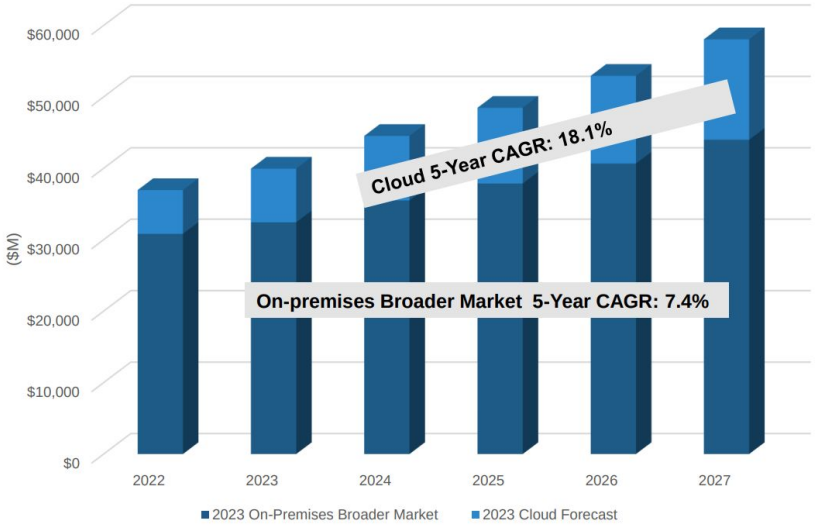
Efficient and Cost-Effective HPC on the Cloud

FlexScience 2025

Aditya Bhosale, Laxmikant Kale, Sara Kokkila-Schumacher

Introduction

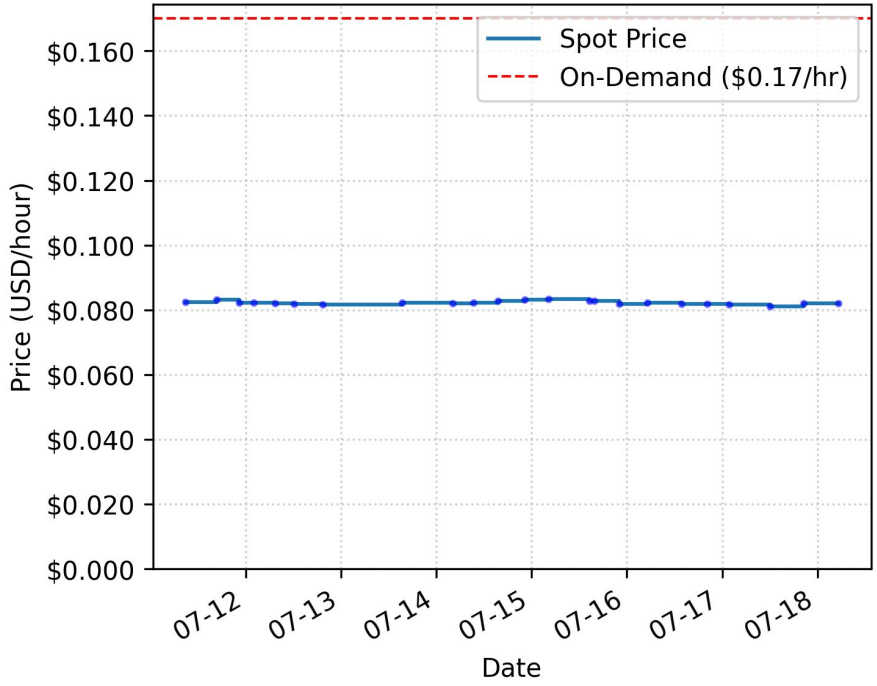
- Increasing availability and adoption of HPC cloud resources over the past few years
- Consumption of HPC cloud resources is projected to grow from $5bn in 2021 to $14bn in 2027
- The demand for HPC cloud resources is higher than ever with the increasing number of AI workloads running on cloud



Source: Hyperion Research, 2023 - https://www.ibm.com/downloads/cas/RGKYOOKB#:~:text=HPC%20in%20the%20cloud%20has,resources%20more%20warmly%20than%20others.

Motivation

- Cloud providers make excess capacity available as spot compute at deep discounts
- Spot vs on-demand price for c5.xlarge AWS instance shows more than 50% discount over the past week
- These instances are preemptible with a 1-2 minute warning if there is a surge in demand



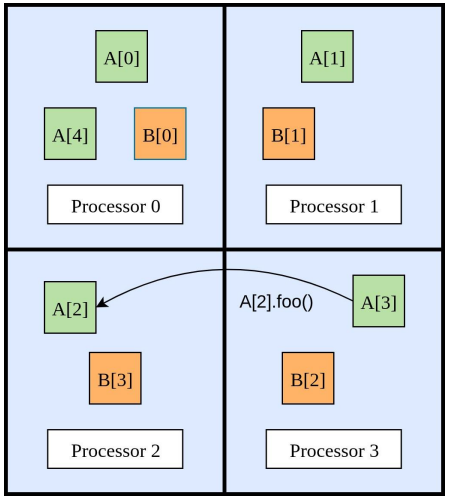
Motivation

- Spot instances ideal for stateless, loosely coupled, fault-tolerant workloads
- HPC applications are
  - Stateful
  - Tightly coupled
  - Not inherently fault-tolerant

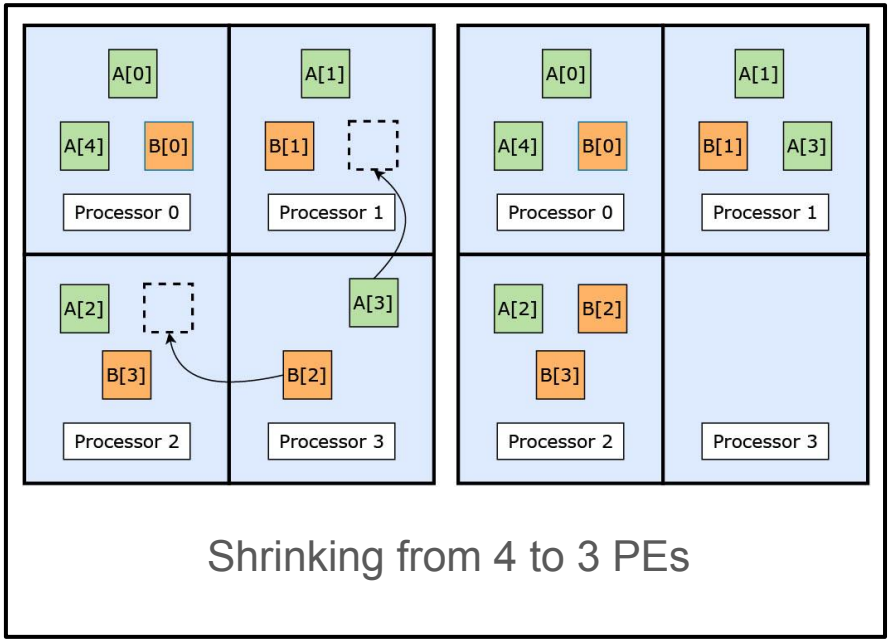
- In this paper we present
  - Charm++ as a programming model for running HPC applications on spot instances with minimal programmer effort and without the need for a shared filesystem
  - CharmCloudManager, a Python utility to automatically monitor and run Charm++ applications on AWS EC2 spot instances

Background

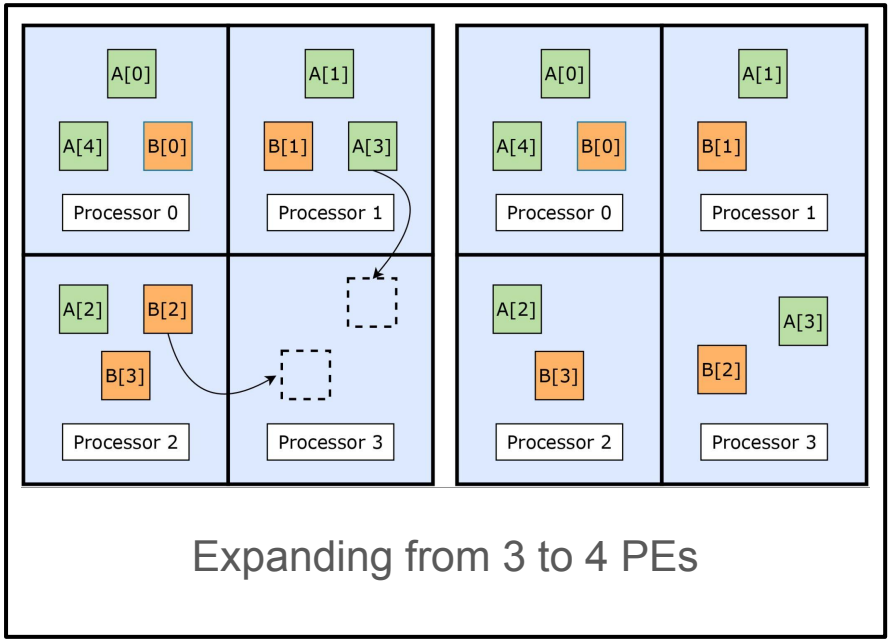
- Charm++ is an asynchronous message-driven parallel programming model
- Users express computation in terms of objects (chares) that communicate via remote method invocations
- Chares are migratable across processors
- Charm++ natively supports -
  - Dynamic load balancing
  - Dynamic rescaling



Rescaling in Charm++



Shrinking from 4 to 3 PEs



Expanding from 3 to 4 PEs

Rescaling in Charm++

- When shrinking -
  - Migrate chares out of the processors to be removed
  - Checkpoint application state in Linux shared memory
  - Restart with reduced number of processors

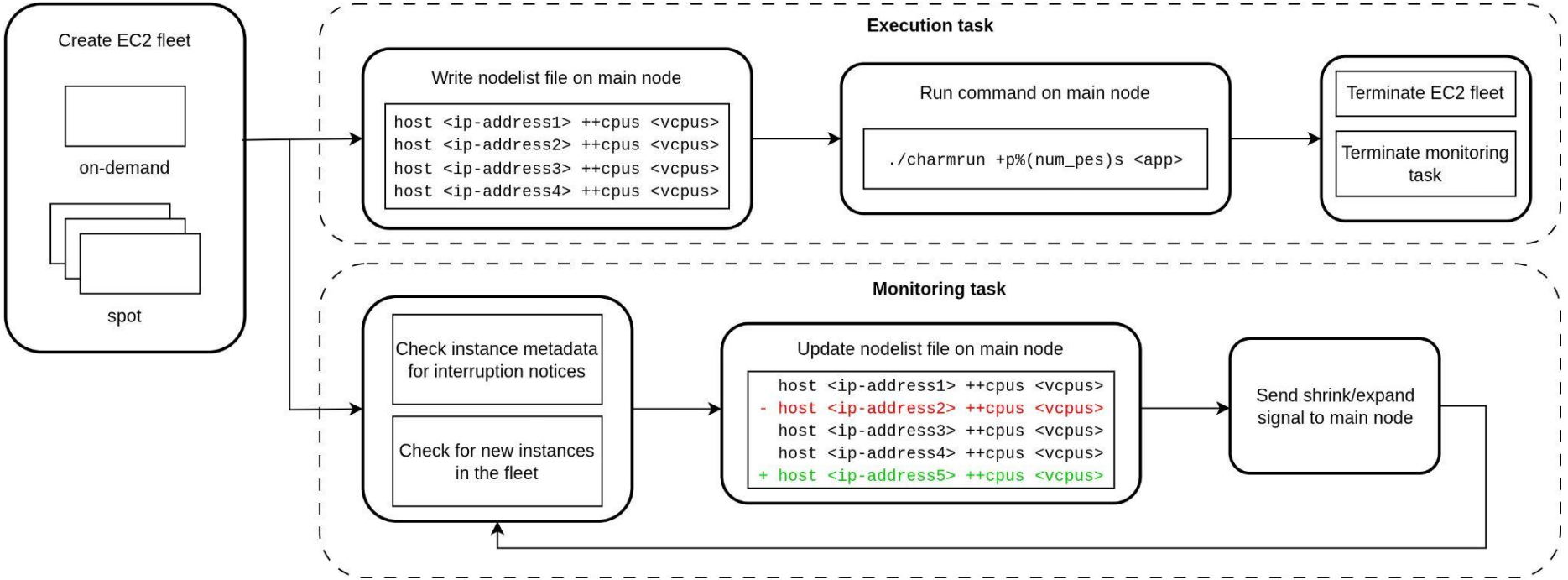
- When expanding
  - Checkpoint application state in Linux shared memory
  - Restart with increased number of processors
  - Load balance to distribute workload evenly

Methodology

CharmCloudManager - a Python utility to run Charm++ applications on EC2 spot instances

- Create a fleet of EC2 spot and on-demand instances
- Monitor instance metadata for interruption notice
- Use the 2 minute interruption warning to shrink the application to remove the interrupted instance
- After instance is replaced, expand to include the new instance

CharmCloudManager



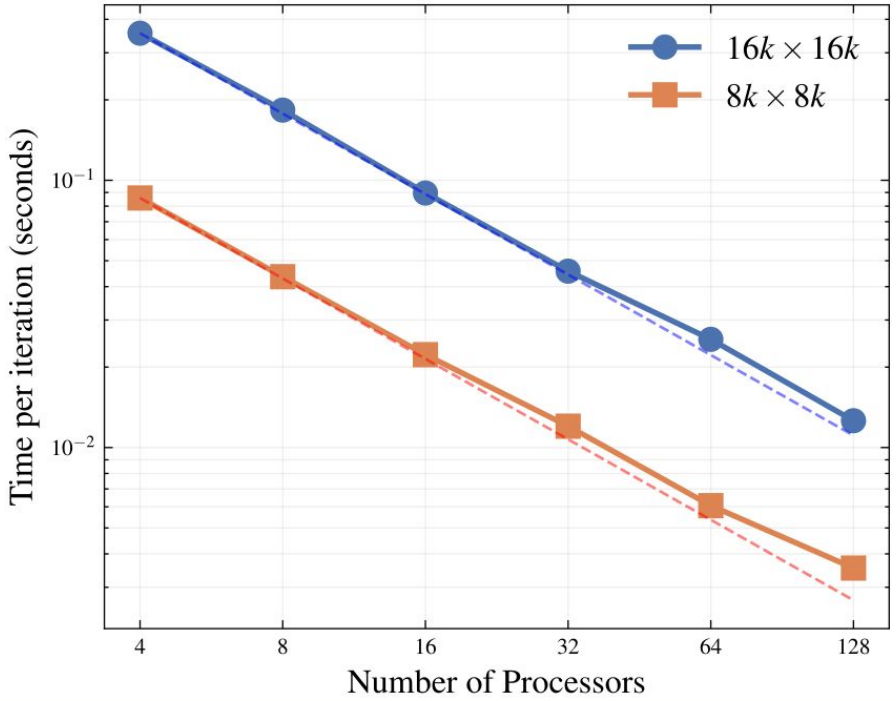
CharmCloudManager

- We used a cluster placement group in a single region, availability zone and subnet to maximize network performance
- Need at least 1 vCPU in on-demand capacity to run the main node of the Charm++ application

Performance

Strong scaling performance

- We used a 2D Stencil benchmark for all experiments
- Instance type - `c5.xlarge`
- Each instance has 4 vCPUs

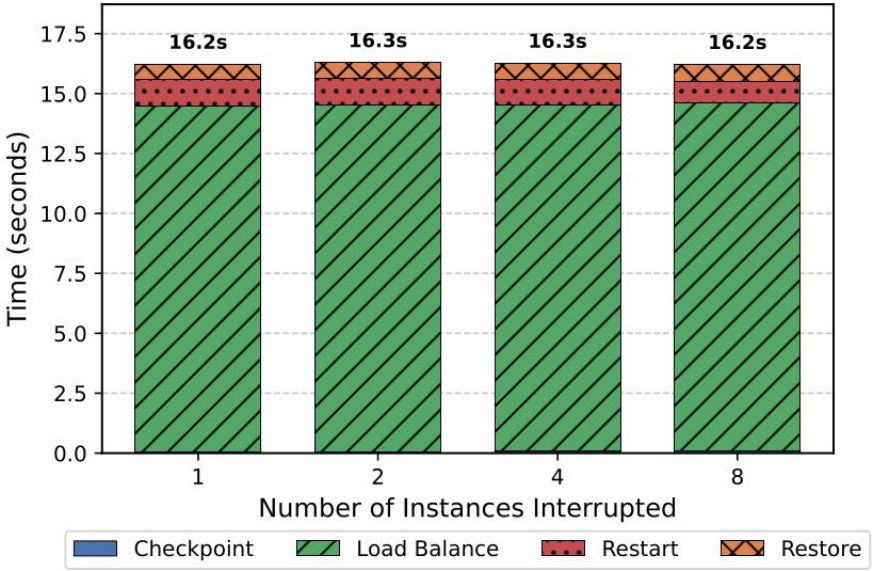


Performance

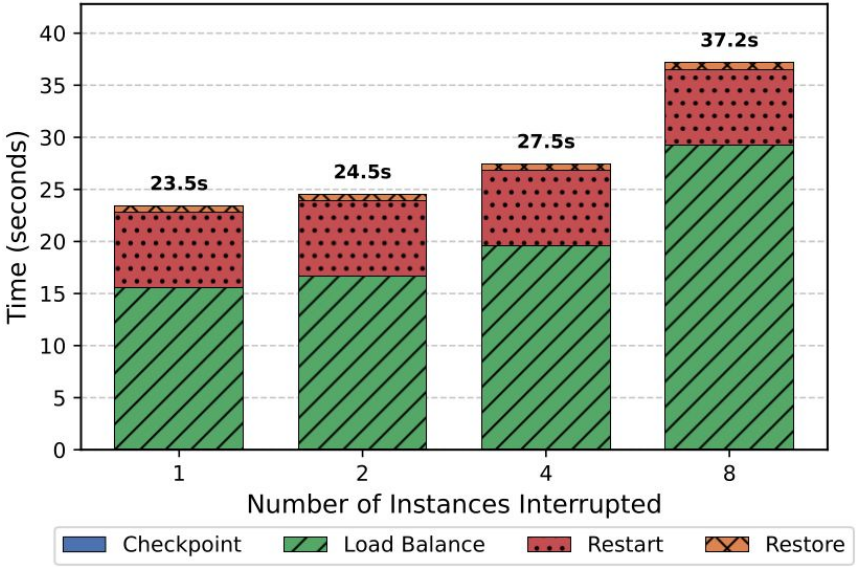
The overhead of spot interruptions can be classified as follows

- Overhead of rescaling
  - Shrink overhead when removing interrupted instances
  - Expand overhead when adding new instances
- Overhead due to drop in capacity when application shrinks

Rescaling overhead



Shrink overhead when instances interrupted



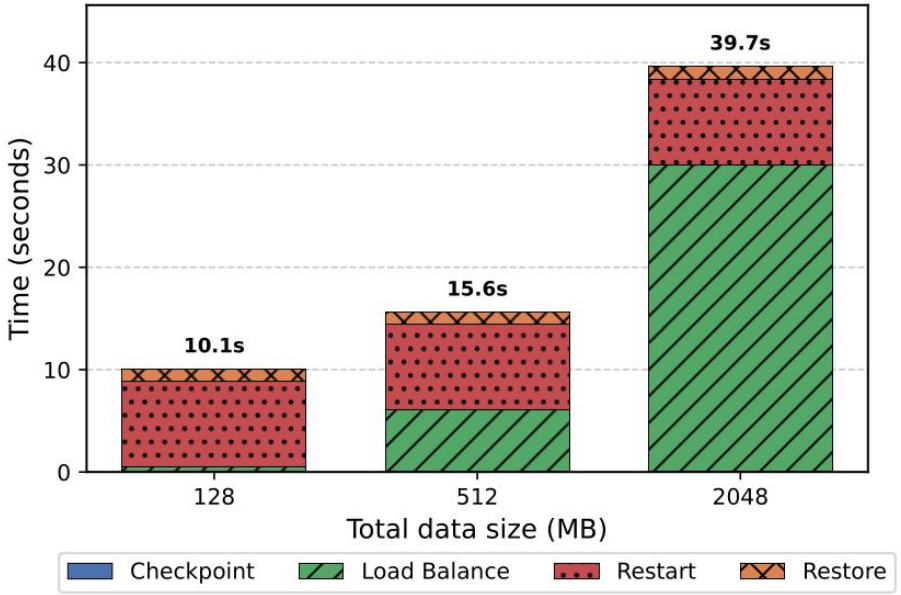
Expand overhead when instances replaced

Contribution of each step involved in rescaling from 16 instances to the total overhead

Rescaling overhead

Rescaling overhead with varying data size

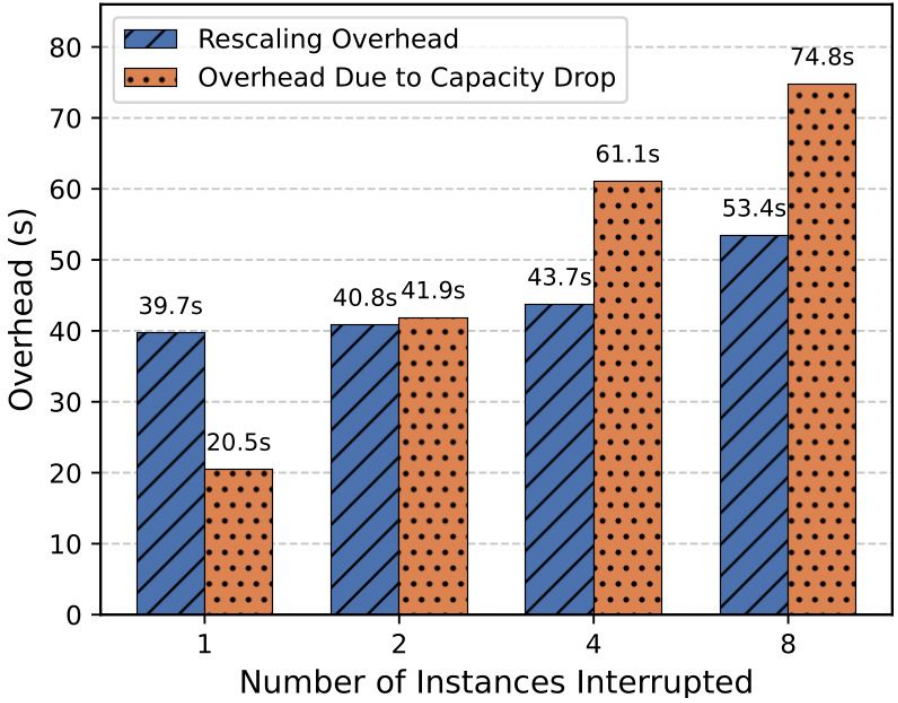
- The cost of in-memory checkpointing is as small as 0.046s for 2GB
- The cost of data movement during load balancing sharply increases with increasing data size



Overhead comparison

Comparison of the two sources of overhead using 16 instances

- The rescaling overhead dominates for smaller number of instances being interrupted
- As more instances are interrupted, the cost of drop in capacity dominates



Cost comparison

- 1 on-demand and 15 spot instances
- Even with a large number of instances being interrupted, the run using spot instances costs 50.71% lower than the all on-demand run

Instances Interrupted	On-Demand Cost (USD)	Spot Cost (USD)	Cost Savings
0	0.43	0.17	59.78%
1	0.43	0.19	55.52%
2	0.43	0.20	53.93%
4	0.43	0.20	52.36%
8	0.43	0.21	50.71%

Future work

- Reducing the overhead of interruptions
  - Using EFA for more efficient communication can reduce the contribution of load balancing to the rescaling overhead
  - Using capacity rebalancing to reduce overhead due to drop in capacity at the cost of potentially more frequent rescaling

- Extending rescaling support to GPUs
  - Additional cost of data-movement between host and device and increase overhead
  - Using GPUDirect RDMA with EFA will be critical to manage rescaling overhead