Brookhaven National Laboratory

U.S. DEPARTMENT OF ENERGY

PanDA/Dask Integration

Paul Nilsson

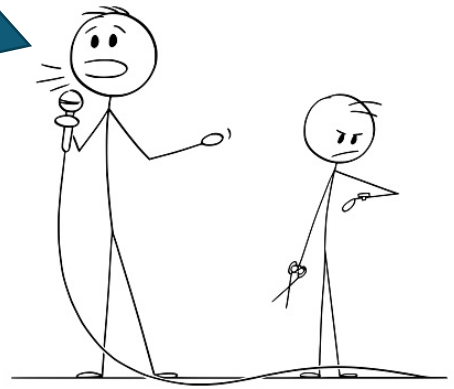
March 3, 2023

NPPS Group Meeting

@BrookhavenLab

Introduction

- As part of the ATLAS Google Project, a “Dask submitter” is being developed for Harvester to achieve PanDA/Dask integration
- The goal is to give users possibility to use Dask on Google resources via PanDA
- The solution is being tested on GCP resources, on which a Kubernetes cluster has been set up
- But what is actually **Dask** about?



dask

Wikipedia

Dask is a flexible open-source Python library for parallel computing. Dask scales Python code from multi-core local machines to large distributed clusters in the cloud. Dask provides a familiar user interface by mirroring the APIs of other libraries in the PyData ecosystem including: Pandas, scikit-learn and NumPy.

- Dask makes it “easy” to scale common Python libraries (NumPy, pandas, scikit-learn, ..)
- Can be used to parallelize Python code
- Composed of two parts
  - **Dynamic task scheduling** optimized for computation
  - **“Big Data” collections** like parallel arrays, dataframes, and lists
- Scales up to thousands of cores
- Dask can be deployed on anything from laptops to HPCs
  - In the PanDA/Dask project, it is deployed on Kubernetes
- For more info, see https://www.dask.org

PanDA/Dask Integration

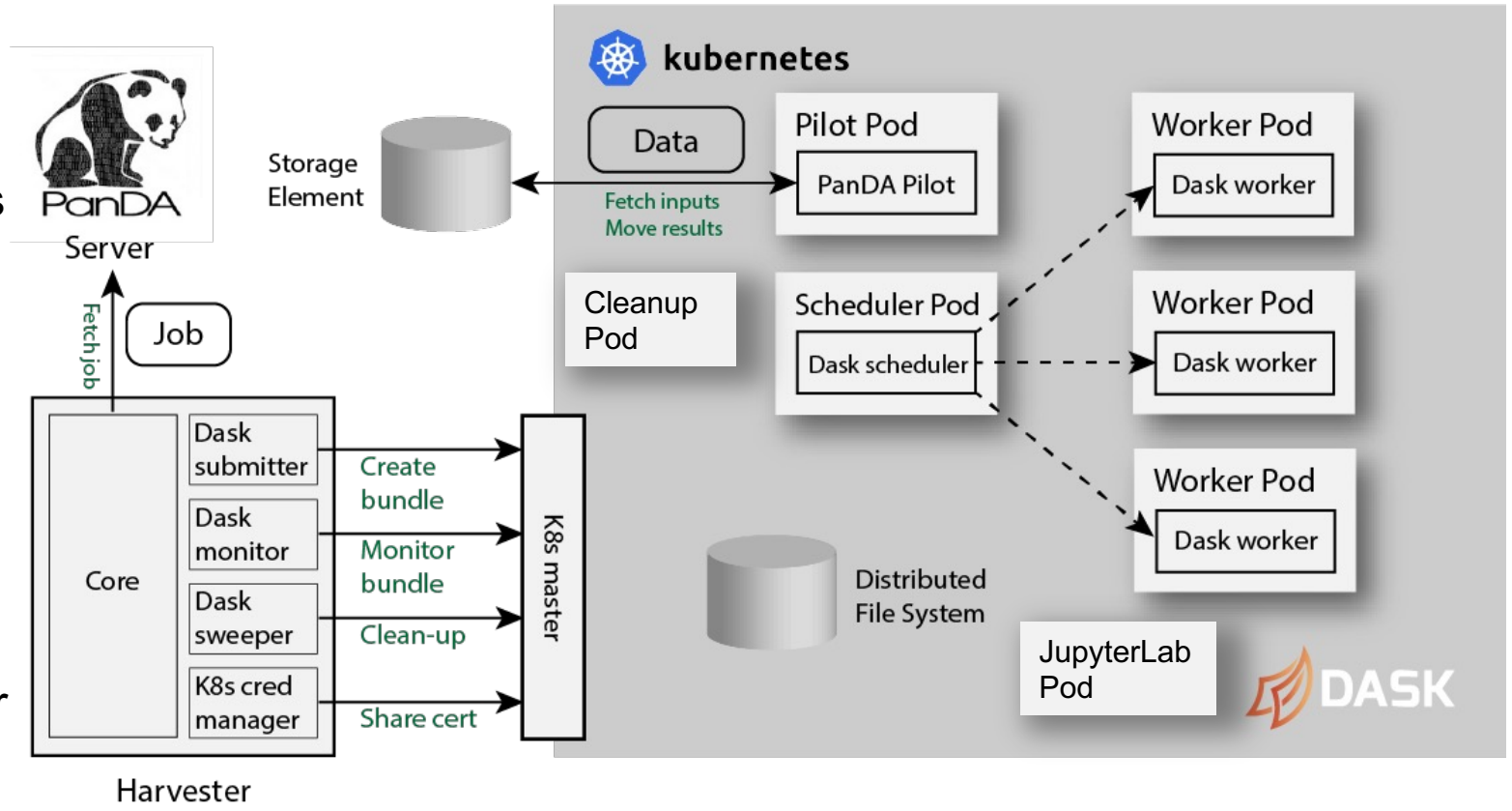
- In our case, what will the user be able to do?
  - A user can [soon] use prun to either create an interactive jupyter session on GCP with a requested number of dask workers (and relevant tools) available, or to submit a dask script that will run on the requested resources much like a grid job
- Preliminary example (with no input data but using secrets)
  - `prun --outDS user.username.`uuidgen` --exec dask_script.py --site GOOGLE_DASK --noBuild –useSecrets` (creates a task in PanDA)
  - Assuming existing secrets dictionary (user name + password) and a user job script (currently not executed – only testing interactive mode at the moment)
  - prun changes for dask job submission are pending users - dask cluster specifics are currently hardcoded but can easily be added to prun when actually needed
- A single job is created from the task which is picked up by Harvester, and handled by dask submitter which in turn prepares a session / runs the code on GCP

Setup

- Kubernetes cluster
  - Private cluster setup on GCP (isolated from the internet, except for authorized networks)
  - Test pool with machine type e2-medium, ~minimal configuration (few nodes, to be increased when it makes sense)
  - Shared file system (Filestore) added to nodes
- PanDA queue
  - A new PanDA queue, GOOGLE_DASK, was created, based on GOOGLE100 queue
- Dev version of Harvester
  - Harvester ID CERN_central_dask
  - Running on VM (aipanda003@CERN)
  - Harvester sets up single-user dask session on cluster when a job is created
    - I.e. one session per job, nothing running when there are no jobs; of course, multiple users can run at the same time

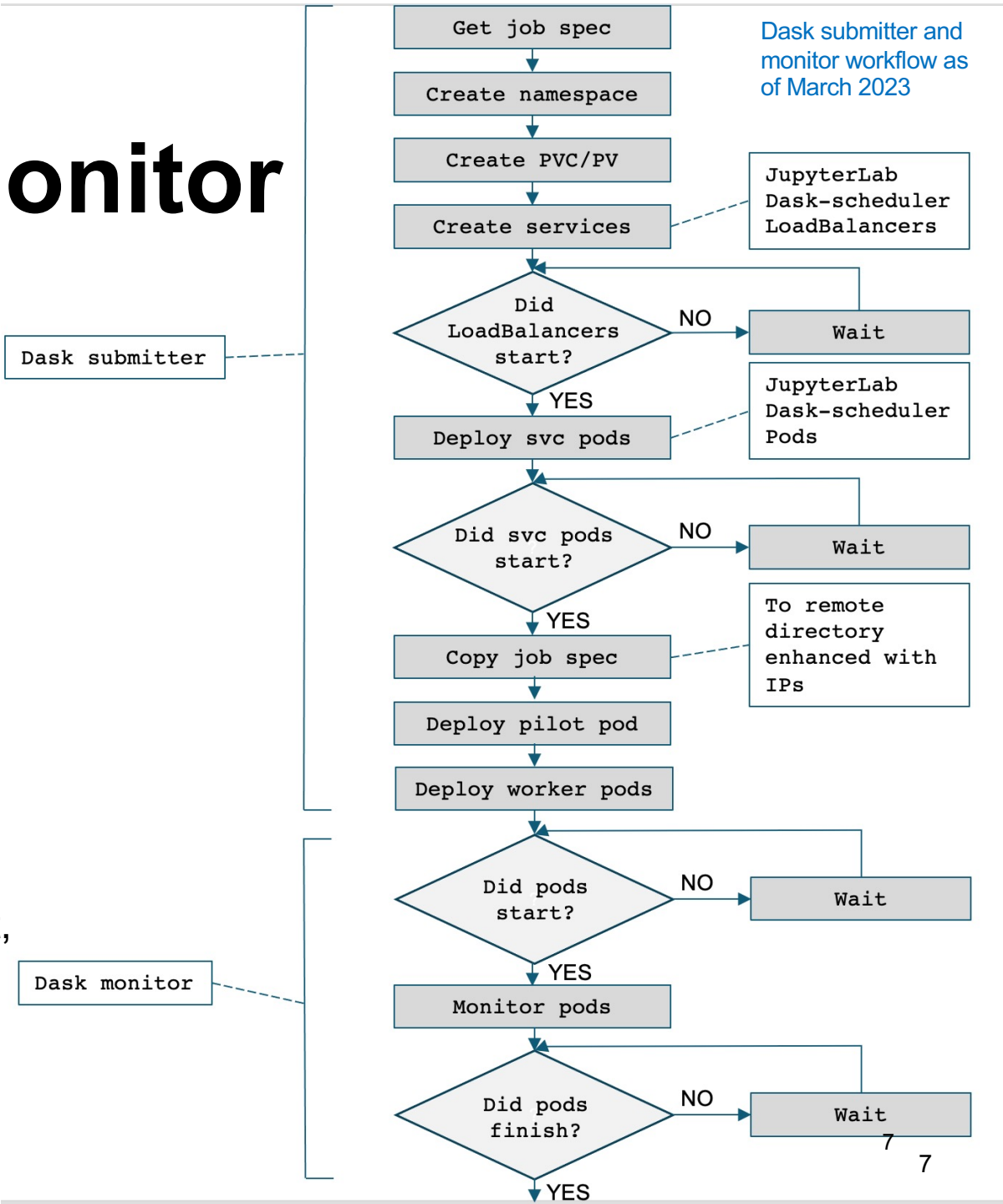
Workflow Overview

- Harvester fetches job from PanDA server
- Job is handed over to Dask submitter which starts multiple pods (pilot, dask scheduler and workers, and optionally jupyterlab)
- Dask monitor awaits startup of pilot and dask workers, keeps track of pod statuses [*in testing*]
- Pilot fetches any input and communicates with server
- A shared file system is used for proxy/certs and work directories (for pilot and job definition)
- Dask sweeper will be used for cleanup [*not fully implemented yet*]



Dask submitter and monitor

- Normally – in Harvester – pods would be started from a certain “submitter”, while a corresponding “monitor” would wait for all pods to start
- In our case, some pods and services must be started in correct order, before some other pods can be started
  - Dask scheduler (and optionally Jupyterlab in interactive sessions) pod starts first
    - When running, the corresponding IP number is extracted and added to the job definition (which is then copied to the remote shared file system)
  - Pilot and dask worker pods are deployed next
  - Monitor will wait for pilot and dask workers to start, and pods to finish
    - Can also terminate pods when out of time
    - To be decided: when to issue time-out ..



7

Images

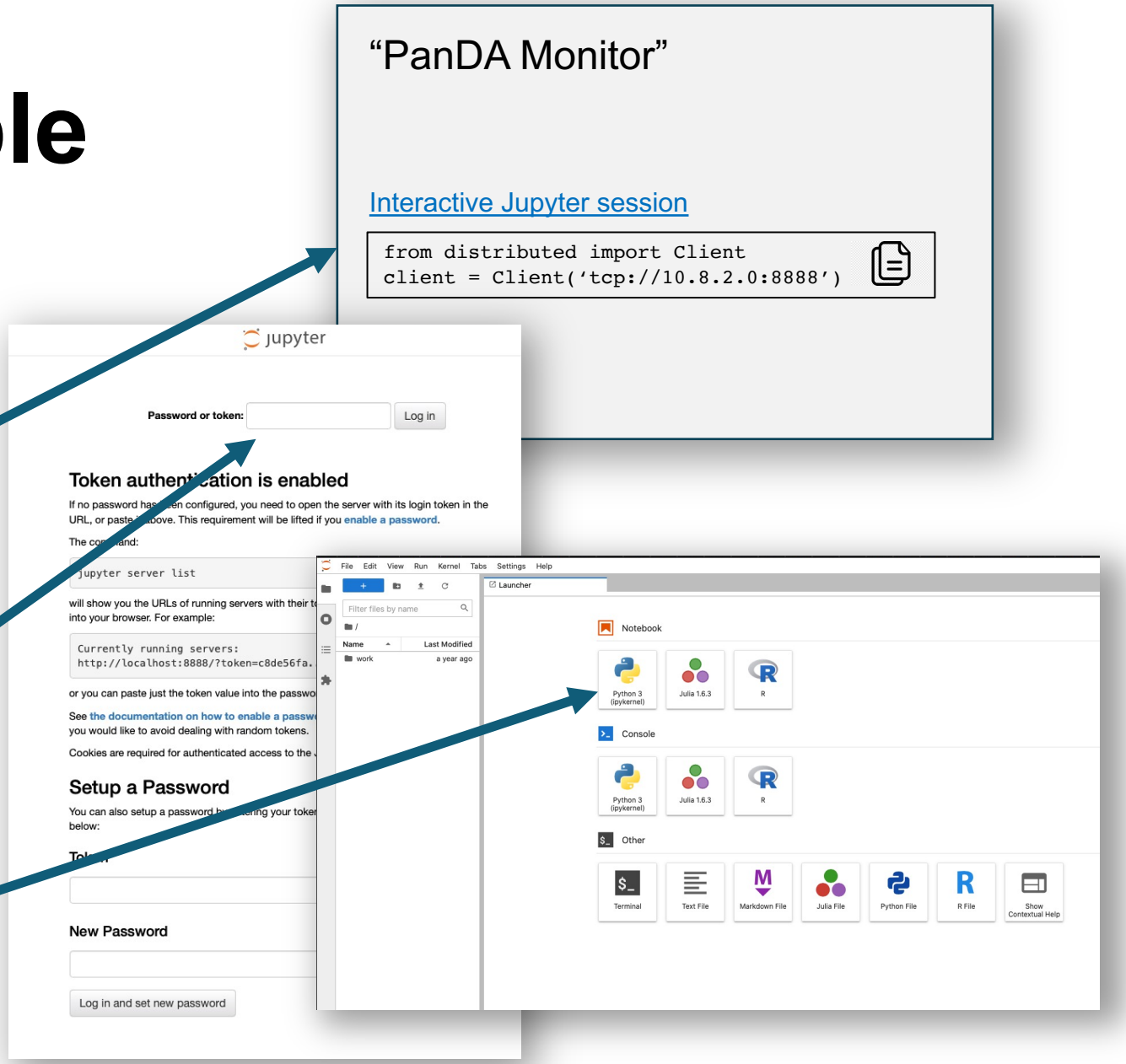
- Dask submitter uses the following images

Image name	Description	Size (MB)
dask_scheduler	Dask scheduler image based on continuumio/miniconda3:4.8.2 (to be updated), https://github.com/PalNilsson/dask-scheduler	424
dask_worker	Dask worker image based on continuumio/miniconda3:4.8.2 (to be updated), https://github.com/PalNilsson/dask-worker	414
jupyterlab	JupyterLab image based on datascience-notebook which comes with dask support (to be updated), https://hub.docker.com/r/jupyter/datascience-notebook/tags (several options to choose between at https://jupyter-docker-stacks.readthedocs.io/en/latest/using/selecting.html)	1400
pilot	Pilot image based on rucio-clients (1.30.5) and pilot3 (latest dev),(to be optimized) https://github.com/PalNilsson/pilot-image	267
remote_cleanup	Remote cleanup image, https://github.com/PalNilsson/remote-cleanup	0.8

- Images are currently built on aiatlas025/034@CERN (ATLAS software development machines)
- All images uploaded to Google Artefact repository for fast access and avoids docker hub
  - Note: still using docker to build images, but planning on switching to podman asap (an earlier attempt failed since there was a problem installing it) which is allegedly a full alternative to docker (worth checking out)

Interactive Example

- Interactive job created with prun (command on slide 4)
- When dask scheduler and jupyterlab IPs are known, pilot sends them to PanDA server
  - PanDA monitor gets the info from DB and displays info NOT YET
- User password given to prun is used to login to jupyter session
  - *Will be obscured from any logs*
- User starts python 3



Interactive Example

- Import and connect to dask scheduler
  - Cut and pasted from job page
  - Warnings are currently ignored
    - Packages will be updated
- Run your code ..

File Edit View Run Kernel Tabs Settings Help

Filter files by name

/

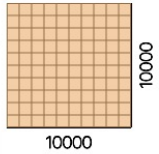
Name	Last Modified
work	a year ago
Untitled.ip...	a minute ago

Untitled.ipynb

Code

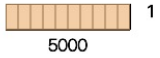
```
[1]: from distributed import Client

[2]: client = Client('tcp://10.8.2.120:8786')

/opt/conda/lib/python3.9/site-packages/distributed/client.py:1128: VersionMismatchWarning: Mismatched versions found

+-------------+---------------+---------------+---------------+
| Package     | client        | scheduler     | workers       |
+-------------+---------------+---------------+---------------+
| blosc       | None          | 1.9.2         | 1.9.2         |
| cloudpickle | 2.0.0         | 2.0.0         | 1.6.0         |
| dask        | 2021.11.1     | 2021.07.2     | 2021.07.2     |
| distributed | 2021.11.1     | 2021.07.2     | 2021.07.2     |
| lz4         | None          | 3.1.3         | 3.1.3         |
| msgpack     | 1.0.2         | 1.0.0         | 1.0.0         |
| numpy       | 1.20.3        | 1.21.1        | 1.21.1        |
| pandas      | 1.3.4         | 1.3.0         | 1.3.0         |
| python      | 3.9.7.final.0 | 3.8.0.final.0 | 3.8.0.final.0 |
| toolz       | 0.11.2        | 0.11.2        | 0.11.1        |
+-------------+---------------+---------------+---------------+
Notes:
-  msgpack: Variation is ok, as long as everything is above 0.6
  warnings.warn(version_module.VersionMismatchWarning(msg[0]["warning"]))
distributed.client - ERROR - Failed to reconnect to scheduler after 30.00 seconds, closing client
_GatheringFuture exception was never retrieved
future: <_GatheringFuture finished exception=CancelledError()>
asyncio.exceptions.CancelledError

[3]: import dask.array as da

[4]: x = da.random.random((10000, 10000), chunks=(1000, 1000))

[5]: x

[5]:
```

	Array	Chunk
Bytes	762.94 MiB	7.63 MiB
Shape	(10000, 10000)	(1000, 1000)
Count	100 Tasks	100 Chunks
Type	float64	numpy.ndarray




```
[6]: y = x + x.T

[7]: z = y[::2, 5000:].mean(axis=1)

[8]: z

[8]:
```

	Array	Chunk
Bytes	39.06 kiB	3.91 kiB
Shape	(5000,)	(500,)
Count	430 Tasks	10 Chunks
Type	float64	numpy.ndarray




```
[9]: z.compute()

[9]: array([0.98971878, 1.01004025, 0.99557568, ..., 1.00180369, 1.00692376,
```

Simple 0 1 Python 3 (ipykernel) | Idle

Status and Immediate Plans

- Basic functionality mostly implemented and mostly works
  - Pilot pod added this week - currently in testing
    - Needed for communicating IPs and sending job status to server (file transfers not attempted yet)
    - Pod starts nicely, but needs proxy to communicate to server (proxy not copied yet; now debugging ..)
  - Sort out remaining technical issues and questions (incl. how to end interactive session fairly and properly)
  - Finish implementing dask sweeper
    - Currently minimal implementation (or Harvester complains), so cleanup is occasionally manual ..
  - Time line for “basic functionality implemented and working”: April 2023
- Update images
  - Not needed for current tests, but probably good to do anyway