Analyzing MOM6 with a python/xarray/dask/zarr software stack

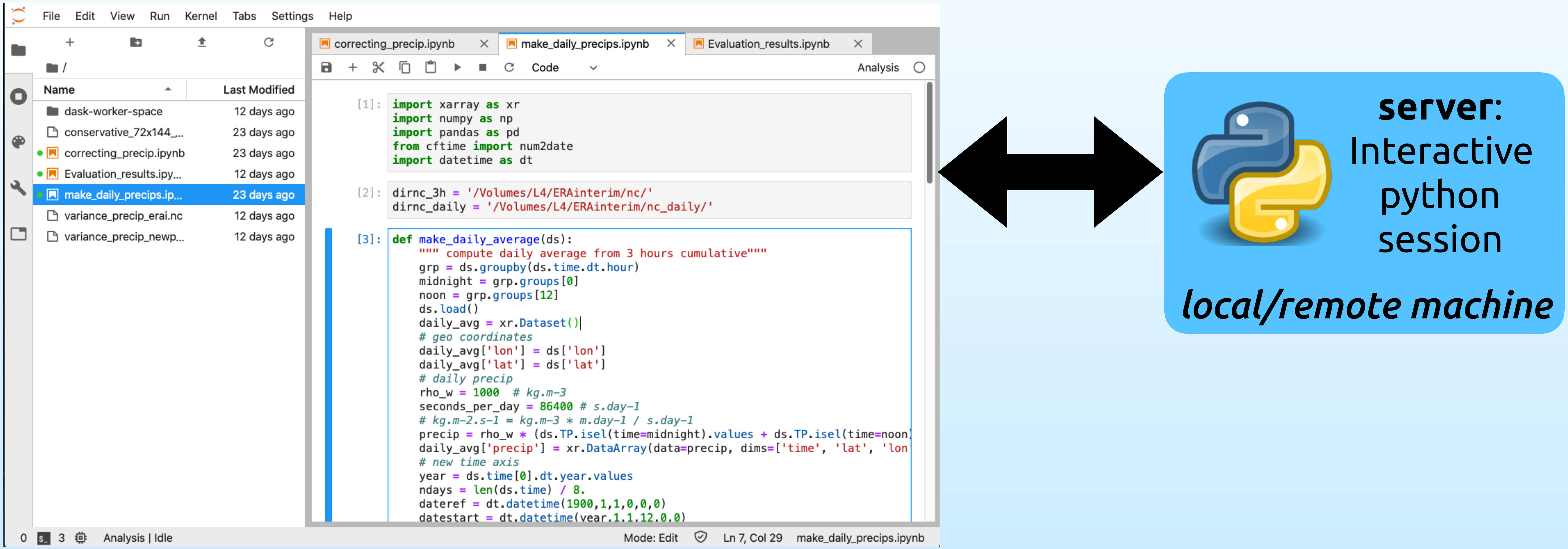
R. Dussin

CESM OWG May 11th 2020

Outline

- IDE: Jupyter and extensions
- Compute: Xarray and Dask
- Storage: Zarr
- Useful links
- demo with MOM6

Jupyter ecosystem



client: local web browser

Jupyterlab = more functional IDE

Jupyterhub = multi-user Jupyter server

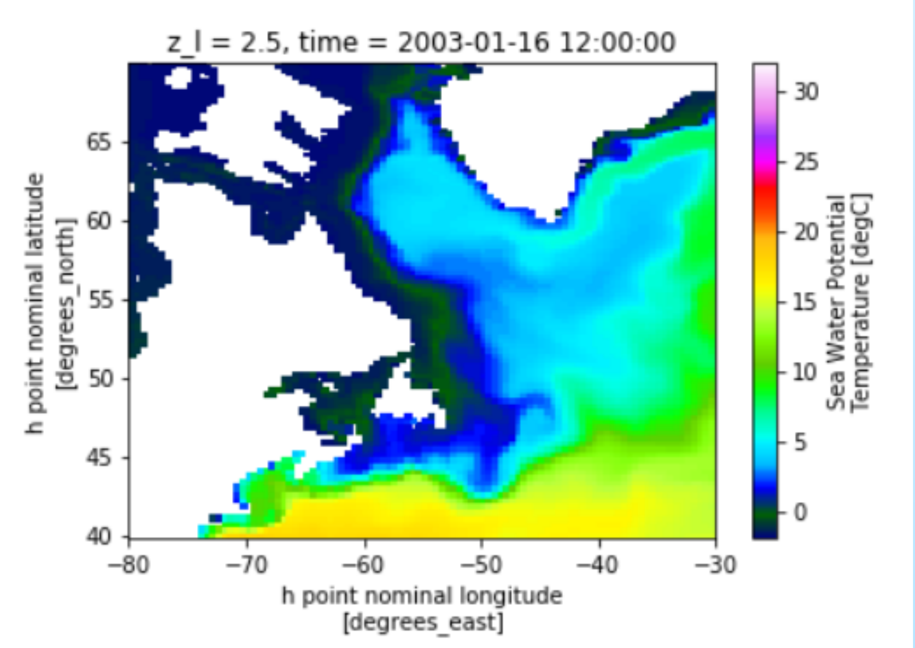
e.g. jupyterhub.ucar.edu, ocean.pangeo.io

xarray: “label-aware” arrays

- Philosophically similar to netcdf data model
- Dataset = set of DataArrays
- Datasets can be build from N files
- DataArrays have labelled dimensions/coords
- We can use methods working on these labels

```
ds['thetao'].sel(xh=slice(-80,-30), yh=slice(40,70),
                 z_l=2.5, time='2003-01').plot(vmin=-2, vmax=32,
                                               cmap='gist_ncar')
```

```
clim = ds.mean(dim='time')
```



- xgcm: adds staggered grid awareness to xarray

dask: lazy, parallel and OOC

- xarray runs either numpy or dask under the hood
- if chunks are specified, then dask is the backend
- dask operates in lazy mode, numpy in eager mode
- dask build graph of operations, delays execution
- dask only executes when data is requested (plot,...)
- execution is multi-threaded on cluster (local, k8s, jobqueue)
- can handle dataset size larger than memory (OOC)

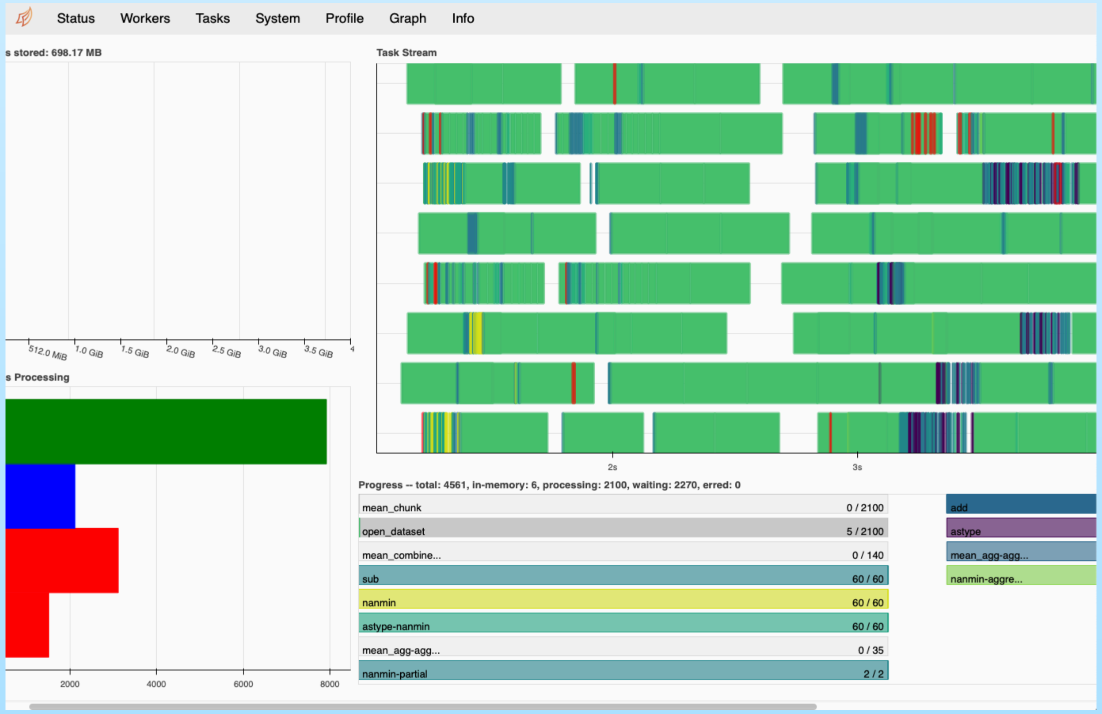
```python
from dask.distributed import Client, LocalCluster
cluster = LocalCluster()
client = Client(cluster)
client
```

Client

Scheduler: tcp://127.0.0.1:63195
Dashboard: http://127.0.0.1:63196/status

Cluster

Workers: 4
Cores: 8
Memory: 17.18 GB



Zarr: optimized cloud storage

Why Bother with a new format?

```
dmdu -sh my_OM4p125_run/*

6.8T    history
9.4T    pp
1.8T    restart
2.7T    zstore
```

- zarr have BLOSC compression
- designed for cloud object storage
- chunk size matters (10-100 Mo)
- stores can be of different types (zip/directory/…)

```
temp_tendency
├── temp_tendency/0.0.0.0
├── temp_tendency/0.1.0.0
├── temp_tendency/0.2.0.0
├── temp_tendency/0.3.0.0
├── temp_tendency/0.4.0.0
├── temp_tendency/0.5.0.0
├── temp_tendency/0.6.0.0
├── temp_tendency/0.7.0.0
├── temp_tendency/0.8.0.0
├── temp_tendency/0.9.0.0
├── temp_tendency/1.0.0.0
├── temp_tendency/10.0.0.0
```

```
├── ./tosga
│   └── ./tosga/gn
│       └── ./tosga/gn/v1
│           ├── ./tosga/gn/v1/tosga.yml
│           └── ./tosga/gn/v1/tosga.zip
├── ./umo
│   └── ./umo/gn_d2
│       └── ./umo/gn_d2/v1
│           ├── ./umo/gn_d2/v1/umo.yml
│           └── ./umo/gn_d2/v1/umo.zip
├── ./uo
│   └── ./uo/gn_d2
│       └── ./uo/gn_d2/v1
│           ├── ./uo/gn_d2/v1/uo.yml
│           └── ./uo/gn_d2/v1/uo.zip
```

Zarr: optimized cloud storage

zarr ZipStore vs DirectoryStore

1. in DirectoryStore, 1 chunk = 1 file. For 3d monthly variable (60 yr run), this amounts to a lot. ZipStore = 1 file!!!

```
directory_store/./thetao/.zmetadata
(base) PPAN: Raphael.Dussin@an104  perf_tests: find directory_store/. -type f | wc -l
25697
(base) PPAN: Raphael.Dussin@an104  perf_tests: find zipstore/. -type f | wc -l
1
```

2. Similar performance using dask cluster:

```
[4]: rootdir = '/work/Raphael.Dussin/zarr_stores/perf_tests/'

     zds = xr.open_zarr(f'{rootdir}/zipstore/thetao.zip', consolidated=True)
     dds = xr.open_zarr(f'{rootdir}/directory_store/thetao', consolidated=True)
```

```
[44]: zm = zds['thetao'].mean(dim='time')

[45]: %%time
      zm.load()

CPU times: user 3min 31s, sys: 10.1 s, total: 3min 41s
Wall time: 11min 51s
```

```
[46]: dm = dds['thetao'].mean(dim='time')

[47]: %%time
      dm.load()

CPU times: user 3min 43s, sys: 11.2 s, total: 3min 54s
Wall time: 13min 45s
```

3. ZipStore not as commonly used as DirectoryStore hence some bugs found along the way (and fixed)

Useful doc for MOM6

https://mom6-analysiscookbook.rtfd.io

MOM6-AnalysisCookbook

latest

Search docs

CONTENTS:

Docs » Cookbook

Edit on GitHub

Cookbook

Here are recipes for doing some xarray-based analysis with MOM6.

- Setting up a DASK cluster using dask-jobqueue
  - Your DASK cluster at work
- Setting up a DASK cluster on your local machine
  - Sample computation:
- Getting started with MOM6
  - grid variables
  - building a xgcm grid object
  - A note on geographical coordinates
  - Plotting
- Time-based operations
  - 1. Computing climatologies for SST
  - 2. Selecting based on dates
- Spatial Operations
  - 2D horizontal averaging
  - Zonal average
  - 3D average
  - Using xgcm
- Vorticity-based diagnostics
  - Relative vorticity
  - Potential vorticity $(\zeta + f)/h$
- Computations for Potential density, buoyancy and geostrophic shear
  - Potential density
  - Buoyancy
  - Geostrophic shear
- Horizontal Remapping
  - Remapping model output to a 1x1 degree grid
  - Remapping onto the model grid
- Creating nice maps with xarray
  - Polar projections
- Comparing MOM6 data to hydrographic section

Useful doc for MOM6

https://xgcm.rtfd.io

XGCM 0.3 Site ▾ Page ▾ « MITgcm ECCOv4 Example Generate miss... »

GETTING STARTED WITH XGCM FOR MOM6

- MOM6 variables are staggered according to the Arakawa C-grid
- It uses a north-east index convention
- center points are labelled (xh, yh) and corner points are labelled (xq, yq)
- important: variables xh/yh, xq/yq that are named “nominal” longitude/latitude **are not** the true geographical coordinates and are not suitable for plotting (more later)

See indexing for details.

[1]:
```
import xarray as xr
from xgcm import Grid
import warnings
import matplotlib.pylab as plt
from cartopy import crs as ccrs
import numpy as np
```

[2]:
```
%matplotlib inline
warnings.filterwarnings("ignore")
```

For this tutorial, we are going to use sample data for the $\frac{1}{2}^{\circ}$ global model OM4p05 hosted on a GFDL thredds server:

[3]:
```
dataurl = 'http://35.188.34.63:8080/thredds/dodsC/OM4p5/'

ds = xr.open_dataset(f'{dataurl}/ocean_monthly_z.200301-200712.nc4',
                     chunks={'time':1, 'z_l': 1}, drop_variables=['average_DT',
                                                                  'average_T1',
                                                                  'average_T2'],
                     engine='pydap')
```

[4]:
```
ds
```

[4]:
```
<xarray.Dataset>
Dimensions:       (nv: 2, time: 60, xh: 720, xq: 720, yh: 576, yq: 576, z_i: 36, z_l
Coordinates:
  * nv            (nv) float64 1.0 2.0
  * xh            (xh) float64 -299.8 -299.2 -298.8 -298.2 ... 58.75 59.25 59.75
  * xq            (xq) float64 -299.5 -299.0 -298.5 -298.0 ... 59.0 59.5 60.0
```

Demo time

notebook available at

https://github.com/raphaeldussin/presentations/blob/master/MOM6_Webinar_May11_2020/demo_overflows.ipynb

Concluding remarks

- Jupyter:
  - same user experience on every platform
  - easy to prototype analysis
  - deployment in production workflow with *papermill*

- Xarray:
  - easily write high level diagnostics
  - xgcm adds model staggered grid awareness

- Dask:
  - easily run performant parallel computations
  - chunking matter!!

- Zarr:
  - great compression
  - choice of chunking is also important

- Importance of community software dev:
  - no “one size fits all” monolithic package
  - contribute to existing packages