CDO's python bindings

Ralf Müller

MPI Met Hamburg

October 17, 2018

Max-Planck-Institut
für Meteorologie

What to expect

Overview:

- General features
- Installation
- What it's *not*

Source Code Examples:

- Basics Usage
- Work with temporary files
- Parallelization with Python
- Integration with numpy/xarray/...

Max-Planck-Institut
für Meteorologie

WHAT ...

... is offered

- integration of CDO into python/ruby scripts like a native library

Max-Planck-Institut
für Meteorologie

WHAT ...

... is offered

- integration of CDO into python/ruby scripts like a native library
- keeps CDOs main feature: **operator chaining**

Max-Planck-Institut
für Meteorologie

WHAT ...

... is offered

- integration of CDO into python/ruby scripts like a native library
- keeps CDOs main feature: **operator chaining**
- multiple types of return values:
  - output files, numpy arrays, masked arrays, XArray
  - netCDF4 or XDataset handles
  - strings for operators, which write to stdout
  - None on error (optional)

Max-Planck-Institut
für Meteorologie

WHAT ...

... is offered

- integration of CDO into python/ruby scripts like a native library
- keeps CDOs main feature: **operator chaining**
- multiple types of return values:
  - output files, numpy arrays, masked arrays, XArray
  - netCDF4 or XDataset handles
  - strings for operators, which write to stdout
  - None on error (optional)
- access to all options
  - -f *file format*
  - -P *OpenMP-threads*
  - ...
- environment settings
- GPL-2 licensed like CDO itself

Max-Planck-Institut
für Meteorologie

HOW ...

... to get it

- prebuild debian packages: python-cdo, python3-cdo
- installation via pip or conda (conda-forge)
- or spack (https://spack.io)

Max-Planck-Institut
für Meteorologie

HOW ...

... to get it

- prebuild debian packages: python-cdo, python3-cdo
- installation via pip or conda (conda-forge)
- or spack (https://spack.io)

... to work with it

- IO: provide automatic tempfile handling
- IO: optional use of existing files if present
- interactive help
- use different CDO binaries for different tasks

Max-Planck-Institut
für Meteorologie

WTH ... internals

cdo.{rb,py}

- is a *smart* caller of a CDO binary (with all the pros and cons)
- doesn't need to be re-installed for a new CDO version
- isn't a shared library, which keeps everything in memory
- doesn't allow write access to files via the numpy or masked arrays

See MPI-MET ort github page:

https://code.zmaw.de/projects/cdo/wiki/Cdo{rbpy}
https://github.com/Try2Code/cdo-bindings

Max-Planck-Institut
für Meteorologie

Basic Python 2.7/3.x

```
1   from cdo import Cdo
2   import glob
3
4   cdo = Cdo()
5
6   # use a special binary
7   cdo.setCdo('/sw/rhel6-x64/cdo/cdo-1.9.5-gcc64/bin/cdo')
8
9   # concatenate list of files into a temp file with relative time axis
10  ofile = cdo.cat(input = glob.glob('*.nc'), options = '-r')
11
12  # vertical interpolation
13  Temp3d = cdo.intlevel(100,200,500,1000, options = '-f grb',
14                        input  = ofile,
15                        output = 'TempOnTargetLevels.grb')
16
17  # perform zonal mean after interpolation in nc4 classic format with 8 OpenMP threads
18  zonmeanFile =  cdo.zonmean(input = "-remapbil,r1400x720 %s"%(Temp3d),
19                             options = '-P 8 -f nc4c')
```

Max-Planck-Institut
für Meteorologie

Possible issues with tempfiles

Using tempfiles can become a problem

Tempfiles are usually removed at the end of a script. But in long-lasting or SIGKILLed interactive session (ipython/jupyter notebooks) with possibly many users per node the system tempdir can get filled up sooner or later.
How to avoid a reboot?

Max-Planck-Institut
für Meteorologie

Possible issues with tempfiles

Using tempfiles can become a problem

Tempfiles are usually removed at the end of a script. But in long-lasting or SIGKILLed interactive session (ipython/jupyter notebooks) with possibly many users per node the system tempdir can get filled up sooner or later.
How to avoid a reboot?

Solution

Manual clean-up for all files created by cdo.py belonging to the current user

```
cdo.cleanTempDir()
```

Max-Planck-Institut für Meteorologie

Possible issues with tempfiles

Using tempfiles can become a problem

Tempfiles are usually removed at the end of a script. But in long-lasting or SIGKILLed interactive session (ipython/jupyter notebooks) with possibly many users per node the system tempdir can get filled up sooner or later.
How to avoid a reboot?

Solution

Manual clean-up for all files created by cdo.py belonging to the current user

```
cdo.cleanTempDir()
```

Solution

Use other tempdir like `/dev/shm`

```
cdo = Cdo(tempdir='/dev/shm/{0}'.format(os.environ['USER']))
```

Max-Planck-Institut
für Meteorologie

Parallelism with Python

```
1   from cdo import Cdo
2   from multiprocessing import Pool
3
4   # define methods to use with the Pool
5   def cdozonmean(infile):
6       ofile = cdo.zonmean(input=infile)
7
8   files = sorted([s for s in glob.glob(nicam_path+'*/sa_tppn.nc')])[0:20]
9
10  # create the Pool and a dict for collecting the results
11  pool, results    = Pool(4), dict()
12
13  # fill and run the Pool, keep the connection of input and output
14  for file in files:
15      results[file] = pool.apply_async(cdozonmean,(file,))
16  pool.close()
17  pool.join()
18
19  # retrieve the _real_ results from the Pool (i.e. filenames)
20  for k,v in results.items():
21      results[k] = v.get()
22
23  cdo.cat(input = [results[x] for x in files],output = wrk_dir+'test.nc')
```

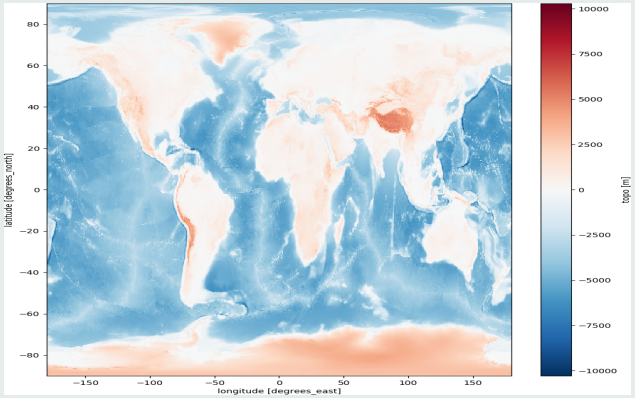
Max-Planck-Institut
für Meteorologie

XArray/Numpy interaction

XArray

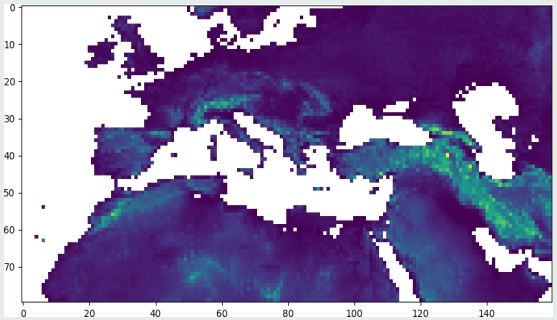
```
# plotting with XArray
cdo.topo(returnXArray='topo').plot()

# IO with XDataset
dataSet = xarray.open_dataset(cdo.topo('global_0.1',
                                       options = '-f nc'))
dataSet['topo'] = 1.0 + np.abs(dataSet['topo'])
cdo.fldmin(input=dataSet,returnArray='topo').min() == ?
```



numpy/matplotlib-based plotting

```
# or with masked arrays
from matplotlib import pylab
import numpy
oro = cdo.setrtomiss(-20000,0,
                     input='-sellonlatbox,-20,60,20,60  -topo',
                     returnMaArray='topo')
pylab.imshow(numpy.flipud(oro))
pylab.show()
```



für Meteorologie

More Examples at github

Units test for all features available at Github

- numpy or masked arrays, XArray, XDataset, cdf handles ...

key	value	return type
returnArray	varname	numpy array
returnMaArray	varname	numpy masked array
returnXarray	varname	XArray
returnXDataset	Bool	XDataset handle
returnCdf	Bool	netCDF4 file handle

- conditional output
- return None on error
- exception handling
- output operators

... test code is about 1.5 times the library code

Max-Planck-Institut
für Meteorologie

thank you for your attention

???

Max-Planck-Institut
für Meteorologie

Appendix: Constructor

```
1     def __init__(self,
2                  returnCdf          = False,                 # always return netCDF4 filehandle
3                  returnNoneOnError  = False,                 # don't raise exception, return No
4                  forceOutput        = True,                  # global switch for cond. output
5                  cdfMod             = CDF_MOD_NETCDF4,       # set the cdf module to by used
6                  env                = os.environ,            # environment for the object
7                  debug              = False,                 # print commands, return codes, et
8                  tempdir            = tempfile.gettempdir(), # location for temporary files
9                  logging            = False,                 # log commands internally
10                 logFile            = StringIO()):
11
12        # read path to CDO from the environment if given
13        if 'CDO' in os.environ:
14          self.CDO = os.environ['CDO']
15        else:
16          self.CDO = 'cdo'
```

Max-Planck-Institut
für Meteorologie

Appendix: Pool.apply_async syntax explained

```
1   from multiprocessing import Pool
2
3   def f(x, *args, **kwargs):
4       print x, args, kwargs
5
6   args, kw = (1,2,3), {'cat': 'dog'}
7
8   print "# Normal call"
9   f(0, *args, **kw)
10
11  print "# Multicall"
12  P = Pool()
13      sol = [P.apply_async(f, (x,) + args, kw) for x in range(2)]
14  P.close()
15  P.join()
16
17  for s in sol: s.get()
```

Max-Planck-Institut
für Meteorologie

Appendix: Parallel with Ruby

```ruby
require 'parallel'
require 'cdo'

cdo = Cdo.new
files = Dir.glob("*nc")

ofiles = Parallel.map(files,:in_processes => nWorkers).each {|file|
    basename = file[0..-(File.extname(file).size+1)]
    ofile = cdo.remap(targetGridFile,targetGridweightsFile,
                      :input => file,
                      :output => "remapped_#{basename}.nc")
}

# Merge all the results together
cdo.merge(:input => ofiles.join(" "),:output => 'mergedResults.nc')
```

Max-Planck-Institut
für Meteorologie