Radium: Transparent Distributed Execution via Process Virtualization

Aidan Cully, Dusan Veljko, Hyojong Kim, Vance Miller, Husheng Zhou, Joel Zambrano, Mazhar Memon

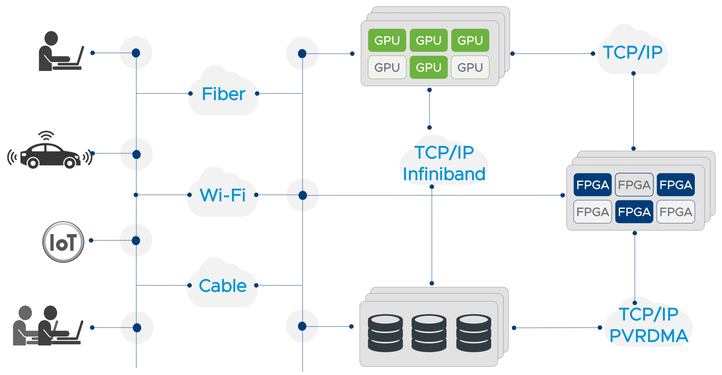
Cloud Infrastructure Business Group, VMware, Inc.

Motivation

- Accelerators are growing in compute and memory capacity but do not provide facilities for flexible sharing.
- The user's data is not co-located with available compute resources. Moving data over the network is slow.
- By remoting applications to remote accelerators without performance loss we provide opportunities for the system to share accelerators and increase utilization

Radium Solution

- Radium migrates execution and memory contents between nodes across the network to achieve the best locality to data and compute acceleration
- Radium minimizes the amount of data movement by using containers on the remote machine to provide needed application software and only migrating user data from the client when it is ready to be used by the accelerator.



Radium Use Cases

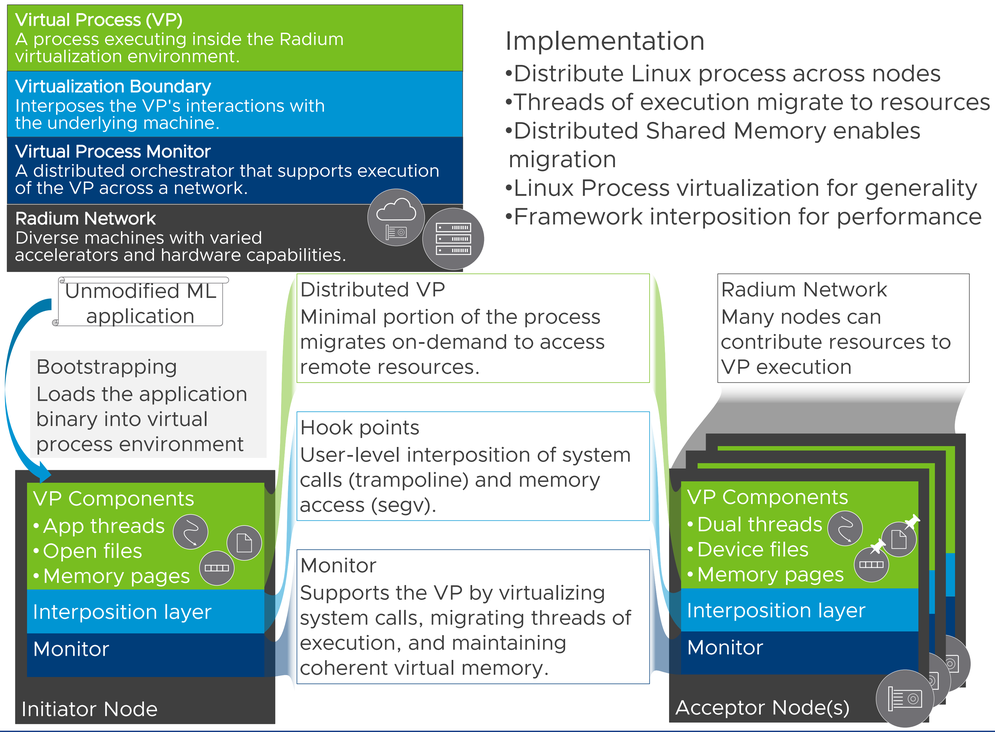
- Unmodified binaries:

```
Acceptor node:
  radium wait --bind localhost:5500
Initiator node:
  radium run --remote server -- /bin/gpu-app
```

- Python integration:

```
import tensorflow as tf
import radium
print("TensorFlow version:", tf.__version__)
radium.migrate_execution_to(DATA_NODE)
m = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = m.load_data()
...
radium.migrate_execution_to(GPU_NODE)
model.fit(x_train, y_train, epochs=5)
...
radium.migrate_execution_to(INITIATOR_NODE)
model.save_weights()
```

Design & Implementation

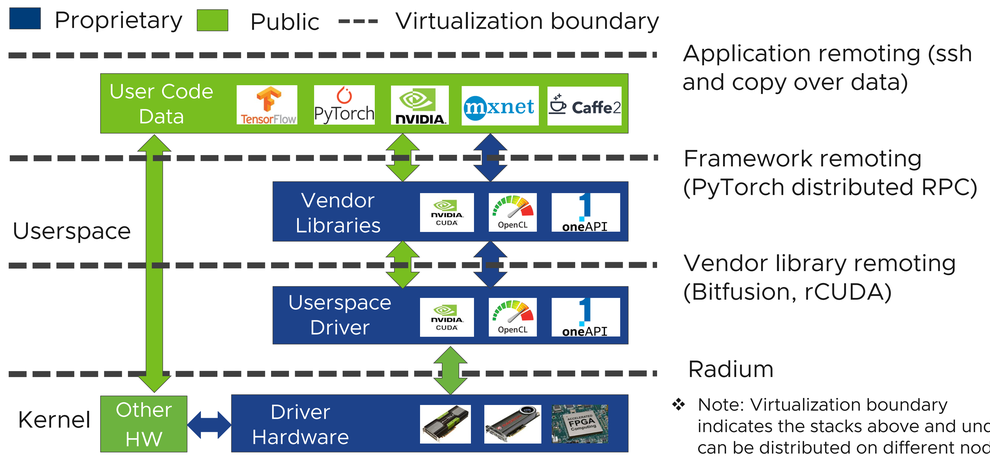


Implementation

- Distribute Linux process across nodes
- Threads of execution migrate to resources
- Distributed Shared Memory enables migration
- Linux Process virtualization for generality
- Framework interposition for performance

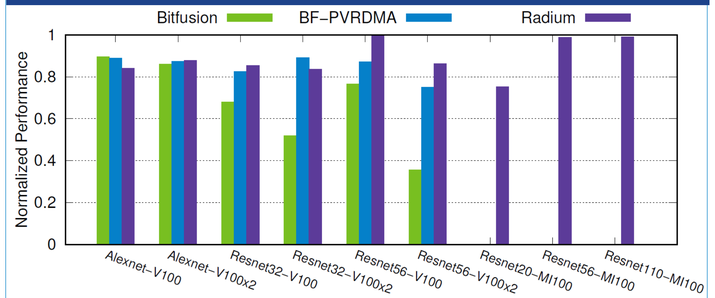
Virtualization Boundary Choice

	Application Remoting	Framework Remoting	Vendor Library Remoting	Radium
Incremental effort to support new hardware	Easy	Easy	Hard	Easy
Accelerated access to client files on remote	No	No	No	Yes
Transparent to users	No	No	No	Yes
Resilient to version changes	Yes	No	Some	Yes
Resilient to network latency	Yes	No	No	Yes



- Note: Virtualization boundary indicates the stacks above and under can be distributed on different nodes

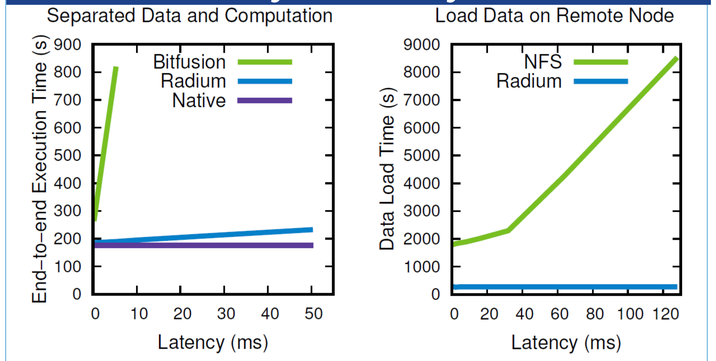
Performance



Configs	Initiator	Acceptor
OS	Ubuntu 22.04	Ubuntu 22.04
CPU	Intel Xeon Gold 6230R	Intel Xeon Platinum 8260
RAM	128 GB	128 GB
GPU	N/A	2xV100 / 1xMI100
Network	TCP/IP, PVRDMA	

- Radium has decent performance compared to baseline
- Outperforms current state-of-the-art solution
- Radium supports both NVIDIA and AMD GPUs

Network Latency Resiliency



- We run pytorch-dcgan with Radium and Bitfusion on nodes where data and computation are separated. And we measure the load time of 64MB data on remote node.
- Bitfusion requires network round-trips at every CUDA synchronization point.
- NFS involves network round-trips when reading data.
- Radium loads the dataset on its node and can hide communications overhead for later parts of the dataset while the GPU is processing earlier portions.

Conclusion and Future Work

- Radium is a userspace framework that uses process virtualization to provide transparent distributed execution among local and remote locations.
- Radium enables resource aggregation.
- With Radium, novel hardware can be rapidly deployed and transparently support existing HW/SW stacks.
- Radium performs well even on a high-latency network by hiding data transfer from critical path.

vmware