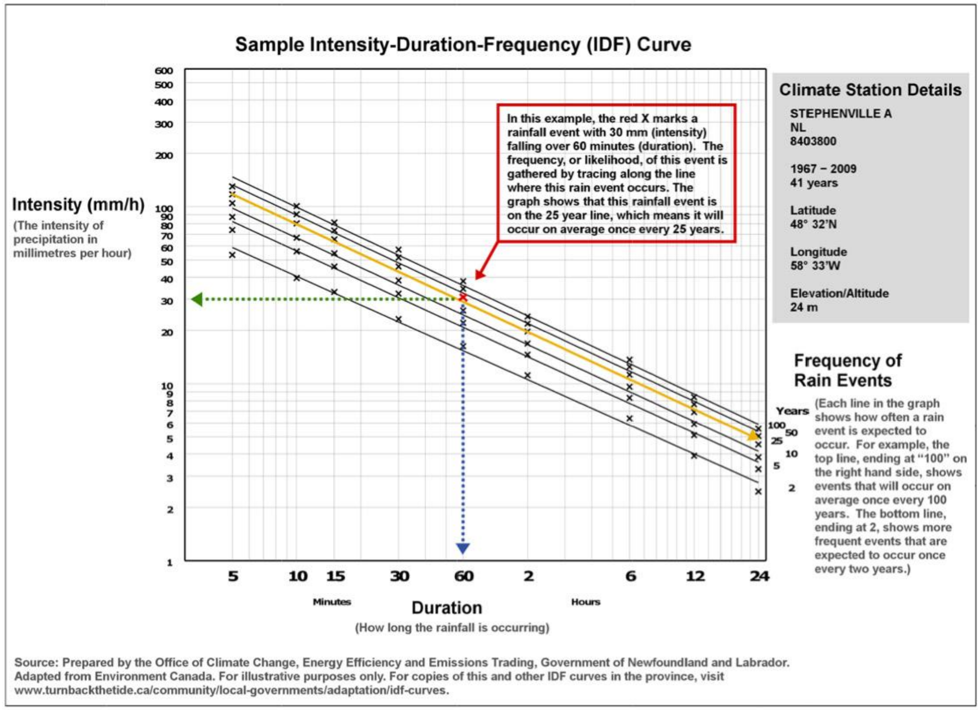
Using open source tools to reduce large data sets in a distributed environment

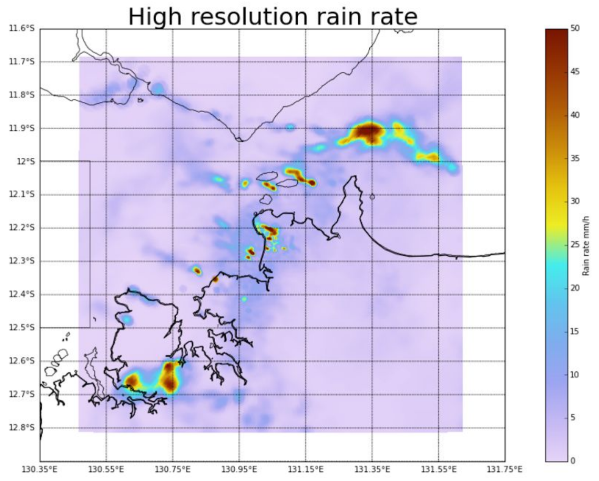
A view from a reluctant and very poor data guy

Scott Collis and Jonathan Helmus and many others!

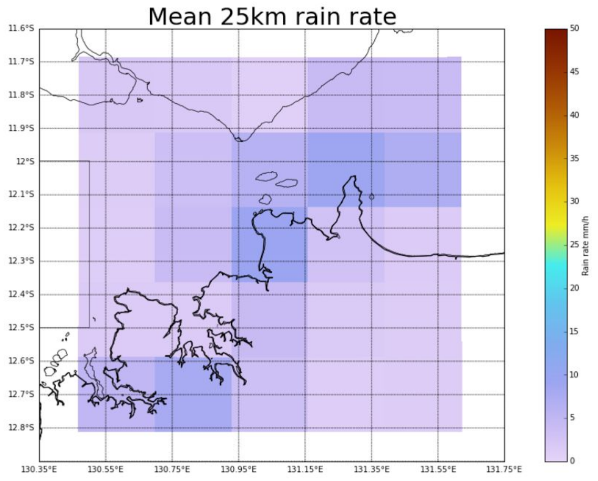
The Problem:

- Currently many centers are downscaling GCM runs using WRF to 12.5km to look at local rainfall.
- This does not match to high density hydrological systems (cities). Or natural scales of precipitation.
- Most urban centers have a WSR-88D radar nearby capable of resolutions less than 4km.
- We are investigating the impact of resolution on modelled streamflow.

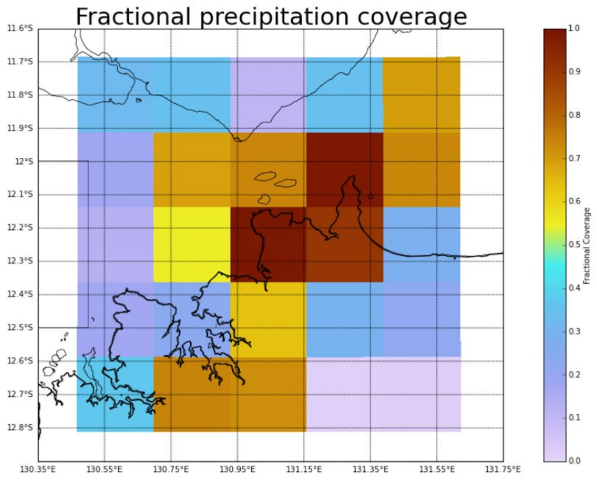




(a)



(b)



(c)

Solution: Radar. New Problem: Lots of data.

- A radar system images precipitating systems.
- It measures reflectivity factor, or the 6th moment of DSD as well as a number of other measurements depending on the sophistication of the system.
- So many formats. Complex geometry.
- Produces many thousands of files per year.
- Fortunately the problem is pleasantly parallel. The data is highly granular.

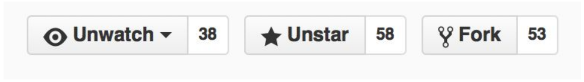
National Severe Storms Laboratory
National Storm Prediction Center
1313
NOAA
By Original Photograph By: Andrew J Oldaker - http://en.wikipedia.org/wiki/Image:LabNexrad.jpg, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=3124873

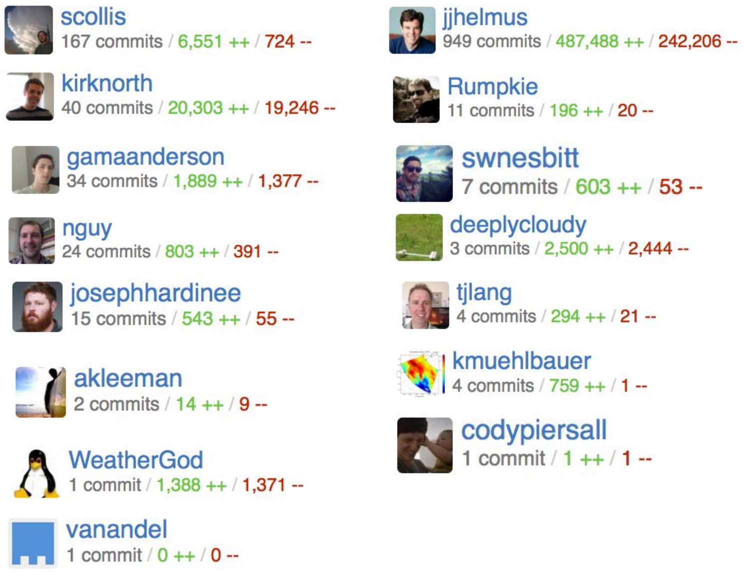
Open source software + Cluster tools make the problem tractable

Py-ART the Python ARM Radar Toolkit

https://github.com/ARM-DOE/pyart

- Py-ART uses a moderately complex data model which closely mirrors the CF-Radial community file format.
- It makes heavy use of Python dictionaries to form a self –describing radar object. Rich IO layer allowing a very large number file formats to be read into the data model. Primary output format is CF-Radial. We are investigating adding ODIM as an output format.
- Community codebase on GitHub, main fork is DoE maintained and moderated. Set of core dependences (all in Anaconda Python) with many optional dependencies which, when present, increase functionality.




scollis
167 commits / 6,551 ++ / 724 --

jjhelmus
949 commits / 487,488 ++ / 242,206 --

kirknorth
40 commits / 20,303 ++ / 19,246 --

Rumpkie
11 commits / 196 ++ / 20 --

gamaanderson
34 commits / 1,889 ++ / 1,377 --

swnesbitt
7 commits / 603 ++ / 53 --

nguy
24 commits / 803 ++ / 391 --

deeplycloudy
3 commits / 2,500 ++ / 2,444 --

josephhardinee
15 commits / 543 ++ / 55 --

tjlang
4 commits / 294 ++ / 21 --

akleeman
2 commits / 14 ++ / 9 --

kmuehlbauer
4 commits / 759 ++ / 1 --

WeatherGod
1 commit / 1,388 ++ / 1,371 --

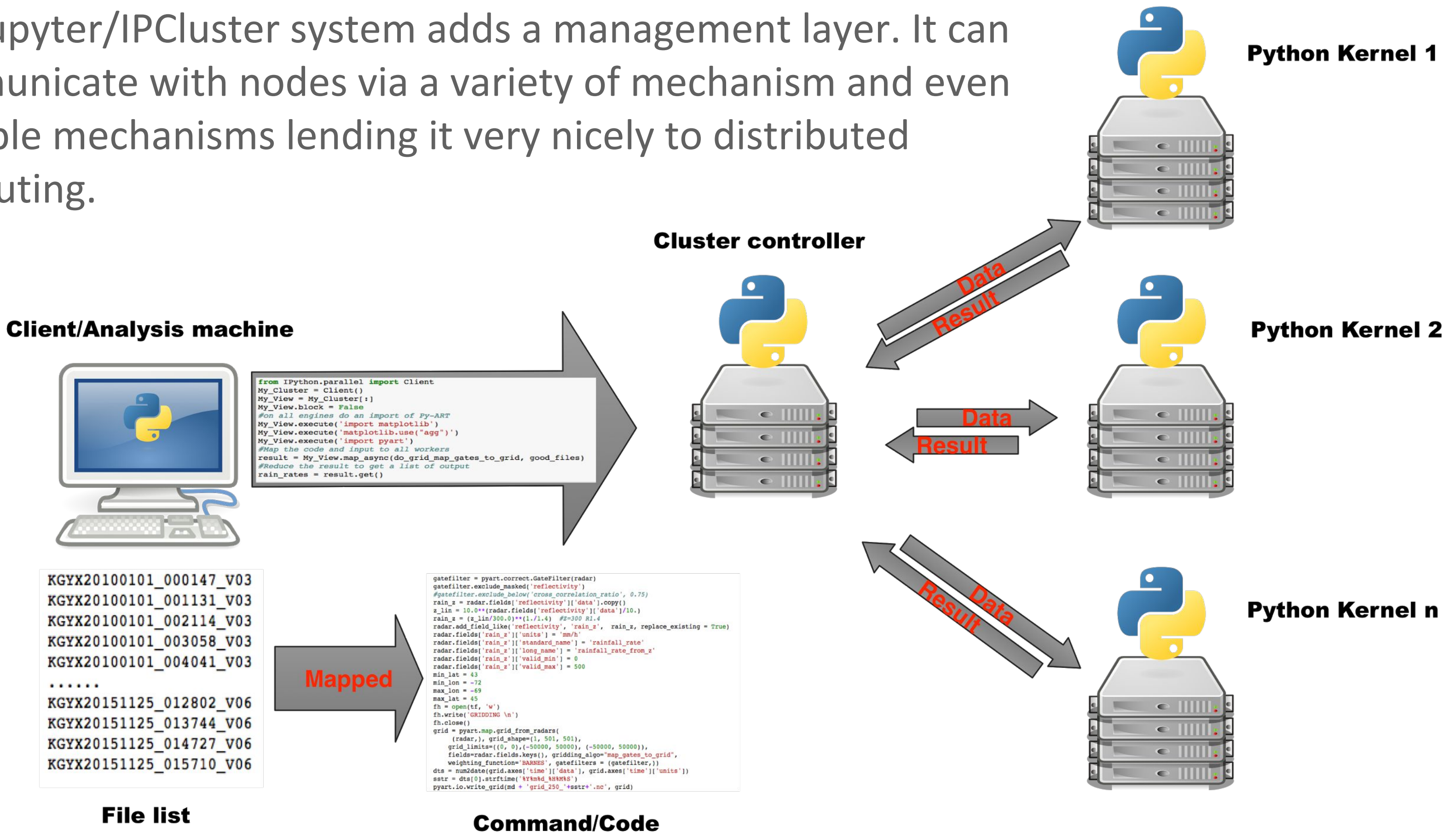
codypiersall
1 commit / 1 ++ / 1 --

vanandel
1 commit / 0 ++ / 0 --

ARM
CLIMATE RESEARCH FACILITY

IPython/Jupyter Cluster. Easy map reduce

- In the quest for reproducible, collaborative science, on thing the Jupyter project has done is to work with “detached kernels”.
- This is how notebooks work.. But you can have an array of kernels waiting for 0MQ packets to do work and return results.
- The Jupyter/IPCluster system adds a management layer. It can communicate with nodes via a variety of mechanism and even multiple mechanisms lending it very nicely to distributed computing.



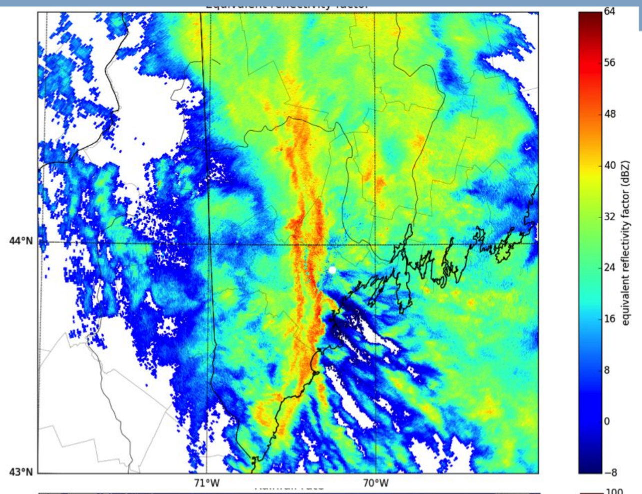
```
from IPython.parallel import Client
My_Cluster = Client()
My_View = My_Cluster[:]
My_View.block = False
#on all engines do an import of Py-ART
My_View.execute('import matplotlib')
My_View.execute('matplotlib.use("agg")')
My_View.execute('import pyart')
#Map the code and input to all workers
result = My_View.map_async(do_grid_map_gates_to_grid, good_files)
#Reduce the result to get a list of output
rain_rates = result.get()
```

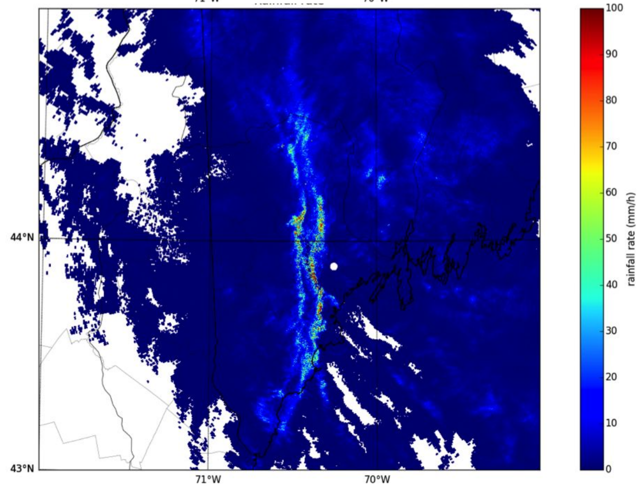
```
KGYX20100101_000147_V03
KGYX20100101_001131_V03
KGYX20100101_002114_V03
KGYX20100101_003058_V03
KGYX20100101_004041_V03
......
KGYX20151125_012802_V06
KGYX20151125_013744_V06
KGYX20151125_014727_V06
KGYX20151125_015710_V06
```

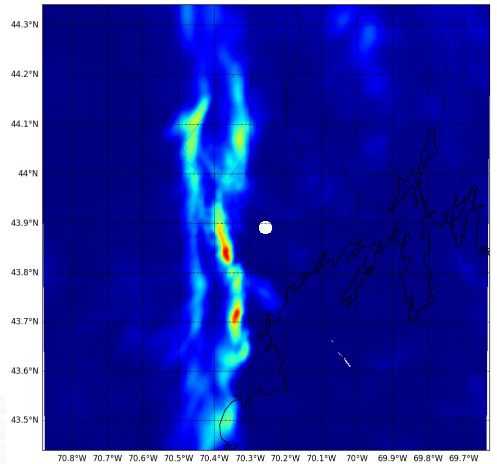
```
gatefilter = pyart.correct.GateFilter(radar)
gatefilter.exclude_masked('reflectivity')
#gatefilter.exclude_below('cross_correlation_ratio', 0.75)
rain_z = radar.fields['reflectivity']['data'].copy()
z_lin = 10.0**(radar.fields['reflectivity']['data']/10.)
rain_z = (z_lin/300.0)**(1./1.4)  #Z=300 R1.4
radar.add_field_like('reflectivity', 'rain_z',  rain_z, replace_existing = True)
radar.fields['rain_z']['units'] = 'mm/h'
radar.fields['rain_z']['standard_name'] = 'rainfall_rate'
radar.fields['rain_z']['long_name'] = 'rainfall_rate_from_z'
radar.fields['rain_z']['valid_min'] = 0
radar.fields['rain_z']['valid_max'] = 500
min_lat = 43
min_lon = -72
max_lon = -69
max_lat = 45
fh = open(tf, 'w')
fh.write('GRIDDING \n')
fh.close()
grid = pyart.map.grid_from_radars(
     (radar,), grid_shape=(1, 501, 501),
    grid_limits=((0, 0),(-50000, 50000), (-50000, 50000)),
    fields=radar.fields.keys(), gridding_algo="map_gates_to_grid",
    weighting_function='BARNES', gatefilters = (gatefilter,))
dts = num2date(grid.axes['time']['data'], grid.axes['time']['units'])
sstr = dts[0].strftime('%Y%m%d_%H%M%S')
pyart.io.write_grid(md + 'grid_250_'+sstr+'.nc', grid)
```

The study: Rainfall around Portland, Maine.

- Portland is the target for a Department of Homeland Security Regional Resiliency Assessment Program study.
- Step 1 : Download A LOT of data.. This is the **most challenging step.** We downloaded 10 years worth of data from NCEI and staged it on an optimized storage setup.
- Step 2: Develop your technique for one file.
- Step 3: Google like mad to find out how to set up a Jupyter cluster on a PBS managed system. Fortunately this step stays done.. and you have a lasting resource!
- Step 4: Map the file list from step 1 to the technique in step 2.





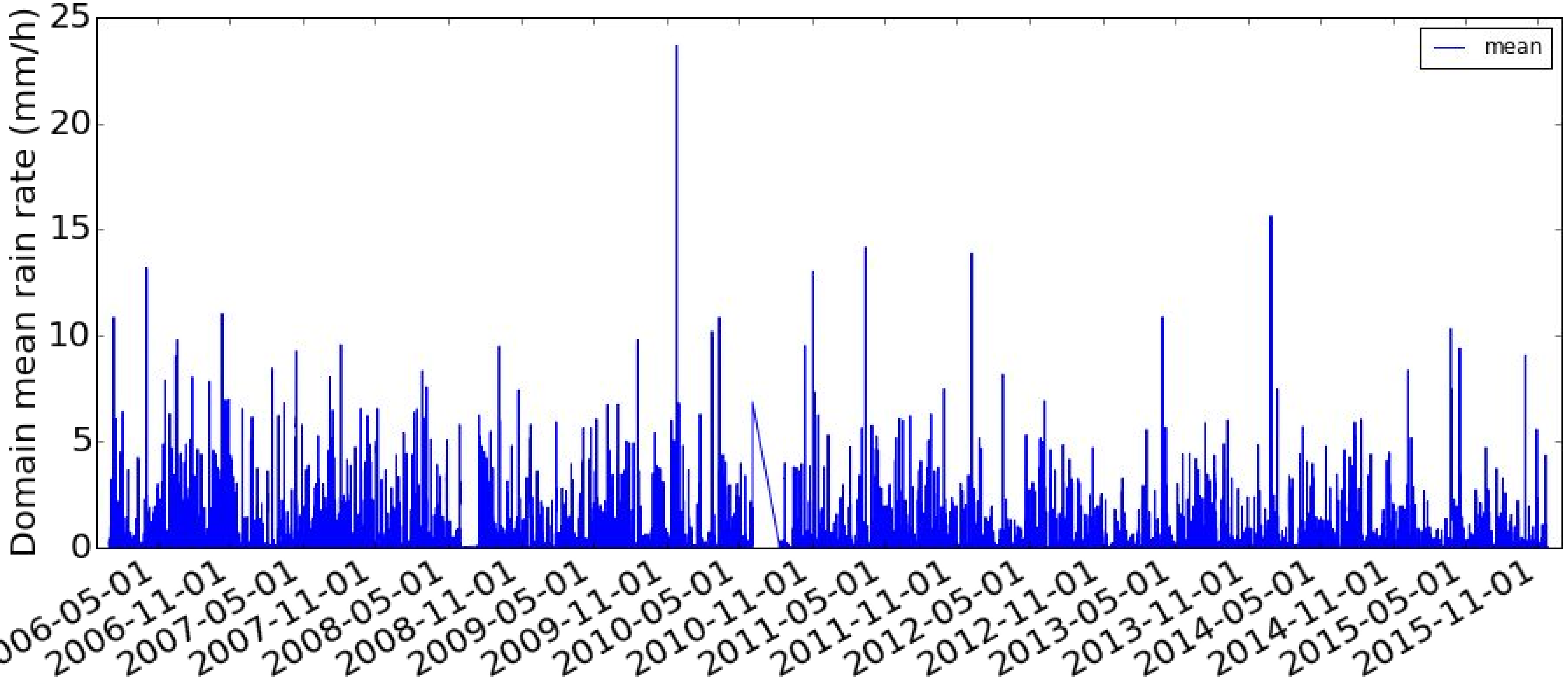


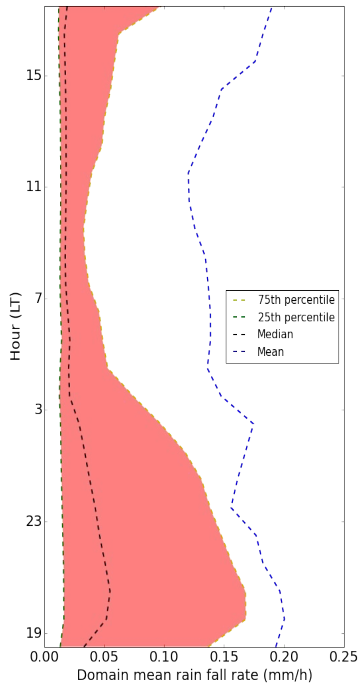
Demo 1: Very simple rainfall grid with Py-ART

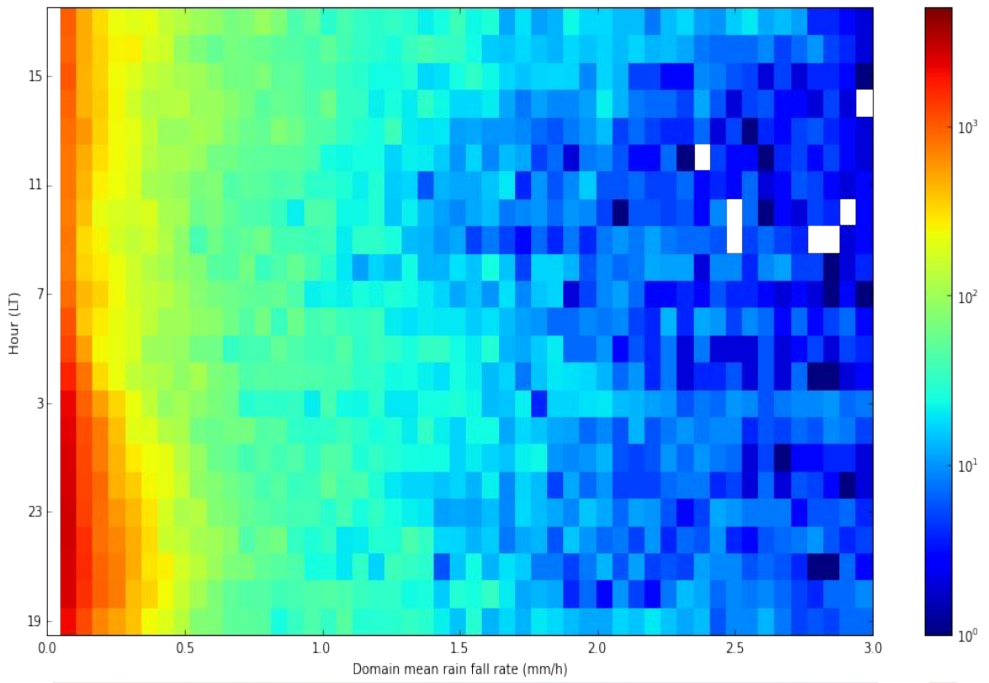
Demo 2: Simple demo of the Jupyter Cluster.

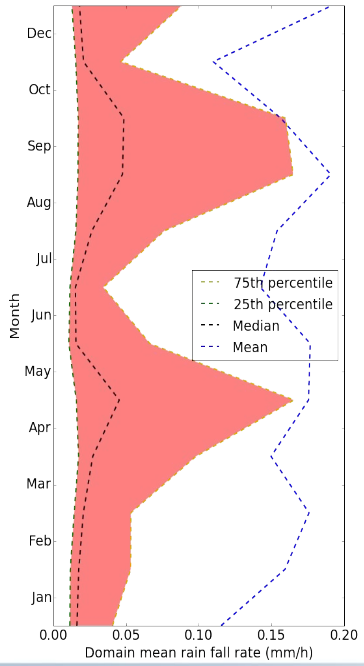
Some Results

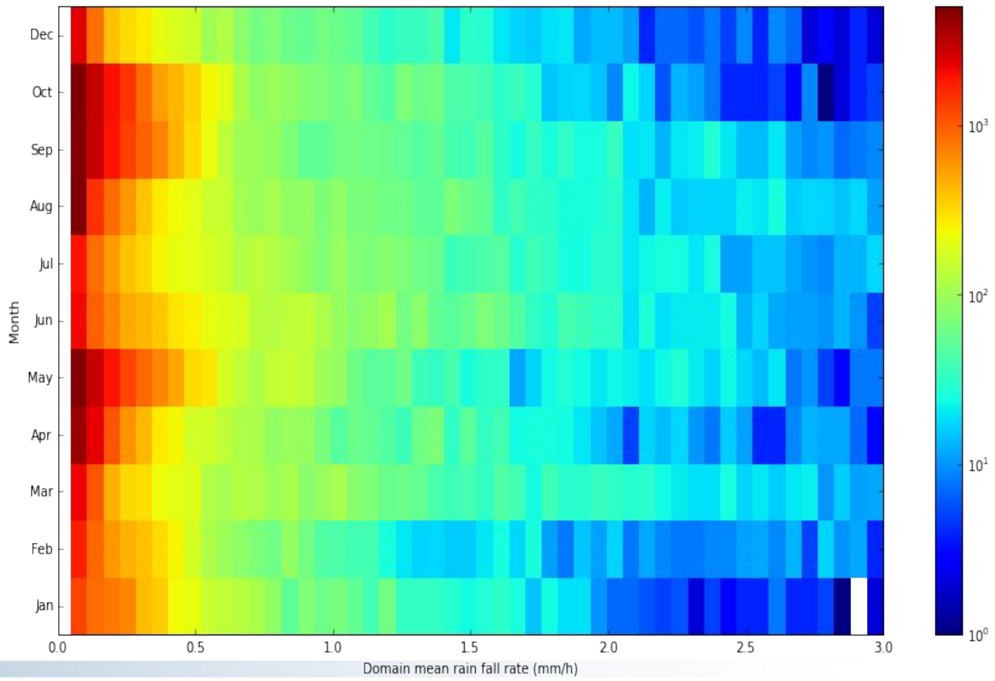
- 700,000 files, 10+TB was reduced to grids and domain mean rainfall calculated.
- We saved these off as a 100MB times series for easy analysis and data indexing.
- We then, because we can, stratified by hour of the day and monthly.




Hour (LT)
Domain mean rain fall rate (mm/h)
75th percentile
25th percentile
Median
Mean


Hour (LT)
Domain mean rain fall rate (mm/h)


Month
Domain mean rain fall rate (mm/h)
75th percentile
25th percentile
Median
Mean


Month
Domain mean rain fall rate (mm/h)

Some concluding thoughts

- Ability to push single "granule" code to a distributed environment is key.
- This allows very quick revisit times for development and makes decadal+ processing possible.
- Good, thread safe, well tested open source software,**even if it is written without the explicit desire to be parallel** is key.
- As is a good system to map the problem to many cores.
- This was, well.. very messy. Good data tools are a key need.. would love it if NEXRAD data came with some descriptive inventory data.
- Py-ART has grown in strides. We have benefited greatly from unfunded contributions. Key in encouraging this is using funds to grow out accepted formats base. This is a key communique to programs: open mindedness pays dividends..
- We are now working with Amazon to see how far we can push this.

amazon web services

https://github.com/scollis/high_resolution_hydrology