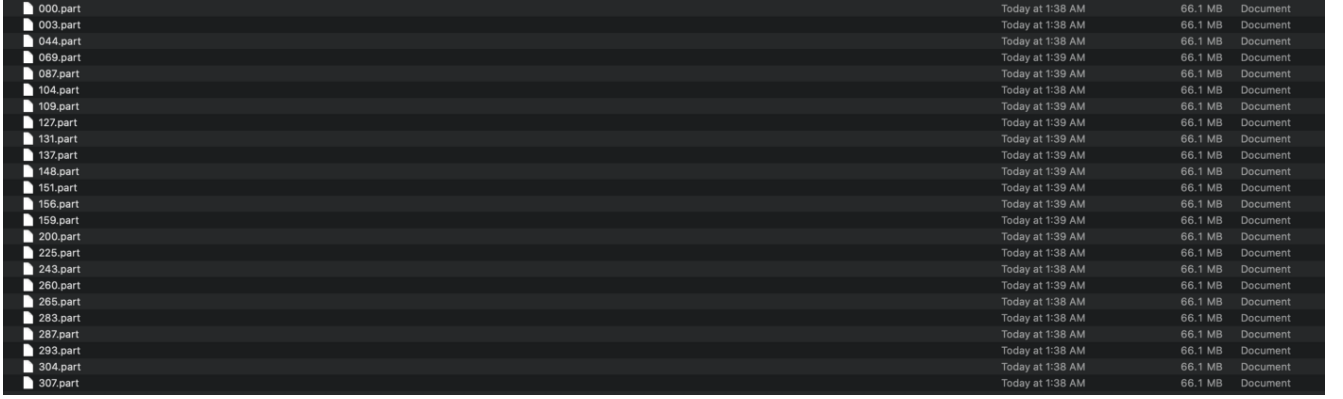
Dask does not support these two things as well. There are numerous other things for which you'll have to use pandas.

Saving a Dataframe to a file — Pandas & Dask:

```
In [ ]: # saving a file using pandas
start_time = datetime.now()
uniquePandas.to_csv('/Users/Kunal/Downloads/Data/finalFile.csv')
time_elapsed = datetime.now() - start_time
print('Time elapsed (hh:mm:ss.ms) {}'.format(time_elapsed))
```

Pandas does a good job of saving the file. It took around 3 minutes for me to save the filtered file.

```
In [ ]: # saving a file using dask
start_time = datetime.now()
daskFinalFile.to_csv('/Users/Kunal/Downloads/Data/DaskFinalFile.csv')
time_elapsed = datetime.now() - start_time
print('Time elapsed (hh:mm:ss.ms) {}'.format(time_elapsed))
```



Dask does not save the file properly. It breaks the file into multiple chunks and saves these files in a folder with that mentioned name. Another problem is that you can't read this saved file ever again. It is just a waste of time.

The solution for saving a dask data frame to a file is to convert it into a pandas data frame like this and then save the pandas data frame to a file.

```
In [93]: # Convert dask dataframe to pandas dataframe if you want to save the results to a file
         start_time = datetime.now()
         daskFinalFile.compute()
         time_elapsed = datetime.now() - start_time
         print('Time elapsed (hh:mm:ss.ms) {}'.format(time_elapsed))

         Time elapsed (hh:mm:ss.ms) 0:07:31.888239
```

> ***Conclusion:***
>
> ***It is always the best option to use pandas and dask together because one can fill other's limitations very well. When used individually, I suppose that you might run into different issues. Hence, we conclude that Pandas with Dask can save you a lot of time and resources.***

Note:

Dask is faster because it doesn't really execute anything until we use .compute(). Although, This can save a lot, a

lot of time and give us more speed!

Tip:

```
# Use this to make pandas run faster
pd.read_csv(filepath, engine = 'c')
```

Do let me know if you are aware of any better alternatives than dask.

> ***Jupyter Notebook (Code used) :*** *https://github.com/kunaldhariwal/Medium-12-Amazing-Pandas-NumPy-Functions*
>
> ***LinkedIn****: https://bit.ly/2u4YPoF*

I hope that this has helped you to enhance your knowledge base :)

Follow me for more!

Thanks for your read and valuable time!