Building Dask Bags & Globbing

PARALLEL COMPUTING WITH DASK

Dhavide Aruliah
Director of Training, Anaconda

Sequences to bags

```
nested_containers = [[0, 1, 2, 3],{},
                     [6.5, 3.14], 'Python',
                     {'version':3}, '' ]

import dask.bag as db

the_bag = db.from_sequence(nested_containers)

the_bag.count()
```

```
6
```

```
the_bag.any(), the_bag.all()
```

```
True, False
```

Reading text files

```python
import dask.bag as db
zen = db.read_text('zen')

taken = zen.take(1)
type(taken)
```

```
tuple
```

Reading text files

```
taken
```

```
('The Zen of Python, by Tim Peters\n',)
```

```
zen.take(3)
```

```
('The Zen of Python, by Tim Peters\n',
 '\n',
 'Beautiful is better than ugly.\n')
```

Glob expressions

```
import dask.dataframe as dd
df = dd.read_csv('taxi/*.csv', assume_missing=True)
```

- `taxi/*.csv` is a *glob expression*
- `taxi/*.csv` matches:

```
taxi/yellow_tripdata_2015-01.csv
taxi/yellow_tripdata_2015-02.csv
taxi/yellow_tripdata_2015-03.csv
...
taxi/yellow_tripdata_2015-10.csv
taxi/yellow_tripdata_2015-11.csv
taxi/yellow_tripdata_2015-12.csv
```

Using Python's glob module

```
%ls
```

```
Alice  Dave  README  a02.txt a04.txt b05.txt b07.txt b09.txt b11.t
Bob    Lisa  a01.txt a03.txt a05.txt b06.txt b08.txt b10.txt taxi
```

```
import glob
txt_files = glob.glob('*.txt')
txt_files
```

```
['a01.txt',
 'a02.txt',
     ...
 'b10.txt',
 'b11.txt']
```

More glob patterns

```
glob.glob('b*.txt')
```

```
['b05.txt',
 'b06.txt',
 'b07.txt',
 'b08.txt',
 'b09.txt',
 'b10.txt',
 'b11.txt']
```

```
glob.glob('b?.txt')
```

```
glob.glob('?0[1-6].txt')
```

```
['a01.txt',
 'a02.txt',
 'a03.txt',
 'a04.txt',
 'a05.txt',
 'b05.txt',
 'b06.txt']
```

```
[]
```

More glob patterns

```
glob.glob('??[1-6].txt')
```

```
['a01.txt',
 'a02.txt',
 'a03.txt',
 'a04.txt',
 'a05.txt',
 'b05.txt',
 'b06.txt',
 'b11.txt']
```

Permissible glob patterns

- Filename characters (e.g., `file-02_tmp.txt`)
- Wildcard character `*` : matches 0 or more
- Wildcard character `?` : matches exactly 1
- Character ranges (e.g., `[0-5]` , `[a-m]` , `[A-Z0-9]`)

Let's practice!

PARALLEL COMPUTING WITH DASK

Functional Approaches using Dask Bags

PARALLEL COMPUTING WITH DASK

Dhavide Aruliah
Director of Training, Anaconda

Functional programming

- Functions: *first-class* data
- **Higher-order functions**:
  - functions as *input* or *output* to functions
- Functions replacing loops with:
  - **map** operations
  - **filter** operations
  - **reduction** operations (or **aggregations**)

Using map

```
def squared(x):
    return x ** 2

squares = map(squared, [1, 2, 3, 4, 5, 6])
squares
```

```
<map at 0x1037a1b70>
```

```
squares = list(squares)
squares
```

```
[1, 4, 9, 16, 25, 36]
```

Using filter

```
def is_even(x):
...:     return x % 2 == 0

evens = filter(is_even, [1, 2, 3, 4, 5, 6])
list(evens)
```

```
[2, 4, 6]
```

```
even_squares = filter(is_even, squares))
list(even_squares)
```

```
[4, 16, 36]
```

Using dask.bag.map

```
import dask.bag as db
numbers = db.from_sequence([1, 2, 3, 4, 5, 6])
squares = numbers.map(squared)
squares
```

```
dask.bag<map-squared, npartitions=6>
```

```
result = squares.compute() # Must fit in memory
result
```

```
[1, 4, 9, 16, 25, 36]
```

Using dask.bag.filter

```
numbers = db.from_sequence([1, 2, 3, 4, 5, 6])
evens = numbers.filter(is_even)

evens.compute()
```

```
[2, 4, 6]
```

```
even_squares = numbers.map(squared).filter(is_even)
even_squares.compute()
```

```
[4, 16, 36]
```

Using .str & string methods

```
zen = db.read_text('zen.txt')
uppercase = zen.str.upper()

uppercase.take(1)
```

```
('THE ZEN OF PYTHON, BY TIM PETERS\n',)
```

```
def my_upper(string):
...:     return string.upper()
my_uppercase = zen.map(my_upper)
my_uppercase.take(1)
```

```
('THE ZEN OF PYTHON, BY TIM PETERS\n',)
```

A bigger example I

```
def load(k):
    template = 'yellow_tripdata_2015-{:02d}.csv'
    return pd.read_csv(template.format(k))
def average(df):
    return df['total_amount'].mean()

def total(df):
    return df['total_amount'].sum()
data = db.from_sequence(range(1, 13)).map(load)
data
```

```
dask.bag<map-loa..., npartitions=12>
```

A bigger example II

```
totals = data.map(total)
averages = data.map(average)
totals.compute()
```

```
[1175217.5200009614,
 947282.0900005419,
 956752.3400005258,
 1304602.4800011297,
 1354966.290001166,
 1251511.6500010253,
 1167936.1000008786,
 915174.880000469,
 994643.300000564,
 1273267.4800010026,
 1158279.990000822,
 1166242.130000856]
```

```
averages.compute()
```

```
[14.75051171665384,
 15.463557844570461,
 15.790076907851297,
 15.971334410669527,
 16.477159899324676,
 16.250654434978838,
 16.163639508987067,
 16.164026987891997,
 16.364647910506154,
 16.544750841370114,
 16.385807916489675,
 16.28056690958003]
```

Reductions (aggregations)

```
t_sum, t_min, t_max, = totals.sum(), totals.min(), totals.max()
t_mean, t_std, = totals.mean(), totals.std()
stats = [t_sum, t_min, t_max, t_mean, t_std]
%time [s.compute() for s in stats]
```

```
CPU times: user 142 ms, sys: 101 ms, total: 243 ms
Wall time: 4.57 s
[13665876.250009943,
 915174.880000469,
 1354966.290001166,
 1138823.0208341617,
 144025.81874405374]
```

Reductions (aggregations)

```
import dask
%time dask.compute(t_sum, t_min, t_max, t_mean, t_std)
```

```
CPU times: user 63.7 ms, sys: 29.1 ms, total: 92.7 ms
Wall time: 852 ms
(13665876.250009943,
 915174.880000469,
 1354966.290001166,
 1138823.0208341617,
 144025.81874405374)
```

Let's practice!

Analyzing Congressional Legislation

PARALLEL COMPUTING WITH DASK

Dhavide Aruliah
Director of Training, Anaconda

JSON data files

- JavaScript Object Notation:
  - stored as plain text
  - common web format
  - direct mapping to Python lists & dictionaries

Sample JSON FIle: items.json

`items.json`

```
[
  {
    "name": "item1",
    "content": ["a","b","c"]
  },
  {
    "name": "item2",
    "content": {"a": 0, "b": 1}
  }
]
```

Using json module

```
import json
with open('items.json') as f:
    items = json.load(f)
type(items)
```

```
list
```

```
items[0]
items[1]
items[1]['content']['b']
```

```
{'content': ['a', 'b', 'c'], 'name': 'item1'}
{'content': {'a': 0, 'b': 1}, 'name': 'item2'}
1
```

JSON Files into Dask Bags

`items-by-line.json`

```
{"name": "item1", "content": ["a", "b", "c"]}
{"name": "item2", "content": {"a": 0, "b": 1}}
```

```
import dask.bag as db
items = db.read_text('items-by-line.json')
items.take(1)  # Note: tuple containing a *string*
```

```
('{"name": "item1", "content": ["a", "b", "c"]}\n',)
```

JSON Files into Dask Bags

```
dict_items = items.map(json.loads) # converts strings -> other data
dict_items.take(2) # Note: tuple containing dicts
```

```
({'content': ['a', 'b', 'c'], 'name': 'item1'},
 {'content': {'a': 0, 'b': 1}, 'name': 'item2'})
```

Plucking values

```
type(dict_items.take(2))
```

```
tuple
```

```
dict_items.take(2)[1]['content'] # Chained indexing
```

```
{'a': 0, 'b': 1}
```

```
dict_items.take(1)[0]['name']     # Chained indexing
```

```
'item1'
```

Plucking values

```
contents = dict_items.pluck('content')
names = dict_items.pluck('name')
contents
names
```

```
dask.bag<pluck-5..., npartitions=1>
dask.bag<pluck-3..., npartitions=1>
```

```
contents.compute()
names.compute()
```

```
[['a', 'b', 'c'], {'a': 0, 'b': 1}]
['item1', 'item2']
```

Congressional legislation metadata

- 23 JSON files
  - metadata about congressional bills
  - up to 1500 pieces of legislation per congress.
- Load *all* into Dask Bag
  - use `current_status` to count vetoed bills
  - use date info to compute average times

Metadata keys

- Selected dictionary keys

```
'bill_type'
'title_without_number'
'related_bills'
'id'
'titles'
'display_number'
'major_actions'
'current_status_description'
'link'
'current_status_date'
'committee_reports'
'current_status_label'
'introduced_date'
'sponsor'
'current_status'
'title'
```

- **Warning:** Not all available for every bill

Let's practice!

PARALLEL COMPUTING WITH DASK