SC21 St. Louis, MO | science & beyond.

Evaluating Multi-Level Checkpointing for Distributed Deep Neural Network Training

Quentin Anthony*, Donglai Dai

{q.anthony, d.dai}@x-scalesolutions.com

anthony.301@osu.edu

* This work has been done through an internship at X-ScaleSolutions while being a student at the Ohio State University

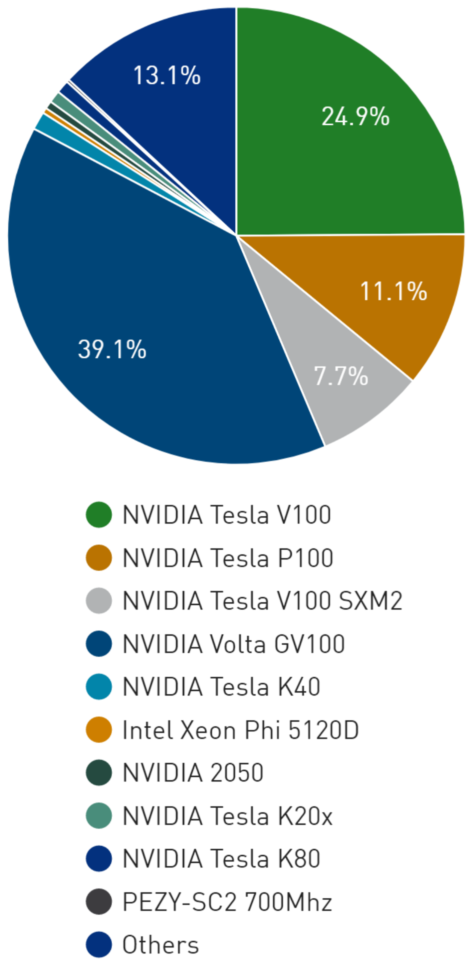
Agenda

- **Introduction**
- Background
- Research Challenges
- Methodologies
- Performance Evaluation
  - Evaluation Platforms and Software Libraries
  - Scaling Results
- Conclusion

Deep Learning, CPUs, and GPUs

- *NVIDIA GPUs - main driving force for faster training of Deep Neural Networks (DNNs)*
- The ImageNet Challenge - (ILSVRC)
  - DNNs like AlexNet, ResNet, and VGG
  - 90% of the ImageNet teams used GPUs in 2014*
  - *And, GPUs are growing in the HPC arena as well! – Top500 (May '21)*

Accelerator/CP Family Performance Share



https://www.top500.org/

*https://blogs.nvidia.com/blog/2014/09/07/imagenet/

Deep Learning Frameworks

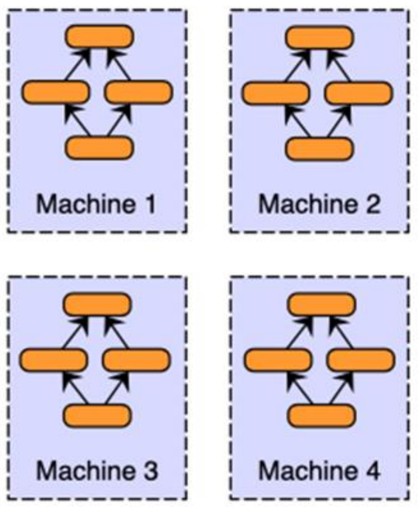
- Easily implement and experiment with Deep Neural Networks
  - Several Deep Learning (DL) frameworks have emerged
- PyTorch, TensorFlow, and MXNet are the major DL frameworks
  - *Focus on **PyTorch** in this work*
- Most frameworks are optimized for NVIDIA GPUs (for now!)
- Distributed DL frameworks built on top of DL frameworks are gaining steam (e.g. Horovod, DeepSpeed)

Agenda

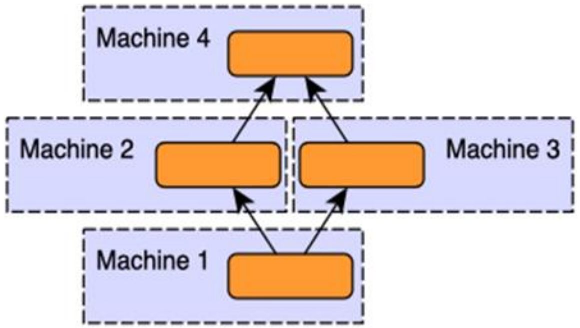
- Introduction
- **Background**
- Research Challenges
- Methodologies
- Performance Evaluation
  - Evaluation Platforms and Software Libraries
  - Scaling Results
- Conclusion

Background: Distributed DNN Training

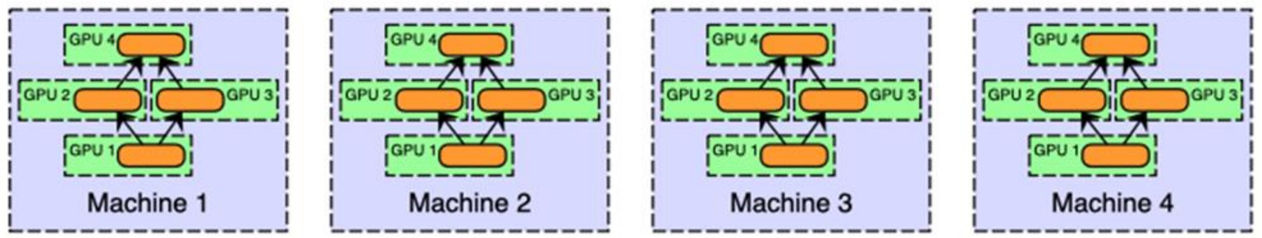
- Deep Neural Network training consists of two phases
  - Forward pass
  - Backward pass
- Training is a compute intensive task
  - Large datasets
  - Complex Deep Learning Models
  - MPI-driven training is on the rise
- Three approaches to Distribute DNN training
  - **Data Parallelism (focus of this paper)**
  - Model Parallelism
  - Hybrid Parallelism



Data Parallelism



Model Parallelism



Hybrid Parallelism

Courtesy: https://blog.skymind.ai/distributed-deep-learning-part-1-an-introduction-to-distributed-training-of-neural-networks/

Background: MVAPICH2-GDR

- MVAPICH2-GDR is an MPI library designed to efficiently support NVIDIA and AMD GPUs over Mellanox InfiniBand adapters
  - Based on MVAPICH2
  - Support for ARM, x86, and **OpenPOWER** 8/**9** systems
- Support for many GPU features including:
  - Non-Blocking Collectives
  - CUDA managed memory
  - GDRCOPY and Loopback
  - CUDA IPC and registration cache
  - Large-message collectives for DL frameworks
- More Information: http://mvapich.cse.ohio-state.edu/userguide/gdr/

Background: SCR-Exa

- **Based on LLNL Scalable Checkpoint Restart (SCR) library**
- Built in collaboration with LLNL in a DOE SBIR Phase-I (currently)

- **Focus**:
  - New features on top of open-source SCR
  - Some new features go to SCR
  - Some new features remain in SCR-Exa
  - Optimization foci:
    - New applications (DL, ML, and AI)
    - Cloud environments
    - New systems, schedulers, etc.

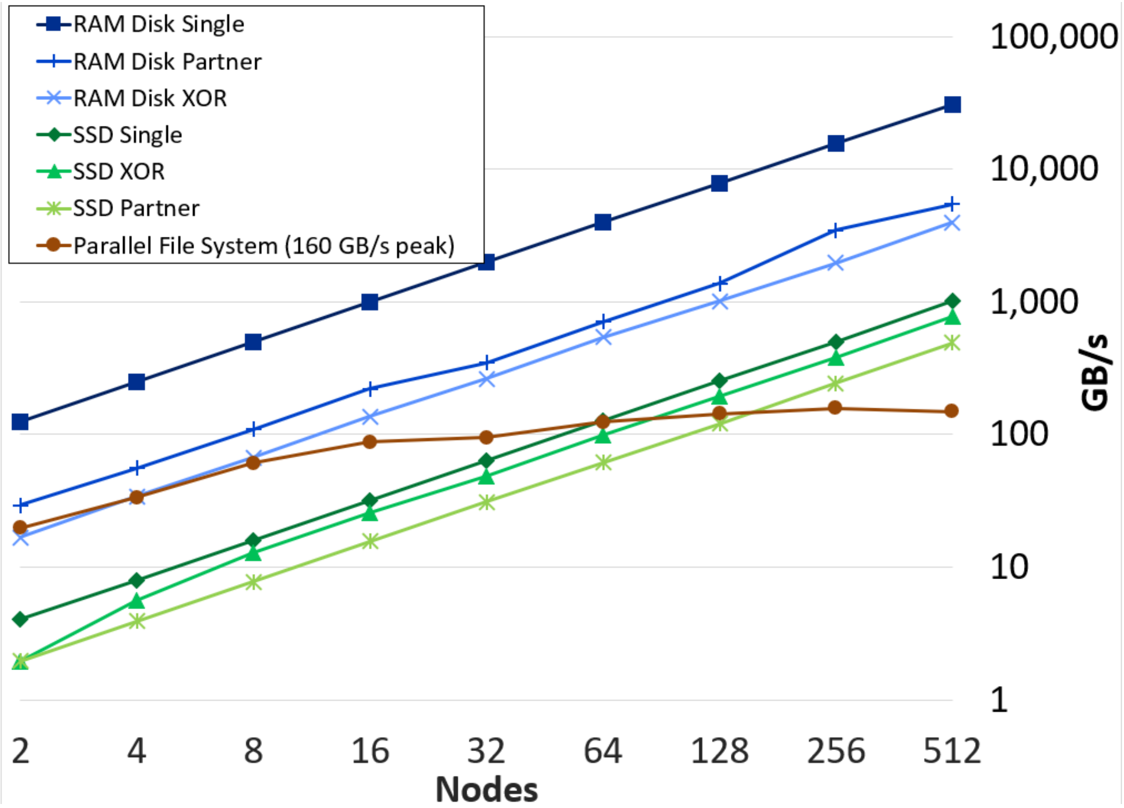
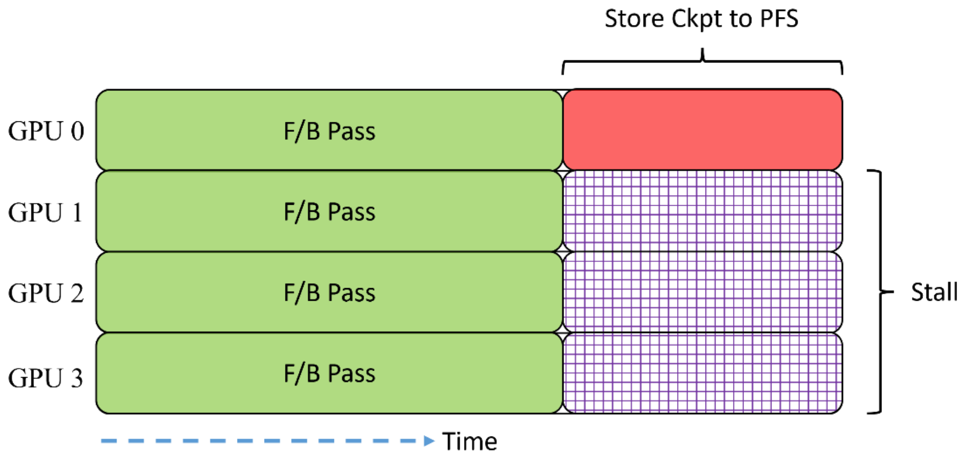


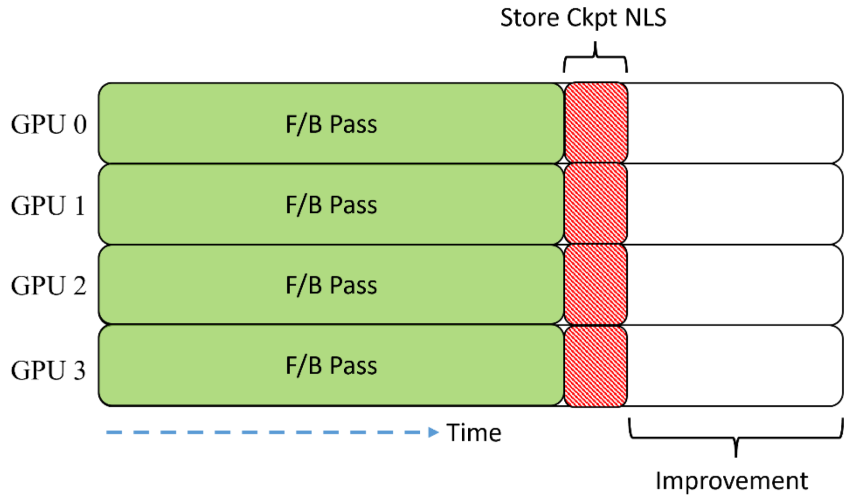
Image Courtesy: **https://computing.llnl.gov/projects/scalable-checkpoint-restart-for-mpi/multilevel-checkpointing-research**

Background: SCR-Exa

- Root Checkpoints/Restarts
  - In standard distributed DL applications the root rank saves a checkpoint to the parallel file system (See top figure)
  - For restarts, the root rank loads a checkpoint and MPI_Bcast's it to all nodes in the job
- SCR and SCR-Exa Checkpoints
  - Checkpoints are saved to node-local storage (NLS)
  - Every Nth checkpoint is "flushed" to the parallel file system
  - Two redundancy schemes are used in this work:
    - **Single**: Every rank saves the checkpoint to its own NLS only
    - **Partner**: Every rank saves the checkpoint to its own NLS and the neighboring node's NLS
- SCR and SCR-Exa Restarts
  - Cached checkpoints within a job are used (bypassing the parallel file system)



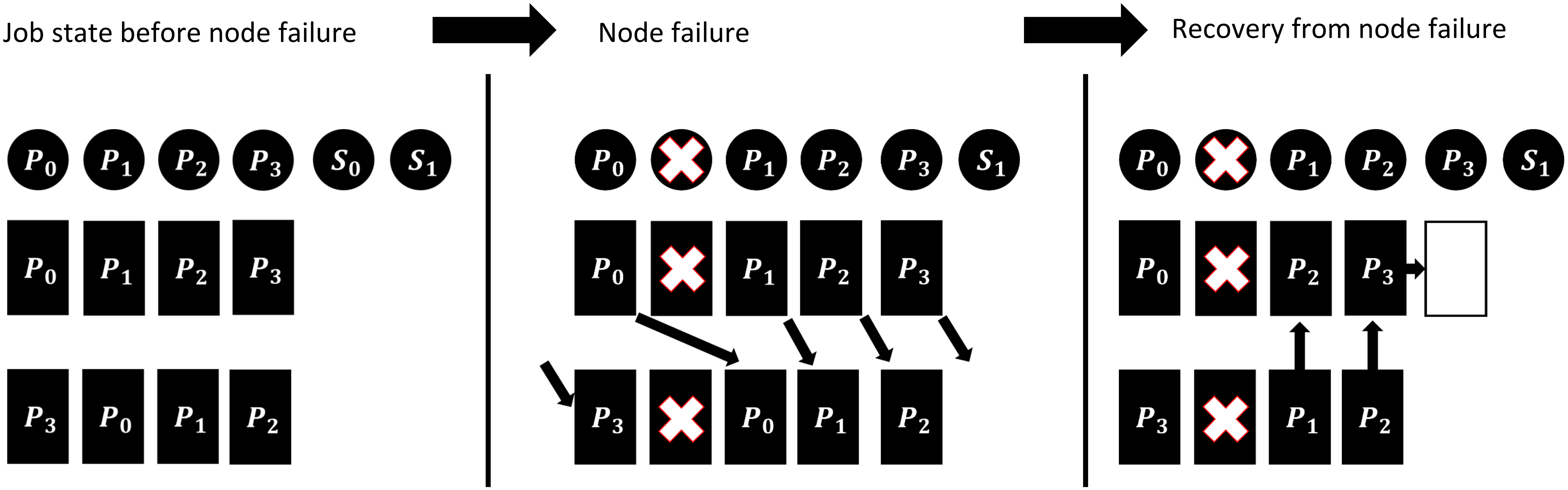
Saving root ckpt on last training step of epoch



Saving SCR(-Exa) ckpt to NLS

Background: SCR-Exa (cont'd)

- SCR and SCR-Exa support **hot** and **cold** restarts
  - A **cold** restart uses a checkpointing cache within the same allocation
  - A **hot** restart replaces faulty nodes within the allocation with idle spare nodes (**see below**)



Background: SCR-Exa for DL Applications

- Periodically saving a snapshot of a DL model's parameters during training can save work in the event of interruptions
  - DL training on a single machine often requires weeks or months to complete
  - DL training at scale on HPC systems is more susceptible to hardware or software failures
- Single-machine DL training jobs can simply load/store the DL model every *N* epochs
- What about distributed training jobs? DL frameworks recommend the following naïve scheme:

```
for n in num_epochs:
    if rank == 0 and n % checkpoint_freq == 0:
        save_DNN()
    MPI_Barrier()
    …
    if rank == 0 and interruption:
        load_DNN()
    MPI_Bcast(DNN_params)
```

- However, this scheme requires all ranks to block on rank 0 while it writes to the PFS every *checkpoint_freq* epochs!
- What if we add support for distributed multi-level checkpointing via SCR-Exa's Python API?

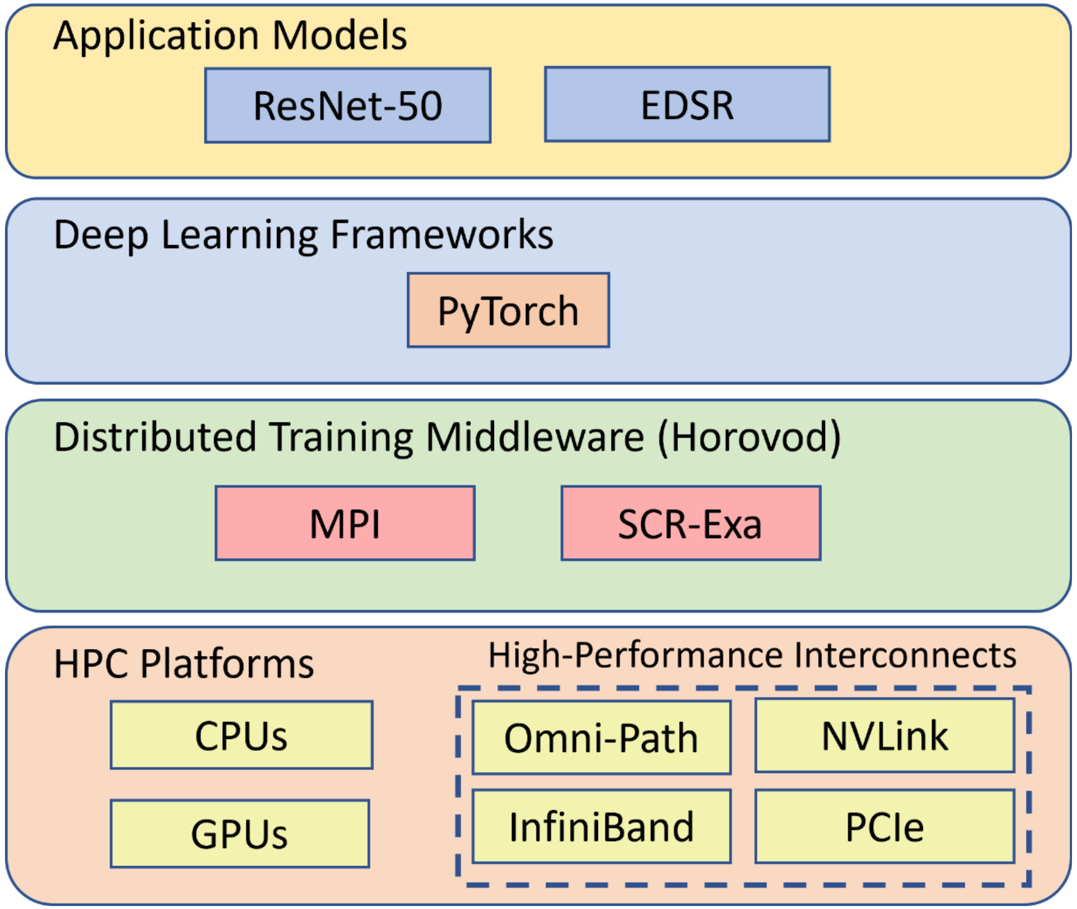
Background: Horovod

- Horovod is a distributed DNN training framework that employs data parallelism
  - **Acts as middleware** between DL framework (Tensorflow, Pytorch, etc) and communication backend (MPI, NCCL, etc)
  - Performance is strongly dependent on **Allreduce**

- Before carrying out evaluations, we have added full checkpointing support to EDSR and ResNet-50 training with Horovod and SCR-Exa

Full Program Stack

- Horovod standardizes data-parallel training
- SCR-Exa can be used directly from the application layer



Agenda

- Introduction
- Background
- **Research Challenges**
- Methodologies
- Performance Evaluation
  - Evaluation Platforms and Software Libraries
  - Scaling Results
- Conclusion

Problem Statement

Can we reduce the DNN checkpoint overhead by adding support for multi-level checkpointing into the distributed DL framework?

Agenda

- Introduction
- Background
- Research Challenges
- **Methodologies**
- Performance Evaluation
  - Evaluation Platforms and Software Libraries
  - Scaling Results
- Conclusion

Adding SCR-Exa support to Horovod (Similar for torch.dist)

- SCR-Exa was directly applied to the Horovod training script
- In addition to the basic SCR-Exa code, a full SCR-Exa configuration needs to be defined
  - Options such as asynchronous flush and the redundancy scheme are set here
- Our runs use the following config options:
  - SCR_FLUSH_ASYNC=1
  - SCR_COPY_TYPE=SINGLE
  - SCR_FLUSH=10

```
def save_checkpoint(epoch):
  if scr.need_checkpoint():
    ...
    # All processes tell SCR-Exa a new checkpoint is about to start
    scr.start_output(name, scr.FLAG_CHECKPOINT)
    ...
    # Get the full path and file name SCR-Exa will need to access the checkpoint file
    newfname = scr.route_file(fname)
    ...
    # Save DNN
    torch.save(ddp_model.state_dict(), newfname)
    ...
    # All processes tell SCR-Exa the checkpoint is over
    scr.complete_output(valid)
```

```
def restart(epoch):
  while True:
    # Is there a checkpoint available?
    if not scr.have_restart()
    break
    # All processes tell SCR-Exa that a new restart is about to start
    name = scr.start_restart()
    ...
    # Get the full path and file name SCR-Exa will need to access the checkpoint file
    newfname = scr.route_file(fname)
    ...
    # Load DNN
    torch.load(newfname, map_location=map_location)
    ...
    # All processes tell SCR-Exa the restart is over
    scr.complete_restart(valid):
```

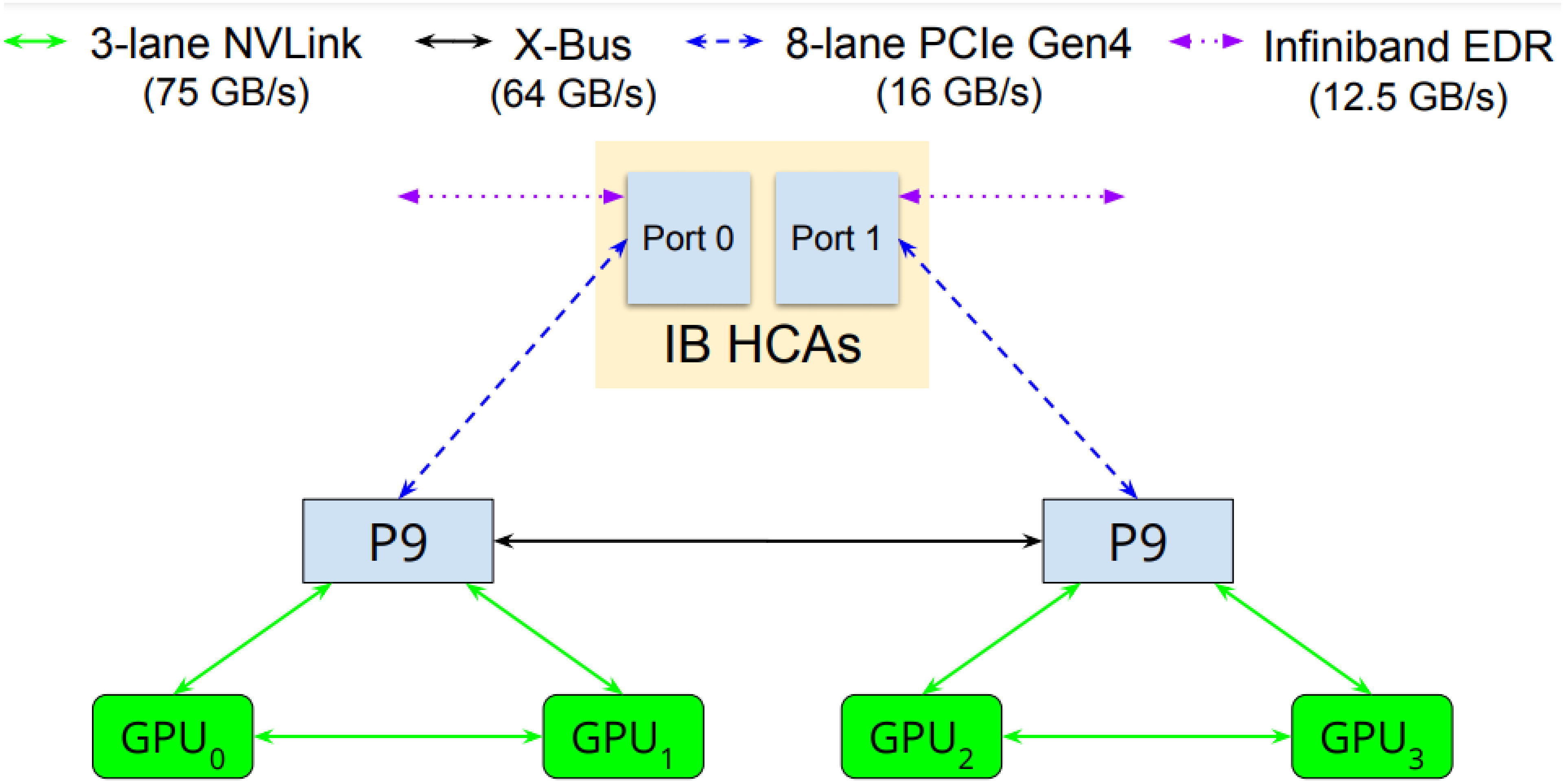
Agenda

- Introduction
- Background
- Research Challenges
- Methodologies
- **Performance Evaluation**
  - **Evaluation Platforms and Software Libraries**
  - Scaling Results
- Conclusion

Evaluation Platform

- **Lassen Supercomputer at Lawrence Livermore National Laboratory**
  - #17 on TOP500.org
  - 792 GPU Nodes
  - Two IBM POWER 9 processors
  - **4 NVIDIA Volta GPUs (16 GB HBM2)**
  - **NVIDIA NVLINK (GPU-GPU and CPU-GPU)**
  - 256 GB CPU Memory/Node
  - Mellanox InfiniBand, EDR (12.5 GB/s)

Evaluation Platform



Courtesy: **Performance Evaluation of MPI Libraries on GPU-enabled OpenPOWER Architectures: Early Experiences, IWOPH '19**

Software Libraries

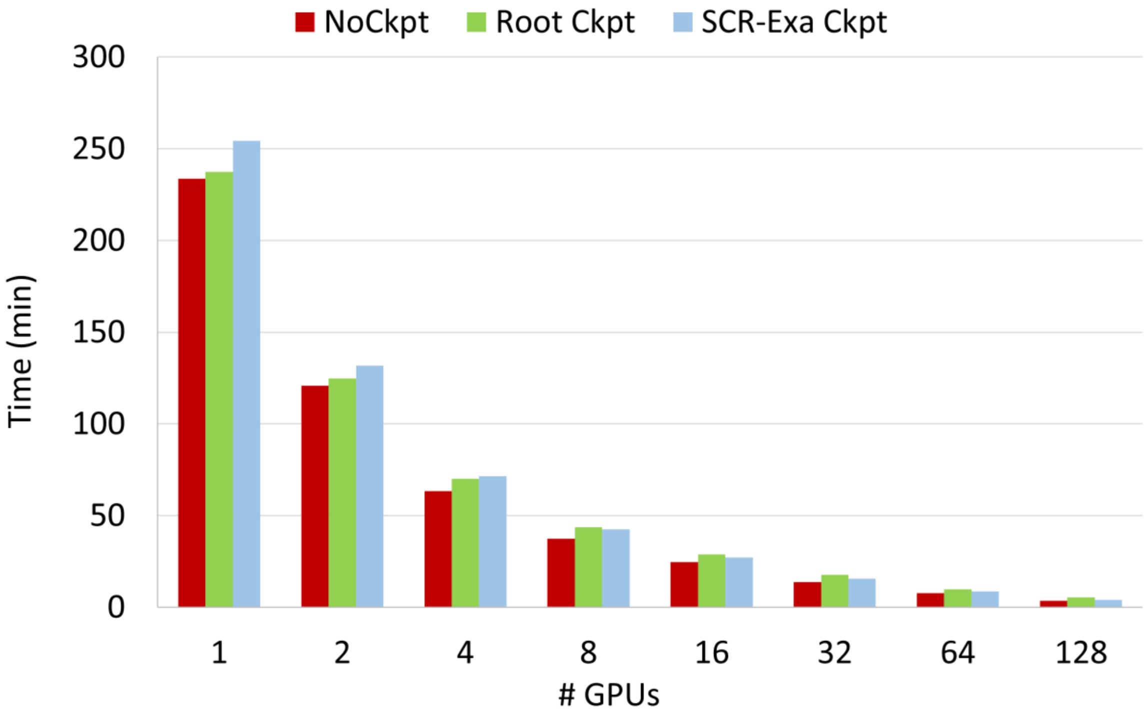
- Deep Learning Frameworks
  - PyTorch v1.9.0
- CUDA 10.2
- cuDNN 7.6.5
- Horovod Distributed Training middleware (0.22.1)
- MPI Library: MVAPICH2-GDR 2.3.6
- DL Models: EDSR from publicly-available github
  - https://github.com/sanghyun-son/EDSR-PyTorch

Agenda

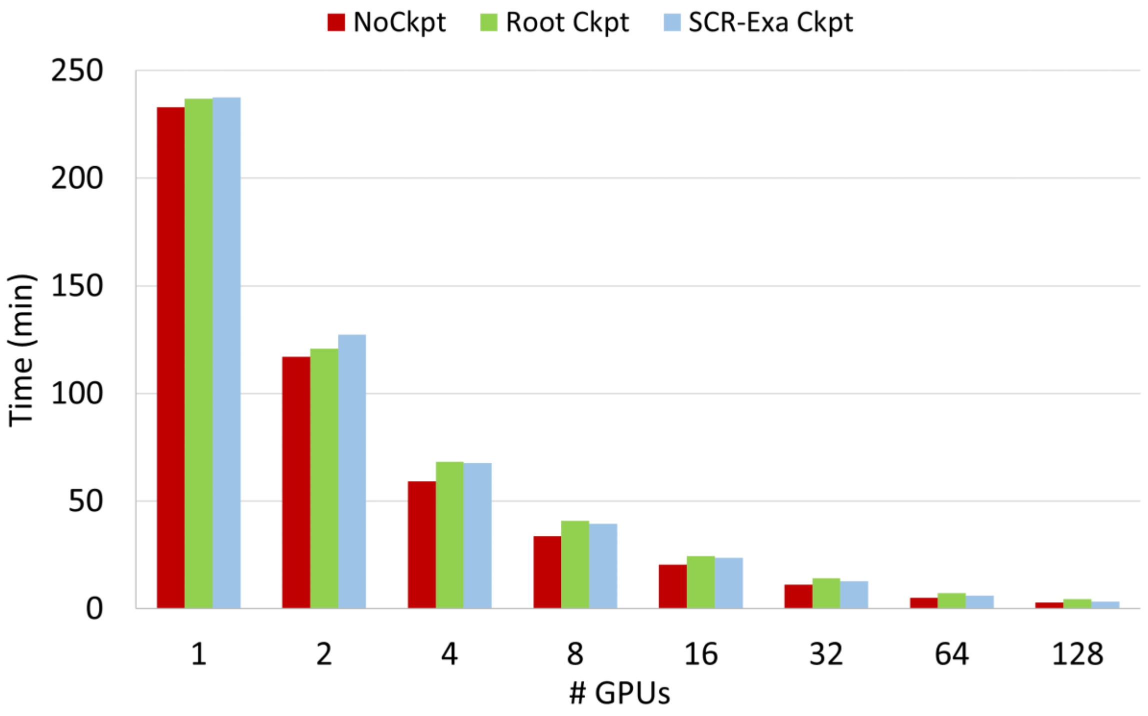
- Introduction
- Background
- Research Challenges
- Methodologies
- **Performance Evaluation**
  - Evaluation Platforms and Software Libraries
  - **Scaling Results**
- Conclusion

Performance Improvement: Save Ckpt

- We take the end-to-end training time of 100 epochs of ResNet-50 training with two distributed DL frameworks
  - PyTorch's distributed module
  - Horovod
- Compared root ckpt method with SCR-Exa performance, with 1 ckpt per epoch



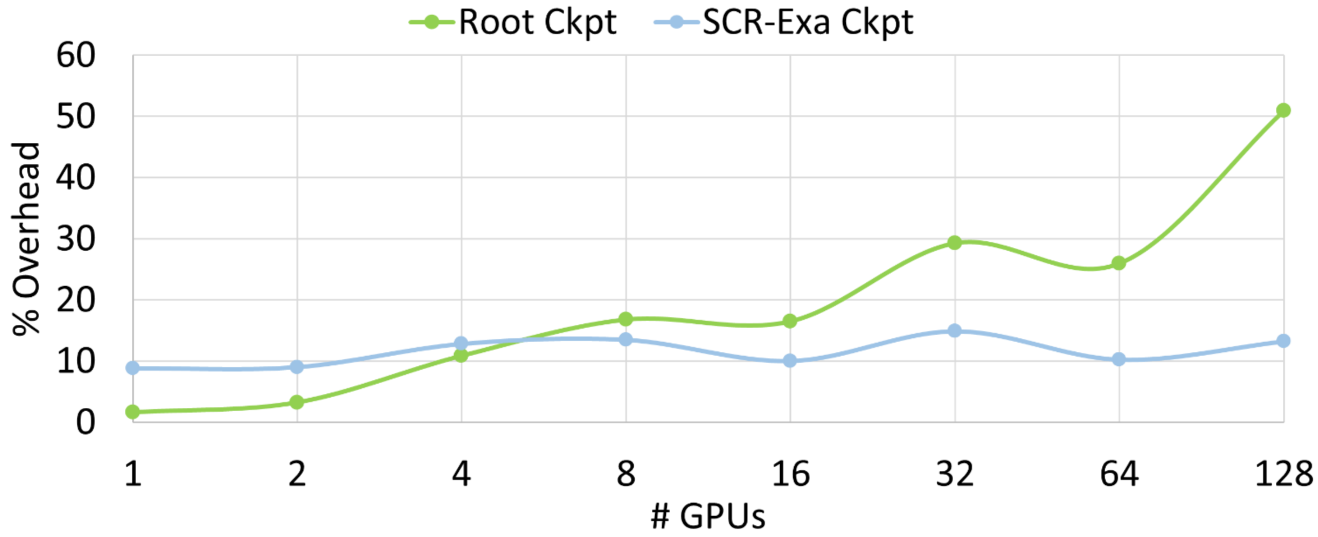
ResNet-50 PyTorch distributed module training time



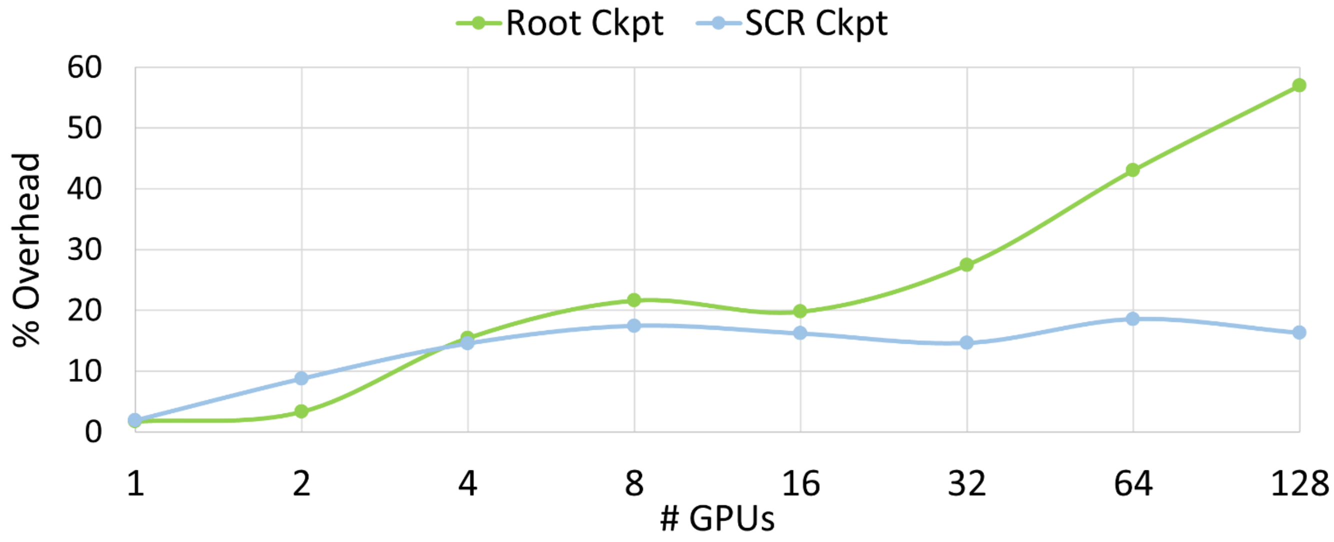
ResNet-50 Horovod training time

Performance Improvement: Save Ckpt (Cont'd)

- Every 10th ckpt is flushed to the PFS with SCR-Exa
- Benefits are clearer when looking at % overhead
- SCR-Exa overhead remains below ~20%
- Root checkpointing overhead increases linearly with GPU count
- SCR-Exa introduces additional overhead below 4 GPUs (1 node)



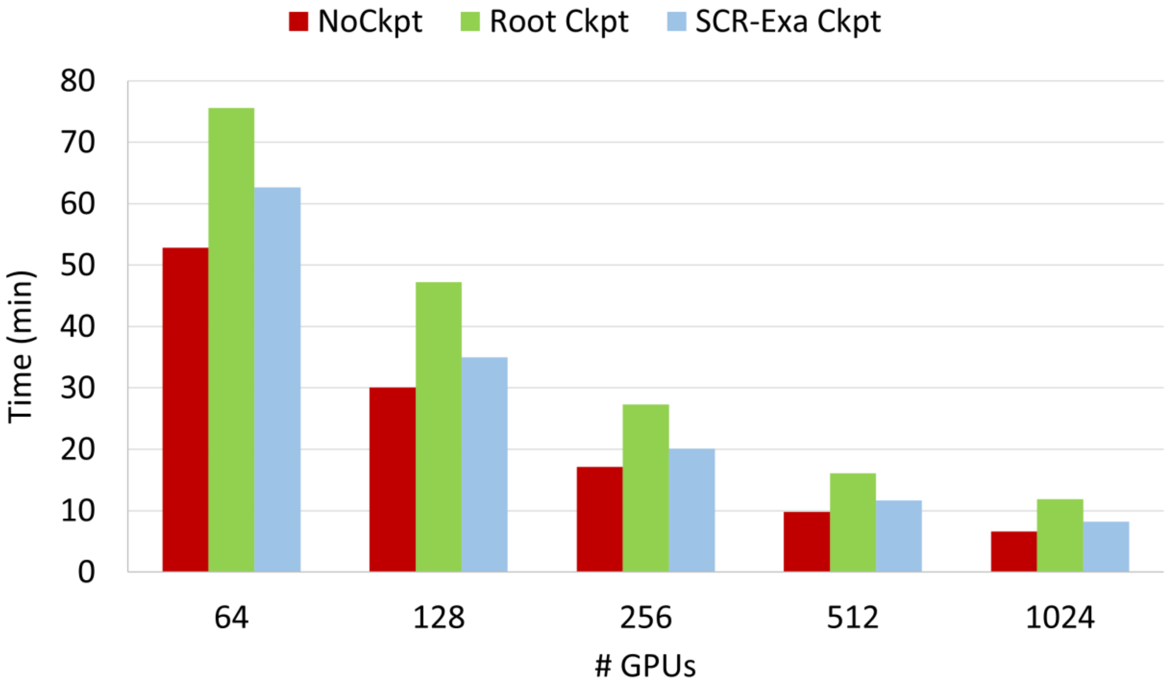
ResNet-50 PyTorch distributed module ckpt overhead



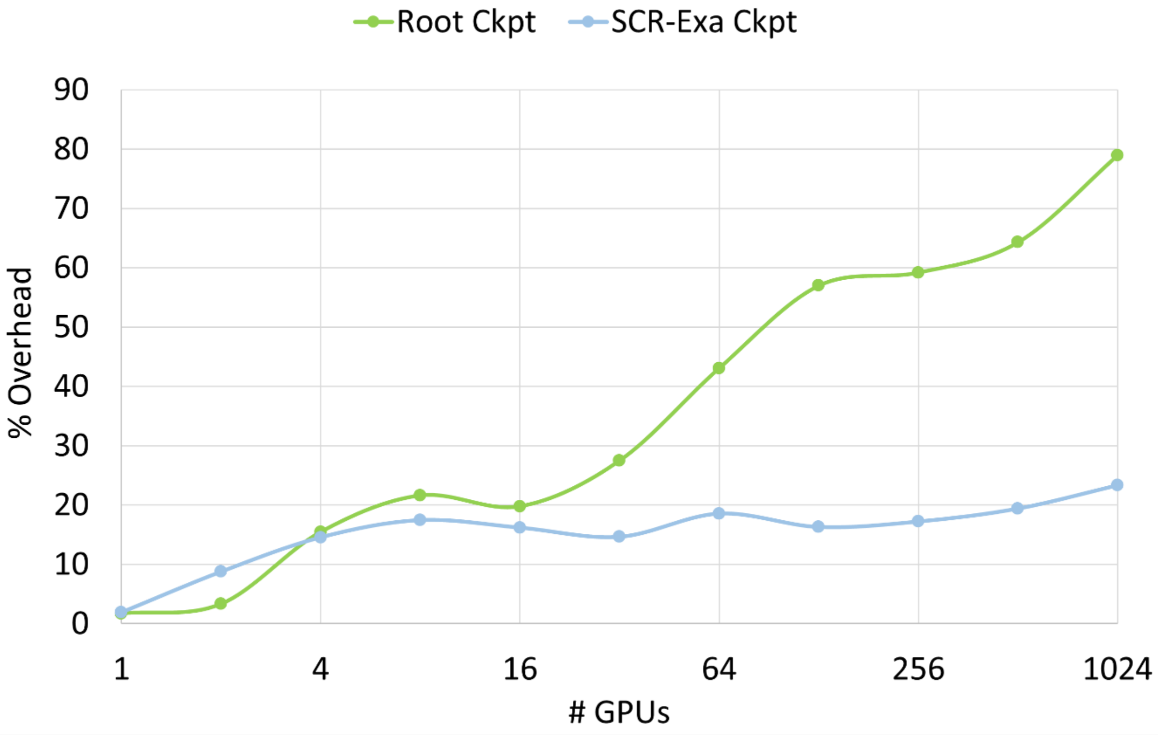
ResNet-50 Horovod ckpt overhead

Performance Improvement: Save Ckpt (cont'd)

- We take the end-to-end training time of 100 epochs of **EDSR** training with only **Horovod**
- **Again** compared root ckpt method with SCR-Exa performance, with 1 ckpt per epoch
- Similar trends as ResNet-50



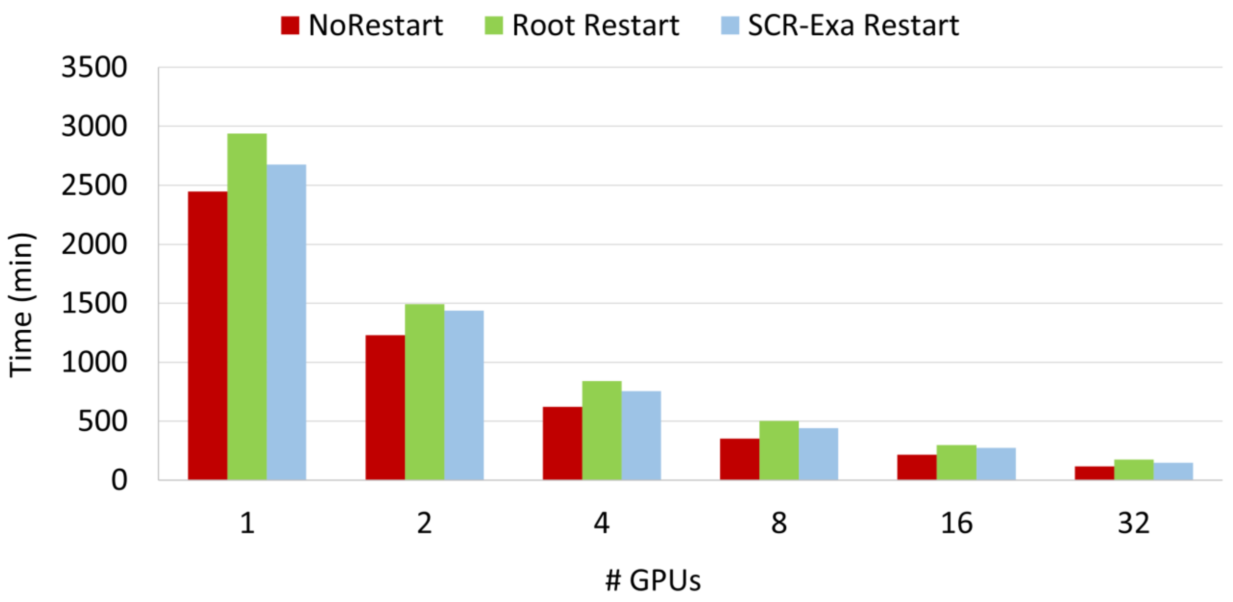
EDSR Horovod training time



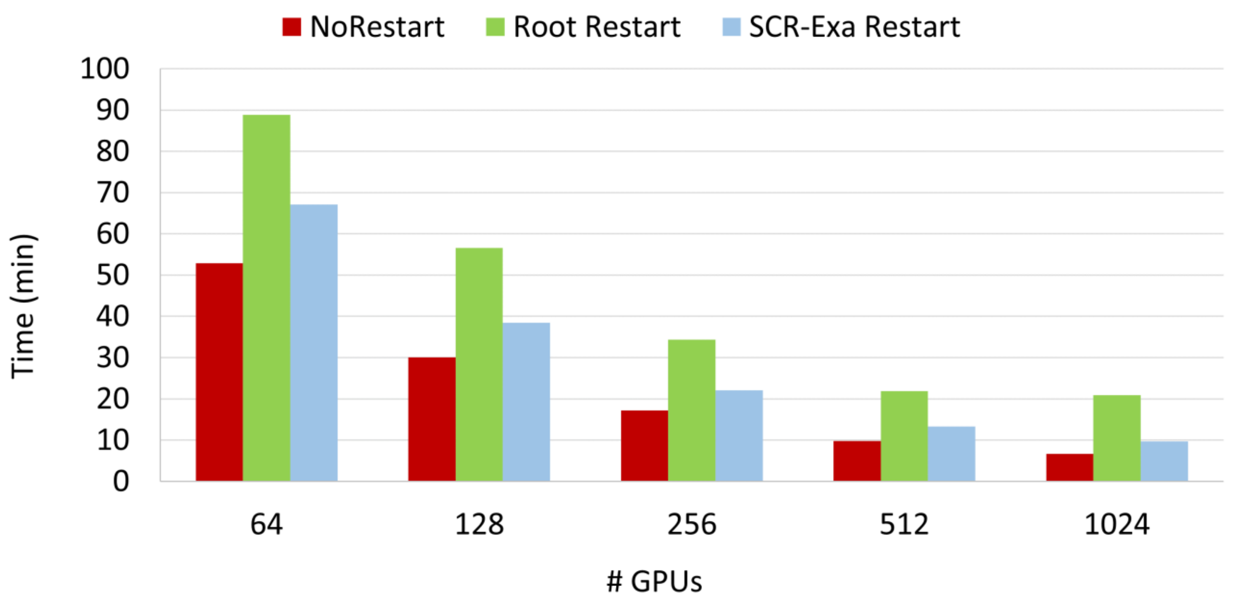
EDSR Horovod overhead

Performance Improvement: Load Ckpt

- Now test ckpt load performance with cold restarts
  - Restart training within the allocation every 10 epochs



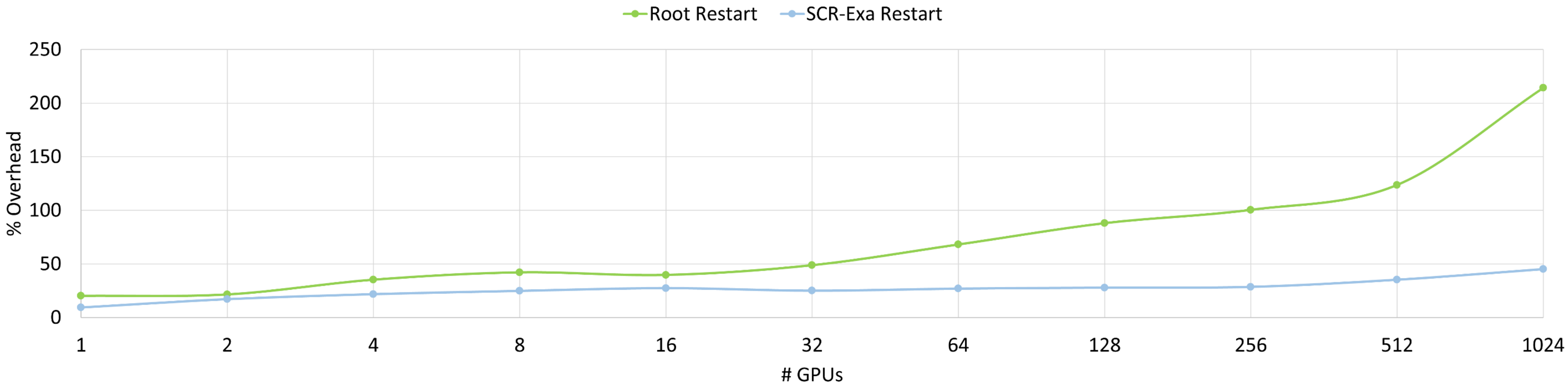
Small-scale EDSR Horovod training time



Large-scale EDSR Horovod training time

Performance Improvement: Load Ckpt (cont'd)

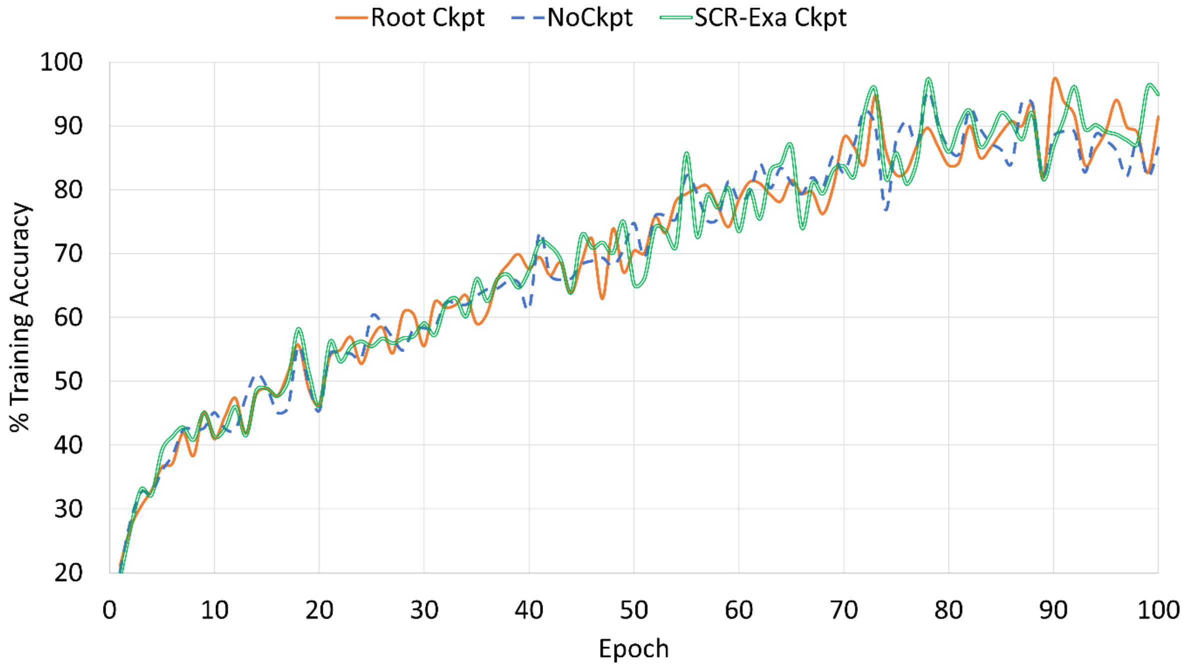
- SCR-Exa significantly outperforms root restarts for cold checkpoints
  - Performance improvement is due to **restart caching**



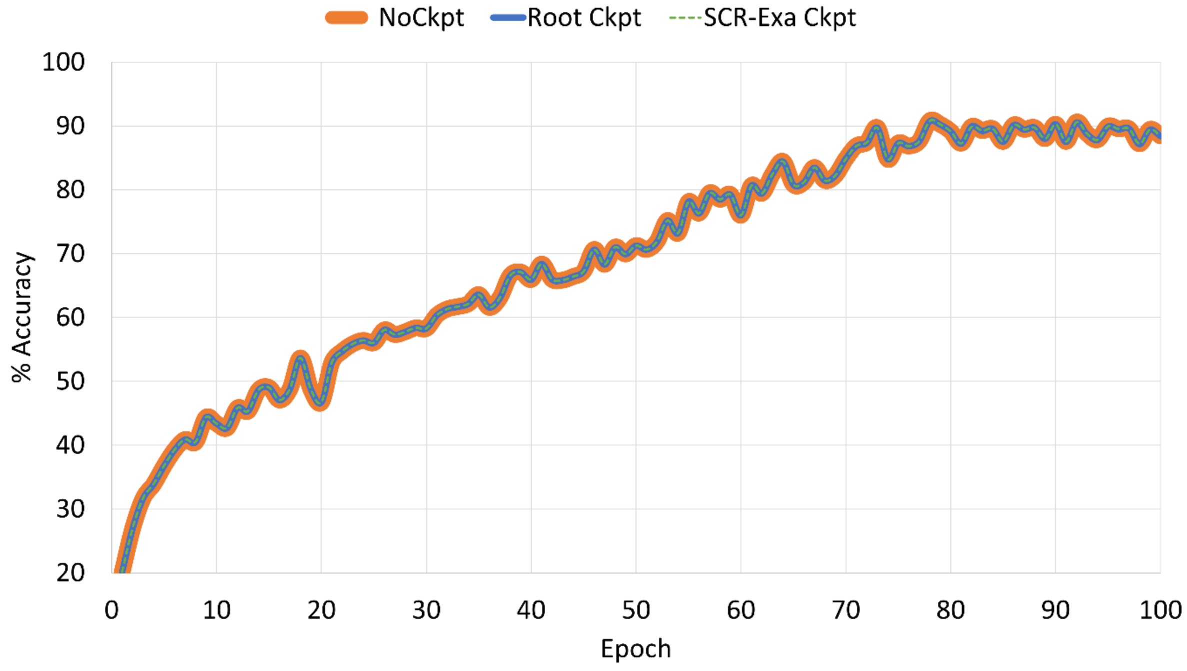
EDSR Horovod restart overhead

Performance Improvement: Training Logs

- We want to ensure checkpoints are saved/loaded without affecting the DNN
- Take training accuracy logs with/without checkpointing with SCR-Exa
- Double-check with deterministic PyTorch and dependencies (i.e. cuDNN and NumPy)



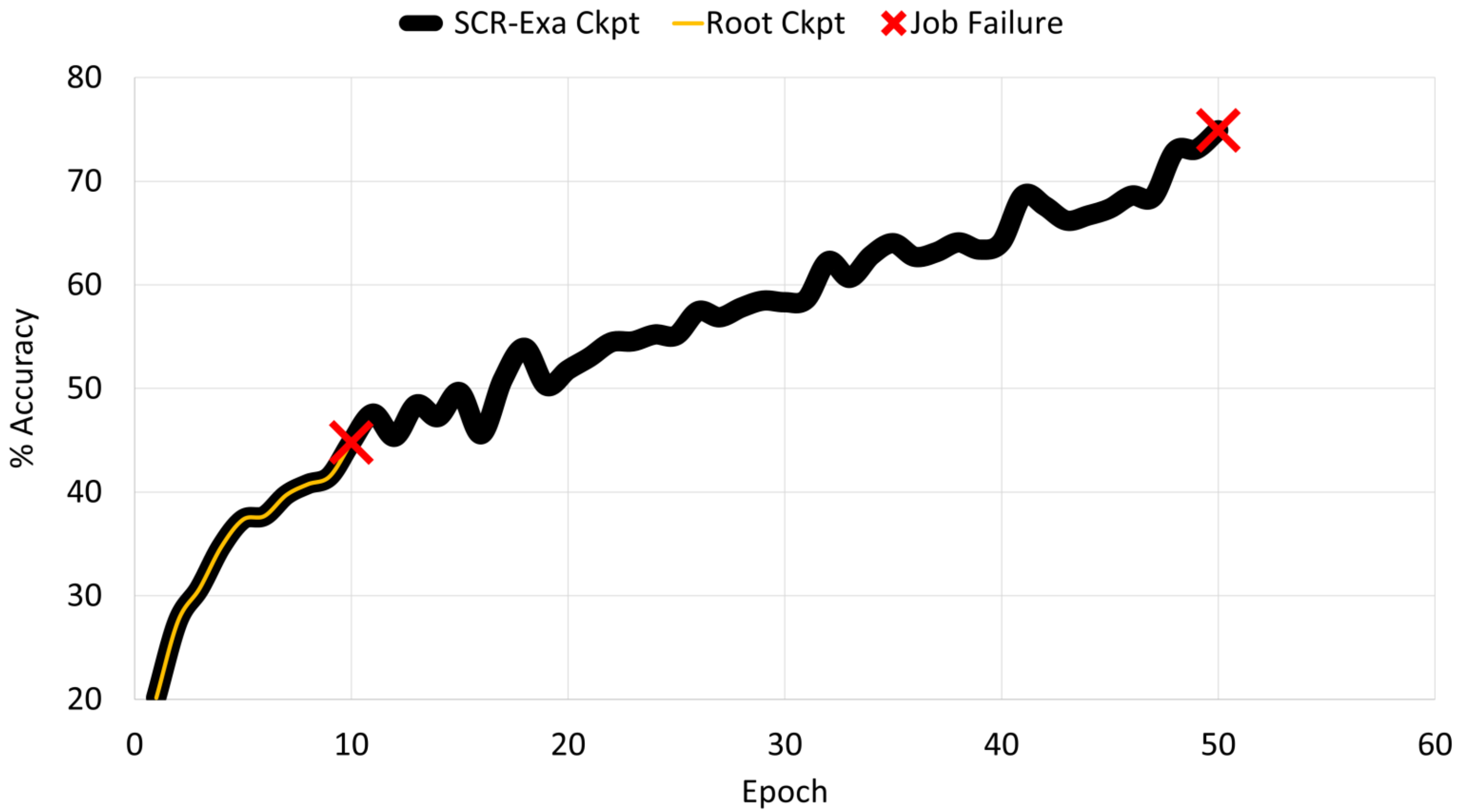
ResNet-50 training logs with ckpt restart



ResNet-50 training logs with deterministic ckpt restart

Performance Improvement: Training Logs

- To demonstrate SCR-Exa's hot restart capability, we simulated a node error every 10 epochs
- We ran with 8 nodes for root checkpointing, and 12 nodes for SCR-Exa (4 spare nodes)
- We expect SCR-Exa to withstand 4 training failures before job failure
- We expect root checkpointing job to fail after the first error



ResNet-50 training logs with deterministic ckpt restart

Agenda

- Introduction
- Background
- Research Challenges
- Methodologies
- Performance Evaluation
  - Evaluation Platforms and Software Libraries
  - Scaling Results
- **Conclusion**

Conclusion

- For bandwidth-bound checkpointing functions (e.g. torch.save() for relatively small DNNs), root checkpointing is not scalable
- Multi-level checkpointing schemes can reduce dependence on the parallel filesystem
- HPC checkpointing tools such as SCR-Exa can be integrated with DL frameworks via Python bindings
- Neither root checkpointing nor SCR-Exa affect DNN convergence

Future Work

- Apply SCR-Exa to other parallelism schemes and distributed DL frameworks
- Introduce detailed profiling to better understand the performance difference between SCR-Exa and root checkpointing

Thank You!

q.anthony@x-scalesolutions.com

http://x-scalesolutions.com/