Cloud computing made easy

in

Joblib

Alexandre Abadie

Outline

An overview of Joblib

Joblib for cloud computing

Future work

Joblib in a word

A Python package to make your algorithms run faster

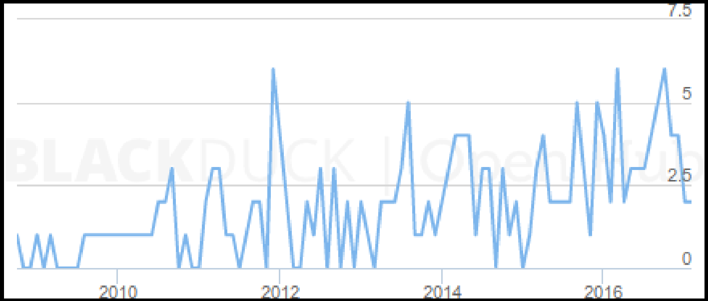
Joblib in a word

A Python package to make your algorithms run faster

http://joblib.readthedocs.io

The ecosystem

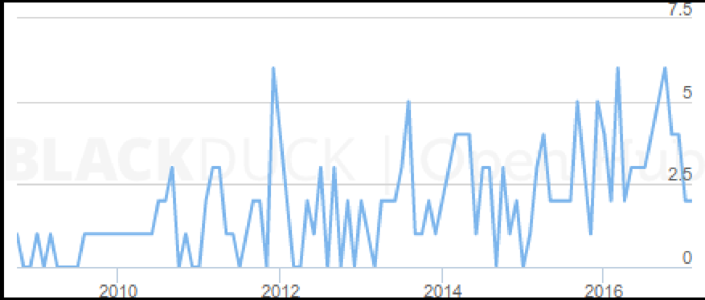
- **54 different contributors** since the beginning in 2008



Contributors per month

The ecosystem

- **54 different contributors** since the beginning in 2008

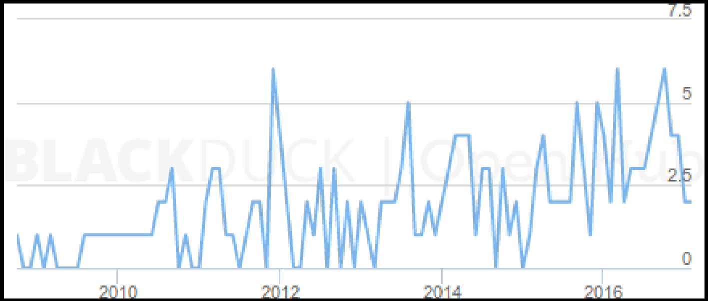


Contributors per month

- Joblib is the computing backend **used by Scikit-Learn**

The ecosystem

- **54 different contributors** since the beginning in 2008



Contributors per month

- Joblib is the computing backend **used by Scikit-Learn**

- **Stable and mature** code base

https://github.com/joblib/joblib

Why Joblib?

Why Joblib?

- Because we want to **make use of all available computing resources**

Why Joblib?

- Because we want to **make use of all available computing resources**

  ⇒ **And ensure algorithms run as fast as possible**

Why Joblib?

- Because we want to **make use of all available computing resources**

  ⇒ **And ensure algorithms run as fast as possible**

- Because we work on **large datasets**

Why Joblib?

- Because we want to **make use of all available computing resources**

  ⇒ **And ensure algorithms run as fast as possible**

- Because we work on **large datasets**

  ⇒ **Data that just fits in RAM**

Why Joblib?

- Because we want to **make use of all available computing resources**

  ⇒ **And ensure algorithms run as fast as possible**

- Because we work on **large datasets**

  ⇒ **Data that just fits in RAM**

- Because we want the **internal algorithm logic** to remain **unchanged**

Why Joblib?

- Because we want to **make use of all available computing resources**

  ⇒ **And ensure algorithms run as fast as possible**

- Because we work on **large datasets**

  ⇒ **Data that just fits in RAM**

- Because we want the **internal algorithm logic** to remain **unchanged**

  ⇒ **Adapted to embarrassingly parallel problems**

Why Joblib?

- Because we want to **make use of all available computing resources**

  ⇒ **And ensure algorithms run as fast as possible**

- Because we work on **large datasets**

  ⇒ **Data that just fits in RAM**

- Because we want the **internal algorithm logic** to remain **unchanged**

  ⇒ **Adapted to embarrassingly parallel problems**

- Because we love **simple APIs**

Why Joblib?

- Because we want to **make use of all available computing resources**

  ⇒ **And ensure algorithms run as fast as possible**

- Because we work on **large datasets**

  ⇒ **Data that just fits in RAM**

- Because we want the **internal algorithm logic** to remain **unchanged**

  ⇒ **Adapted to embarrassingly parallel problems**

- Because we love **simple APIs**

  ⇒ **And parallel programming is not user friendly in general**

How?

- Embarrassingly Parallel computing helper

  ⇒ **make parallel computing easy**

How?

- Embarrassingly Parallel computing helper

  ⇒ **make parallel computing easy**

- Efficient disk caching to avoid recomputation

  ⇒ **computation resource friendly**

How?

- Embarrassingly Parallel computing helper

  ⇒ **make parallel computing easy**

- Efficient disk caching to avoid recomputation

  ⇒ **computation resource friendly**

- Fast I/O persistence

  ⇒ **limit cache access time**

How?

- Embarrassingly Parallel computing helper

  ⇒ **make parallel computing easy**

- Efficient disk caching to avoid recomputation

  ⇒ **computation resource friendly**

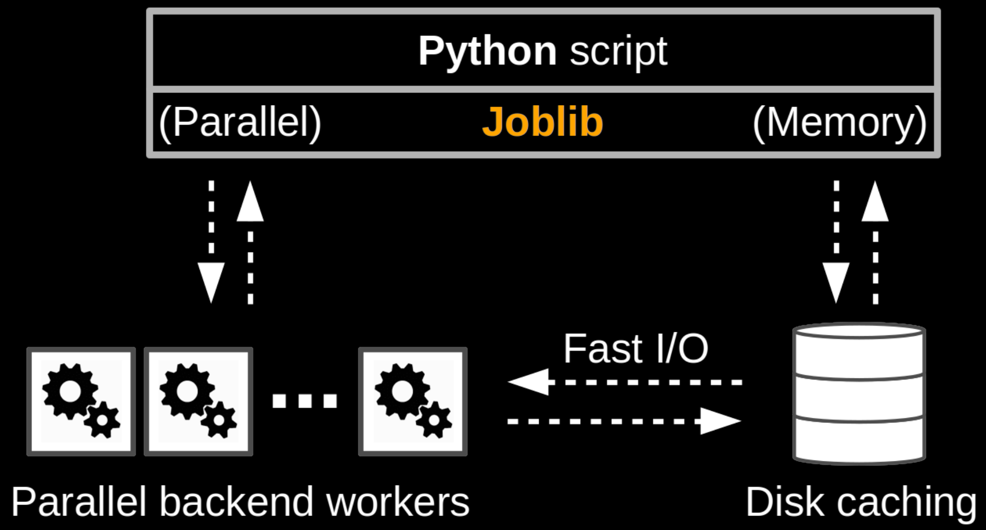
- Fast I/O persistence

  ⇒ **limit cache access time**

- No dependencies, optimized for numpy arrays

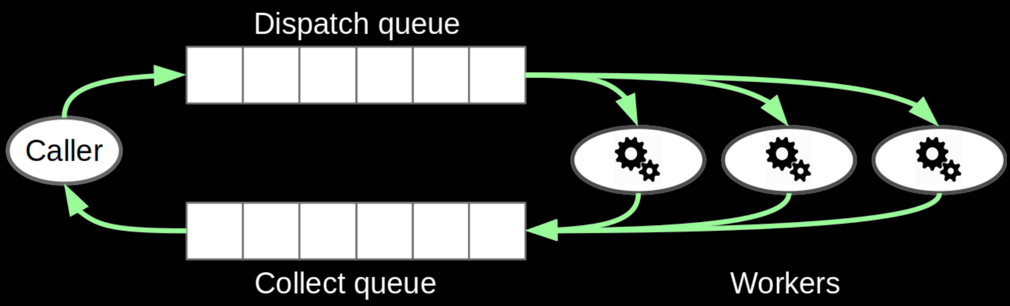
  ⇒ **simple installation and integration in other projects**

Overview


Python script
(Parallel)
Joblib
(Memory)
Fast I/O
Parallel backend workers
Disk caching

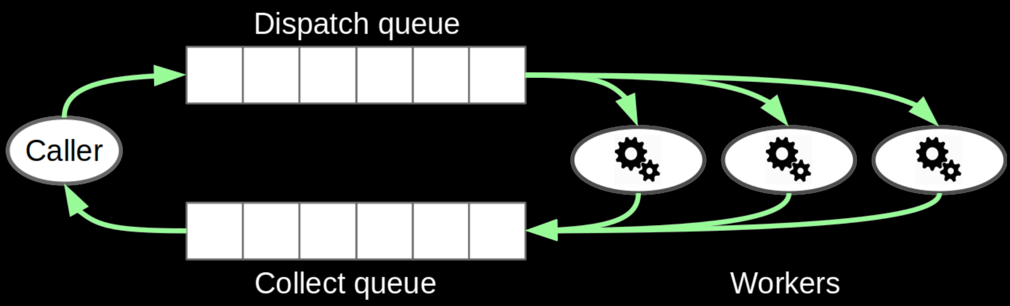
Parallel helper

```
>>> from joblib import Parallel, delayed
>>> from math import sqrt

>>> Parallel(n_jobs=3, verbose=50)(delayed(sqrt)(i**2) for i in range(6))
[Parallel(n_jobs=3)]: Done   1 tasks      | elapsed:    0.0s
[...]
[Parallel(n_jobs=3)]: Done   6 out of   6 | elapsed:    0.0s finished
[0.0, 1.0, 2.0, 3.0, 4.0, 5.0]
```



Parallel helper

```
>>> from joblib import Parallel, delayed
>>> from math import sqrt

>>> Parallel(n_jobs=3, verbose=50)(delayed(sqrt)(i**2) for i in range(6))
[Parallel(n_jobs=3)]: Done   1 tasks      | elapsed:    0.0s
[...]
[Parallel(n_jobs=3)]: Done   6 out of   6 | elapsed:    0.0s finished
[0.0, 1.0, 2.0, 3.0, 4.0, 5.0]
```



⇒ **API can be extended with external backends**

Parallel backends

- Single machine backends: works on a Laptop

  ⇒ **threading**, **multiprocessing** and soon **Loky**

Parallel backends

- Single machine backends: works on a Laptop

  ⇒ **threading**, **multiprocessing** and soon **Loky**

- Multi machine backends: available as optional extensions

  ⇒ **distributed**, **ipyparallel**, **CMFActivity**, **Hadoop Yarn**

Parallel backends

- Single machine backends: works on a Laptop

  ⇒ **threading**, **multiprocessing** and soon **Loky**

- Multi machine backends: available as optional extensions

  ⇒ **distributed**, **ipyparallel**, **CMFActivity**, **Hadoop Yarn**

```
>>> from distributed.joblib import DistributedBackend
>>> from joblib import (Parallel, delayed,
>>>                     register_parallel_backend, parallel_backend)

>>> register_parallel_backend('distributed', DistributedBackend)
>>> with parallel_backend('distributed', scheduler_host='dscheduler:8786'):
>>>     Parallel(n_jobs=3)(delayed(sqrt)(i**2) for i in range(6))
[...]
```

Parallel backends

- Single machine backends: works on a Laptop

  ⇒ **threading**, **multiprocessing** and soon **Loky**

- Multi machine backends: available as optional extensions

  ⇒ **distributed**, **ipyparallel**, **CMFActivity**, **Hadoop Yarn**

```
>>> from distributed.joblib import DistributedBackend
>>> from joblib import (Parallel, delayed,
>>>                     register_parallel_backend, parallel_backend)

>>> register_parallel_backend('distributed', DistributedBackend)
>>> with parallel_backend('distributed', scheduler_host='dscheduler:8786'):
>>>     Parallel(n_jobs=3)(delayed(sqrt)(i**2) for i in range(6))
[...]
```

- Future: new backends for **Celery**, **Spark**

Caching on disk

- Use a **memoize** pattern with the **Memory** object

```
>>> from joblib import Memory
>>> import numpy as np
>>> a = np.vander(np.arange(3)).astype(np.float)

>>> mem = Memory(cachedir='/tmp/joblib')
>>> square = mem.cache(np.square)
```

Caching on disk

- Use a **memoize** pattern with the **Memory** object

```
>>> from joblib import Memory
>>> import numpy as np
>>> a = np.vander(np.arange(3)).astype(np.float)

>>> mem = Memory(cachedir='/tmp/joblib')
>>> square = mem.cache(np.square)


>>> b = square(a)
________________________________________________________________________________
[Memory] Calling square...
square(array([[ 0.,  0.,  1.],
       [ 1.,  1.,  1.],
       [ 4.,  2.,  1.]]))
___________________________________________________________square - 0...s, 0.0min

>>> c = square(a) # no recomputation
array([[ 0.,  0.,  1.],
[...]
```

Caching on disk

- Use a **memoize** pattern with the **Memory** object

```
>>> from joblib import Memory
>>> import numpy as np
>>> a = np.vander(np.arange(3)).astype(np.float)

>>> mem = Memory(cachedir='/tmp/joblib')
>>> square = mem.cache(np.square)


>>> b = square(a)
________________________________________________________________________________
[Memory] Calling square...
square(array([[ 0.,  0.,  1.],
       [ 1.,  1.,  1.],
       [ 4.,  2.,  1.]]))
___________________________________________________________square - 0...s, 0.0min

>>> c = square(a) # no recomputation
array([[ 0.,  0.,  1.],
[...]
```

- **Least Recently Used (LRU)** cache replacement policy

Persistence

- Convert/create **an arbitrary object** into/from a **string of bytes**
- **Streamable persistence** to/from file or socket objects

```
>>> import numpy as np
>>> import joblib
>>> obj = [('a', [1, 2, 3]), ('b', np.arange(10))]
>>> joblib.dump(obj, '/tmp/test.pkl')
['/tmp/test.pkl']
>>> with open('/tmp/test.pkl', 'rb') as f:
>>>     joblib.load(f)
[('a', [1, 2, 3]), ('b', array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]))]
```

Persistence

- Convert/create **an arbitrary object** into/from a **string of bytes**
- **Streamable persistence** to/from file or socket objects

```
>>> import numpy as np
>>> import joblib
>>> obj = [('a', [1, 2, 3]), ('b', np.arange(10))]
>>> joblib.dump(obj, '/tmp/test.pkl')
['/tmp/test.pkl']
>>> with open('/tmp/test.pkl', 'rb') as f:
>>>     joblib.load(f)
[('a', [1, 2, 3]), ('b', array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]))]
```

- Use **compression for fast I/O**:
  support for **zlib, gz, bz2, xz and lzma** compressors

```
>>> joblib.dump(obj, '/tmp/test.pkl.gz', compress=True, cache_size=0)
['/tmp/test.pkl.gz']
>>> joblib.load('/tmp/test.pkl.gz')
```

Outline

Joblib in a word

⇒**Joblib for cloud computing**

Future work

The Cloud trend

- Lots of Cloud providers on the market:

The Cloud trend

- Lots of Cloud providers on the market:
- Existing solutions for processing Big Data:

The Cloud trend

- Lots of Cloud providers on the market:

- Existing solutions for processing Big Data:

- Existing container orchestration solutions: Docker SWARM, Kubernetes

The Cloud trend

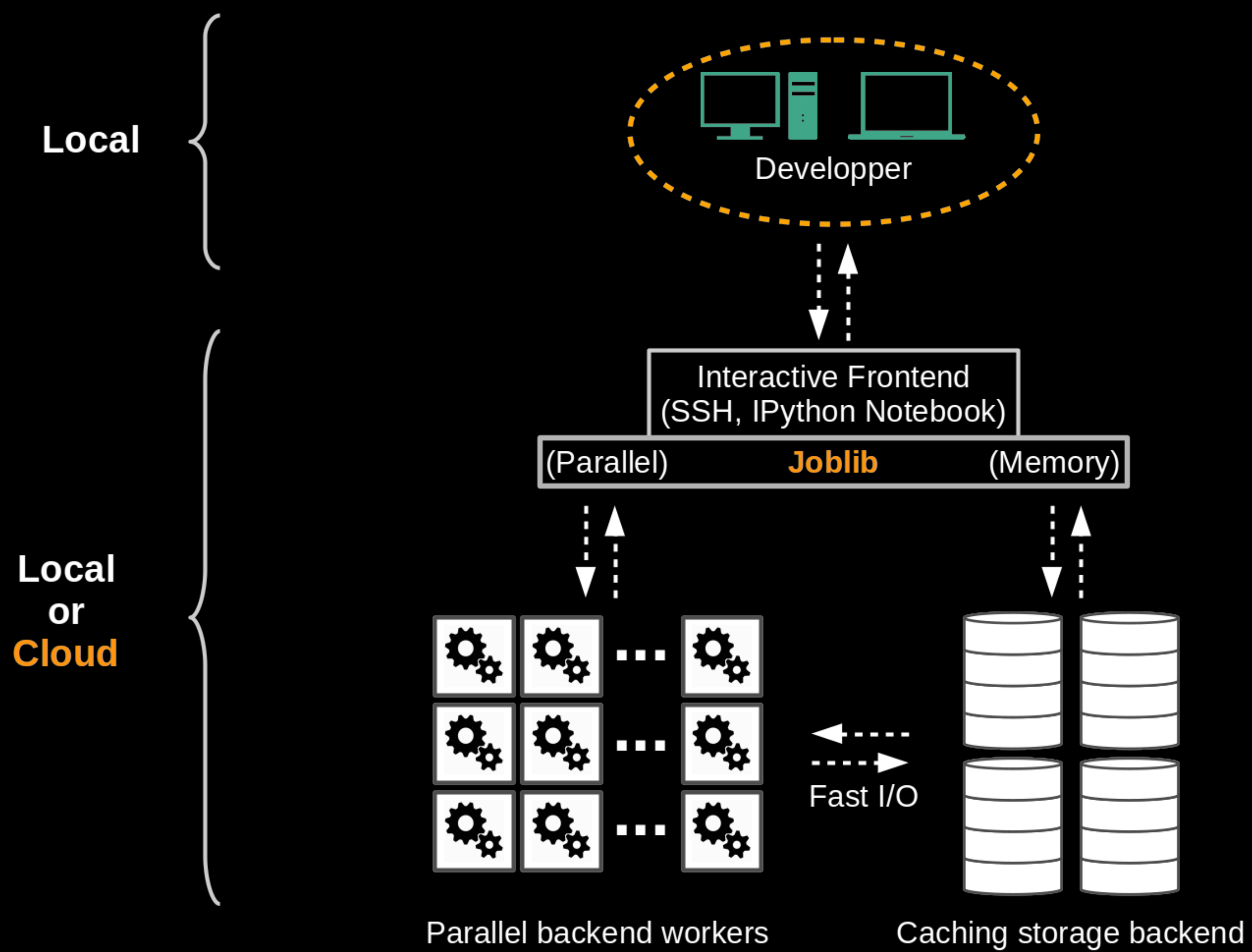
- Lots of Cloud providers on the market:

- Existing solutions for processing Big Data:

- Existing container orchestration solutions: Docker SWARM, Kubernetes

How can Joblib be used with them?

The general idea



Use pluggable multi-machine parallel backends

Principle: configure your backend and wrap the calls to Parallel

```
>>> import time
>>> import ipyparallel as ipp
>>> from ipyparallel.joblib import register as register_joblib
>>> from joblib import parallel_backend, Parallel, delayed

# Setup ipyparallel backend
>>> register_joblib()
>>> dview = ipp.Client()[:]

# Start the job
>>> with parallel_backend("ipyparallel", view=dview):
>>>     Parallel(n_jobs=20, verbose=50)(delayed(time.sleep)(1) for i in range(10))
```

Use pluggable multi-machine parallel backends

Principle: configure your backend and wrap the calls to Parallel

```
>>> import time
>>> import ipyparallel as ipp
>>> from ipyparallel.joblib import register as register_joblib
>>> from joblib import parallel_backend, Parallel, delayed

# Setup ipyparallel backend
>>> register_joblib()
>>> dview = ipp.Client()[:]

# Start the job
>>> with parallel_backend("ipyparallel", view=dview):
>>>     Parallel(n_jobs=20, verbose=50)(delayed(time.sleep)(1) for i in range(10))
```

Complete examples exist for:

- Dask distributed: https://github.com/ogrisel/docker-distributed
- Hadoop Yarn: https://github.com/joblib/joblib-hadoop

Use pluggable store backends

- Extends Memory API with other store providers
- Not available upstream yet:
  ⇒ PR opened at https://github.com/joblib/joblib/pull/397

Use pluggable store backends

- Extends Memory API with other store providers
- Not available upstream yet:
  ⇒ PR opened at https://github.com/joblib/joblib/pull/397

```
>>> import numpy as np
>>> from joblib import Memory
>>> from joblibhadoop.hdfs import register_hdfs_store_backend

# Register HDFS store backend provider
>>> register_hdfs_store_backend()
# Persist data in hdfs://namenode:9000/user/john/cache/joblib
>>> mem = Memory(location='cache', backend='hdfs',
>>>              host='namenode', port=9000, user='john', compress=True)
multiply = mem.cache(np.multiply)
```

Use pluggable store backends

- Extends Memory API with other store providers
- Not available upstream yet:
  ⇒ PR opened at https://github.com/joblib/joblib/pull/397

```
>>> import numpy as np
>>> from joblib import Memory
>>> from joblibhadoop.hdfs import register_hdfs_store_backend

# Register HDFS store backend provider
>>> register_hdfs_store_backend()
# Persist data in hdfs://namenode:9000/user/john/cache/joblib
>>> mem = Memory(location='cache', backend='hdfs',
>>>              host='namenode', port=9000, user='john', compress=True)
multiply = mem.cache(np.multiply)
```

Store backends available:

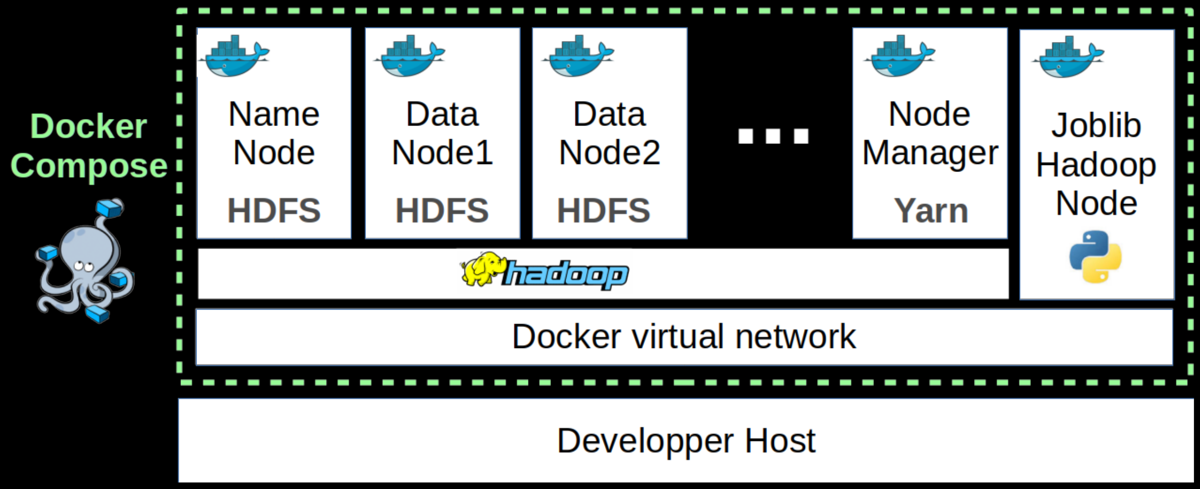
- Amazon S3: https://github.com/aabadie/joblib-s3
- Hadoop HDFS: https://github.com/joblib/joblib-hadoop

Using Hadoop with Joblib

- joblib-hadoop package: https://github.com/joblib/joblib-hadoop

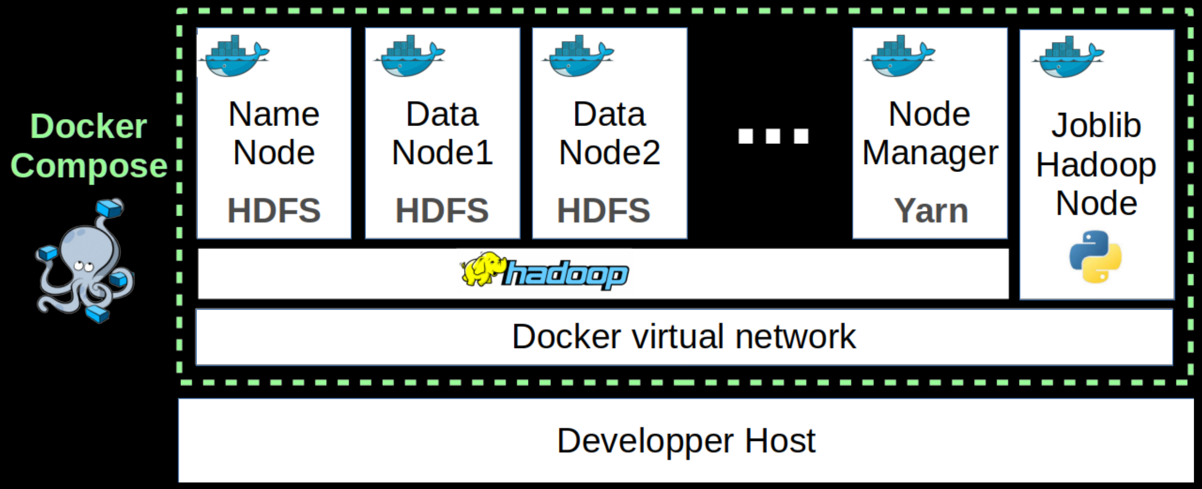
Using Hadoop with Joblib

- joblib-hadoop package: https://github.com/joblib/joblib-hadoop
- Provides **docker containers helpers** for developing and testing



Using Hadoop with Joblib

- joblib-hadoop package: https://github.com/joblib/joblib-hadoop
- Provides **docker containers helpers** for developing and testing



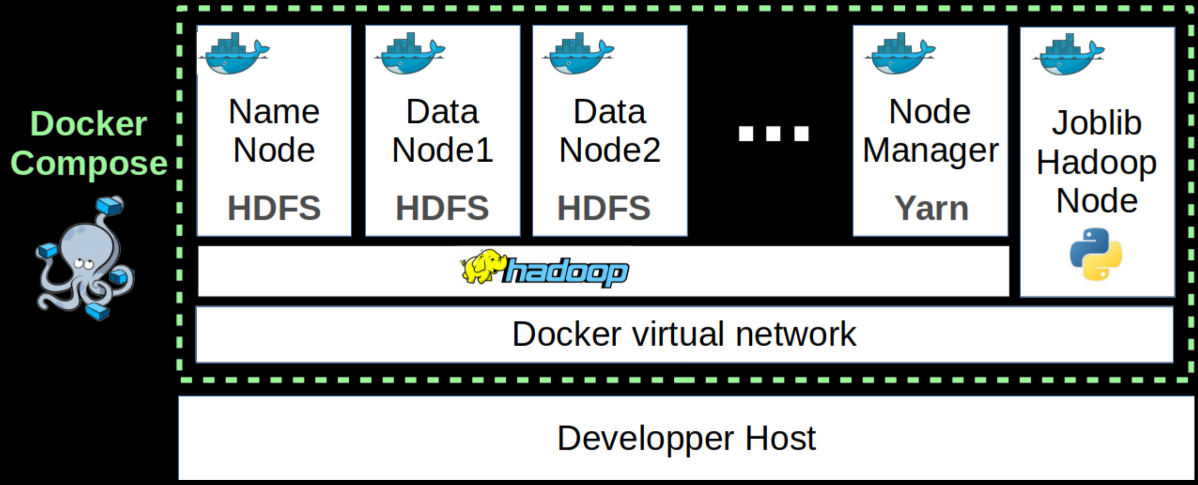
⇒ no need for a production Hadoop cluster

⇒ make developer life easier: **CI on Travis is possible**

⇒ local repository on host is shared with Joblib-hadoop-node container

Using Hadoop with Joblib

- joblib-hadoop package: https://github.com/joblib/joblib-hadoop
- Provides **docker containers helpers** for developing and testing



⇒ no need for a production Hadoop cluster

⇒ make developer life easier: **CI on Travis is possible**

⇒ local repository on host is shared with Joblib-hadoop-node container

Outline

Joblib in a word

Joblib for cloud computing

⇒**Future work and conclusion**

Future work

- In-memory object caching

  ⇒ Should save RAM during a parallel job

Future work

- In-memory object caching

  ⇒ Should save RAM during a parallel job

- Allow *overriding* of parallel backends

  ⇒ See PR: https://github.com/joblib/joblib/pull/524

  ⇒ Seamless distributed computing in scikit-learn

Future work

- In-memory object caching

  ⇒ Should save RAM during a parallel job

- Allow *overriding* of parallel backends

  ⇒ See PR: https://github.com/joblib/joblib/pull/524

  ⇒ Seamless distributed computing in scikit-learn

- Replace multiprocessing parallel backend with **Loky**

  ⇒ See PR: https://github.com/joblib/joblib/pull/516

Future work

- In-memory object caching

  ⇒ Should save RAM during a parallel job

- Allow *overriding* of parallel backends

  ⇒ See PR: https://github.com/joblib/joblib/pull/524

  ⇒ Seamless distributed computing in scikit-learn

- Replace multiprocessing parallel backend with **Loky**

  ⇒ See PR: https://github.com/joblib/joblib/pull/516

- Extend Cloud providers support

  ⇒ Using **Apache libcloud**: give access to a lot more Cloud providers

Conclusion

Conclusion

- Parallel helper is **adapted to embarassingly parallel problems**

Conclusion

- Parallel helper is **adapted to embarassingly parallel problems**

- Already a lot of **parallel backends available**

  ⇒ threading, multiprocessing, loky, CMFActivity distributed, ipyparallel, Yarn

Conclusion

- Parallel helper is **adapted to embarassingly parallel problems**

- Already a lot of **parallel backends available**

  ⇒ threading, multiprocessing, loky, CMFActivity distributed, ipyparallel, Yarn

- Use **caching techniques** to avoid recomputation

Conclusion

- Parallel helper is **adapted to embarassingly parallel problems**

- Already a lot of **parallel backends available**

  ⇒ threading, multiprocessing, loky, CMFActivity distributed, ipyparallel, Yarn

- Use **caching techniques** to avoid recomputation

- Extra **Store backends** available ⇒ **HDFS (Hadoop)** and **AWS S3**

Conclusion

- Parallel helper is **adapted to embarassingly parallel problems**

- Already a lot of **parallel backends available**

  ⇒ threading, multiprocessing, loky, CMFActivity distributed, ipyparallel, Yarn

- Use **caching techniques** to avoid recomputation

- Extra **Store backends** available ⇒ **HDFS (Hadoop)** and **AWS S3**

- Use Joblib either **on your laptop** or **in a Cloud** with **very few code changes**

Thanks!