

Simplifying HPC and ML Application Deployment across the Computing Continuum

Jack Brassil

Dept. of Computer Science
Princeton University

Goal – Support mix of HPC and AI/ML workloads across platforms

- Diverse population of computational scientists needing Python horizontal and vertical scaling
- AI/ML research and training needs growing rapidly in Humanities and Social Sciences
- Some degree of cluster hardware divergence today (e.g., dedicated campus system for LLM research)
- Need to rationalize different software frameworks in use across campus; hard to train, hard to support, expensive
- Investigating *RAY* to provide a “single” (primary) framework to support on the widest range of target platforms

This material is based upon work partially supported by a National Science Foundation award under Grant No. OAC-2429485 (CC* Integration-Small: Unifying and Accelerating Campus Computational Science with Ray)

Goal – Easily run workloads across the computing continuum

1. Public compute clouds
2. DoE Leadership Class Systems
3. NSF Shared SuperComputing Centers
4. NSF Computer & Networking Testbeds (FABRIC, CloudLab)
5. Regional AI Hubs
6. Conventional campus HPC clusters
7. Dedicated AI GPU clusters (Princeton Language & Intelligence)
8. Desktops & departmental clusters
9. IoT (e.g., Jetson, Raspberry Pi ?)

What is RAY?

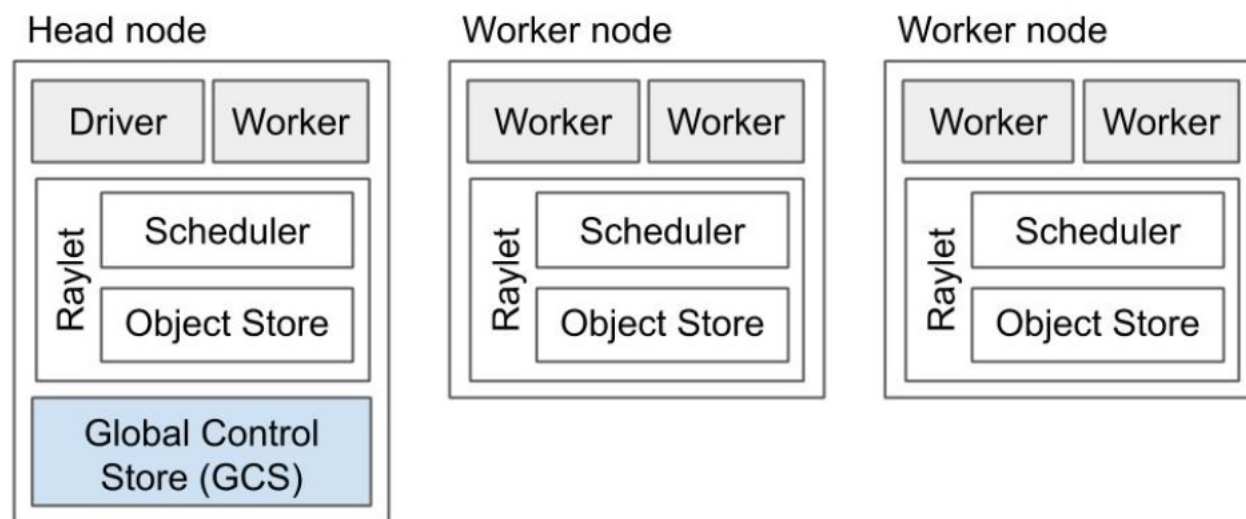
- Python open-source distributed computing framework
- Easy to use, simple multi-programming abstractions, and integrated scheduling and cluster management
- Integrations with common data science and scientific computing tools
- Focus on support for simulation, training, and model serving for RL applications
- Supported by founders from UC Berkeley RISELab at Anyscale, Inc.

What is RAY?

Programming Model highlights:

- A *task* represents the execution of a remote function on a stateless worker. A future representing the result of the task is returned immediately. Futures can be retrieved using `ray.get()`
- An *actor* represents a stateful computation. Each actor (Class) exposes methods that can be invoked remotely and returns a future.
- *Dynamic task graph* computation – the execution of both remote functions and actor methods is automatically triggered when their inputs become available.

What is RAY?



The major components in a cluster: a Global Control Store (GCS), a distributed scheduler, and a distributed object store. Each component is horizontally scalable and fault-tolerant.

Simple Ray programming example

```
import numpy as np
```

```
N = 4000
```

```
def f(x):
```

```
    arr = np.random.randn(N, N)
```

```
    inv_arr = np.linalg.inv(arr)
```

```
    return x
```

```
futures = [f(i) for i in range(96)]
```

```
print(ray.get(futures))
```

```
import numpy as np, ray
```

```
N = 4000
```

```
ray.init()
```

```
@ray.remote
```

```
def f(x):
```

```
    arr = np.random.randn(N, N)
```

```
    inv_arr = np.linalg.inv(arr)
```

```
    return x
```

```
futures = [f.remote(i) for i in range(96)]
```

```
print(ray.get(futures))
```

Parallelizing 96 4000x4000 matrix inversions

Ray dashboard

← → ↻ ⚠ Not secure 172.17.0.121:8265/#/cluster ☆ 🏠 👤 ⋮

 Overview Jobs Serve **Cluster** Actors Metrics Logs 📄 🗨

NODES

Auto Refresh: ☒

Request Status: Node summary fetched.

Node Statistics

TOTALx 1 ALIVEx 1

Node List

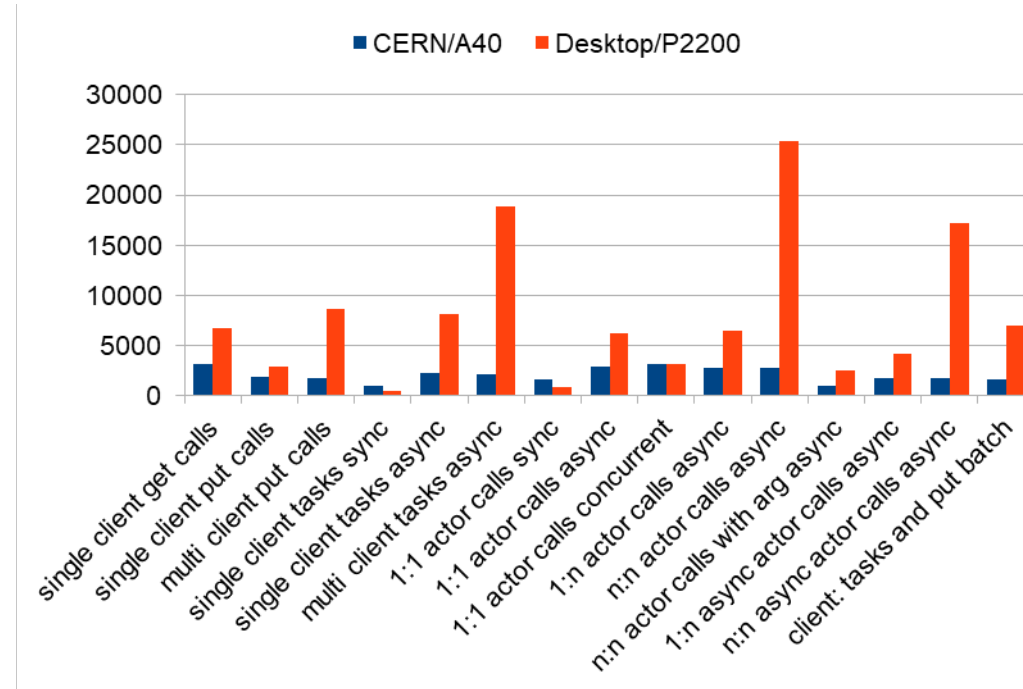
Host 🔍 IP 🔍 State ▼ Page Size 🔍 Sort By ▼

Reverse: ☐ TABLE CARD

< 1 >

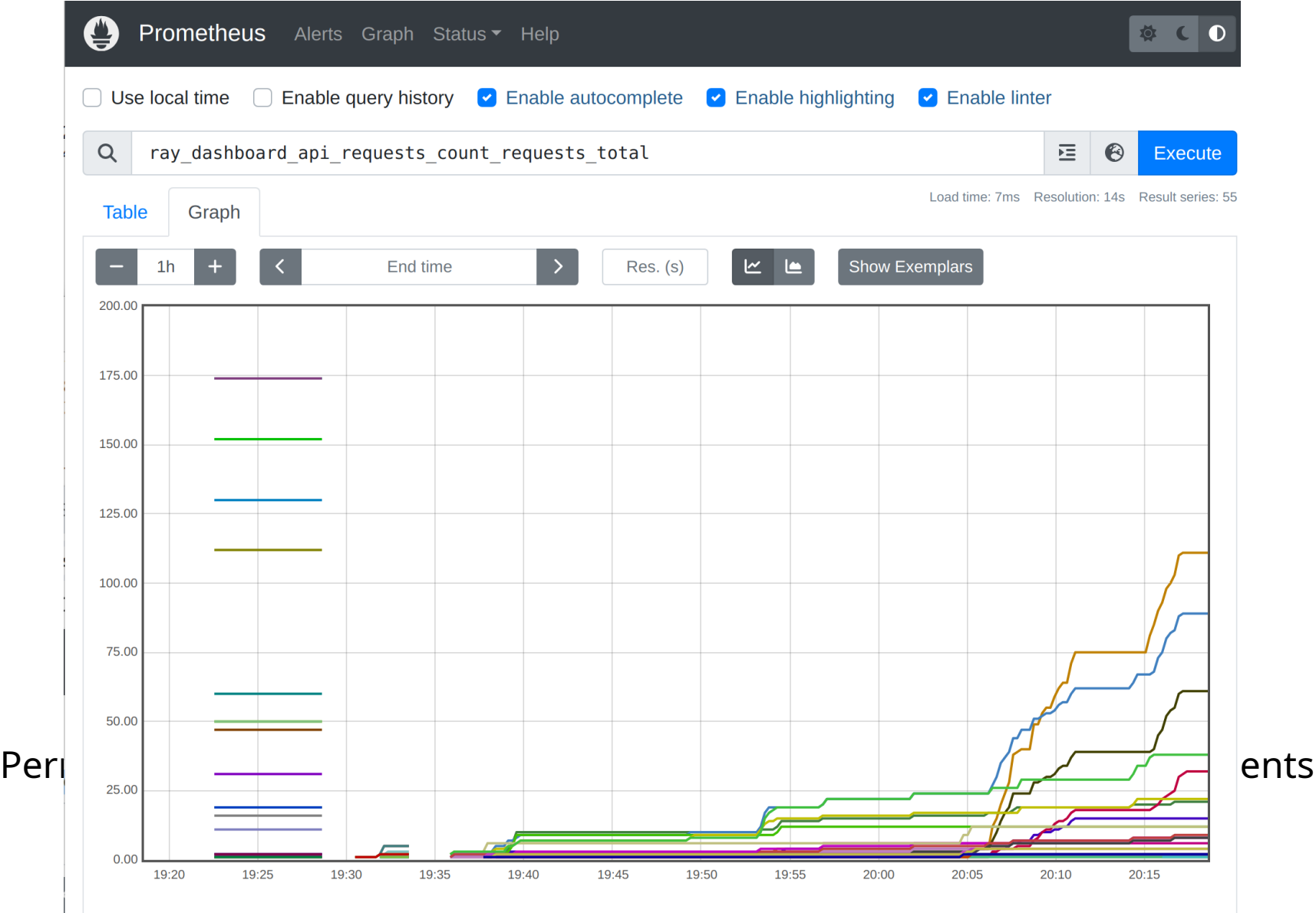
	Host / Worker Process name	State	ID	IP / PID	Actions	CPU ?	Memory ?	
>	jtb-dell-6515-2	ALIVE	a3629...	172.17.0.121 (Head)	Log	2.6%	21.11GB/62.38GB(33.8%)	N/

Ray microbenchmarks



Permits quick performance comparison of different execution environments

Ray Integrations (Visualization)



RAY on NSF Research Infrastructures

jackbrasil / ray-on-research-infrastructures

Type to search

<> Code Issues Pull requests Actions Projects Wiki Security Insights Settings

Files

master

Go to file

- cern_cluster
- cloudlab
- fabric**
- gpu-tests
- invertMatrix
- train
- LICENSE
- README.md
- requirements.txt

ray-on-research-infrastructures / fabric /

Add file

jackbrasil Add files via upload d7a0cb8 · 5 months ago History

Name	Last commit message	Last commit date
..		
README.txt	Add files via upload	5 months ago
fabric_GPU_A30_2nodes.ipynb	Add files via upload	5 months ago
fabric_GPU_A40.ipynb	Add files via upload	5 months ago
fabric_GPU_A40_2nodes-1.ipynb	Add files via upload	5 months ago
fabric_GPU_A40_2nodes.ipynb	Add files via upload	5 months ago
fabric_gpu-rtx6000.ipynb	Add files via upload	5 months ago
fabric_gpu-teslat4.ipynb	Add files via upload	5 months ago
fabric_gpu.ipynb	Add files via upload	5 months ago
facility_port_PRIN-2local.ipynb	Add files via upload	5 months ago
installer.sh	Add files via upload	5 months ago
invertArray.py	Add files via upload	5 months ago

- CloudLab and FABRIC testbeds as illustrative cluster settings
- Advantages include access to small clusters, diverse GPUs, ease of experimentation, experiment in wild, testing and deployment of RL apps, federated learning apps
- All necessary artifacts on Project github

Case 1: NSF CloudLab

- Experiments with 5 bare metal GPU nodes at CloudLab Wisconsin
- Easy to instrument cluster for monitoring RAY, cluster management, task placement, Ray jobs API
- Methodology – run single, multiple, and mixed easy to understand jobs (e.g., large matrix inversions, training a Pytorch Lightning Image Classifier on MNIST data) various platforms)



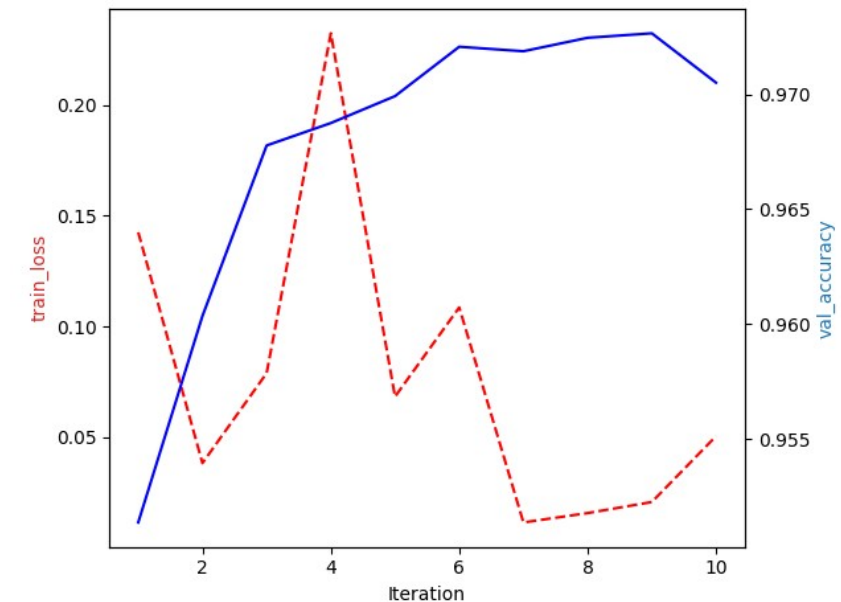
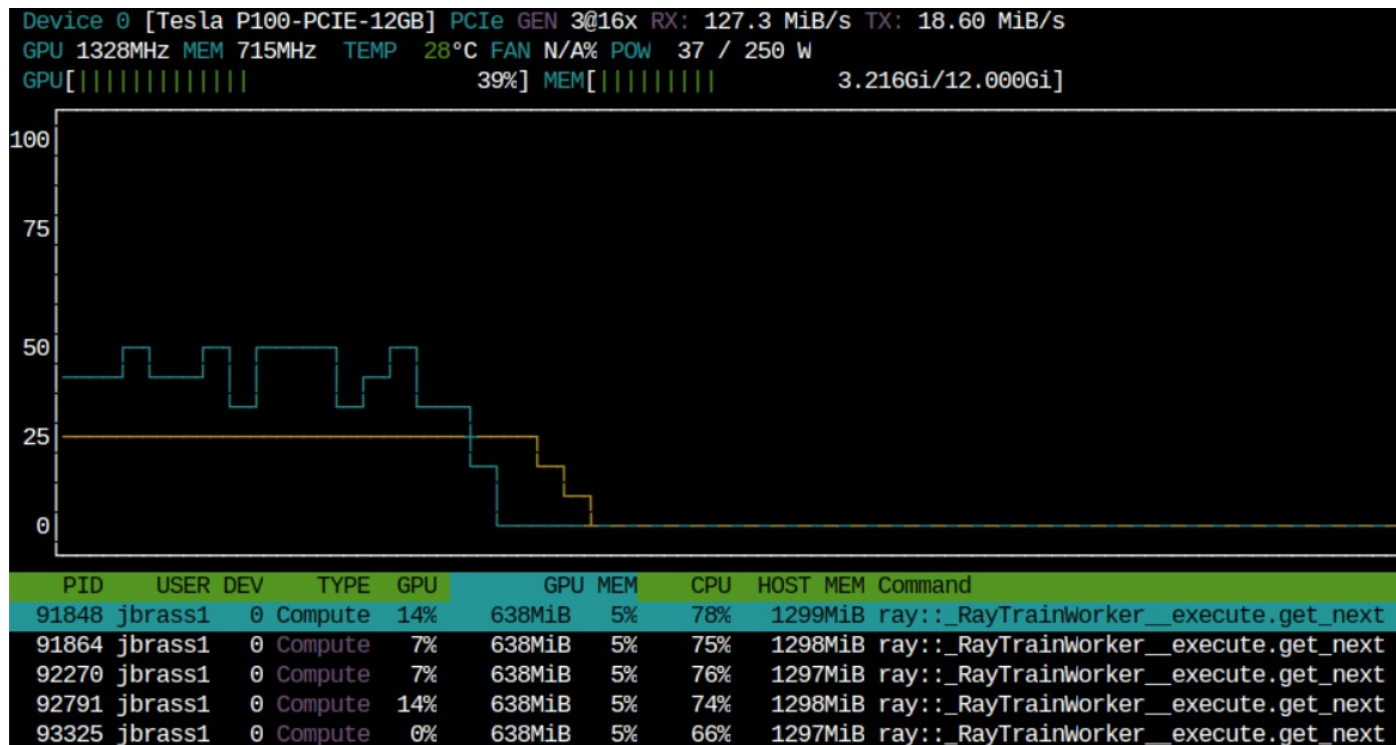
```
head: ray start --head --node-ip-address=  
'128.105.144.55:6379'
```

```
workers: ray start --address='128.105.144.55:6379'  
--node-ip-address= 128.105.144.[44,45,57,59]
```

```
RAY_ADDRESS='http://127.0.0.1:8265' ray job  
submit  
--working-dir . -- python ./invertMatrix.py &
```

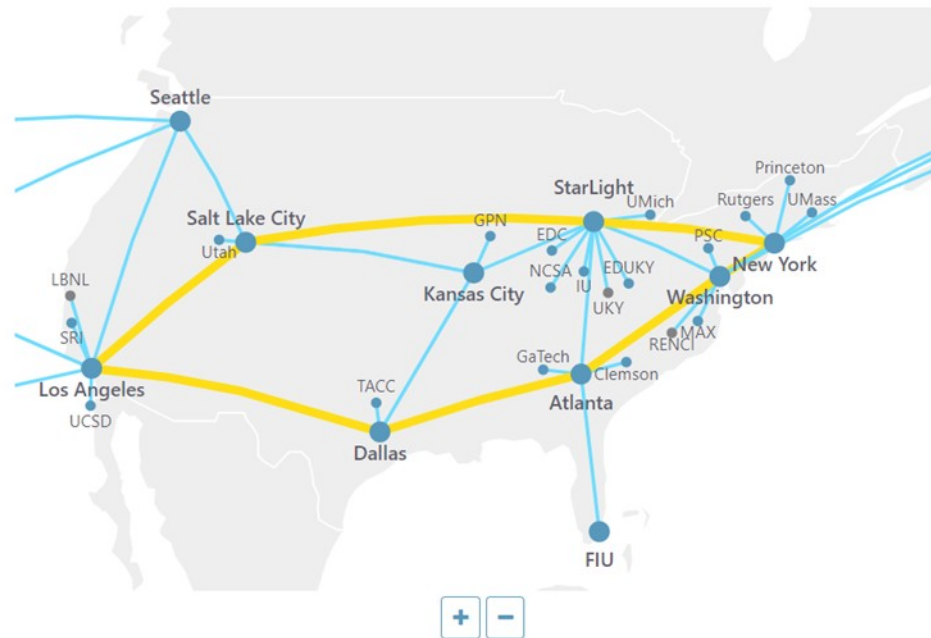
CloudLab – Example

- Simple Pytorch image classifier training
- GPU fully busy at < 10 simultaneous jobs
- Online and offline monitoring (nvidia-smi, btop, TensorBoard)



Case 2: NSF FABRIC Research Infrastructure

- Testbed connecting major instruments, compute clouds, and universities
- Permits collaborative sharing of on- and off-campus computing systems
- whatisfabric.net



TACC (TACC) Active	
Resource	available / total
Cores	388/640
Disk (GB)	103703/107463
RAM (GB)	1598/2390
GPU	4/10
NVME	16/16
SmartNIC	5/6
SharedNIC	616/635
FPGA	1/1

Ray over FABRIC

- Step 1 - FABRIC Preliminaries. Build a global network topology

```
# Preliminaries (e.g., import FABlib libraries)

from ipaddress import ip_address,
IPv4Address, IPv6Address, IPv4Network, IPv6Network
from fabrictestbed_extensions.fablib.fablib
import FablibManager as fablib_manager
fablib = fablib_manager()

#Create FABRIC slice
slice_name = '4node-2site'
node1_name = 'prin1'; node2_name = 'prin2';
node3_name = 'ucsd3'; node4_name = 'ucsd4';
net_name='net1'
slice = fablib.new_slice(name=slice_name)

# Add network
net1 = slice.add_l2network(name=net_name,
subnet=IPv4Network("192.168.100.0/24"))
```

Ray over FABRIC

- Step 2 – Deploy VMs across FABRIC sites. Deploy contributed bare metal nodes across sites.

```
# Node1
node1 = slice.add_node(name=node1_name,
    site='PRIN', cores=4, ram=16, disk=32,
    image='default_ubuntu_22')
iface1 = node1.add_component(model='NIC_Basic',
    name='nic1').get_interfaces()[0]
iface1.set_mode('auto')
net1.add_interface(iface1)
.
.
.
# Node4
node4 = slice.add_node(name=node4_name,
    site='UCSD', cores=4, ram=16, disk=32,
    image='default_ubuntu_22')
iface4 = node4.add_component(model='NIC_Basic',
    name='nic1').get_interfaces()[0]
iface4.set_mode('auto')
net1.add_interface(iface4)

# Create topology

slice.submit()
```


Ray over FABRIC

- Step 3 – Deploy executables & tools

```
# Upload configuration files

result1 = node1.upload_file
('config_script.sh', 'config_script.sh')
.
.
.
result4 = node4.upload_file
('config_script.sh', 'config_script.sh')

# Additional script arguments

script_args="net-tools wireshark"

# Run configurations

stdout, stderr = node1.execute(f'chmod +x
config_script.sh && ./config_script.sh
{script_args} >> config1.log')
.
.
.
stdout, stderr = node4.execute(f'chmod +x
config_script.sh && ./config_script.sh
{script_args} >> config4.log')
```

Ray over FABRIC

- Step 4 – Run Ray benchmarks

```
-----config_script.sh-----
#!/bin/bash
args=$@
sudo apt install -y $args

pip install -U "ray[default]"

if (worker_node):
    ray start
    --address='192.168.100.3:6379'
    --node-ip-address=192.168.100.3
    ray microbenchmark
else:
    ray start --head
    --node-ip-address=192.168.100.3
    --metrics-export-port=8080
    --dashboard-host=192.168.100.3
```

Example: Using Ray on FABRIC testbed

```
ubuntu@gpu-node2: ~  
Training saved a checkpoint for iteration 10 at: (local)/tmp/ray_results/ptl-mnist-example/TorchTrainer_d2fa8_00000_0_2024-10-02_14-37-17/checkpoint_000009  
[RayTrainWorker pid=55856] Checkpoint successfully created at: Checkpoint(filesystem=local, path=/tmp/ray_results/ptl-mnist-example/TorchTrainer_d2fa8_00000_0_2024-10-02_14-37-17/checkpoint_000009)  
[RayTrainWorker pid=55856] `Trainer.fit` stopped: `max_epochs=10` reached.  
Epoch 9: 100%|██████████| 430/430 [00:05<00:00, 76.99it/s, v_num=0]  
[RayTrainWorker pid=55856] LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]  
Testing: | 0/? [00:00<?, ?it/s]  
Testing DataLoader 0: 9%|███████| 7/79 [00:00<00:00, 87.22it/s]  
Testing DataLoader 0: 24%|███████| 19/79 [00:00<00:00, 105.50it/s]  
Testing DataLoader 0: 37%|███████| 29/79 [00:00<00:00, 105.85it/s]  
Testing DataLoader 0: 51%|███████| 40/79 [00:00<00:00, 103.87it/s]  
Testing DataLoader 0: 63%|███████| 50/79 [00:00<00:00, 104.34it/s]  
Testing DataLoader 0: 76%|███████| 60/79 [00:00<00:00, 103.91it/s]  
Testing DataLoader 0: 90%|███████| 71/79 [00:00<00:00, 103.36it/s]  
Testing DataLoader 0: 100%|███████| 79/79 [00:00<00:00, 108.79it/s]  
[RayTrainWorker pid=55856]  
[RayTrainWorker pid=55856]  
[RayTrainWorker pid=55856]  
[RayTrainWorker pid=55856]  
[RayTrainWorker pid=55856]  
[RayTrainWorker pid=55856]  
Training completed after 10 iterations at 2024-10-02 14:38:32. Total running time: 1min 14s  
2024-10-02 14:38:32,121 INFO tune.py:1009 -- Wrote the latest version of all result files and experiment state to '/tmp/ray_r
```

Test metric	DataLoader 0
test_accuracy	0.9775000214576721

AMST

gpu-node1

gpu-node1-gpu1

gpu-node1-nic1

net1

Network Service

gpu-node2-nic1

gpu-node2-gpu2

gpu-node2

Open Challenges

- Characterizing RAY performance across multiple, diverse academic platforms.
- Understanding how RAY can streamline teaching distributed systems to a growing cohort of students spanning multiple disciplines.
- Create best practices for multiple job entry points for conventional SLURM clusters
- Optimize RAY performance on hybrid clouds, multiclouds, lower performance interconnects, edge systems

Thanks!

Conclusion

- Characterizing RAY performance across multiple, diverse academic platforms.
- Understanding how RAY can streamline teaching distributed systems to a growing cohort of students spanning multiple disciplines.
- Seeking to build and strengthen a RAY community across university Research Computing organizations!

Thanks!

This material is based upon work partially supported by an National Science Foundation under Grant No. OAC-2429485.