dask `compute()` Deferred Computing

We're going to build a somewhat interesting workload and then run it a couple of different ways. Let's start by loading the NYC flight data.

This exercise will reinforce dask dataframe programming concepts by building a set of analyses. We will then use these type of `groupby` and aggregate queries to look at execution properties.

Code that you need to write is indicated with #TODO. I've left the output of the reference implementation in the cells so that you can refer to it for correctness. You can refer to the read-only shared version for this output.

```
%pip install fsspec
%pip install gcsfs
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-whe
Requirement already satisfied: fsspec in /usr/local/lib/python3.7/dist-packages
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-whe
Collecting gcsfs
  Downloading gcsfs-2022.10.0-py2.py3-none-any.whl (25 kB)
Requirement already satisfied: google-auth-oauthlib in /usr/local/lib/python3.7/
Requirement already satisfied: google-cloud-storage in /usr/local/lib/python3.7/
Requirement already satisfied: decorator>4.1.2 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: aiohttp!=4.0.0a0,!=4.0.0a1 in /usr/local/lib/pyth
Requirement already satisfied: google-auth>=1.2 in /usr/local/lib/python3.7/dist
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: fsspec==2022.10.0 in /usr/local/lib/python3.7/dis
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.7/dis
Requirement already satisfied: charset-normalizer<3.0,>=2.0 in /usr/local/lib/py
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.7/d
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: typing-extensions>=3.7.4 in /usr/local/lib/python
Requirement already satisfied: yarl<2.0,>=1.0 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: async-timeout<5.0,>=4.0.0a3 in /usr/local/lib/pyt
Requirement already satisfied: asynctest==0.13.0 in /usr/local/lib/python3.7/dis
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.7/dist
Requirement already satisfied: cachetools<5.0,>=2.0.0 in /usr/local/lib/python3.
Requirement already satisfied: six>=1.9.0 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: setuptools>=40.3.0 in /usr/local/lib/python3.7/di
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.7
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in /usr/local/lib/python3.7/
Requirement already satisfied: idna>=2.0 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/di
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dis
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/l
Requirement already satisfied: google-cloud-core<2.0dev,>=1.0.0 in /usr/local/li
Requirement already satisfied: google-resumable-media<0.5.0dev,>=0.3.1 in /usr/l
```

```
Requirement already satisfied: google-api-core<2.0.0dev,>=1.14.0 in /usr/local/l
Requirement already satisfied: pytz in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: packaging>=14.3 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: googleapis-common-protos<2.0dev,>=1.6.0 in /usr/l
Requirement already satisfied: protobuf<4.0.0dev,>=3.12.0 in /usr/local/lib/pyth
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/python
Installing collected packages: gcsfs
Successfully installed gcsfs-2022.10.0
```

Read in the NYC Flights data from Google cloud storage and then print the dataframe metadata.

```
import dask.dataframe as dd

df = dd.read_csv('gcs://nycflights/*.csv',
                 storage_options={'token': 'anon'},
                 parse_dates={'Date': [0, 1, 2]},
                 dtype={'TailNum': str,
                        'CRSElapsedTime': float,
                        'Cancelled': bool})
df
```

Dask DataFrame Structure:

	Date	DayOfWeek	DepTime	CRSDepTime	ArrTime	CRSAr
npartitions=10						
	datetime64[ns]	int64	float64	int64	float64	
	...	...	...	...	...	
...	...	...	...	...	...	
	...	...	...	...	...	
	...	...	...	...	...	

Dask Name: read-csv, 10 tasks

Let's build a set of queries around the performance of particular planes, identified by tail number. The pattern will be to `groupby('TailNum')` and then compute statistics.

Query: What is the average departure delay 'DepDelay' for each plane?

```
#TODO
df_delay = df.groupby('TailNum')['DepDelay'].mean()
delay = df_delay.compute()
```

```
print(delay.max())
delay
```

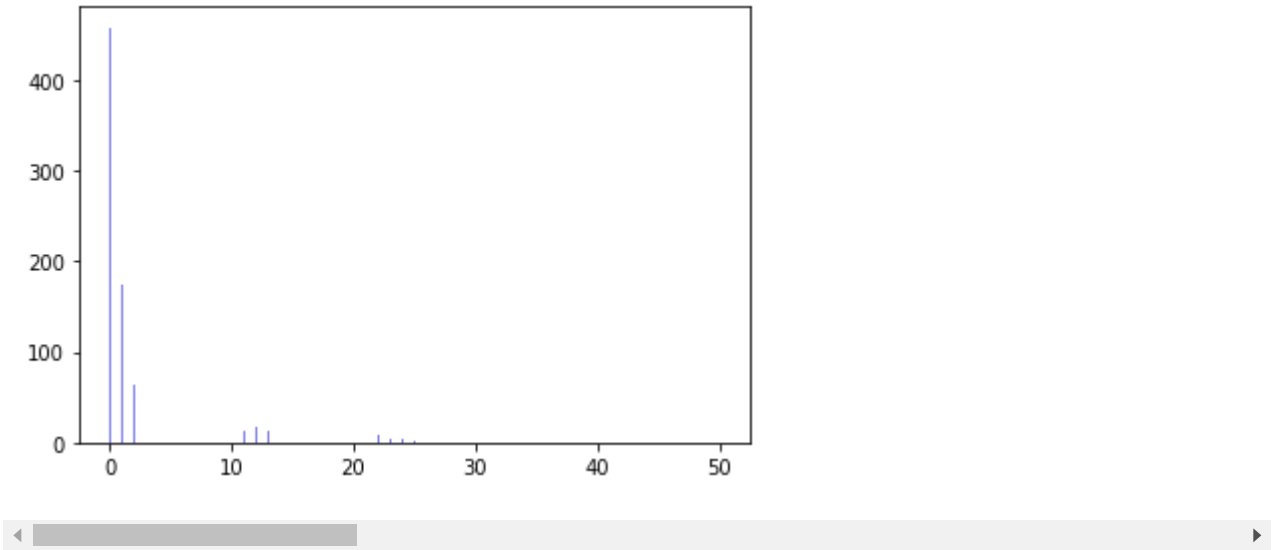
```
241.0
TailNum
EI-BWD    11.213501
EI-CAL    23.846154
EI-CAM    26.611511
EI-CIW    12.918182
N050AA     9.180180
             ...
N976TW    -2.294118
N978TW     0.000000
N979TW     5.250000
N980TW     3.428571
N982TW    14.000000
Name: DepDelay, Length: 3712, dtype: float64
```

Interesting, some planes were early, lets plot a histrogram of the distribution with 1000 bins.

```
import seaborn as sns
import dask.array as da
import matplotlib.pyplot as plt

%matplotlib inline
h, bins = da.histogram(df_delay, bins=1000,range=[0,50])
n, bins, patches = plt.hist(h, bins, facecolor='blue', alpha=0.5)
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/dask/array/core.py:1645: FutureWarni
  FutureWarning,
```



OK, we have very few chronically bad planes. Let's find those that are 30 (or more) minutes late on average.

```
import numpy as np
lateplanes = delay.loc[lambda x : x > 30].index.to_numpy()
```

```
print(np.sort(lateplanes))
```

```
['N101UW' 'N102UW' 'N104UW' 'N106UW' 'N133JC' 'N133TW' 'N134TW' 'N14249'
 'N144JC' 'N147US' 'N151AW' 'N151UA' 'N152UA' 'N153US' 'N154AA' 'N154AW'
 'N155US' 'N156AW' 'N158AW' 'N161US' 'N168AW' 'N169AW' 'N17010' 'N17011'
 'N1738D' 'N1739D' 'N174AW' 'N174GM' 'N174UA' 'N175UA' 'N17789' 'N1854U'
 'N195UA' 'N199UA' 'N224DA' 'N224NW' 'N225NW' 'N235NW' 'N303TW' 'N304AW'
 'N305AW' 'N305TW' 'N307TW' 'N322AW' 'N328AW' 'N33021' 'N3310L' 'N331AW'
 'N375DA' 'N376DL' 'N379DL' 'N382DA' 'N53110' 'N53116' 'N534TW' 'N6700'
 'N701UW' 'N706UW' 'N708UW' 'N713DA' 'N713UW' 'N716DA' 'N719DA' 'N724DA'
 'N727UW' 'N733DS' 'N735D' 'N737D' 'N760DH' 'N78019' 'N787DL' 'N789DL'
 'N802DE' 'N805DE' 'N817AA' 'N8911E' 'N93104' 'N93107' 'N93108' 'N93109'
 'N93119' 'N96S' 'N971Z' 'N976UA' 'N993UA' 'NEIDLA' 'UNKNOW']
```

OK, this is a hard query. Build a dataframe that is a subset all the data associated with the late planes. There are many ways to solve this problem. I would recommend looking at the `isin()` function in dask.

```
df_late = df.loc[df['TailNum'].isin(lateplanes),:]
df_late
```

	Date	DayOfWeek	DepTime	CRSDepTime	ArrTime	CRSArrTime	UniqueCar
8	1995-01-09	1	NaN	1950	NaN	2337	
39	1995-01-18	3	NaN	1640	NaN	2002	
248	1995-01-11	3	NaN	1815	NaN	2202	

Double check that the planes indexes match.

```
import numpy as np
latelist = df_late['TailNum'].unique()
print(np.sort(latelist))
```

```
['N101UW' 'N102UW' 'N104UW' 'N106UW' 'N133JC' 'N133TW' 'N134TW' 'N14249'
 'N144JC' 'N147US' 'N151AW' 'N151UA' 'N152UA' 'N153US' 'N154AA' 'N154AW'
 'N155US' 'N156AW' 'N158AW' 'N161US' 'N168AW' 'N169AW' 'N17010' 'N17011'
 'N1738D' 'N1739D' 'N174AW' 'N174GM' 'N174UA' 'N175UA' 'N17789' 'N1854U'
 'N195UA' 'N199UA' 'N224DA' 'N224NW' 'N225NW' 'N235NW' 'N303TW' 'N304AW'
 'N305AW' 'N305TW' 'N307TW' 'N322AW' 'N328AW' 'N33021' 'N3310L' 'N331AW'
 'N375DA' 'N376DL' 'N379DL' 'N382DA' 'N53110' 'N53116' 'N534TW' 'N6700'
 'N701UW' 'N706UW' 'N708UW' 'N713DA' 'N713UW' 'N716DA' 'N719DA' 'N724DA'
 'N727UW' 'N733DS' 'N735D' 'N737D' 'N760DH' 'N78019' 'N787DL' 'N789DL'
 'N802DE' 'N805DE' 'N817AA' 'N8911E' 'N93104' 'N93107' 'N93108' 'N93109'
 'N93119' 'N96S' 'N971Z' 'N976UA' 'N993UA' 'NEIDLA' 'UNKNOW']
```

Now, let's get a sense of what airports these planes fly out of. For the planes in the late_list, let's find out the total delay at these airports, the average delay by airport and the total number of flights at each airport.

```
#TODO total DepDelay for planes by Origin airport
df_late.groupby("Origin")["DepDelay"].sum().compute()
```

```
Origin
EWR    16982.0
JFK    61684.0
LGA    27669.0
Name: DepDelay, dtype: float64
```

```
#TODO average DepDelay for planes by Origin airport
df_late.groupby("Origin")["DepDelay"].mean().compute()
```

```
Origin
EWR    43.101523
JFK    41.763033
LGA    36.027344
Name: DepDelay, dtype: float64
```

```
#TODO number of late flights by Origin airport
df_late.groupby("Origin")["DepDelay"].count().compute()
```

```
Origin
EWR     394
JFK    1477
LGA     768
Name: DepDelay, dtype: int64
```

I don't know that these statistics all make sense, but that's to debug.

Deferred computing

We are going to show the value of deferred computation by timing the following queries in two different ways:

```
df1 = df.groupby(['Origin','TailNum']).DepDelay.mean()
df2 = df.groupby(['TailNum','Origin']).DepDelay.mean()
df3 = df.groupby(['Origin','TailNum']).DepDelay.max()
df4 = df.groupby(['TailNum','Origin']).DepDelay.max()
```

1. In one cell, add these lines and then call `compute()` on every step.
2. In the next cell, add the lines and only call compute at the end.

   First reload the data:

```
import dask.dataframe as dd
df = dd.read_csv('gs://nycflights/*.csv',
                 storage_options={'token': 'anon'},
                 parse_dates={'Date': [0, 1, 2]},
                 dtype={'TailNum': str,
                        'CRSElapsedTime': float,
                        'Cancelled': bool})
```

Run the workload calling `compute()` on every line.

```
%%time

#TODO
df1 = df.groupby(['Origin','TailNum']).DepDelay.mean().compute()
df2 = df.groupby(['TailNum','Origin']).DepDelay.mean().compute()
df3 = df.groupby(['Origin','TailNum']).DepDelay.max().compute()
df4 = df.groupby(['TailNum','Origin']).DepDelay.max().compute()
```

```
CPU times: user 24.2 s, sys: 2.23 s, total: 26.4 s
Wall time: 21 s
```

Load the data again to make sure that intermediate results are not cached and run the entire workload calling `compute()` just once.

```
import dask.dataframe as dd
df = dd.read_csv('gs://nycflights/*.csv',
                 storage_options={'token': 'anon'},
                 parse_dates={'Date': [0, 1, 2]},
                 dtype={'TailNum': str,
                        'CRSElapsedTime': float,
                        'Cancelled': bool})
```

```
%%time

import dask

df1 = df.groupby(['Origin','TailNum']).DepDelay.mean()
df2 = df.groupby(['TailNum','Origin']).DepDelay.mean()
df3 = df.groupby(['Origin','TailNum']).DepDelay.max()
df4 = df.groupby(['TailNum','Origin']).DepDelay.max()

df1,df2,df3,df4 = dask.compute(df1,df2,df3,df4)

#TODO
print(df1)
print(df2)
print(df3)
print(df4)
```

```
Origin  TailNum
EWR     EI-BWD      9.355140
        EI-CIW     16.283019
        N050AA      8.309677
        N051AA      5.949275
        N052AA     21.845070
                       ...
LGA     N993UA     55.000000
        N994UA     -3.250000
        N995UA     14.600000
        N996UA      9.333333
        N998UA      1.750000
Name: DepDelay, Length: 8861, dtype: float64
TailNum  Origin
EI-BWD   EWR        9.355140
         JFK       11.575758
         LGA       11.626866
```

```
EI-CAL   JFK       23.846154
EI-CAM   JFK       26.611511
                     ...
N993UA   LGA       55.000000
N994UA   LGA       -3.250000
N995UA   LGA       14.600000
N996UA   LGA        9.333333
N998UA   LGA        1.750000
Name: DepDelay, Length: 8861, dtype: float64
Origin  TailNum
EWR     EI-BWD     177.0
        EI-CIW     331.0
        N050AA     248.0
        N051AA     140.0
        N052AA     996.0
                   ...
LGA     N993UA     227.0
        N994UA       0.0
        N995UA      59.0
        N996UA      17.0
        N998UA       8.0
Name: DepDelay, Length: 8861, dtype: float64
TailNum  Origin
EI-BWD   EWR       177.0
         JFK       414.0
         LGA       208.0
EI-CAL   JFK       350.0
EI-CAM   JFK       225.0
                   ...
N993UA   LGA       227.0
N994UA   LGA         0.0
N995UA   LGA        59.0
N996UA   LGA        17.0
N998UA   LGA         8.0
Name: DepDelay, Length: 8861, dtype: float64
CPU times: user 7.67 s, sys: 645 ms, total: 8.31 s
Wall time: 6.1 s
```

Outcomes

- Wrestled with dataframes syntax and concepts. Good for you.
- Witnessed the benefit of deferred computation.

Questions

1. On computational reuse in execution graphs:

   a. How much faster is it to defer the computation to the end versus calling `compute()` on every line?

20 seconds faster.

b. What computations are shared in the workflow? Be specific, i.e. identify the code.

The groupby operation is shared.

c. Explain the speedup realized in 1(a). Why is it not faster? Why is it not slower?