Using Dask DataFrames

PARALLEL COMPUTING WITH DASK

Dhavide Aruliah
Director of Training, Anaconda

Reading CSV

```
import dask.dataframe as dd
```

- `dd.read_csv()` function
  - Accepts single filename or *glob* pattern (with wildcard `*`)
  - Does not read file immediately (*lazy evaluation*)
  - File(s) need not fit in memory

Reading multiple CSV files

```
%ls
```

```
quarter1.csv  quarter2.csv  quarter3.csv  quarter4.csv
```

```
transactions = dd.read_csv('*.csv')
```

```
transactions.head()

transactions.tail()
```

```
    id    names  amount        date
0  131  Norbert   -1159  2016-01-01
1  342    Jerry    1149  2016-01-01
2  485      Dan    1380  2016-01-01
3  513   Xavier    1555  2016-01-02
4  849  Michael     363  2016-01-02
```

```
      id      names  amount        date
195  838      Wendy      87  2016-12-28
196  915        Bob     852  2016-12-30
197  749   Patricia    1741  2016-12-31
198  743    Michael    1191  2016-12-31
199  889      Wendy     336  2016-12-31
```

Building delayed pipelines

```
is_wendy = (transactions['names'] == 'Wendy')

wendy_amounts = transactions.loc[is_wendy, 'amount']

wendy_amounts
```

```
Dask Series Structure:
npartitions=4
None    int64
None      ...
None      ...
None      ...
None      ...
Name: amount, dtype: int64
Dask Name: loc-series, 24 tasks
```

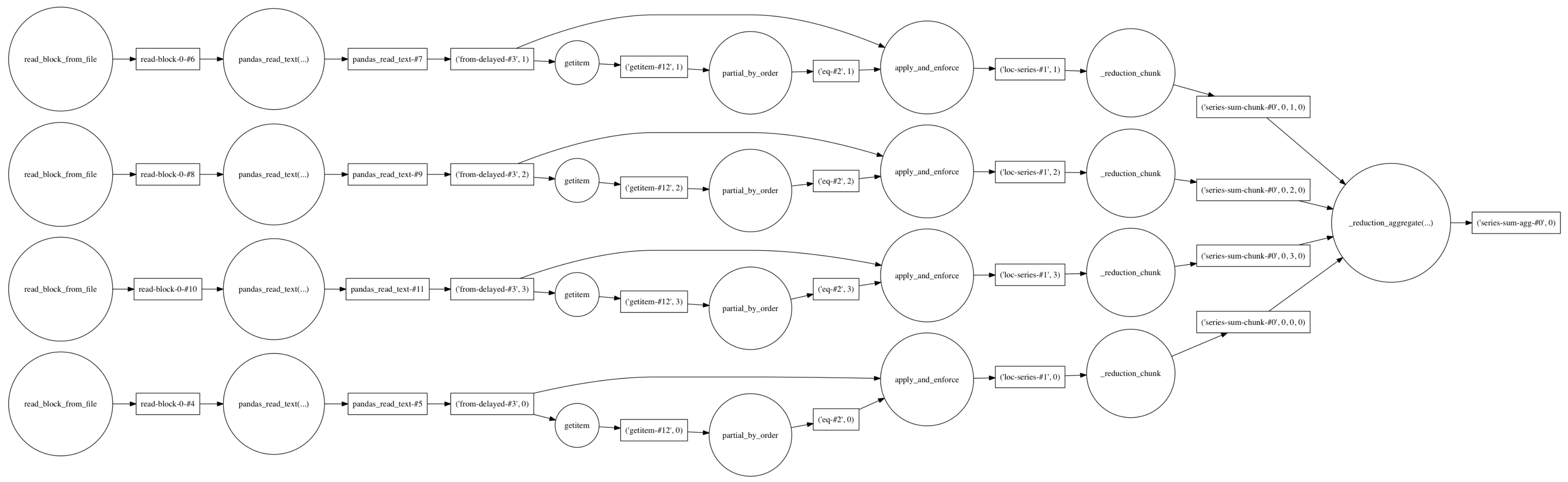
Building delayed pipelines

```
wendy_diff = wendy_amounts.sum()
wendy_diff
```

```
dd.Scalar<series-..., dtype=int64>
```

```
wendy_diff.visualize(rankdir='LR')
```

Visualizing pipelines



Compatibility with Pandas API

Unavailable in `dask.dataframe` :

- some unsupported file formats (e.g., `.xls` , `.zip` , `.gz`)
- sorting

Available in `dask.dataframe` :

- indexing, selection, & reindexing
- aggregations: `.sum()` , `.mean()` , `.std()` , `.min()` , `.max()` etc.
- grouping with `.groupby()`
- datetime conversion with `dd.to_datetime()`

Let's practice!

PARALLEL COMPUTING WITH DASK

Timing DataFrame Operations

PARALLEL COMPUTING WITH DASK

Dhavide Aruliah
Director of Training, Anaconda

How big is big data?

Data size M	Required hardware
$M < 8\,\mathrm{GB}$	RAM (single machine)
$8\,\mathrm{GB} < M < 10\,\mathrm{TB}$	hard disk (single machine)
$M > 10\,\mathrm{TB}$:	*specialized hardware*

Two key questions:

- Data fits in RAM (random access memory)?
- Data fits on hard disk?

Taxi CSV files

```
%ll -h yellow_tripdata_2015-*.csv
```

```
-rw-r--r--  1 user  staff   1.8G 31 Jul 16:43 yellow_tripdata_2015-0
-rw-r--r--  1 user  staff   1.8G 31 Jul 16:43 yellow_tripdata_2015-0
-rw-r--r--  1 user  staff   1.9G 31 Jul 16:43 yellow_tripdata_2015-0
-rw-r--r--  1 user  staff   1.9G 31 Jul 16:43 yellow_tripdata_2015-0
-rw-r--r--  1 user  staff   1.9G 31 Jul 16:43 yellow_tripdata_2015-0
-rw-r--r--  1 user  staff   1.8G 31 Jul 16:43 yellow_tripdata_2015-0
-rw-r--r--  1 user  staff   1.7G 31 Jul 16:43 yellow_tripdata_2015-0
-rw-r--r--  1 user  staff   1.6G 31 Jul 16:43 yellow_tripdata_2015-0
-rw-r--r--  1 user  staff   1.6G 31 Jul 16:43 yellow_tripdata_2015-0
-rw-r--r--  1 user  staff   1.8G 31 Jul 16:43 yellow_tripdata_2015-1
-rw-r--r--  1 user  staff   1.7G 31 Jul 16:43 yellow_tripdata_2015-1
-rw-r--r--  1 user  staff   1.7G 31 Jul 16:43 yellow_tripdata_2015-1
```

Timing I/O & computation: Pandas

```python
import time, pandas as pd
t_start = time.time();
df = pd.read_csv('yellow_tripdata_2015-01.csv');
t_end = time.time();
print('pd.read_csv(): {} s'.format(t_end-t_start)) # time [s]
```

```
pd.read_csv: 43.820565938949585 s
```

```python
t_start = time.time();
m = df['trip_distance'].mean();
t_end = time.time();
print('.mean(): {} ms'.format((t_end-t_start)*1000)) # time [ms]
```

```
.mean(): 17.752885818481445 ms
```

Timing I/O & computation: Dask

```
import dask.dataframe as dd, time
t_start = time.time();
df = dd.read_csv('yellow_tripdata_2015-*.csv');
t_end = time.time();
print('dd.read_csv: {} ms'.format((t_end-t_start)*1000))  # time [ms
```

```
dd.read_csv: 404.7999382019043 ms
```

```
t_start = time.time();
m = df['trip_distance'].mean();
t_end = time.time();
print('.mean(): {} ms'.format((t_end-t_start)*1000))  # time [ms]
```

```
.mean(): 2.289295196533203 ms
```

Timing I/O & computation: Dask

```
t_start = time.time();
result = m.compute();
t_end = time.time();
print('.compute(): {} min'.format((t_end-t_start)/60)) # time [min]
```

```
.compute(): 3.4004417498906454 min
```

Timing in the IPython shell

```python
m = df['trip_distance'].mean()

%time result = m.compute()
```

```
CPU times: user 9min 50s, sys: 1min 16s, total: 11min 7s
Wall time: 3min 1s
```

Is Dask or Pandas appropriate?

- How big is dataset?
- How much RAM available?
- How many threads/cores/CPUs available?
- Are Pandas computations/formats supported in Dask API?
- Is computation *I/O-bound* (disk-intensive) or *CPU-bound* (processor intensive)?

Best use case for Dask

- Computations from Pandas API available in Dask
- Problem size close to limits of RAM, fits on disk

Let's practice!

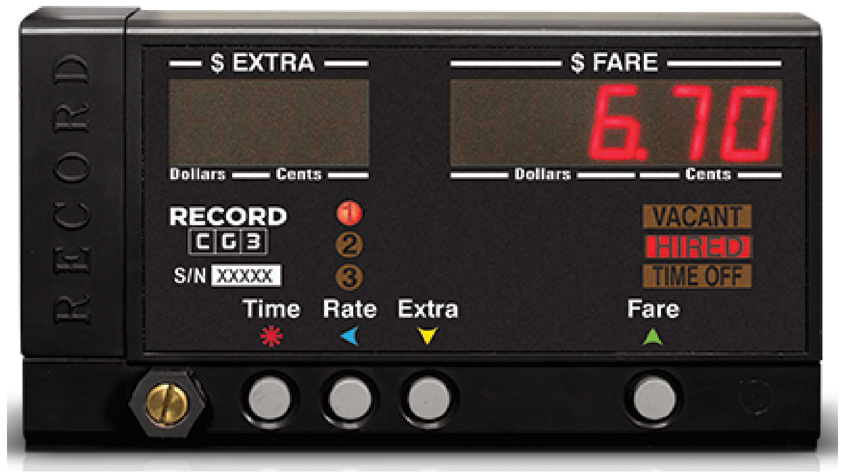
PARALLEL COMPUTING WITH DASK

Analyzing NYC Taxi Rides

PARALLEL COMPUTING WITH DASK

Dhavide Aruliah
Director of Training, Anaconda

The New York taxi dataset



Taxi CSV files

```
%ll -h yellow_tripdata_2015-*.csv
```

```
-rw-r--r--  1 user  staff   1.8G 31 Jul 16:43 yellow_tripdata_2015-01.csv
-rw-r--r--  1 user  staff   1.8G 31 Jul 16:43 yellow_tripdata_2015-02.csv
-rw-r--r--  1 user  staff   1.9G 31 Jul 16:43 yellow_tripdata_2015-03.csv
-rw-r--r--  1 user  staff   1.9G 31 Jul 16:43 yellow_tripdata_2015-04.csv
-rw-r--r--  1 user  staff   1.9G 31 Jul 16:43 yellow_tripdata_2015-05.csv
-rw-r--r--  1 user  staff   1.8G 31 Jul 16:43 yellow_tripdata_2015-06.csv
-rw-r--r--  1 user  staff   1.7G 31 Jul 16:43 yellow_tripdata_2015-07.csv
-rw-r--r--  1 user  staff   1.6G 31 Jul 16:43 yellow_tripdata_2015-08.csv
-rw-r--r--  1 user  staff   1.6G 31 Jul 16:43 yellow_tripdata_2015-09.csv
-rw-r--r--  1 user  staff   1.8G 31 Jul 16:43 yellow_tripdata_2015-10.csv
-rw-r--r--  1 user  staff   1.7G 31 Jul 16:43 yellow_tripdata_2015-11.csv
-rw-r--r--  1 user  staff   1.7G 31 Jul 16:43 yellow_tripdata_2015-12.csv
```

- Exercises use smaller files...

Taxi data features

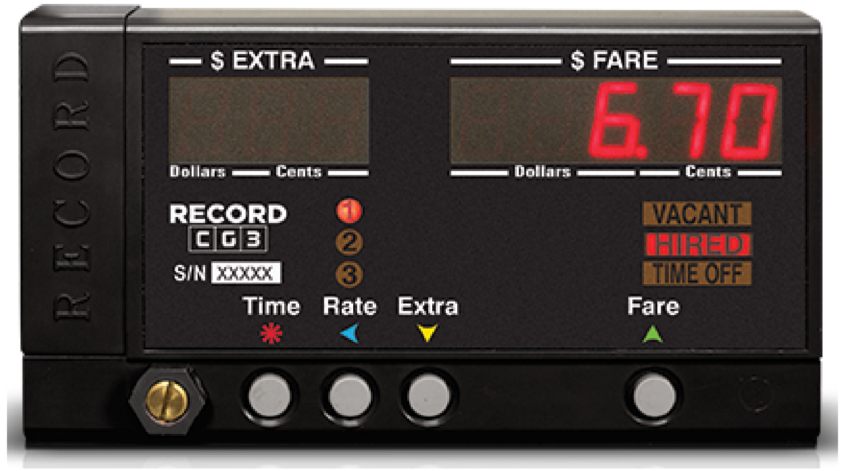
```python
import pandas as pd
df = pd.read_csv('yellow_tripdata_2015-01.csv')
df.shape

df.columns
```

```
(12748986, 19)
Index(['VendorID', 'tpep_pickup_datetime', 'tpep_dropoff_datetime',
       'passenger_count', 'trip_distance', 'pickup_longitude',
       'pickup_latitude', 'RateCodeID', 'store_and_fwd_flag',
       'dropoff_longitude', 'dropoff_latitude', 'payment_type',
       'fare_amount','extra', 'mta_tax', 'tip_amount',
       'tolls_amount','improvement_surcharge', 'total_amount'],
      dtype='object')
```

Amount paid

- How much was each ride?
  - `fare_amount` : cost of ride
  - `tolls_amount` : charges for toll roads
  - `extra` : additional charges
  - `tip_amount` : amount tipped (credit cards only)
  - `total_amount` : total amount paid by passenger



Payment type

```
df['payment_type'].value_counts()
```

```
1    7881388
2    4816992
3      38632
4      11972
5          2
Name: payment_type, dtype: int64
```

Let's practice!

PARALLEL COMPUTING WITH DASK