Quick Pandas and Dask comparison processing large csv files. Real world example that you can run now!

The goal of this article is to show the benefit of using Dask over Pandas.

The goal is to get people to see the value of Dask. Everything you need, data and all the commands used are here for you to download and run.

1. Introduction

Before we start you must:

- Have python 3.5 or older installed with venv installed
- At the moment a Linux-based system or a little bit of knowledge to translate the commands (I don't have a Windows machine close but give me a shout if you want me to translate the non-compatible commands)

Dask is made out of two parts, a Dynamic task scheduler (something to schedule and lunch Dask tasks to process data and other things) and a data collection part. This second part is what you can directly compare to Pandas.

You can think about it as a DataFrame that you can divide into sections and run each section in parallel in a different location. You could do that manually by subdividing a Pandas DataFrame and call some type of async function. Dask does that for you out of the box.

Now the important bit, sometimes using Pandas is a better idea (better performance, more convenient etc). One of the main variables that influences whether you will get better performance in Pandas or Dask is the size of your data. We will now compare both in exactly this aspect.

Please note that this is a complex topic and many variables will influence this decision but hopefully this article will get you started!

2. Environment preparation and data download

We are going to create a virtual environment, install Pandas, Dask and Jupyter Notebooks (this last one just to run our code).

We will now create the main folder called pandasdask and a virtual environment called venv inside:

```
mkdir pandasdask
cd pandasdask
python3 -m venv venv
```

Now we will activate the virtual environment and install the packages we are going to need

```
source venv/bin/activate
pip install "dask[complete]==2.27.0" pandas==1.1.2 jupyter==1.0.0
```

When I was installing it, some wheels failed but the packages were installed correctly nonetheless.

Before we move on to our notebook, let's download the data we are going to use. We will use the uk gov housing price paid data. Make sure you read the usage guidelines here https://www.gov.uk/government/statistical-data-sets/price-paid-data-downloads#using-or-publishing-our-price-paid-data

The copyright disclaimer:

Contains HM Land Registry data © Crown copyright and database right 2020. This data is licensed under the Open Government Licence v3.0.

We will download all the data for 2019 and all the data ever recorded. This second file is 3.6 GB at the time of writing (Sept 2020) which will allow showcasing Dask's capabilities.

```
mkdir data
cd data
wget http://prod.publicdata.landregistry.gov.uk.s3-website-eu-west-1.amazonaws.com/pp-2019.csv
wget http://prod.publicdata.landregistry.gov.uk.s3-website-eu-west-1.amazonaws.com/pp-complete.csv
cd ..
```

We created a data folder, went into it, downloaded the data and now we are back at the root of our directory.

3. Running the comparison

Now, let's start jupyter notebook by running (make sure your virtual environment is activated)

```
jupyter notebook
```

Then create a new notebook and copy these sections into separated sections. First we import the libraries we are going to need

```
import pandas as pd
import time
import dask
import dask.dataframe as dd
from dask.delayed import delayed
import time
from dask.distributed import Client, progress
```

Then we define our data location and the columns names

```
one_year_data = "data/pp-2019.csv"
all_data = "data/pp-complete.csv"
columns = ["transaction", "price", "date", "postcode", "prop_type", "old_new", "duration", "paon", "saon", "street", "locality", "city", "district", "county
```

Now we run the Pandas version

```
start = time.time()
df = pd.read_csv(one_year_data,  header=0, names=columns)
df_by_county = df.groupby("county")["price"].sum()
print(df_by_county)
end = time.time()
print("time elapsed {}".format(end-start))
```

Once that finishes, we initialise the Dask workers. This is were you can start playing with different configurations. I'm going to use 4 workers and 4 threads because it suits my particular architecture. I recommend you to change these settings and see what happens when you run the code afterwards. Try matching the workers to your cores and changing the number of threads for example.

```
client = Client(threads_per_worker=4, n_workers=4)
client
```

You can click in the link that appears when running this command to have a look at how thing are being processed (the link will be output when running the command in pandas).

Finally, we run the Dask version of the code. Here I'm using a blocksize of 32 MB. Again this is something you should change to 16 or 64 to see the difference it makes.

```
start = time.time()
df = dd.read_csv(one_year_data,  blocksize=32 * 1024 * 1024, header=0, names=columns)
df_by_county = df.groupby("county")["price"].sum().compute()
print(df_by_county)
end = time.time()
print("time elapsed {}".format(end-start))
```

Here are results. For the one_year_data file, I got:

```
pandas - time elapsed 2.32 seconds (rounded)
dask - time elapsed 1.37 seconds (rounded)
```

We are already seeing a gain. More fundamentally, we should be seeing a much large gain on the amount of memory we are using. You can have a quick approximate look at this by using a command such as htop or a resources monitor. Now, you should run this same exercise several times with the same and different parameters to look for an average rather than a punctual result. I found my results were fairly stable when doing this.

WARNING!!!! THE NEXT STEP MIGHT USE ALL YOUR COMPUTER'S MEMORY AND CRASH IT. MAKE SURE YOU HAVE

SAVED ALL PROGRESS AND HAVE ENOUGH MEMORY TO LOAD THE 3.6 GB FILE (you will need around 10 GB for this).

Now, let's run the same code but processing the large all data file. Your code should look like this:

```
start = time.time()
df = pd.read_csv(all_data,  header=0, names=columns)
df_by_county = df.groupby("county")["price"].sum()
print(df_by_county)
end = time.time()
print("time elapsed {}".format(end-start))
```

and for dask.

```
start = time.time()
df = dd.read_csv(all_data,  blocksize=32 * 1024 * 1024, header=0, names=columns)
df_by_county = df.groupby("county")["price"].sum().compute()
print(df_by_county)
end = time.time()
print("time elapsed {}".format(end-start))
```

The results

```
pandas - time elapsed 55.36 seconds (rounded) (around 10GB memory used)
dask - time elapsed 19.6 seconds (rounded) (around 4.5 GB memory used)
```

That is a larger difference and it is what you would expect by running the process in parallel.

If your machine ran out of memory (it runs in mine but I have 32 GB of ram), try using a smaller file by dividing the file using the following command. This command will divide in approximately 4 sections each of a maximum of 7000 lines

```
split -l 7000000 data/pp-complete.csv data/half_data.csv
```

4. Conclusion and final thoughts

We saw considerable gains in the larger file by using Dask. Pandas can be made more efficient and to run "chunks" as well but the main idea was to illustrate out of the box "vanilla" behaviour. You can read more about this here:

https://stackoverflow.com/questions/25962114/how-do-i-read-a-large-csv-file-with-pandas

It is important to remember that this is a very complicated topic and there are many factor to consider before concluding that Dask will solve all your problems. It is a different more complex setup that might require extra monitoring for example.

Also, it will be heavily dependent on your file size (Pandas should beat Dask in file of around 10 MB and smaller) and your architecture. One interesting thing to test for example is make your computer run something heavy like Slack or a spammy web page (a tabloid would be ideal here) and re-run the exercise. You will see that if you are already taking out a couple of cores with other processes, the difference between Pandas and Dask might shorten in relative terms (being Pandas a single core process). Even in some cases I saw Pandas run faster than Dask in the smaller file.