Machine Learning eResearch Platform (MLeRP)

How serving a subcommunity creates different design decisions

Mitchell Hargreaves - Monash University

NCRIS National Research Infrastructure for Australia
An Australian Government Initiative

Australian Research Data Commons

MONASH University

QCIF

The University Of Queensland
AUSTRALIA

Co-Authors

- Dr. Chris Hines - Monash University
- Dr. Slava Kitaeff - Monash University
- Miss Kiowa Scott-Hurley - formerly Monash University
- Mr. Oliver Cairncross - University of Queensland

Project Objectives

- Machine Learning development involves:
  - Code development
  - Debugging
  - Labelling
  - Model training
  - Model inference
- Some of this is well suited towards a batch job environment
  - But some parts need interactivity
- The objective: Build a platform to support all stages of ML development

Interactive - GPU backed notebooks

Pros

- Ideal for dataset exploration or development
- Familiar environment (many tutorials take this form)
  - Google Collab
- Low barrier to entry
  - Cloud platforms allow anyone with a Google account
- Highly responsive (once allocated)
- Short iterations

Cons

- Compute sits idle during development
- Inefficient hardware use
- 1 GPU/user leads to limited seats
  - Long wait times
  - Requires user to be at the device once allocated

Non interactive - Job submission

Pros

- Ideal for mature workloads
- Access to large and flexible amounts of compute
- Efficient use of hardware

Cons

- Unknown wait time
- High learning curve
- Workloads must be refactored to scale well to HPC hardware
- Must wait in queue, even for simple tests
- Long iterations

Target User

- Prioritise interactivity
  - Many come from Google Collab
- Some need for batch processing
- Little prior experience with HPC
- Need control over Python environment

Our approach

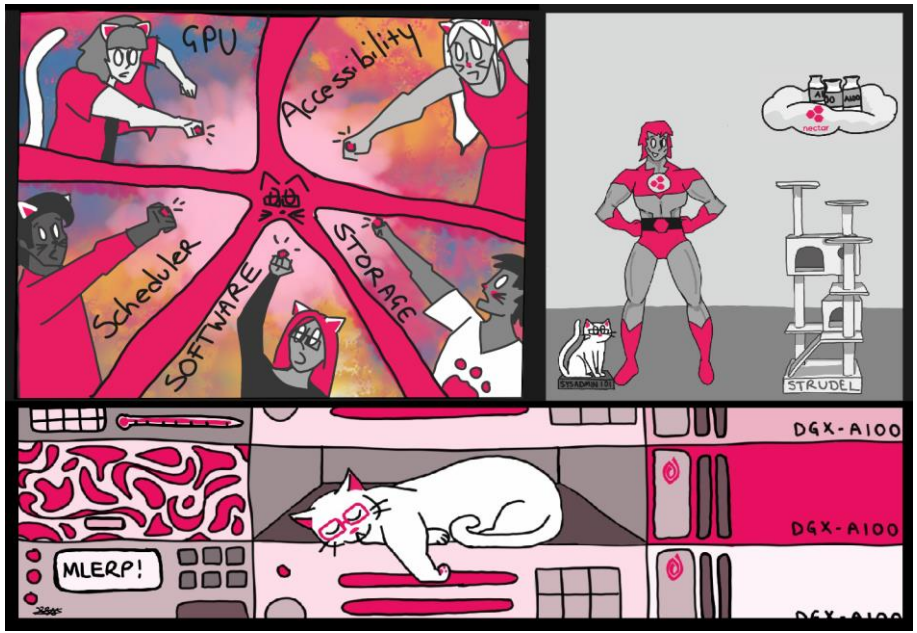
- Create a middle ground
- Support Python first
- Aim for a premium notebook experience
  - Persistent storage
  - Familiar package control with conda
- Aim to share GPU compute effectively
- Prioritise accessibility
  - Allow users to take on new concepts when they’re ready

Why don’t we have both?

What if we a notebook could interactively submit to a queue?

We sacrifice some responsiveness, but this let’s us:

- Ease transition between development and training batch scripts
- Scale up and down compute
- Release compute once finished
- Increase uptime on hardware
- Serve more users and reduce initial wait times
- Empower users to change requested compute from within the notebook



Docs

https://docs.mlerp.cloud.edu.au/

How do we do this?

Docs

Master Process

Worker Processes

How do we do this?

Define a cluster of workers with whatever requirements we wish

In [8]:

```
from dask_jobqueue import SLURMCluster
from distributed import Client
cluster = SLURMCluster(
    memory="190g", processes=1, cores=20, job_extra_directives=["--gres=gpu:1"], nanny=False
)

cluster.scale(1)
client = Client(cluster)
```

Then submit a python function along with its inputs to the cluster

In [9]:

```
trainloader_future = client.scatter(trainloader)
client.submit(train, trainloader_future, test=True).result()
```

Out [9]:

```
2.3033061027526855
```

The result of the job is returned to the notebook by calling '.result()'

Full tutorial: https://docs.mlerp.cloud.edu.au/tutorials/dask-pytorch.html

Docs

Offloading efficiency experiment

Aim:
Determine the impact of offloading to remote hardware

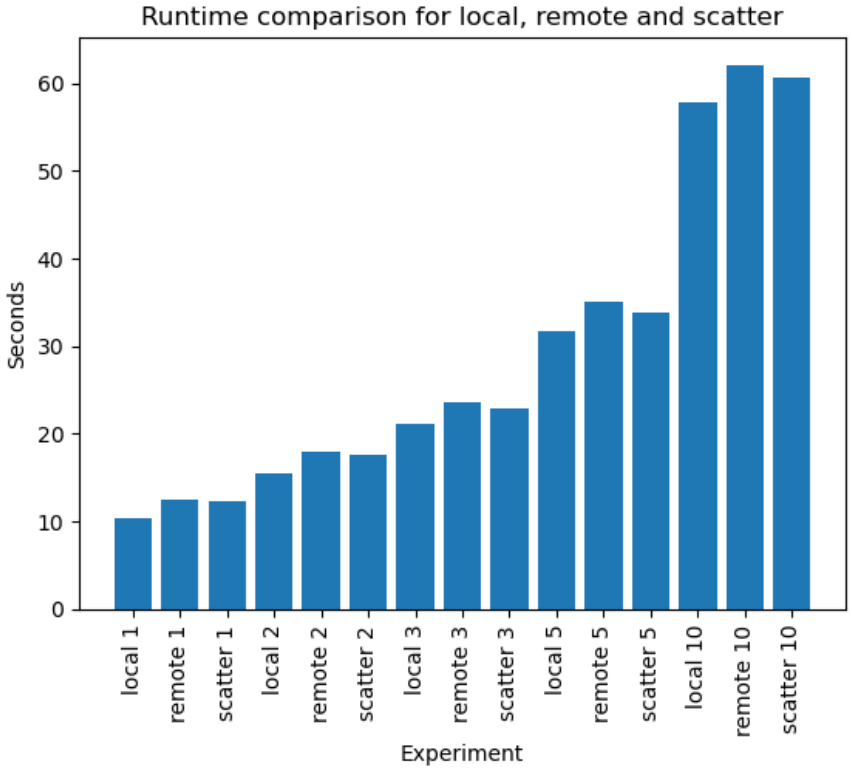
Method:
Train a model (ResNet18) with on a simple classification problem (CIFAR10).

Time how long it takes to train n epochs.

Repeat for local hardware, remote hardware, and remote hardware when ‘scattering’ the dataset

Results:
Time to execute is not dramatically different when holding epoch constant

Conclusion:
Offloading can be effectively used without significant impacting the performance



Full tutorial: https://docs.mlerp.cloud.edu.au/tutorials/dask-pytorch.html

Benefits

- Dask gives you the flexibility to write code with lazy execution or asynchronous code
  - New optimisations are now possible
    - Run your evaluation while next training loop begins

- Users can now request different compute for different cells
  - Use many small CPU workers for preprocessing
  - Use a job with large memory for analysis
  - Use a GPU backed process for training

Docs

Trade Offs

- We have increased the complexity of the notebook code
  - Though we will still offer ‘traditional’ alternatives

- Oversubscription can lead to new fail states
  - What happens if you send off a job but the cluster is at capacity?
  - We need to find the best balance between capacity and responsiveness

Docs

Meeting users where they're at

We offer different Qualities of Service for users of all kinds

Docs

BigCats Partition

CPU only notebooks that control Dask workers

Ideal for:

- Data processing
- Rapid and flexible iteration during development
- Experimenting with techniques

Batch submission

Ideal for:

- Model training
- Heavy duty processing
- Hyperparameter sweeps

HouseCats Partition

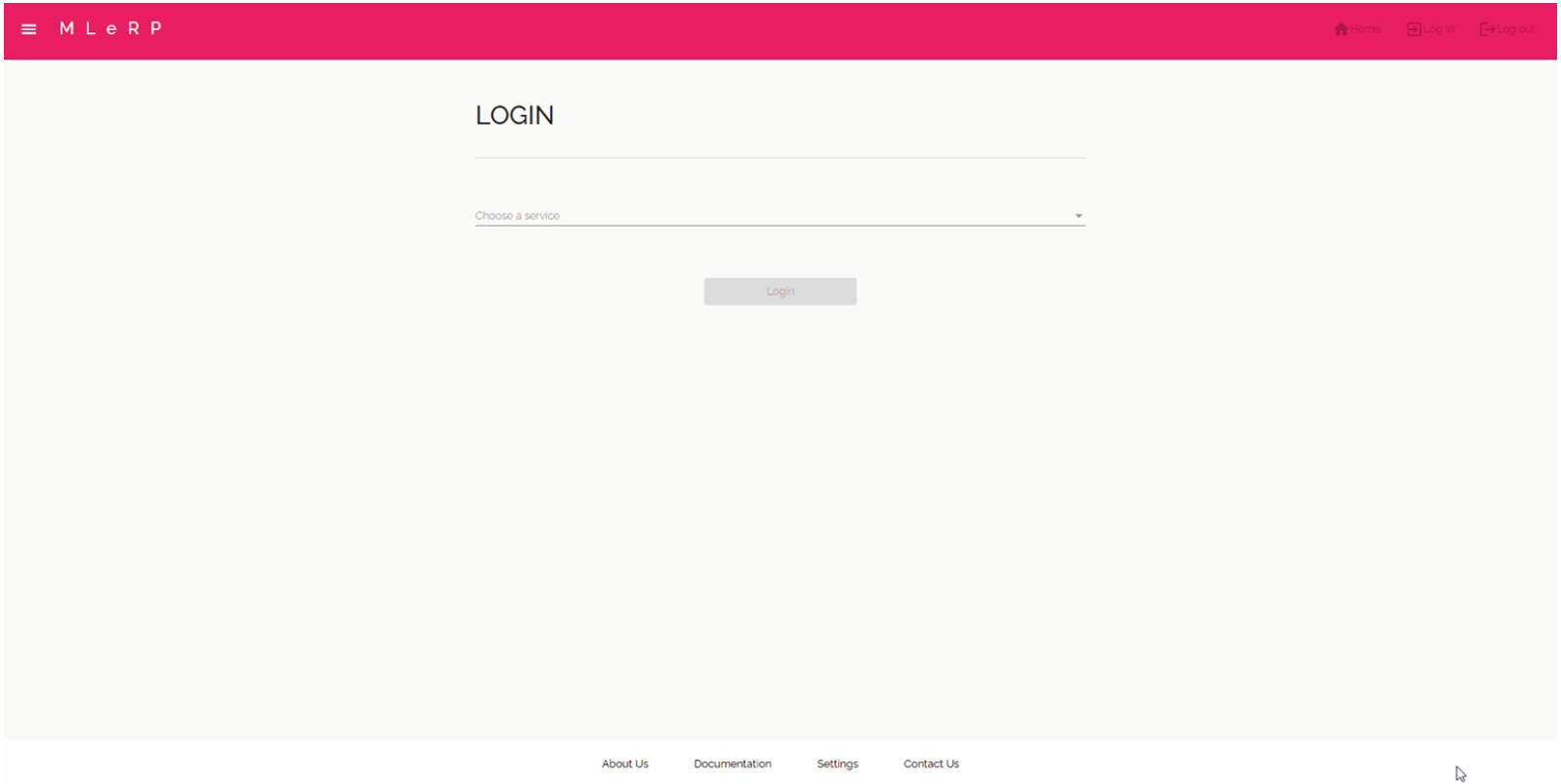
Notebooks backed by GPU reservations

Ideal for:

- Data exploration
- Data visualization
- New users who just want to get started with minimal setup

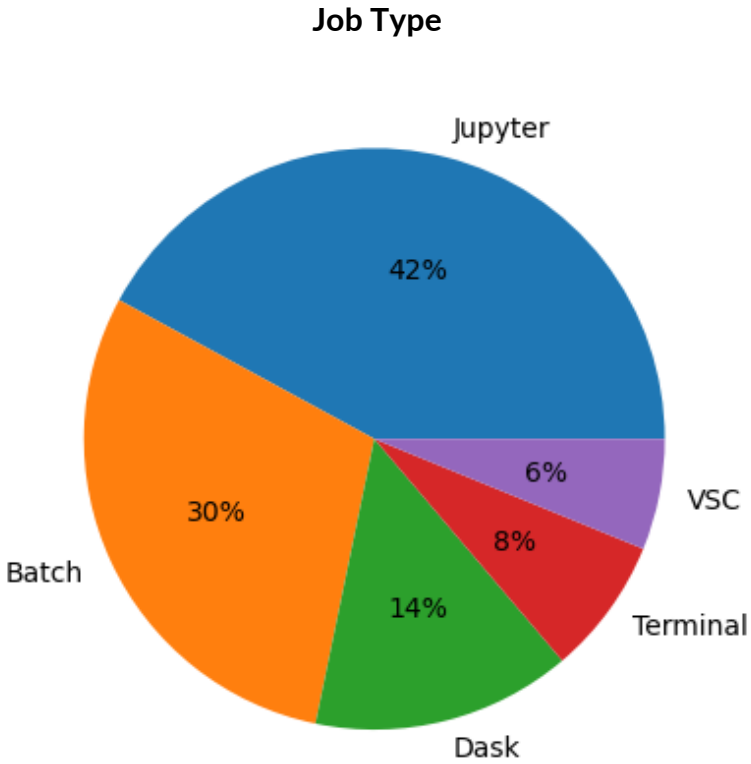
https://docs.mlerp.cloud.edu.au/compute.html

Strudel2


MLeRP
Home
Log in
Log out
LOGIN
Choose a service
Login
About Us
Documentation
Settings
Contact Us

Reception and Learnings

- Users like interactivity!
- Jupyter are favoured as expected
- Batch usage shows users requirements are maturing enough to do training runs

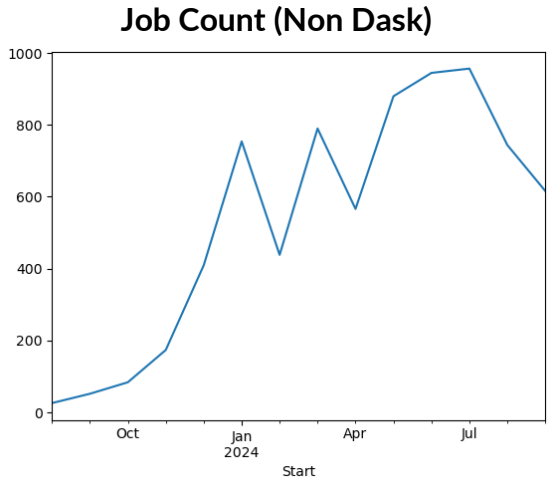


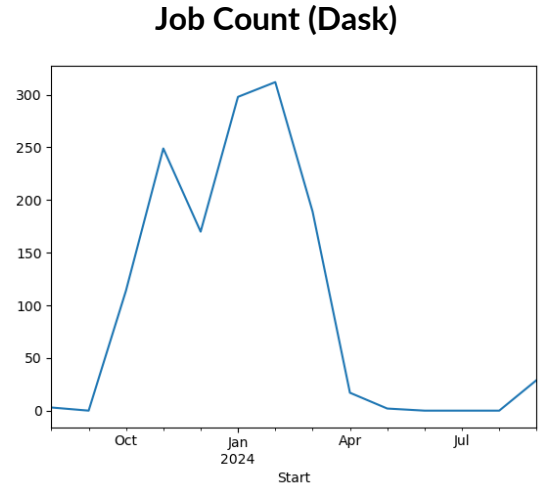
Reception

- Usage is (mostly) increasing over time

- Dask offloading uptake is low
- The peak shows that some users tried it when they first joined, then abandoned it

- Some users report that higher level frameworks are not compatible with the offloading design
  - Concessions need to be made if we are to support these use cases
  - E.g. hugging face or LLM libraries





Antipatterns

Docs

- Reservations have become the default way to use the platform for all users, not just newer users
- Only a rounding error of users actually offload their work
  - But some trial their code with CPU-only notebooks!
  - There is not enough incentive to encourage users to learn dask/submitit
- Users using tmux in interactive jobs like terminals to train models rather than batch submission
- Users installing packages directly in the notebook expecting google collab behaviour
  - !pip install ...

Next Steps

- Not limited by compute yet…
  - How will MLeRP look when this changes?
- Do we push harder to encourage async GPU work?
  - Or will users start to pick it up as demand increases as we intended?
- Or do we observe user behaviour and reinforce the system to empower them to use the system in the way they want to?

Docs

Questions?

https://docs.mlerp.cloud.edu.au/

Meeting users where they're at

We offer different Qualities of Service for users of all kinds

Docs

BigCats Partition

Lion QoS
- 24 hour walltime / 4 job limit
- Job types
  - CPU interactive notebooks and terminals
    - Call out to the Cheetah QoS for GPU
  - Heavy workload batch submissions

Cheetah QoS
- 30 minute walltime / 20 job limit
- Job types
  - Dask workers

Panther QoS
- 7 day walltime / 1 job limit
- Job types
  - CPU interactive notebooks and terminals

HouseCats Partition

Tabby QoS
- 12 hour walltime
- 1 job limit
- Job types
  - GPU interactive notebooks and terminals

https://docs.mlerp.cloud.edu.au/compute.html