PARALLEL COMPUTING WITH DASK

Using Dask DataFrames

Dhavide Aruliah
Director of Training, Anaconda

Reading CSV

```
In [1]: import dask.dataframe as dd
```

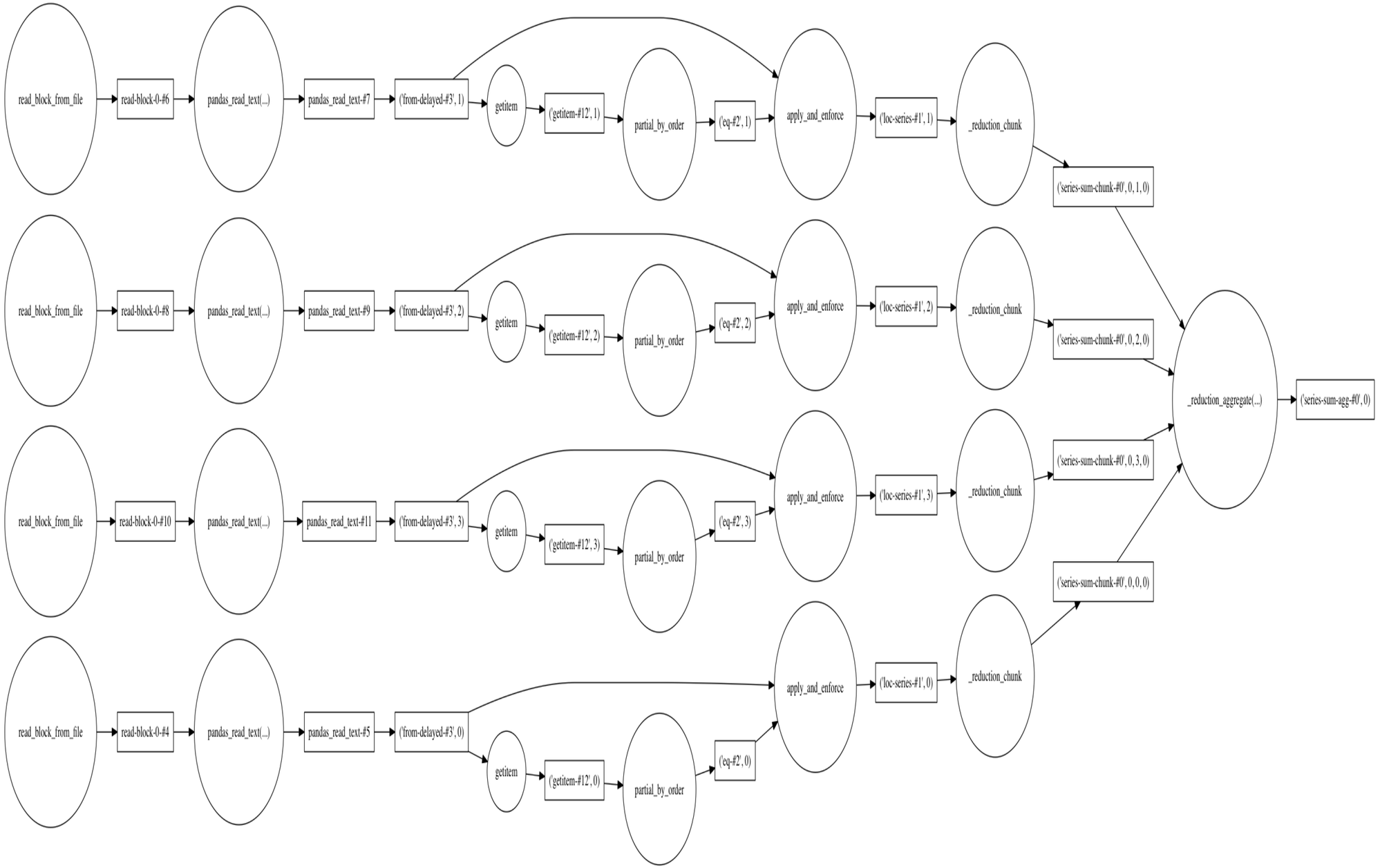
- **`dd.read_csv()`** function
  - Accepts single filename or *glob* pattern (using wildcard `*`)
  - Does not read file immediately (*lazy evaluation*)
  - File(s) need not fit in memory

Reading Multiple CSV Files

```
In [2]: %ls
quarter1.csv  quarter2.csv  quarter3.csv  quarter4.csv

In [3]: transactions = dd.read_csv('*.csv')

In [4]: transactions.head()
Out[4]:
    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

In [5]: transactions.tail()
Out[5]:
      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

```
In [6]: is_wendy = (transactions['names'] == 'Wendy')

In [7]: wendy_amounts = transactions.loc[is_wendy, 'amount']

In [8]: wendy_amounts
Out[8]:
Dask Series Structure:
npartitions=4
None    int64
None      ...
None      ...
None      ...
None      ...
Name: amount, dtype: int64
Dask Name: loc-series, 24 tasks

In [9]: wendy_diff = wendy_amounts.sum()

In [10]: wendy_diff
Out[10]: dd.Scalar<series-..., dtype=int64>

In [11]: wendy_diff.visualize(rankdir='LR')
```


read_block_from_file
read-block-0-#6
pandas_read_text(...)
pandas_read_text-#7
('from-delayed-#3', 1)
getitem
('getitem-#12', 1)
partial_by_order
('eq-#2', 1)
apply_and_enforce
('loc-series-#1', 1)
_reduction_chunk
('series-sum-chunk-#0', 0, 1, 0)
read_block_from_file
read-block-0-#8
pandas_read_text(...)
pandas_read_text-#9
('from-delayed-#3', 2)
getitem
('getitem-#12', 2)
partial_by_order
('eq-#2', 2)
apply_and_enforce
('loc-series-#1', 2)
_reduction_chunk
('series-sum-chunk-#0', 0, 2, 0)
read_block_from_file
read-block-0-#10
pandas_read_text(...)
pandas_read_text-#11
('from-delayed-#3', 3)
getitem
('getitem-#12', 3)
partial_by_order
('eq-#2', 3)
apply_and_enforce
('loc-series-#1', 3)
_reduction_chunk
('series-sum-chunk-#0', 0, 3, 0)
read_block_from_file
read-block-0-#4
pandas_read_text(...)
pandas_read_text-#5
('from-delayed-#3', 0)
getitem
('getitem-#12', 0)
partial_by_order
('eq-#2', 0)
apply_and_enforce
('loc-series-#1', 0)
_reduction_chunk
('series-sum-chunk-#0', 0, 0, 0)
_reduction_aggregate(...)
('series-sum-agg-#0', 0)

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()`

PARALLEL COMPUTING WITH DASK

Let's practice!

PARALLEL COMPUTING WITH DASK

Timing DataFrame Operations

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

```
In [1]: %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
```

Timing I/O & Computation: Pandas

```
In [2]: import time, pandas as pd

In [3]: 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]
Out[3]:
pd.read_csv: 43.820565938949585 s

In [4]: t_start = time.time(); \
   ...: m = df['trip_distance'].mean(); \
   ...: t_end = time.time(); \
   ...: print('.mean(): {} ms'.format((t_end-t_start)*1000)) # time [ms]
Out[4]:
.mean(): 17.752885818481445 ms
```

Timing I/O & Computation: Dask

```
In [5]: import dask.dataframe as dd, time

In [6]: 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]
Out[6]:
dd.read_csv: 404.7999382019043 ms

In [7]: t_start = time.time();\
   ...: m = df['trip_distance'].mean();\
   ...: t_end = time.time();\
   ...: print('.mean(): {} ms'.format((t_end-t_start)*1000))  # time [ms]
Out[7]:
.mean(): 2.289295196533203 ms

In [8]: t_start = time.time(); \
   ...: result = m.compute(); \
   ...: t_end = time.time(); \
   ...: print('.compute(): {} min'.format((t_end-t_start)/60))  # time [min]
Out[8]:
.compute(): 3.4004417498906454 min
```

Timing in the IPython Shell

```
In [9]: m = df['trip_distance'].mean()

In [10]: %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

PARALLEL COMPUTING WITH DASK

Let's practice!

PARALLEL COMPUTING WITH DASK

Analyzing NYC Taxi Rides

Dhavide Aruliah
Director of Training, Anaconda

The New York Taxi Dataset

Taxi CSV Files

```
In [1]: %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

```
In [2]: import pandas as pd

In [3]: df = pd.read_csv('yellow_tripdata_2015-01.csv')

In [4]: df.shape
Out[4]: (12748986, 19)

In [5]: df.columns
Out[5]:
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

```
In [6]: df['payment_type'].value_counts()
Out[6]:
1    7881388
2    4816992
3      38632
4      11972
5          2
Name: payment_type, dtype: int64
```

PARALLEL COMPUTING WITH DASK

Let's practice!