08-DaskDelayed

August 11, 2020

1 Dask

- process data that doesn't fit into memory by breaking it into blocks and specifying task chains
- parallelize execution of tasks across cores and even nodes of a cluster
- move computation to the data rather than the other way around, to minimize communication overheads

http://dask.pydata.org/en/latest/

[1]:
```
import dask
import dask.multiprocessing
```

1.1 Define two slow functions

[2]:
```
from time import sleep

def slowinc(x, delay=1):
    sleep(delay)
    return x + 1

def slowadd(x, y, delay=1):
    sleep(delay)
    return x + y
```

[3]:
```
%%time
x = slowinc(1)
y = slowinc(2)
z = slowadd(x, y)
```

```
CPU times: user 1.52 ms, sys: 195 µs, total: 1.72 ms
Wall time: 3 s
```

1.2 Parallelize with dask.delayed

- Functions wrapped by `dask.delayed` don't run immediately, but instead put those functions and arguments into a task graph.

- The result is computed separately by calling the `.compute()` method.

[4]:
```
from dask import delayed
```

[5]:
```
x = delayed(slowinc)(1)
y = delayed(slowinc)(2)
z = delayed(slowadd)(x, y)
```

[6]:
```
%%time
z.compute()
```

```
CPU times: user 6.61 ms, sys: 362 µs, total: 6.97 ms
Wall time: 2.01 s
```

[6]:
```
5
```

1.3 Dask graph

- Contains description of the calculations necessary to produce the result.
- The z object is a lazy Delayed object. This object holds everything we need to compute the final result. We can compute the result with .compute() as above or we can visualize the task graph for this value with .visualize().

[7]:
```
z.visualize()
```

```
␣
↪---------------------------------------------------------------------------

FileNotFoundError                         Traceback (most recent call␣
↪last)

/usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/graphviz/
↪backend.py in run(cmd, input, capture_output, check, encoding, quiet, **kwargs)
    165     try:
--> 166         proc = subprocess.Popen(cmd, startupinfo=get_startupinfo(),␣
↪**kwargs)
    167     except OSError as e:


/usr/share/miniconda3/envs/big-data/lib/python3.8/subprocess.py in␣
↪__init__(self, args, bufsize, executable, stdin, stdout, stderr, preexec_fn,␣
↪close_fds, shell, cwd, env, universal_newlines, startupinfo, creationflags,␣
↪restore_signals, start_new_session, pass_fds, encoding, errors, text)
    853
--> 854             self._execute_child(args, executable, preexec_fn,␣
↪close_fds,
```

```
      855                                 pass_fds, cwd, env,


      /usr/share/miniconda3/envs/big-data/lib/python3.8/subprocess.py in␣
↪_execute_child(self, args, executable, preexec_fn, close_fds, pass_fds, cwd,␣
↪env, startupinfo, creationflags, shell, p2cread, p2cwrite, c2pread, c2pwrite,␣
↪errread, errwrite, restore_signals, start_new_session)
     1701                         err_msg = os.strerror(errno_num)
  -> 1702                     raise child_exception_type(errno_num, err_msg,␣
↪err_filename)
     1703                 raise child_exception_type(err_msg)


      FileNotFoundError: [Errno 2] No such file or directory: 'dot'


  During handling of the above exception, another exception occurred:


      ExecutableNotFound                          Traceback (most recent call␣
↪last)

      <ipython-input-7-05252b17577d> in <module>
  ----> 1 z.visualize()


      /usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/dask/
↪base.py in visualize(self, filename, format, optimize_graph, **kwargs)
       91         https://docs.dask.org/en/latest/optimize.html
       92         """
  ---> 93         return visualize(
       94             self,
       95             filename=filename,


      /usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/dask/
↪base.py in visualize(*args, **kwargs)
      551         raise NotImplementedError("Unknown value color=%s" % color)
      552
  --> 553     return dot_graph(dsk, filename=filename, **kwargs)
      554
      555


      /usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/dask/dot.
↪py in dot_graph(dsk, filename, format, **kwargs)
      270     """
```

```
    271     g = to_graphviz(dsk, **kwargs)
--> 272     return graphviz_to_file(g, filename, format)
    273
    274


/usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/dask/dot.
↪py in graphviz_to_file(g, filename, format)
    282         format = "png"
    283
--> 284     data = g.pipe(format=format)
    285     if not data:
    286         raise RuntimeError(


/usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/graphviz/
↪files.py in pipe(self, format, renderer, formatter, quiet)
    134         data = text_type(self.source).encode(self._encoding)
    135
--> 136         out = backend.pipe(self._engine, format, data,
    137                            renderer=renderer, formatter=formatter,
    138                            quiet=quiet)


/usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/graphviz/
↪backend.py in pipe(engine, format, data, renderer, formatter, quiet)
    244     """
    245     cmd, _ = command(engine, format, None, renderer, formatter)
--> 246     out, _ = run(cmd, input=data, capture_output=True, check=True,␣
↪quiet=quiet)
    247     return out
    248


/usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/graphviz/
↪backend.py in run(cmd, input, capture_output, check, encoding, quiet, **kwargs)
    167     except OSError as e:
    168         if e.errno == errno.ENOENT:
--> 169             raise ExecutableNotFound(cmd)
    170         else:
    171             raise


ExecutableNotFound: failed to execute ['dot', '-Tpng'], make sure the␣
↪Graphviz executables are on your systems' PATH
```

1.4 Parallelize a loop

[8]:
```
%%time
data = list(range(8))

results = []

for x in data:
    y = slowinc(x)
    results.append(y)

total = sum(results)
total
```

```
CPU times: user 3.4 ms, sys: 480 µs, total: 3.88 ms
Wall time: 8.01 s
```

[8]:
```
36
```

1.4.1 Exercise 8.1

- Parallelize this by appending the delayed `slowinc` calls to the list `results`.
- Display the graph of `total` computation
- Compute time elapsed for the computation.

[9]:
```
from dask import delayed

futures = []

for x in data:
    y = delayed(slowinc)(x)
    futures.append(y)

total = delayed(sum)(futures)
```

[10]:
```
total.visualize()
```

```
␣
↪---------------------------------------------------------------------------

FileNotFoundError                         Traceback (most recent call␣
↪last)

/usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/graphviz/
↪backend.py in run(cmd, input, capture_output, check, encoding, quiet, **kwargs)
```

```
      165     try:
  --> 166         proc = subprocess.Popen(cmd, startupinfo=get_startupinfo(),␣
↪**kwargs)
      167     except OSError as e:


      /usr/share/miniconda3/envs/big-data/lib/python3.8/subprocess.py in␣
↪__init__(self, args, bufsize, executable, stdin, stdout, stderr, preexec_fn,␣
↪close_fds, shell, cwd, env, universal_newlines, startupinfo, creationflags,␣
↪restore_signals, start_new_session, pass_fds, encoding, errors, text)
      853
  --> 854             self._execute_child(args, executable, preexec_fn,␣
↪close_fds,
      855                                 pass_fds, cwd, env,


      /usr/share/miniconda3/envs/big-data/lib/python3.8/subprocess.py in␣
↪_execute_child(self, args, executable, preexec_fn, close_fds, pass_fds, cwd,␣
↪env, startupinfo, creationflags, shell, p2cread, p2cwrite, c2pread, c2pwrite,␣
↪errread, errwrite, restore_signals, start_new_session)
     1701                         err_msg = os.strerror(errno_num)
  -> 1702                     raise child_exception_type(errno_num, err_msg,␣
↪err_filename)
     1703                 raise child_exception_type(err_msg)


      FileNotFoundError: [Errno 2] No such file or directory: 'dot'


  During handling of the above exception, another exception occurred:


      ExecutableNotFound                        Traceback (most recent call␣
↪last)

      <ipython-input-10-d5fa05822eb0> in <module>
  ----> 1 total.visualize()


      /usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/dask/
↪base.py in visualize(self, filename, format, optimize_graph, **kwargs)
       91         https://docs.dask.org/en/latest/optimize.html
       92         """
  ---> 93         return visualize(
       94             self,
       95             filename=filename,
```

```
      /usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/dask/
↪base.py in visualize(*args, **kwargs)
      551         raise NotImplementedError("Unknown value color=%s" % color)
      552
  --> 553     return dot_graph(dsk, filename=filename, **kwargs)
      554
      555


      /usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/dask/dot.
↪py in dot_graph(dsk, filename, format, **kwargs)
      270     """
      271     g = to_graphviz(dsk, **kwargs)
  --> 272     return graphviz_to_file(g, filename, format)
      273
      274


      /usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/dask/dot.
↪py in graphviz_to_file(g, filename, format)
      282         format = "png"
      283
  --> 284     data = g.pipe(format=format)
      285     if not data:
      286         raise RuntimeError(


      /usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/graphviz/
↪files.py in pipe(self, format, renderer, formatter, quiet)
      134         data = text_type(self.source).encode(self._encoding)
      135
  --> 136         out = backend.pipe(self._engine, format, data,
      137                            renderer=renderer, formatter=formatter,
      138                            quiet=quiet)


      /usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/graphviz/
↪backend.py in pipe(engine, format, data, renderer, formatter, quiet)
      244     """
      245     cmd, _ = command(engine, format, None, renderer, formatter)
  --> 246     out, _ = run(cmd, input=data, capture_output=True, check=True,␣
↪quiet=quiet)
      247     return out
      248
```

```
       /usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/graphviz/
↪backend.py in run(cmd, input, capture_output, check, encoding, quiet, **kwargs)
       167     except OSError as e:
       168         if e.errno == errno.ENOENT:
   --> 169             raise ExecutableNotFound(cmd)
       170         else:
       171             raise


       ExecutableNotFound: failed to execute ['dot', '-Tpng'], make sure the␣
↪Graphviz executables are on your systems' PATH
```

[11]:
```
%time total.compute()
```

```
CPU times: user 7.54 ms, sys: 0 ns, total: 7.54 ms
Wall time: 4.01 s
```

[11]:
```
36
```

1.5 Decorator

It is also common to see the delayed function used as a decorator. Same example:

[12]:
```
%%time

@dask.delayed
def slowinc(x, delay=1):
    sleep(delay)
    return x + 1

@dask.delayed
def slowadd(x, y, delay=1):
    sleep(delay)
    return x + y

x = slowinc(1)
y = slowinc(2)
z = slowadd(x, y)
z.compute()
```

```
CPU times: user 4.39 ms, sys: 0 ns, total: 4.39 ms
Wall time: 2 s
```

[12]:
```
5
```

1.6 Control flow

- Delay only some functions, running a few of them immediately. This is helpful when those functions are fast and help us to determine what other slower functions we should call.
- In the example below we iterate through a list of inputs. If that input is even then we want to call `half`. If the input is odd then we want to call `odd_process`. This iseven decision to call `half` or `odd_process` has to be made immediately (not lazily) in order for our graph-building Python code to proceed.

[13]:
```
from random import randint
import dask.delayed

@dask.delayed
def half(x):
    sleep(1)
    return x // 2

@dask.delayed
def odd_process(x):
    sleep(1)
    return 3*x+1

def is_even(x):
    return not x % 2

data = [randint(0,100) for i in range(8)]
data
```

[13]:
```
[52, 94, 61, 91, 54, 14, 17, 68]
```

1.6.1 Exercise 8.2

- Parallelize the sequential code above using dask.delayed
- You will need to delay some functions, but not all
- Visualize and check the computed result

[14]:
```
results = []
for x in data:
    if is_even(x):
        y = half(x)
    else:
        y = odd_process(x)
    results.append(y)

total = delayed(sum)(results)
total.visualize()
```

```
      ␣
↪------------------------------------------------------------------------

       FileNotFoundError                         Traceback (most recent call␣
↪last)

       /usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/graphviz/
↪backend.py in run(cmd, input, capture_output, check, encoding, quiet, **kwargs)
       165     try:
   --> 166         proc = subprocess.Popen(cmd, startupinfo=get_startupinfo(),␣
↪**kwargs)
       167     except OSError as e:


       /usr/share/miniconda3/envs/big-data/lib/python3.8/subprocess.py in␣
↪__init__(self, args, bufsize, executable, stdin, stdout, stderr, preexec_fn,␣
↪close_fds, shell, cwd, env, universal_newlines, startupinfo, creationflags,␣
↪restore_signals, start_new_session, pass_fds, encoding, errors, text)
       853
   --> 854             self._execute_child(args, executable, preexec_fn,␣
↪close_fds,
       855                                 pass_fds, cwd, env,


       /usr/share/miniconda3/envs/big-data/lib/python3.8/subprocess.py in␣
↪_execute_child(self, args, executable, preexec_fn, close_fds, pass_fds, cwd,␣
↪env, startupinfo, creationflags, shell, p2cread, p2cwrite, c2pread, c2pwrite,␣
↪errread, errwrite, restore_signals, start_new_session)
      1701                         err_msg = os.strerror(errno_num)
   -> 1702                     raise child_exception_type(errno_num, err_msg,␣
↪err_filename)
      1703                 raise child_exception_type(err_msg)


       FileNotFoundError: [Errno 2] No such file or directory: 'dot'


   During handling of the above exception, another exception occurred:


       ExecutableNotFound                        Traceback (most recent call␣
↪last)

       <ipython-input-14-5d36a9a9c848> in <module>
         8
         9 total = delayed(sum)(results)
```

```
---> 10 total.visualize()


/usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/dask/
↪base.py in visualize(self, filename, format, optimize_graph, **kwargs)
     91         https://docs.dask.org/en/latest/optimize.html
     92         """
---> 93         return visualize(
     94             self,
     95             filename=filename,


/usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/dask/
↪base.py in visualize(*args, **kwargs)
    551         raise NotImplementedError("Unknown value color=%s" % color)
    552
--> 553     return dot_graph(dsk, filename=filename, **kwargs)
    554
    555


/usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/dask/dot.
↪py in dot_graph(dsk, filename, format, **kwargs)
    270     """
    271     g = to_graphviz(dsk, **kwargs)
--> 272     return graphviz_to_file(g, filename, format)
    273
    274


/usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/dask/dot.
↪py in graphviz_to_file(g, filename, format)
    282         format = "png"
    283
--> 284     data = g.pipe(format=format)
    285     if not data:
    286         raise RuntimeError(


/usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/graphviz/
↪files.py in pipe(self, format, renderer, formatter, quiet)
    134         data = text_type(self.source).encode(self._encoding)
    135
--> 136         out = backend.pipe(self._engine, format, data,
    137                            renderer=renderer, formatter=formatter,
    138                            quiet=quiet)
```

```
       /usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/graphviz/
↪backend.py in pipe(engine, format, data, renderer, formatter, quiet)
       244     """
       245     cmd, _ = command(engine, format, None, renderer, formatter)
   --> 246     out, _ = run(cmd, input=data, capture_output=True, check=True,␣
↪quiet=quiet)
       247     return out
       248


       /usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/graphviz/
↪backend.py in run(cmd, input, capture_output, check, encoding, quiet, **kwargs)
       167     except OSError as e:
       168         if e.errno == errno.ENOENT:
   --> 169             raise ExecutableNotFound(cmd)
       170         else:
       171             raise


       ExecutableNotFound: failed to execute ['dot', '-Tpng'], make sure the␣
↪Graphviz executables are on your systems' PATH
```

1.6.2 Exercise 8.3

- Parallelize the hdf5 conversion from json files
- Create a function `convert_to_hdf`
- Use dask.compute function on delayed calls of the funtion created list
- Is it really faster as expected ?

Hint: Read Delayed Best Practices

[15]:
```python
import os  # library to get directory and file paths
import tarfile # this module makes possible to read and write tar archives

def extract_data(name, where):
    datadir = os.path.join(where,name)
    if not os.path.exists(datadir):
       print("Extracting data...")
       tar_path = os.path.join(where, name+'.tgz')
       with tarfile.open(tar_path, mode='r:gz') as data:
          data.extractall(where)

extract_data('daily-stock','data') # this function call will extract json files
```

[16]:
```
import os, sys
from glob import glob
import pandas as pd
import json

here = os.getcwd() # get the current directory
filenames = sorted(glob(os.path.join(here,'data', 'daily-stock', '*.json')))
```

[17]:
```
def read( fn ):
    with open(fn) as f:
        return [json.loads(line) for line in f]

def convert(data):
    df = pd.DataFrame(data)

    out_filename = fn[:-5] + '.h5'
    df.to_hdf(out_filename, os.path.join(here,'data'))
    return

for fn in filenames:
    data = read( fn)
    convert(data)
```

```
/usr/share/miniconda3/envs/big-data/lib/python3.8/site-
packages/tables/path.py:155: NaturalNameWarning: object name is not a valid
Python identifier: 'big-data'; it does not match the pattern ``^[a-zA-
Z_][a-zA-Z0-9_]*$``; you will not be able to use natural naming to access this
object; using ``getattr()`` will still work, though
  check_attribute_name(name)
```

[18]:
```
%ls data/daily-stock/*.h5
```

```
data/daily-stock/aet.h5   data/daily-stock/bwa.h5  data/daily-stock/jpm.h5
data/daily-stock/afl.h5   data/daily-stock/ge.h5   data/daily-stock/luv.h5
data/daily-stock/aig.h5   data/daily-stock/hal.h5  data/daily-stock/met.h5
data/daily-stock/al.h5    data/daily-stock/hp.h5   data/daily-stock/pcg.h5
data/daily-stock/amgn.h5  data/daily-stock/hpq.h5  data/daily-stock/tgt.h5
data/daily-stock/avy.h5   data/daily-stock/ibm.h5  data/daily-stock/usb.h5
data/daily-stock/b.h5     data/daily-stock/jbl.h5  data/daily-stock/xom.h5
```

[19]:
```
@dask.delayed
def read( fn ):
    " read json file "
    with open(fn) as f:
        return [json.loads(line) for line in f]
```

```
@dask.delayed
def convert(data, fn):
    " convert json file to hdf5 file"
    df = pd.DataFrame(data)
    out_filename = fn[:-5] + '.h5'
    df.to_hdf(out_filename, '/data')
    return fn[:-5]

results = []
for filename in filenames:
    data = read(filename)
    results.append(convert(data, filename))
```

[20]:
```
%time dask.compute(*results)
```

```
CPU times: user 8.54 s, sys: 1.07 s, total: 9.61 s
Wall time: 9.06 s
```

[20]:
```
('/home/runner/work/big-data/big-data/notebooks/data/daily-stock/aet',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/afl',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/aig',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/al',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/amgn',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/avy',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/b',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/bwa',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/ge',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/hal',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/hp',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/hpq',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/ibm',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/jbl',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/jpm',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/luv',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/met',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/pcg',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/tgt',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/usb',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/xom')
```