Xarray: N-D Labeled Arrays and Datasets in Python
Stephan Hoyer (@shoyer)

Originally (2014-2015) developed at

Now, I work at

but this isn’t a Google project.

ECMWF Python Workshop, November 28, 2017
Xarray is part of the scientific Python stack
Why is Python growing so rapidly?

“data science, machine learning and academic research… pandas is the fastest growing Python tag”

stackoverflow.blog/2017/09/14/python-growing-quickly
Pandas makes Python data analysis easy

- data frames!
- labels: indexing & alignment
- groupby: split-apply-combine
- missing data
- time series
- plotting
- scipy/pydata stack
- but not N-dimensional
xarray.Dataset: netCDF meets pandas.DataFrame

Data variables
used for computation

Coordinates
describe data

Indexes
align data

Attributes
metadata ignored by operations

temperature
pressure
elevation
land_cover

latitude
longitude
time
Design goals for xarray

“pandas for N-dimensional arrays”

- build on pandas + NumPy (and now dask)
- copy the pandas API
- use the netCDF data model

Motivated by weather & climate use cases

...but domain agnostic
Xarray operations use names, not numbers

```python
# xarray style
>>> ds.sel(time='2017-11-28').max(dim='station')

# numpy style
>>> array[[0, 1, 2, 3], :, :].max(axis=2)
```
Every operation in xarray is parallelized with Dask

Dask adds two major features to NumPy:

- **Parallelized**: use all your cores
- **Out-of-core**: streaming operations

Dask scales up (to a cluster) and down (to a single machine).

To use Dask in xarray, users specify chunks or call `open_mfdataset()`.
Xarray + Dask makes scalable data analysis easy

```python
import xarray

ds = xarray.open_mfdataset('all/your/data/*.nc')
climatology = ds.groupby('time.season').mean('time')
temperature_range = abs(
    climatology.air.sel(season='JJA')
    - climatology.air.sel(season='DJF'))
temperature_range.plot()
```

...but also easily interoperates with the scientific Python stack
Use `xarray.apply_ufunc` to wrap code for `xarray`

Handles all the boilerplate involved in wrapping a NumPy function.

Example usage:

```python
def spearman_correlation(x, y, dim):
    return xarray.apply_ufunc(
        spearman_correlation_gufunc, x, y,
        input_core_dims=[[dim], [dim]],
        dask='parallelized',
        output_dtypes=[float])
```

- Function that supports NumPy style broadcasting
- Core dimensions over which the computation takes place
- Automatic parallelization with dask!

`New in xarray v0.10.0`
Current data type support in xarray is not enough

Two possible solutions:

- NumPy duck arrays: \_\_array\_ufunc\_\_ (and \_\_array\_concatenate\_\_)?
- Custom NumPy dtypes

<table>
<thead>
<tr>
<th>Physical Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>52.8 ft/s</td>
</tr>
<tr>
<td>= 36 mi/h</td>
</tr>
</tbody>
</table>
Pangeo Data: a community effort for big data geoscience

Domain specific packages building on xarray + dask:

- Data Discovery
- Regions and Shapes
- Regridding
- Signal Processing
- Thermodynamics
- Vector Calculus

pangeo-data.github.io
Xarray is a community project: join us!

Funded by Pangeo

Stephan Hoyer  Joe Hamman  Ryan Abernathy  Matthew Rocklin  Fabien Maussion
Benoit Bovy  Clark Fitzgerald  Maximilian Roos  Keisuke Fujii

+ 74 other contributors!

Not geoscience users!
Backup slides
Example: vectorizing by dimension name

Try vectorized indexing! (new in xarray v0.10.0)
Extending xarray with domain specific logic

(1) Composition

```python
class MyData:
    def __init__(self):
        self.ds = xr.Dataset()
    ...
    def __getitem__(self, ...):
        ...
    def __add__(self, ...):
    def __radd__(self, ...):
        ...
```

Too much work!

(2) Inheritance

```python
class MyDataset(xr.Dataset):
    def _merge(self, ...):
        super()._merge(...)
```

Too fragile!

(3) Custom accessors

```python
@xarray.register_dataset_accessor('my')
class My:
    ...
    ds = xarray.Dataset()
    ds.my.custom_method()
```

Just right?