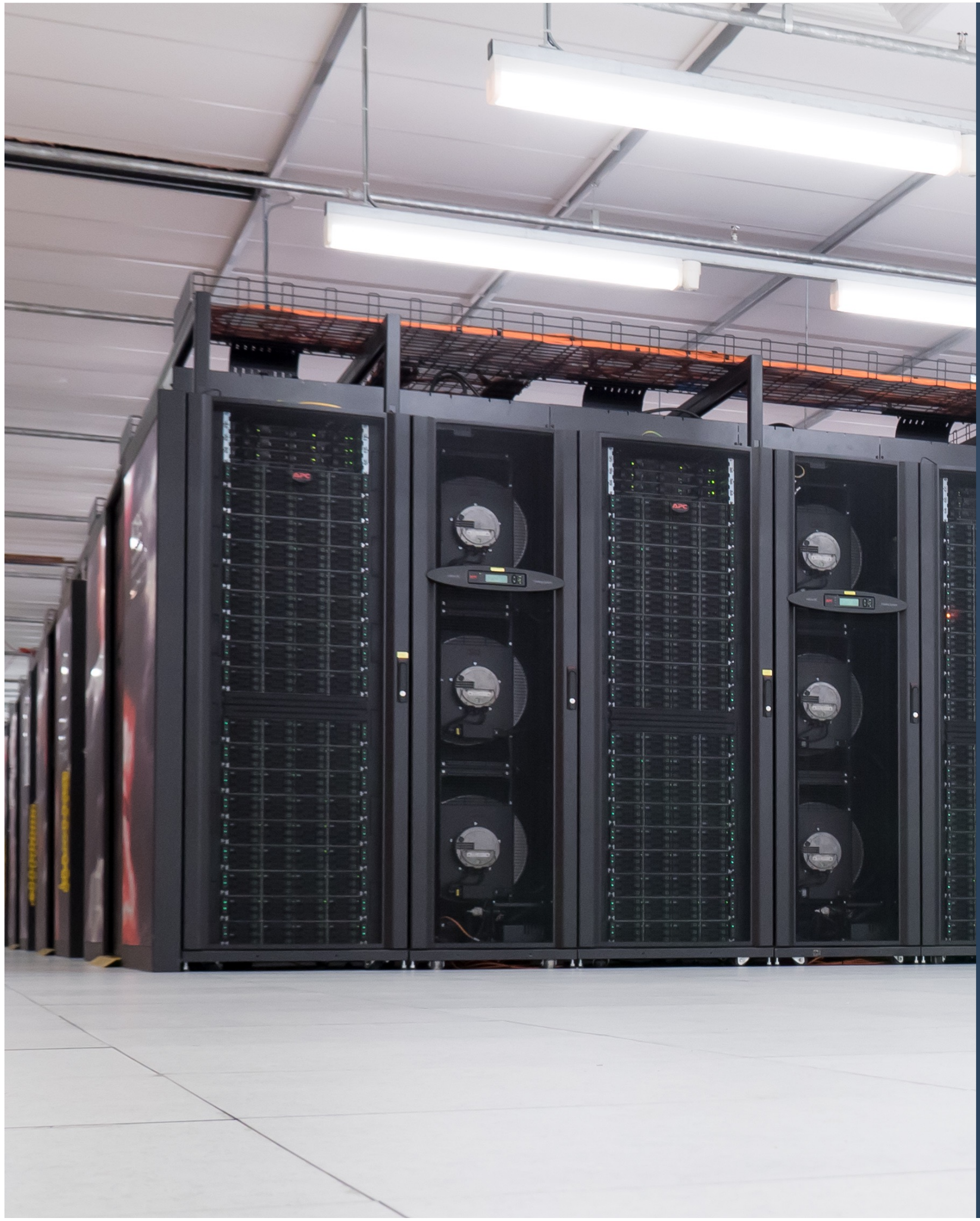


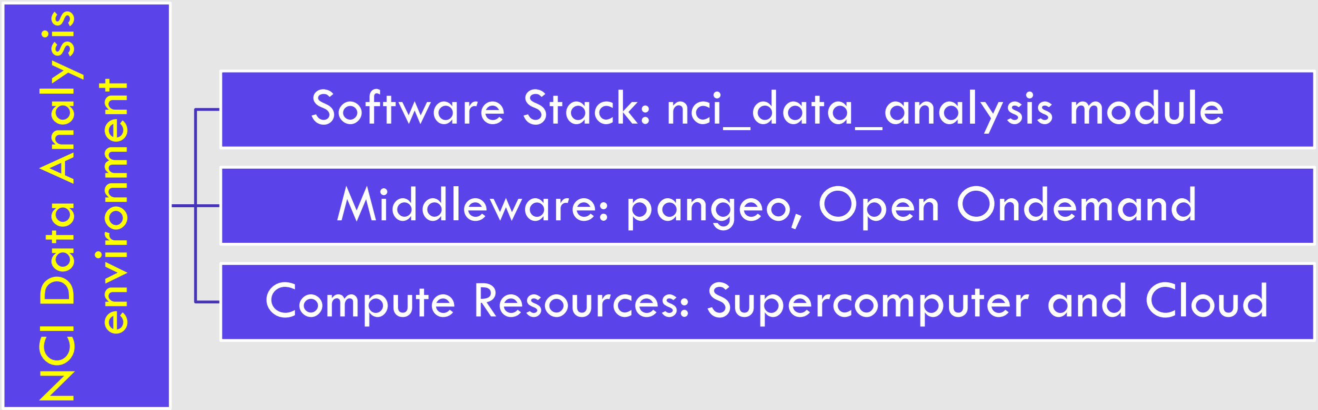
NCI'S JUPYTER-PANGEO ENVIRONMENT FOR DATA ANALYSIS

DATA SCIENCE WEEK 2021

NCI AUSTRALIA

ANALYSING NCI DATA COLLECTIONS

- The NCI Reference Data Collections are organised in a systematic way to enable fast programmatic access for analysis across multiple domains.
- NCI data collections are available for use on NCI's core computing resources like NCI supercomputer and NCI cloud-based platforms.
- NCI established the data analysis environment to help users accessing the data collections in a scalable programming way.



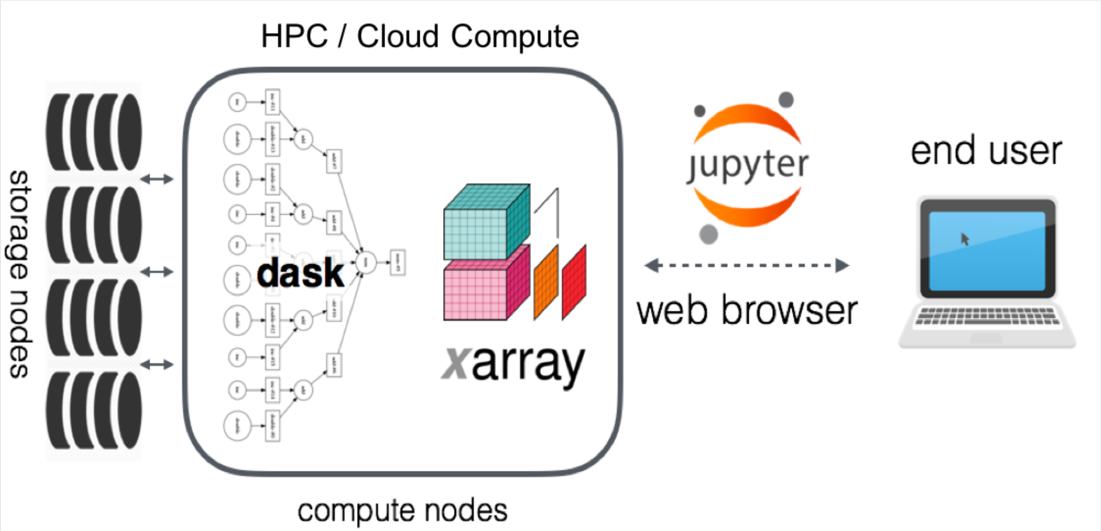
NCI Data Analysis environment

Software Stack: nci_data_analysis module

Middleware: pangeo, Open Ondemand

Compute Resources: Supercomputer and Cloud

TYPICAL NCI DATA ANALYSIS WORKFLOW



https://pangeo.io

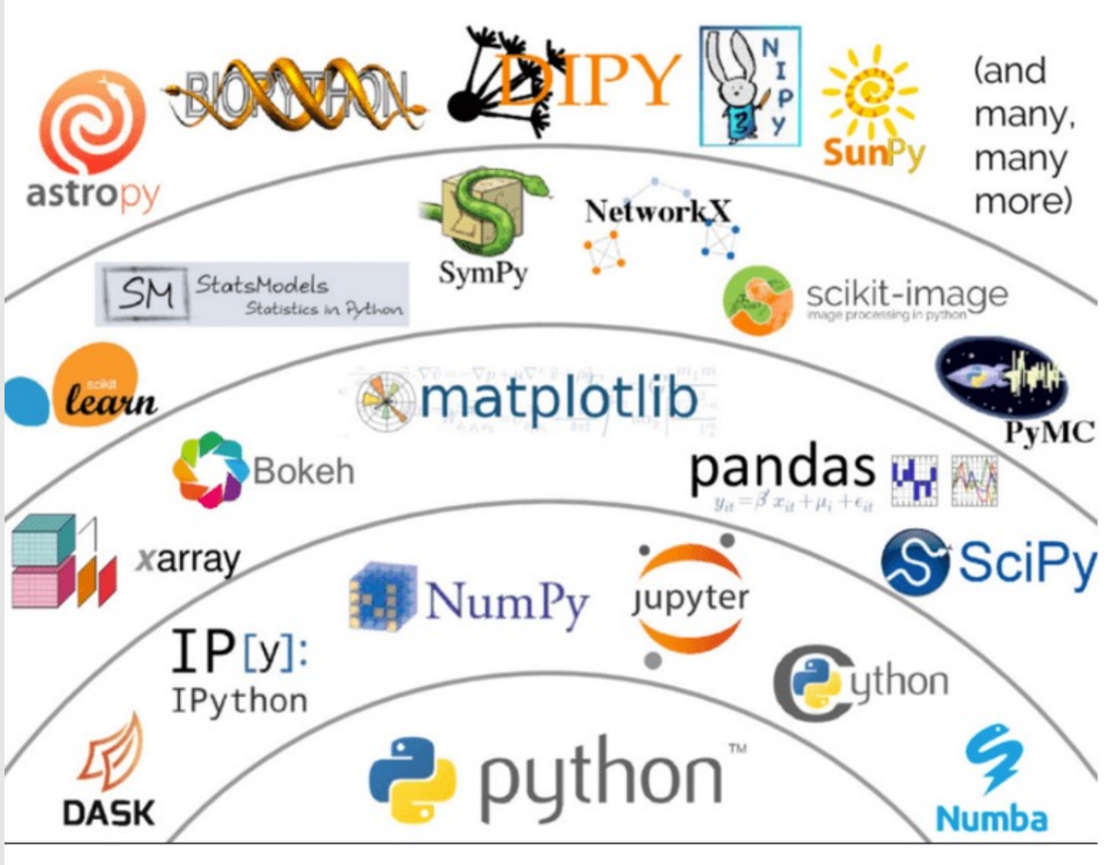
Remote Jupyter job is executing at the NCI scalable resources.

The job utilizes Dask as the parallel computing engine.

Users work with their own desktop web browser to access the remote Jupyter server.

https://opus.nci.org.au/display/Help/Data+Analysis+Environments

SOFTWARE STACK FOR NCI DATA ANALYSIS



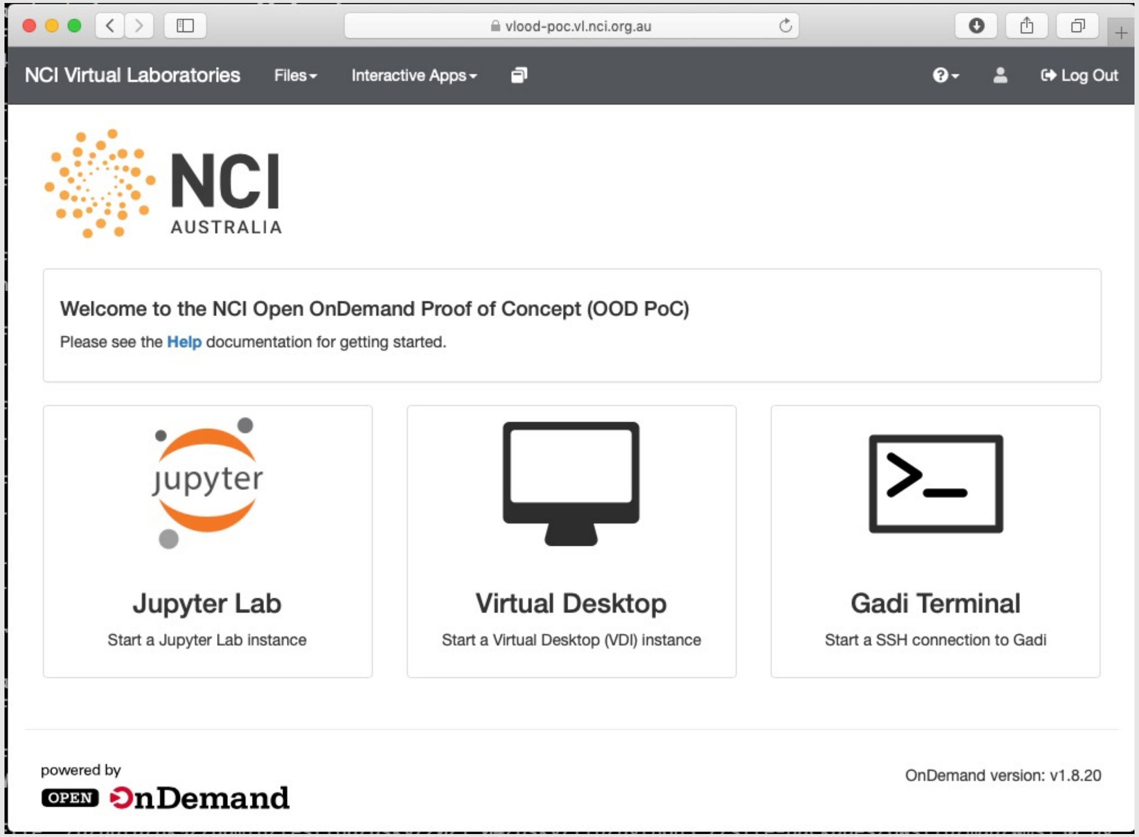
nci-data-analysis/2021.03 module

- 485 general-purpose python libraries including Jupyter, xarray and Dask etc.
- periodically updating (~3 months).
- adding more libs per user's request.

COMPUTE RESOURCES FOR NCI DATA ANALYSIS

	GADI	VDI	JupyterLab
Description	Australia's peak research supercomputer with 4000+ compute nodes including 640 NVIDIA V100 GPUs	Graphical desktop-like interface based on the NCI cloud resource	Jupyter native interface which supports scalable dask cluster jobs
Hardware/Node	**48 CascadeLake CPU cores, 192 GB memory (normal queue)**	16 vCPUs (SandyBridge), 32GB memory	16 vCPUs (SandyBridge), 32GB memory
Job resources	**Multiple nodes**	Single node	**Multiple nodes**
File System	**Lustre**	NFS access to Lustre	NFS access to Lustre
Internet Connection	No	**Yes**	**Yes**
Web Browser	No	**Yes**	**Yes**
Typical job	**compute intensive work**	**data analysis** code development **visualisation**	**data analysis**

OPEN ONDEMAND (OOD) PLATFORM



Welcome to the NCI Open OnDemand Proof of Concept (OOD PoC)

Please see the Help documentation for getting started.

Jupyter Lab

Start a Jupyter Lab instance

Virtual Desktop

Start a Virtual Desktop (VDI) instance

Gadi Terminal

Start a SSH connection to Gadi

OnDemand version: v1.8.20

https://vlood-poc.vl.nci.org.au

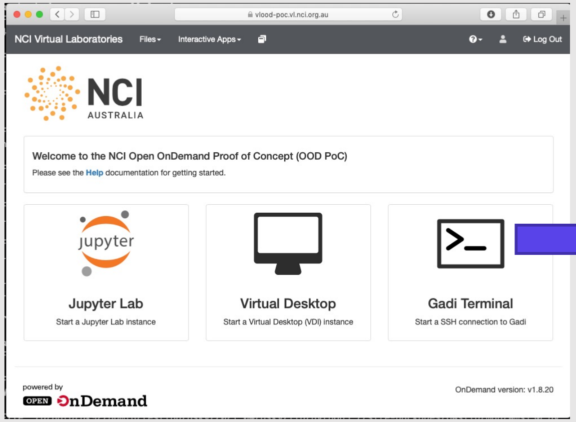
OOD is an online portal giving you access to compute resources at NCI such as VDI, Gadi, or Jupyter Lab. It provides an app based infrastructure which we will extend as new new apps are developed.

3 apps on the OOD:

- Gadi Terminal: simple browser based access to Gadi; the session closes when you close your browser window (or change to another page). Useful for checking Gadi jobs from a web-browser etc.
- VDI: Web-browser based VNC connection to a VDI session (i.e. alternative to Strudel).
- Jupyter Lab: dedicated Jupyter Lab session which can be used to run Dask workload among others.

Currently at the stage of “Proof of Concept”.

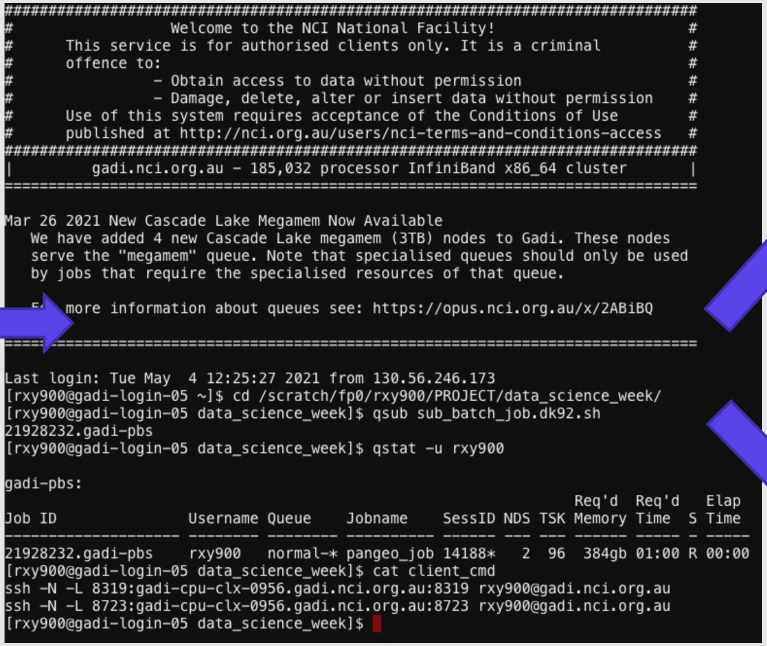
ACCESS GADI VIA OOD



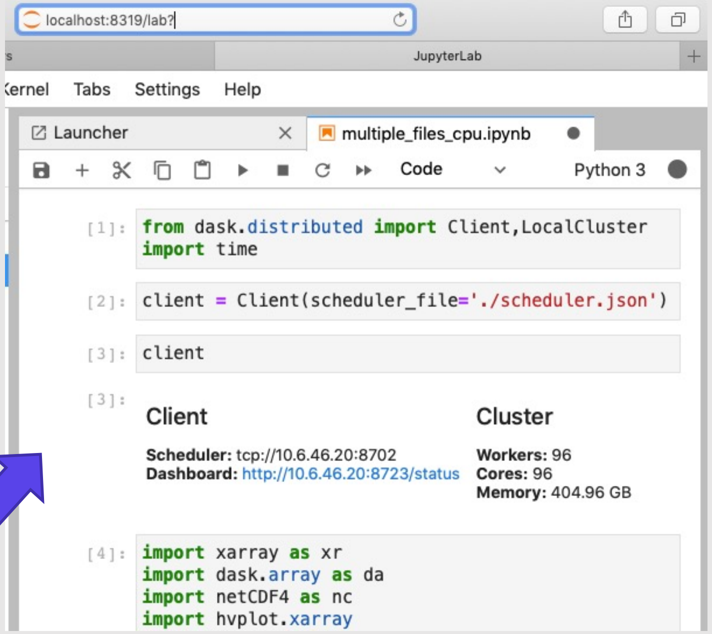
Welcome to the NCI Open OnDemand Proof of Concept (OOD PoC)

Please see the Help documentation for getting started.

Jupyter Lab
Start a Jupyter Lab instance
Virtual Desktop
Start a Virtual Desktop (VDI) instance
Gadi Terminal
Start a SSH connection to Gadi
powered by OPEN OnDemand
OnDemand version: v1.8.20



```
##############################################################################
#                    Welcome to the NCI National Facility!                   #
#       This service is for authorised clients only. It is a criminal        #
#       offence to:                                                          #
#                 - Obtain access to data without permission                 #
#                 - Damage, delete, alter or insert data without permission  #
#       Use of this system requires acceptance of the Conditions of Use      #
#       published at http://nci.org.au/users/nci-terms-and-conditions-access #
##############################################################################
|         gadi.nci.org.au - 185,032 processor InfiniBand x86_64 cluster      |
==============================================================================

Mar 26 2021 New Cascade Lake Megamem Now Available
   We have added 4 new Cascade Lake megamem (3TB) nodes to Gadi. These nodes
   serve the "megamem" queue. Note that specialised queues should only be used
   by jobs that require the specialised resources of that queue.

   For more information about queues see: https://opus.nci.org.au/x/2ABiBQ

==============================================================================

Last login: Tue May  4 12:25:27 2021 from 130.56.246.173
[rxy900@gadi-login-05 ~]$ cd /scratch/fp0/rxy900/PROJECT/data_science_week/
[rxy900@gadi-login-05 data_science_week]$ qsub sub_batch_job.dk92.sh
21928232.gadi-pbs
[rxy900@gadi-login-05 data_science_week]$ qstat -u rxy900

gadi-pbs:
                                                            Req'd  Req'd   Elap
Job ID               Username Queue    Jobname    SessID NDS TSK Memory Time  S Time
-------------------- -------- -------- ---------- ------ --- --- ------ ----- - -----
21928232.gadi-pbs    rxy900   normal-* pangeo_job 14188*   2  96  384gb 01:00 R 00:00
[rxy900@gadi-login-05 data_science_week]$ cat client_cmd
ssh -N -L 8319:gadi-cpu-clx-0956.gadi.nci.org.au:8319 rxy900@gadi.nci.org.au
ssh -N -L 8723:gadi-cpu-clx-0956.gadi.nci.org.au:8723 rxy900@gadi.nci.org.au
[rxy900@gadi-login-05 data_science_week]$
```



```
[1]: from dask.distributed import Client,LocalCluster
     import time

[2]: client = Client(scheduler_file='./scheduler.json')

[3]: client
```

[3]:

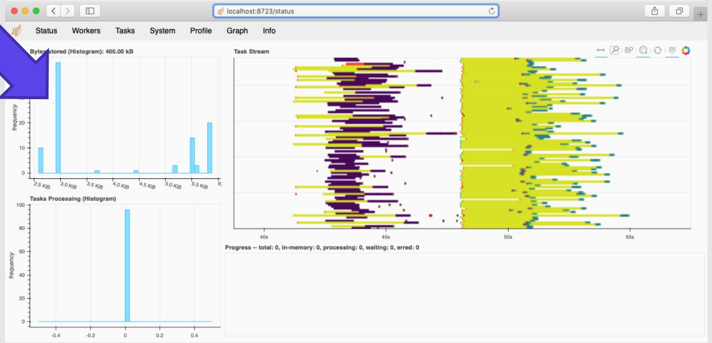
Client

Scheduler: tcp://10.6.46.20:8702
Dashboard: http://10.6.46.20:8723/status

Cluster

Workers: 96
Cores: 96
Memory: 404.96 GB

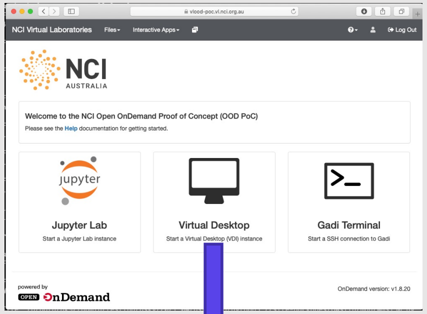
```
[4]: import xarray as xr
     import dask.array as da
     import netCDF4 as nc
     import hvplot.xarray
```



Progress -- total: 0, in-memory: 0, processing: 0, waiting: 0, erred: 0

Pangeo Manual
https://opus.nci.org.au/display/Help/5.+Pangeo+on+Gadi

ACCESS VDI VIA OOD

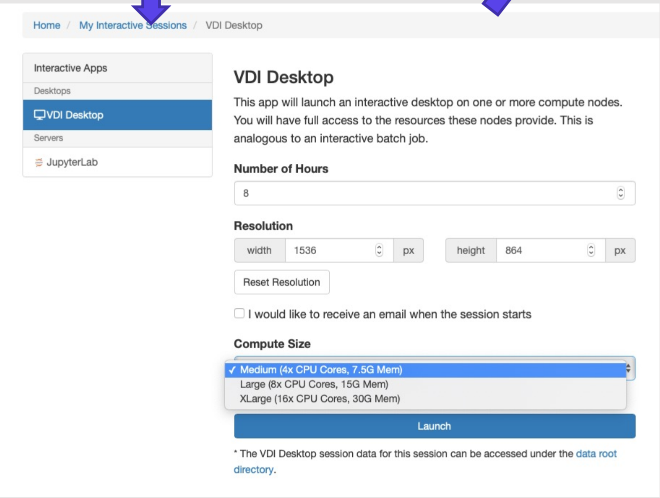


Welcome to the NCI Open OnDemand Proof of Concept (OOD PoC)

Please see the Help documentation for getting started.

Jupyter Lab
Start a Jupyter Lab instance
Virtual Desktop
Start a Virtual Desktop (VDI) instance
Gadi Terminal
Start a SSH connection to Gadi
powered by OPEN OnDemand

OnDemand version: v1.8.20



VDI Desktop

This app will launch an interactive desktop on one or more compute nodes. You will have full access to the resources these nodes provide. This is analogous to an interactive batch job.

Number of Hours: 8

Resolution: width 1536 px, height 864 px

Reset Resolution

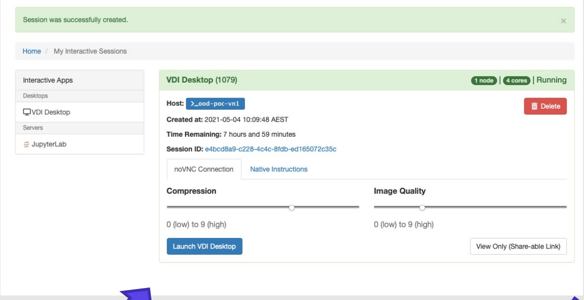
□ I would like to receive an email when the session starts

Compute Size

- ✓ Medium (4x CPU Cores, 7.5G Mem)
- Large (8x CPU Cores, 15G Mem)
- XLarge (16x CPU Cores, 30G Mem)

Launch

* The VDI Desktop session data for this session can be accessed under the data root directory.



Session was successfully created.

VDI Desktop (1079) 1 node | 4 cores | Running

Host: >_ood-poc-vn1

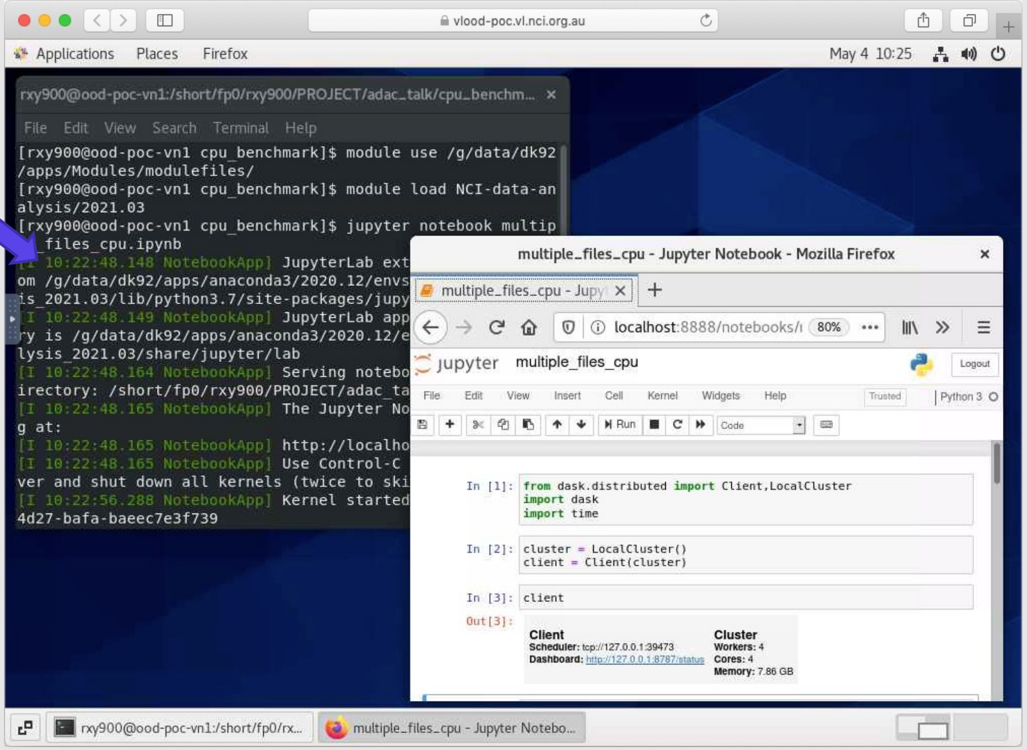
Created at: 2021-05-04 10:09:48 AEST

Time Remaining: 7 hours and 59 minutes

Session ID: e4bcd8a9-c228-4c4c-8fdb-ed165072c35c

Compression: 0 (low) to 9 (high)

Image Quality: 0 (low) to 9 (high)



```
[rxy900@ood-poc-vn1 cpu_benchmark]$ module use /g/data/dk92
/apps/Modules/modulefiles/
[rxy900@ood-poc-vn1 cpu_benchmark]$ module load NCI-data-an
alysis/2021.03
[rxy900@ood-poc-vn1 cpu_benchmark]$ jupyter notebook multip
_files_cpu.ipynb
[I 10:22:48.148 NotebookApp] JupyterLab ext
om /g/data/dk92/apps/anaconda3/2020.12/envs
is_2021.03/lib/python3.7/site-packages/jupy
[I 10:22:48.149 NotebookApp] JupyterLab app
ry is /g/data/dk92/apps/anaconda3/2020.12/e
lysis_2021.03/share/jupyter/lab
[I 10:22:48.164 NotebookApp] Serving notebo
irectory: /short/fp0/rxy900/PROJECT/adac_ta
[I 10:22:48.165 NotebookApp] The Jupyter No
g at:
[I 10:22:48.165 NotebookApp] http://localho
[I 10:22:48.165 NotebookApp] Use Control-C
ver and shut down all kernels (twice to ski
[I 10:22:56.288 NotebookApp] Kernel started
4d27-bafa-baeec7e3f739
```

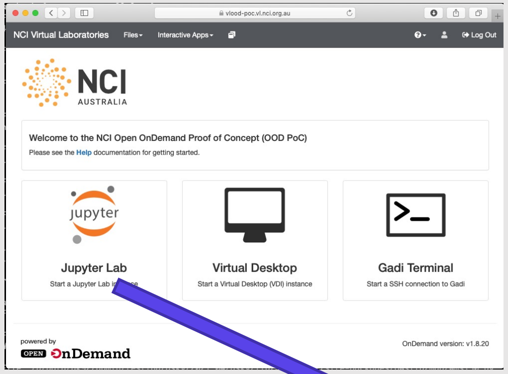
multiple_files_cpu - Jupyter Notebook - Mozilla Firefox
multiple_files_cpu - Jupy
localhost:8888/notebooks/
80%
jupyter multiple_files_cpu
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Python 3
Run
Code

```
In [1]: from dask.distributed import Client,LocalCluster
        import dask
        import time

In [2]: cluster = LocalCluster()
        client = Client(cluster)

In [3]: client
Out[3]:
```

Client	Cluster
Scheduler: tcp://127.0.0.1:39473	**Workers:** 4
Dashboard: http://127.0.0.1:8787/status	**Cores:** 4
	Memory: 7.86 GB

rxy900@ood-poc-vn1:/short/fp0/rx...
multiple_files_cpu - Jupyter Notebo...

ACCESS JUPYTERLAB VIA OOD



vlood-poc.vl.nci.org.au

Welcome to the NCI Open OnDemand Proof of Concept (OOD PoC)

Please see the Help documentation for getting started.

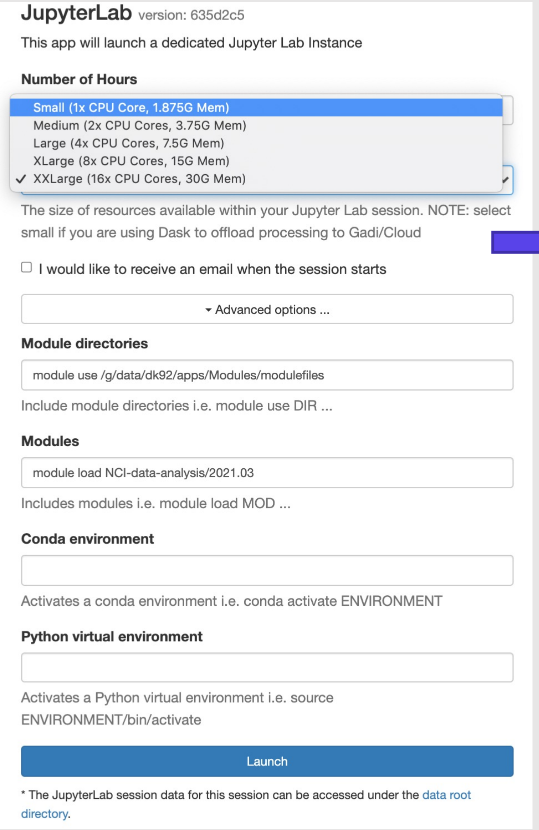
Jupyter Lab
Start a Jupyter Lab instance

Virtual Desktop
Start a Virtual Desktop (VDI) instance

Gadi Terminal
Start a SSH connection to Gadi

powered by OPEN OnDemand

OnDemand version: v1.8.20



JupyterLab version: 635d2c5

This app will launch a dedicated Jupyter Lab Instance

Number of Hours

- Small (1x CPU Core, 1.875G Mem)
- Medium (2x CPU Cores, 3.75G Mem)
- Large (4x CPU Cores, 7.5G Mem)
- XLarge (8x CPU Cores, 15G Mem)
- ✓ XXLarge (16x CPU Cores, 30G Mem)

The size of resources available within your Jupyter Lab session. NOTE: select small if you are using Dask to offload processing to Gadi/Cloud

□ I would like to receive an email when the session starts

▾ Advanced options ...

Module directories

module use /g/data/dk92/apps/Modules/modulefiles

Include module directories i.e. module use DIR ...

Modules

module load NCI-data-analysis/2021.03

Includes modules i.e. module load MOD ...

Conda environment

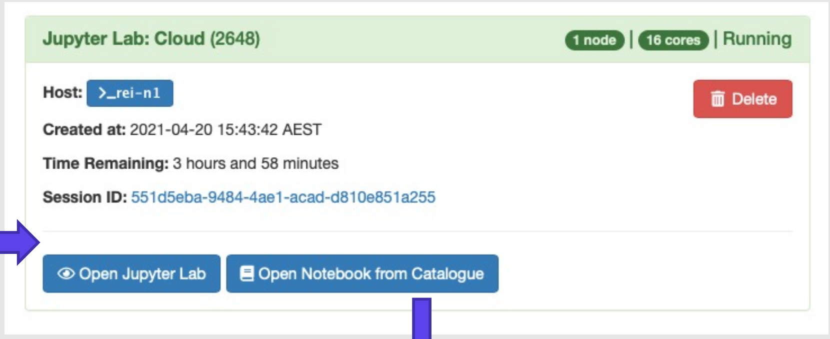
Activates a conda environment i.e. conda activate ENVIRONMENT

Python virtual environment

Activates a Python virtual environment i.e. source ENVIRONMENT/bin/activate

Launch

* The JupyterLab session data for this session can be accessed under the data root directory.



Jupyter Lab: Cloud (2648) 1 node | 16 cores | Running

Host: >_rei-n1

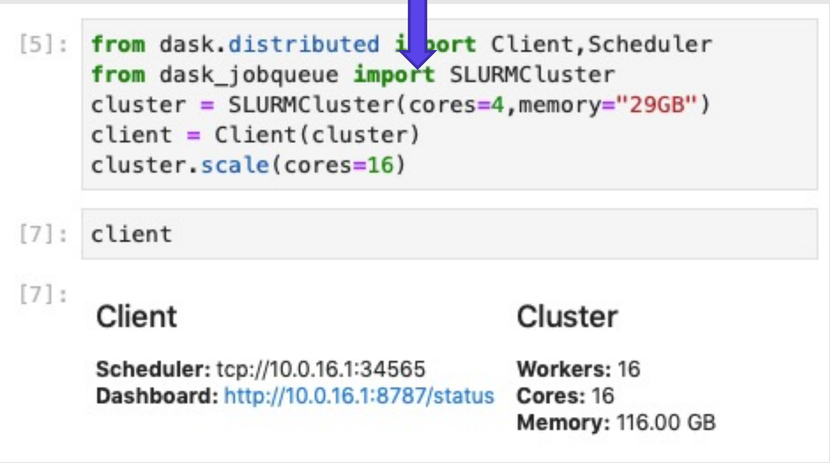
Delete

Created at: 2021-04-20 15:43:42 AEST

Time Remaining: 3 hours and 58 minutes

Session ID: 551d5eba-9484-4ae1-acad-d810e851a255

Open Jupyter Lab | Open Notebook from Catalogue



```
[5]: from dask.distributed import Client,Scheduler
     from dask_jobqueue import SLURMCluster
     cluster = SLURMCluster(cores=4,memory="29GB")
     client = Client(cluster)
     cluster.scale(cores=16)

[7]: client

[7]:
```

Client

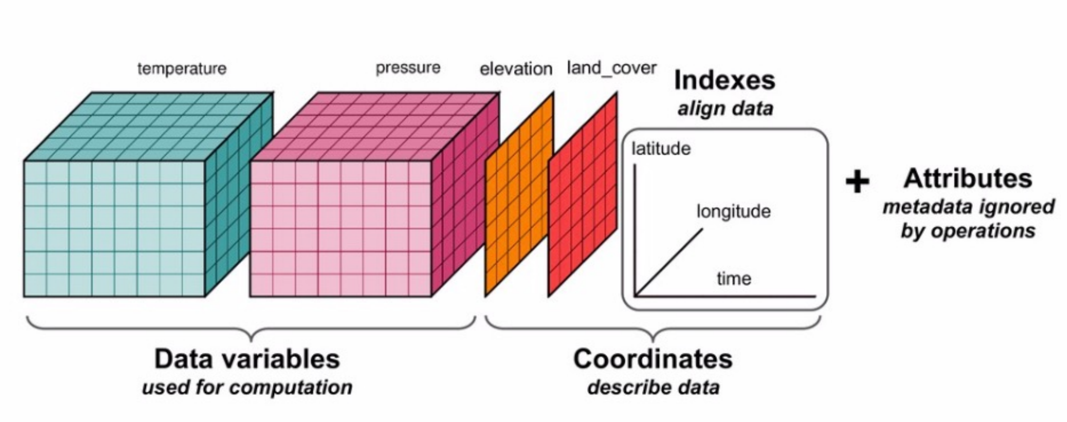
Scheduler: tcp://10.0.16.1:34565
Dashboard: http://10.0.16.1:8787/status

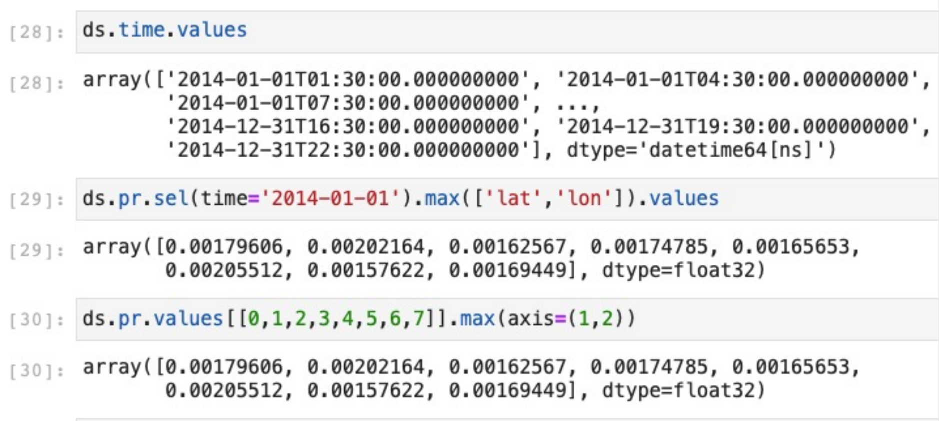
Cluster

Workers: 16
Cores: 16
Memory: 116.00 GB

AN EXAMPLE OF SCALABLE ANALYSIS: XARRAY AND LABELS

- NumPy provides the fundamental data structure and API for working with raw ndarrays.
- Xarray uses metadata in the form of labelled dimensions (e.g., ‘latitude’ or ‘frequency’) and coordinate values (e.g., the date ‘2021-02-05’) to enable a suite of expressive, label based operations.
- Can apply operations over dimensions by name: x.sum(‘time’)
- Can select values by label instead of integer location: x.loc[‘2021-02-05’] or x.sel(time=’2021-02- 05’)
- Mathematical operations (e.g. x - y) vectorize across multiple dimensions based on dimension names, not shape
- split-apply-combine paradigm with groupby, Database-like alignment based on coordinate labels, keep track of arbitrary metadata in the form of a Python dictionary: x.attrs, etc.



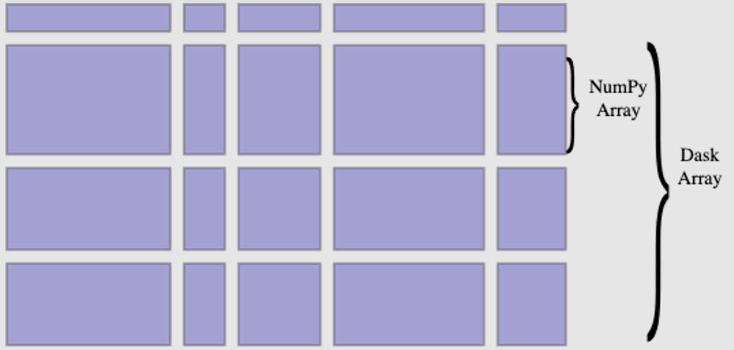


```
[28]: ds.time.values

[28]: array(['2014-01-01T01:30:00.000000000', '2014-01-01T04:30:00.000000000',
             '2014-01-01T07:30:00.000000000', ...,
             '2014-12-31T16:30:00.000000000', '2014-12-31T19:30:00.000000000',
             '2014-12-31T22:30:00.000000000'], dtype='datetime64[ns]')

[29]: ds.pr.sel(time='2014-01-01').max(['lat','lon']).values

[29]: array([0.00179606, 0.00202164, 0.00162567, 0.00174785, 0.00165653,
             0.00205512, 0.00157622, 0.00169449], dtype=float32)

[30]: ds.pr.values[[0,1,2,3,4,5,6,7]].max(axis=(1,2))

[30]: array([0.00179606, 0.00202164, 0.00162567, 0.00174785, 0.00165653,
             0.00205512, 0.00157622, 0.00169449], dtype=float32)
```

AN EXAMPLE OF SCALABLE ANALYSIS:
XARRAY - PARALLEL COMPUTING WITH DASK

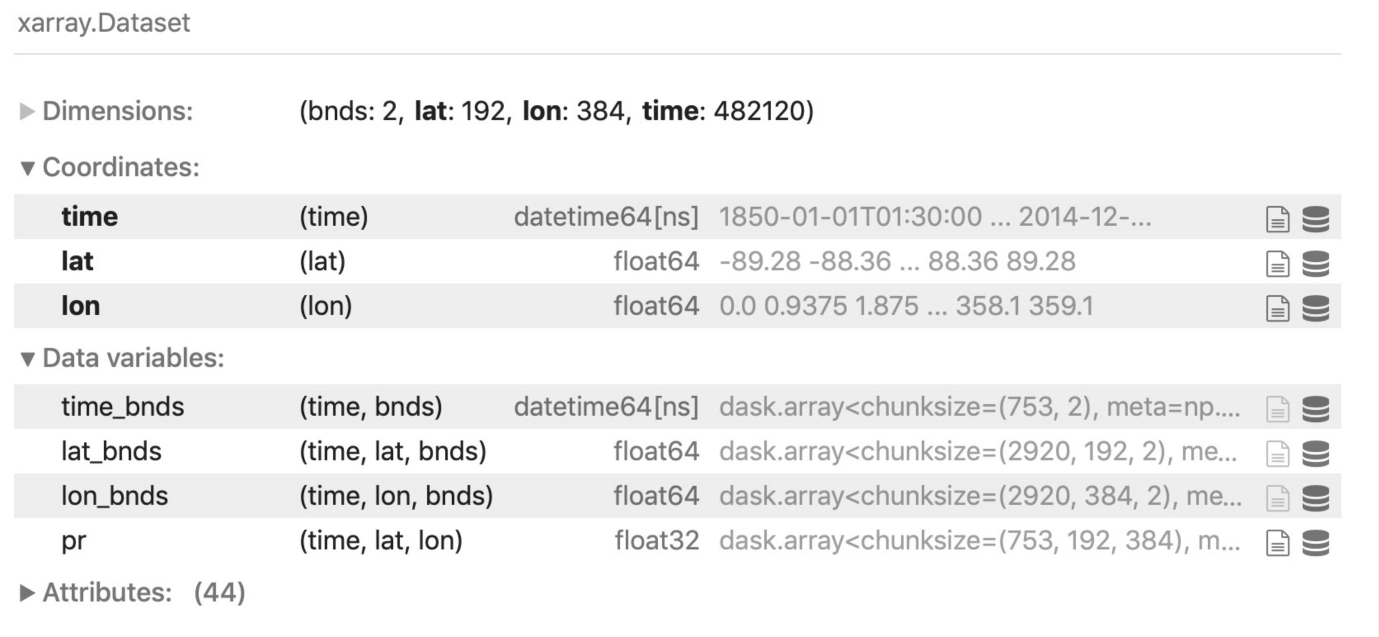
Xarray integrates with Dask to support parallel computations and streaming computation on datasets that don't fit into memory

- Dask divides arrays into many small pieces (chunks), each of which is presumed to be small enough to fit into memory so it can provides multi-core and distributed parallel execution on larger-than-memory datasets
- Dask scales up (to a cluster) and down (to a single machine).
- High level collections like Array, Bag, and DataFrame that mimic NumPy, lists, and Pandas but can operate in parallel on datasets that don't fit into memory.
- Low Level schedulers with low-latency to execute task graphs in parallel.



AN EXAMPLE OF SCALABLE ANALYSIS:
BENCHMARK ON A CMIP6 DATA COLLECTION

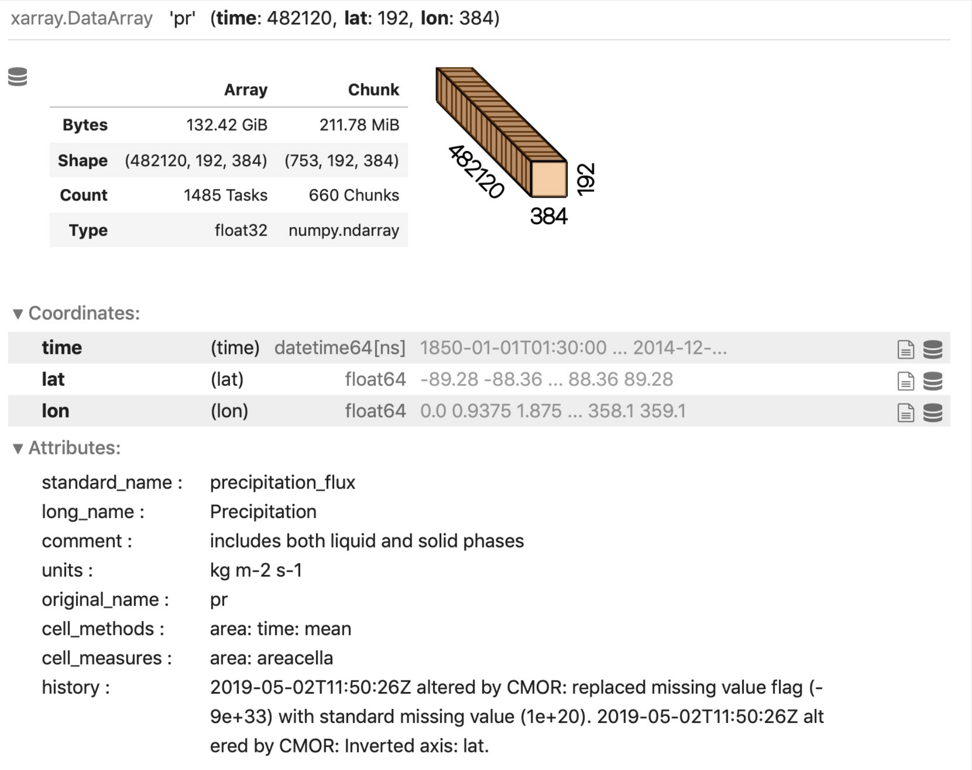
```
import xarray as xr
ds = xr.open_mfdataset(allfiles, chunks={"time":753},combine='by_coords',parallel=True)
len(allfiles)=166               # total number of source NetCDF files.
```



```
xarray.Dataset

▶ Dimensions:        (bnds: 2, lat: 192, lon: 384, time: 482120)
▼ Coordinates:
    time             (time)              datetime64[ns]  1850-01-01T01:30:00 ... 2014-12-...
    lat              (lat)                      float64  -89.28 -88.36 ... 88.36 89.28
    lon              (lon)                      float64  0.0 0.9375 1.875 ... 358.1 359.1
▼ Data variables:
    time_bnds        (time, bnds)        datetime64[ns]  dask.array<chunksize=(753, 2), meta=np....
    lat_bnds         (time, lat, bnds)          float64  dask.array<chunksize=(2920, 192, 2), me...
    lon_bnds         (time, lon, bnds)          float64  dask.array<chunksize=(2920, 384, 2), me...
    pr               (time, lat, lon)           float32  dask.array<chunksize=(753, 192, 384), m...
▶ Attributes:    (44)
```

A model ran by the Alfred Wegener Institute, Helmholtz Centre for Polar and Marine Research, Am Handelshafen 12, 27570 Bremerhaven, Germany (AWI) in native nominal resolutions: atmos: 100 km, land: 100 km, ocean: 25 km, sea Ice: 25 km.

AN EXAMPLE OF SCALABLE ANALYSIS: PRECIPITATION FLUX VARIABLE



xarray.DataArray 'pr' **(time:** 482120, **lat:** 192, **lon:** 384)

	Array	Chunk
Bytes	132.42 GiB	211.78 MiB
Shape	(482120, 192, 384)	(753, 192, 384)
Count	1485 Tasks	660 Chunks
Type	float32	numpy.ndarray

▾ Coordinates:

time	(time)	datetime64[ns]	1850-01-01T01:30:00 ... 2014-12-...
lat	(lat)	float64	-89.28 -88.36 ... 88.36 89.28
lon	(lon)	float64	0.0 0.9375 1.875 ... 358.1 359.1

▾ Attributes:

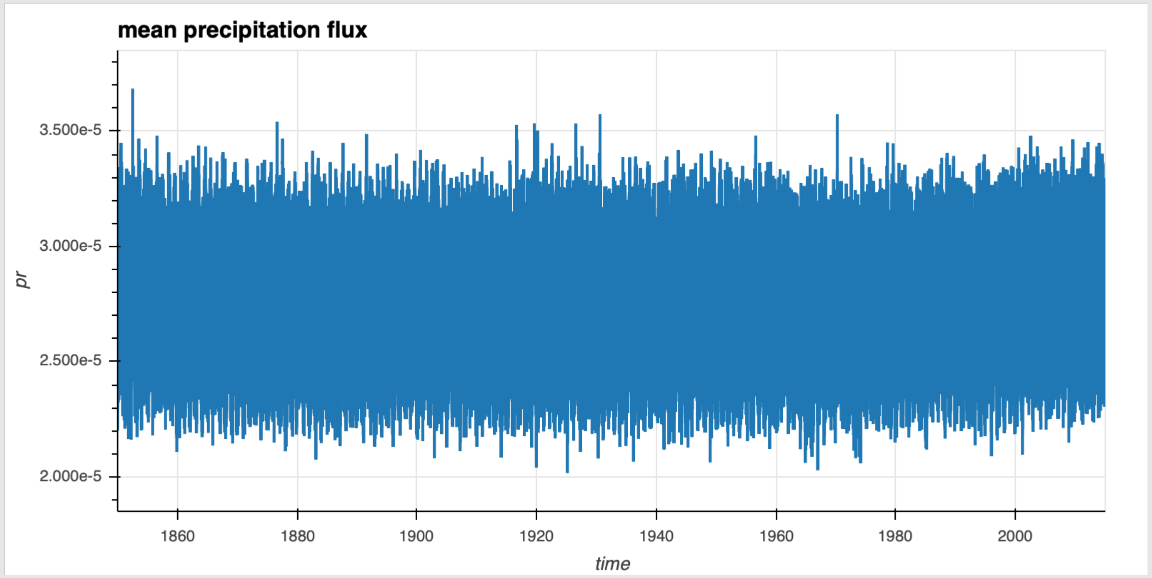
standard_name :	precipitation_flux
long_name :	Precipitation
comment :	includes both liquid and solid phases
units :	kg m-2 s-1
original_name :	pr
cell_methods :	area: time: mean
cell_measures :	area: areacella
history :	2019-05-02T11:50:26Z altered by CMOR: replaced missing value flag (-9e+33) with standard missing value (1e+20). 2019-05-02T11:50:26Z altered by CMOR: Inverted axis: lat.

```
ds.pr.nbytes/1e9=142.1829734#size (GB)
```

The precipitation flux variable has three dimensions. It is a dask.array concatenated over all 166 files in this directory with the total size of 132GiB. The precipitation flux variable is recorded every three hours according to the time stamps above. It's 660 chunks could be processed in parallel over the DASK cluster.

AN EXAMPLE OF SCALABLE ANALYSIS: TIMESERIES OF MEAN PRECIPITATION FLUX

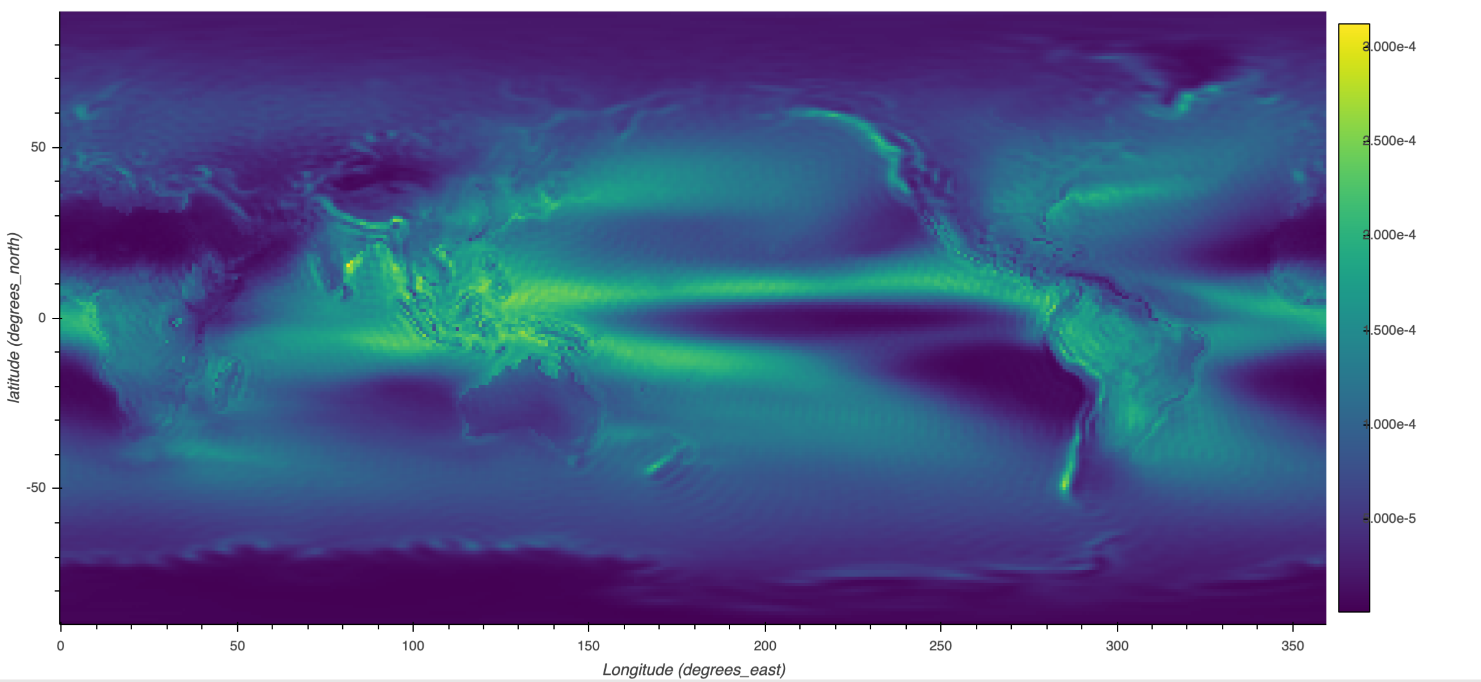
```
pr_mean = ds.pr.mean(dim=('lat','lon’))
pr_full = pr_mean.hvplot(label='full time series dataset from 1850 to
2014', grid=True,title='mean precipitation flux', width=800, height=400)
```



AN EXAMPLE OF SCALABLE ANALYSIS: PRECIPITATION FLUX VARIABILITY

```
pr_std = ds.sel(time=slice("1900-01-01", "2000-12-31")).pr.std(dim='time’)
pr_std.hvplot(colormap='viridis', width=1200, height=550, rasterize=True)
```

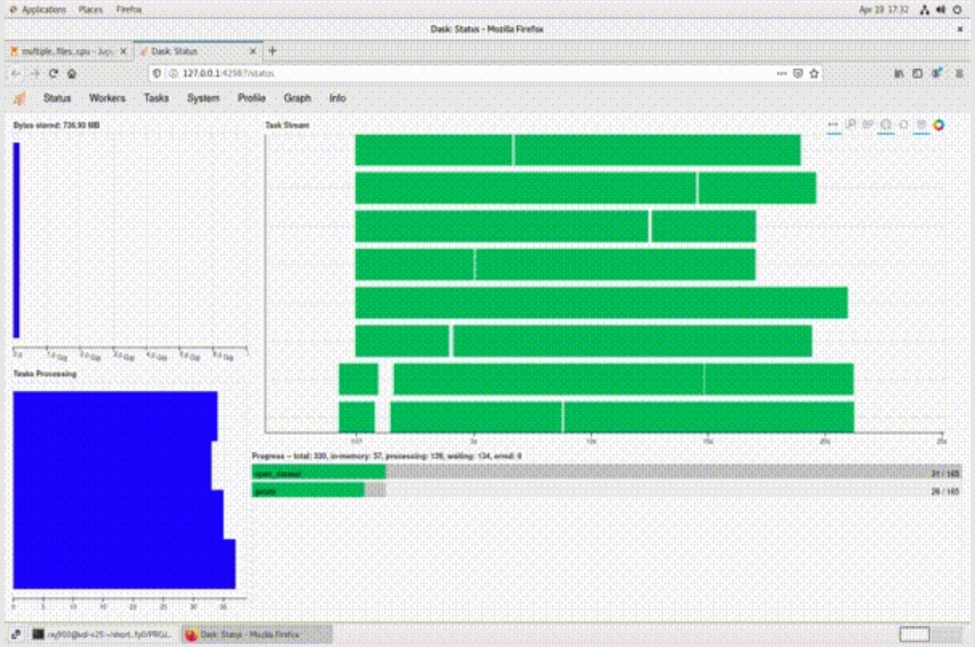
We can examine the natural variability in precipitation flux by looking at its standard deviation over time.



AN EXAMPLE OF SCALABLE ANALYSIS: CODE BLOCK TO ENABLE BENCHMARK

```
from dask.distributed import Client
client = Client(scheduler_file='./scheduler.json’)

ds = xr.open_mfdataset(allfiles, chunks={"time":753},
combine='by_coords',parallel=True)

subds=ds.isel(time=slice(slice_beg,slice_end))

pr_mean=subds.pr.mean(dim=('lat','lon’))
pr_mean.compute()

pr_std = subds.pr.std(dim='time’)
pr_std.compute()
```

AN EXAMPLE OF SCALABLE ANALYSIS:
PARALLEL TASKS WITH THE DASK CLUSTER

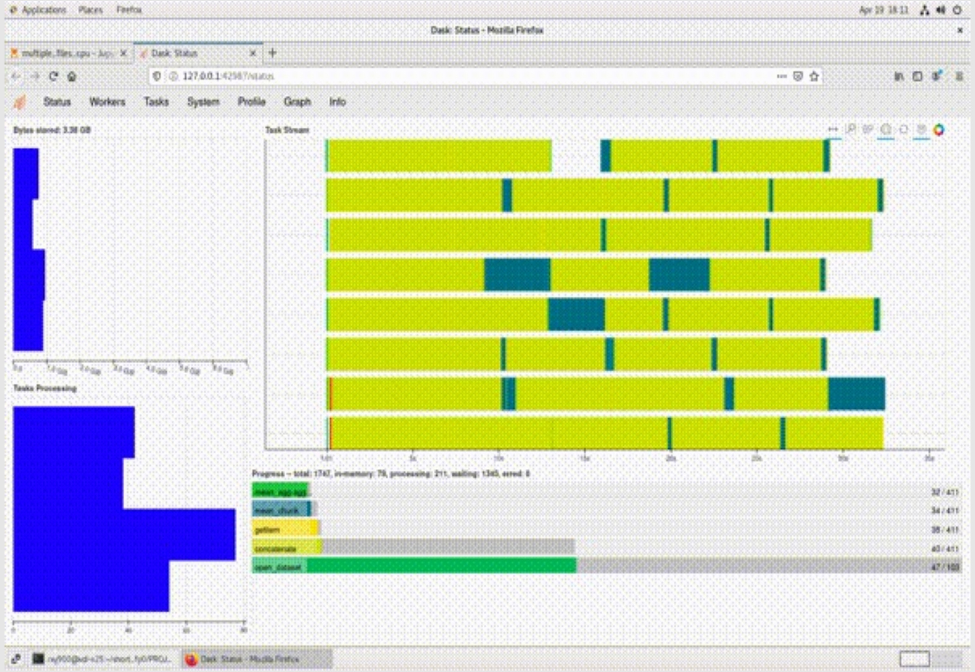


Progress -- total: 330, in-memory: 57, processing: 139, waiting: 134, erred: 0

open_dataset 31 / 165

getattr 26 / 165

`xr.open_mfdataset()`



Progress -- total: 1747, in-memory: 79, processing: 211, waiting: 1345, erred: 0

mean_agg-agg 32 / 411

mean_chunk 34 / 411

getitem 35 / 411

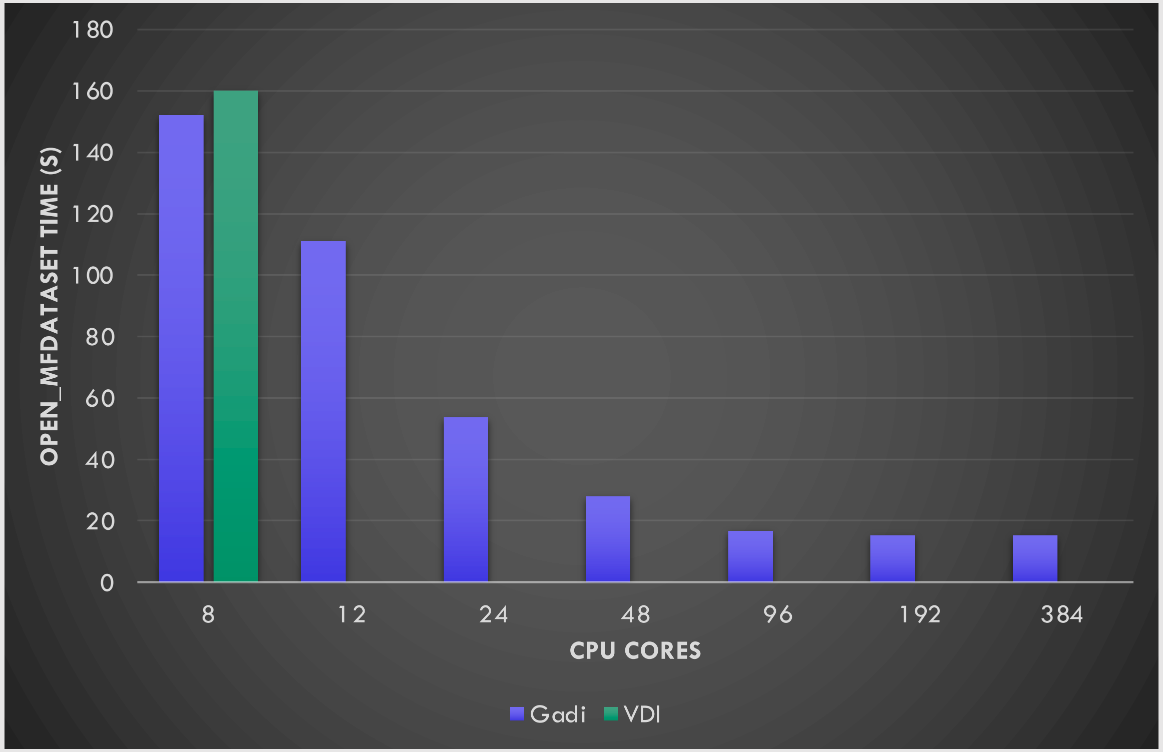
concatenate 40 / 411

open_dataset 47 / 103

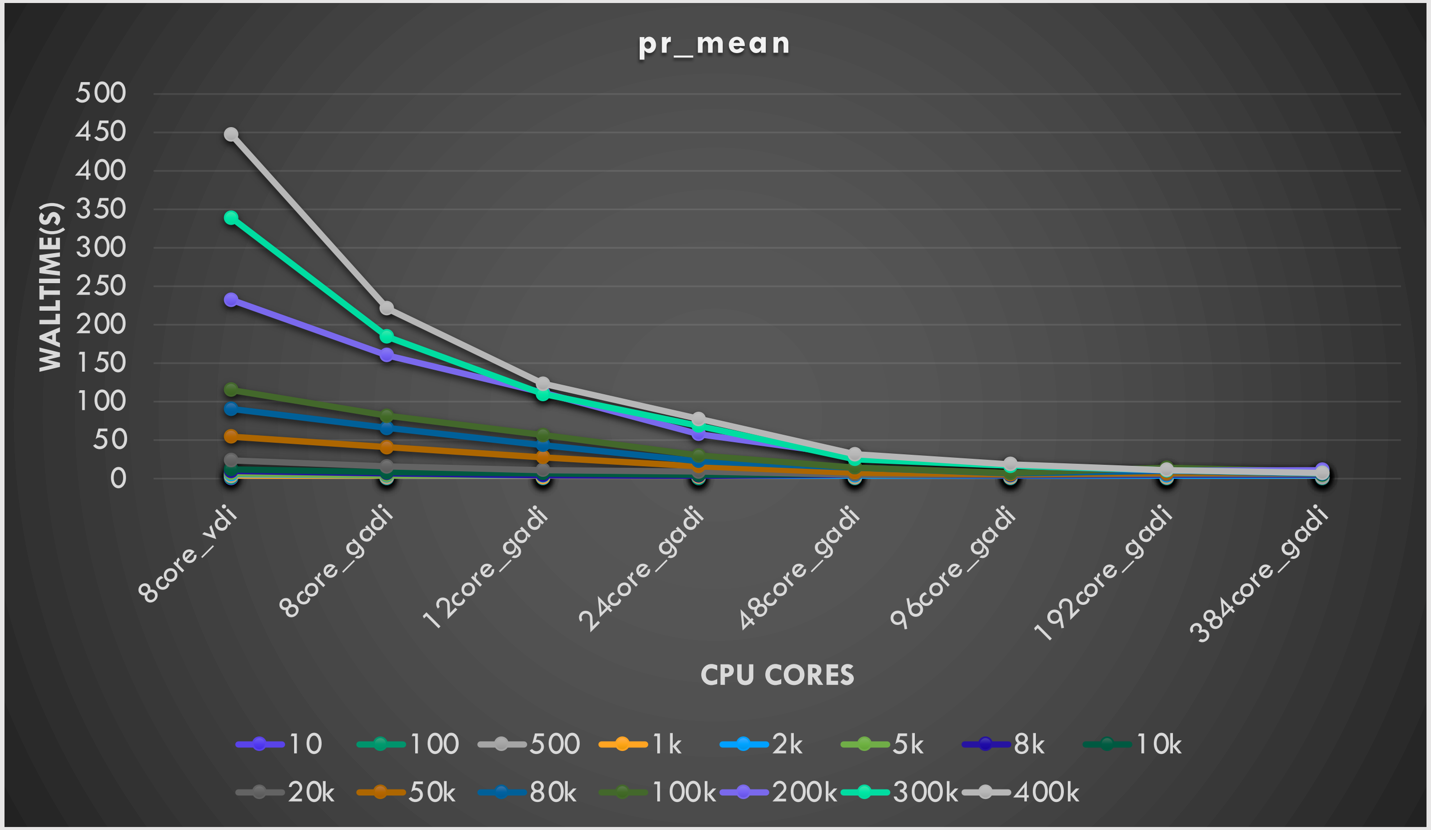
`pr_mean.compute()`

AN EXAMPLE OF SCALABLE ANALYSIS: PERFORMANCE RESULTS OF OPEN_MFDATASET

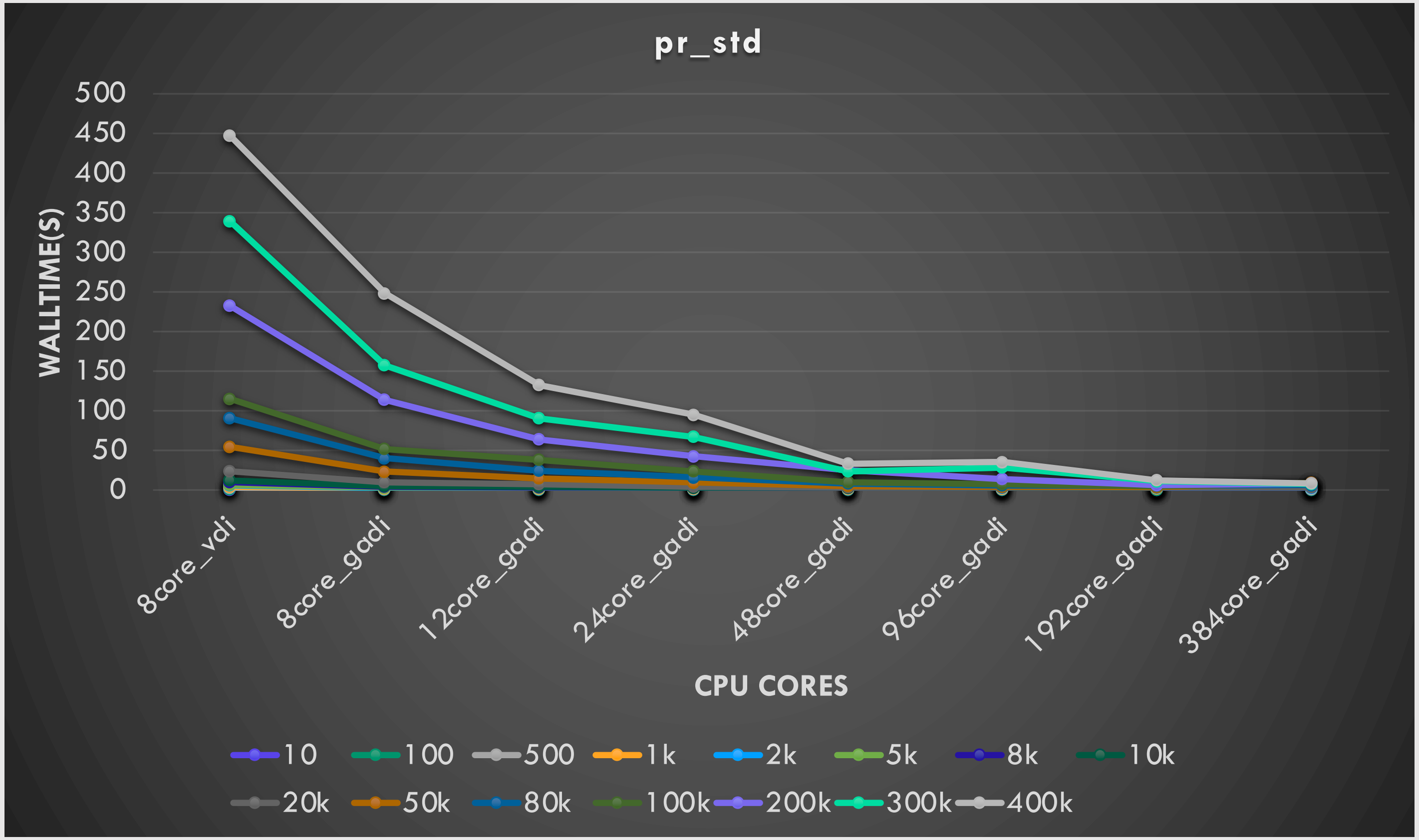
- 2 Threads per Dask worker
- $N_{threads}=N_{cores}$
- CPU resources:
  - VDI: 8 cores in a single node.
  - Gadi: 48 cores/node.



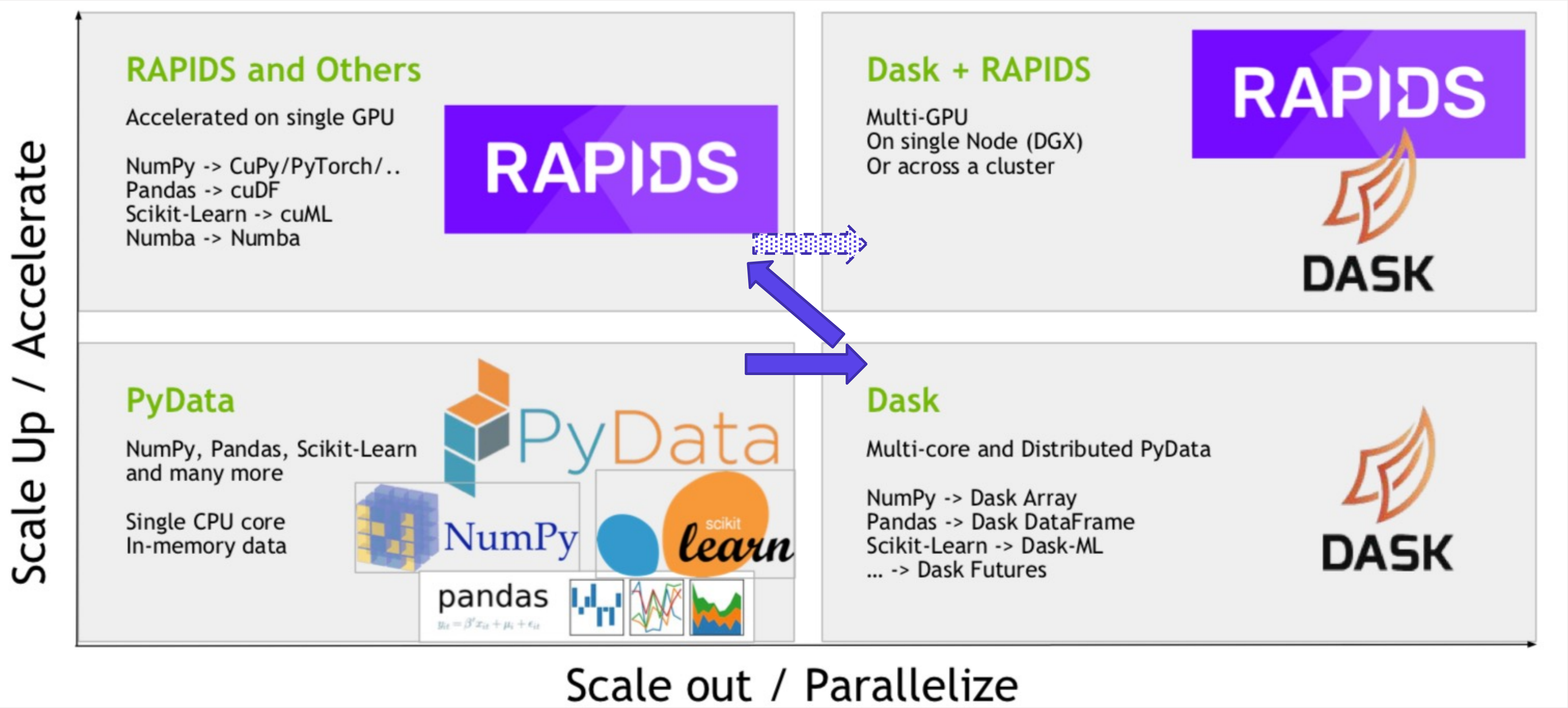
AN EXAMPLE OF SCALABLE ANALYSIS:
PR_MEAN SCALABILITY ON THE SIZE OF TIME SLICES AND CPU



AN EXAMPLE OF SCALABLE ANALYSIS:
PR_STD SCALABILITY ON THE SIZE OF TIME SLICES AND CPU



GPU SUPPORT IN DEVELOPMENT



Scale Up / Accelerate

RAPIDS and Others

Accelerated on single GPU

NumPy -> CuPy/PyTorch/..
Pandas -> cuDF
Scikit-Learn -> cuML
Numba -> Numba

Dask + RAPIDS

Multi-GPU
On single Node (DGX)
Or across a cluster

PyData

NumPy, Pandas, Scikit-Learn and many more

Single CPU core
In-memory data

Dask

Multi-core and Distributed PyData

NumPy -> Dask Array
Pandas -> Dask DataFrame
Scikit-Learn -> Dask-ML
... -> Dask Futures

Scale out / Parallelize

https://developer.download.nvidia.com/video/gputechconf/gtc/2019/presentation/s9797-dask-extensions-and-new-developments-with-rapids.pdf

AN EXAMPLE OF SCALABLE ANALYSIS: SINGLE GPU UTLIZATION

```
Import cupy as cp
from dask.distributed import Client
client = Client(scheduler_file='./scheduler.json’)
ds = xr.open_mfdataset(allfiles,
chunks={"time":753},combine='by_coords',parallel=True)
subds=ds.isel(time=slice(slice_beg,slice_end))
# Additional step to convert data type from numpy to cupy.
 subcds.pr.data = cp.asarray(subcds.pr.data)  # extra convert time cvttime.
pr_mean=subds.pr.mean(dim=(‘lat’,‘lon’))
pr_mean.compute()                              # meantime
pr_std = subds.pr.std(dim=‘time’)
pr_std.compute()                               #stdtime
```

AN EXAMPLE OF SCALABLE ANALYSIS: GPU BENCHMARK RESULTS

Walltime(s)	12 CPUs	12 CPUs	1 GPU	1 GPU	1 GPU		
size	meantime	stdtime	meantime	stdtime	**cvttime**	N_{pr_mean}	N_{pr_std}
10	0.14	0.08	1.63	0.15	**0.10**	1	1
100	0.33	0.34	0.00	0.00	**0.29**	1	1
500	1.31	1.29	0.00	0.01	**2.28**	2	2
1000	2.79	2.17	0.00	0.04	**2.52**	1	1
2000	2.63	2.42	0.00	0.08	**3.70**	1	2
5000	2.77	2.38	0.00	0.21	**8.59**	3	4
8000	2.98	2.20	0.31	0.03	**9.35**	3	4
10000	4.14	4.39	0.00	0.40	**11.14**	3	3
20000	6.28	6.15	0.00	0.81	**23.47**	4	4
50000	14.88	12.02	0.00	2.02	**51.44**	3	4
80000	23.95	19.54	0.00	3.22	**102.01**	4	5
100000	30.44	29.22	1.17	3.65	**142.17**	5	5

Single GPU can accelerate the data analysis workflow with heavy compute operations.

PATHWAY TOWARDS MULTIPLE GPUS

RAPIDS
Open GPU Data Science

DASK

- Set up Dask LocalCUDACluster via dask-cuda library to distribute tasks over multiple GPUs.
- No direct way to read NetCDF files to xarray.Dataset crossing multiple GPUs.
- Convert xarray.Dataset to Dask-cuDF (can read csv, json, orc, parquet directly).
- New implementations & developments in DASK+RAPIDS are on the way.

SUMMARY

NCI set up a data analysis environment by integrating both python software stack and NCI compute resources. It could help users to efficiently process large scale datasets in a scalable way with powerful NCI compute resources.

NCI Gadi User Guide
https://opus.nci.org.au/display/Help/Gadi+User+Guide

NCI VDI User Guide
https://opus.nci.org.au/display/Help/VDI+User+Guide

NCI Data Analysis Environments
https://opus.nci.org.au/display/Help/Data+Analysis+Environments

Xarray tutorial
https://github.com/NCI-data-analysis-platform/examples-dask.git

Dask tutorial
https://github.com/NCI-data-analysis-platform/examples-xarray.git

Please connect various NCI projects to access the data sources for above tutorials.

Thank you.