Methods, Tools and Best Practices for Coupling Simulation and AI on ALCF Systems

Riccardo Balin, Christine Simpson, Harikrishna Tummalapalli

Argonne National Laboratory, LCF

ALCF Developer Session
August 27th, 2025

Overview

- ❑ Motivation and methods for coupling AI and simulation
- ❑ Tools and examples on ALCF systems:
  - ➢ LibTorch for performant inference
  - ➢ ADISO2 for I/O and data streaming
  - ➢ In-memory distributed data-stores with SmartSim and DragonHPC
  - ➢ Use of node-local memory/storage
- ❑ Summary and best practices

Overview

❑ Motivation and methods for coupling AI and simulation

❑ Tools and examples on ALCF systems:

  - ➢ LibTorch for performant inference
  - ➢ ADISO2 for I/O and data streaming
  - ➢ In-memory distributed data-stores with SmartSim and DragonHPC
  - ➢ Use of node-local memory/storage

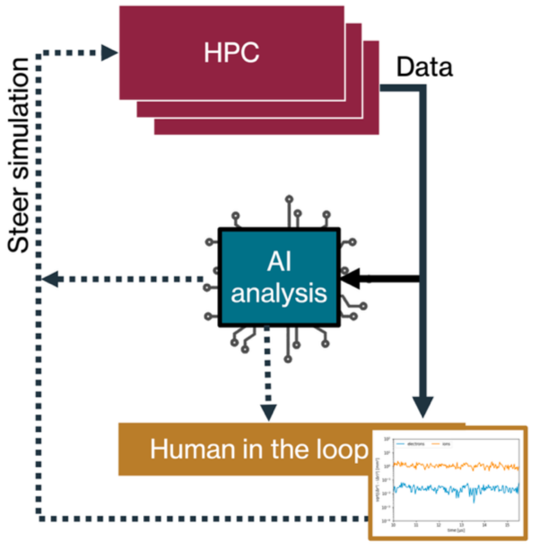
❑ Summary and best practices

Why Couple HPC Simulations with AI/ML?

- ❑ Substitute inaccurate or expensive components of simulation with ML models
    - ➢ E.g.: Closure or surrogate modeling
- ❑ Optimize simulation parameters on-the-fly
    - ➢ E.g.: Select solver parameters at runtime based on AI inference
- ❑ Avoid IO bottleneck and disk storage issues during offline training
    - ➢ E.g.: In situ/online training through data streaming or in-memory staging
- ❑ Active learning and model fine-tuning
    - ➢ E.g.: Continuous fine-tuning and deployment of model
    - ➢ E.g.: Access training data not available during offline pre-training
- ❑ Steering of simulation ensembles
    - ➢ E.g.: Design space exploration or parameter optimization guided by AI

How to Couple HPC Simulations and AI/ML?

Steering of Ensembles



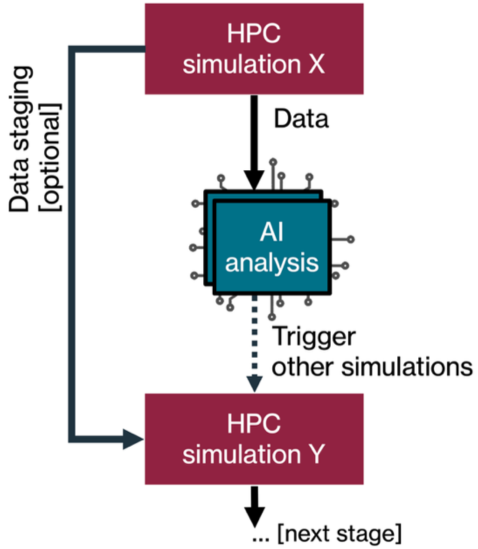
HPC

Data

Steer simulation

AI analysis

Human in the loop



HPC simulation X

Data

Data staging [optional]

AI analysis

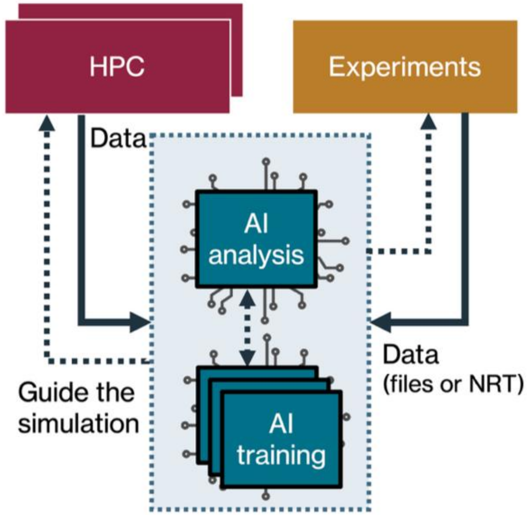
Trigger other simulations

HPC simulation Y

... [next stage]

Optimize Simulation Parameters

Active Learning/ Online Fine-Tuning



HPC

Experiments

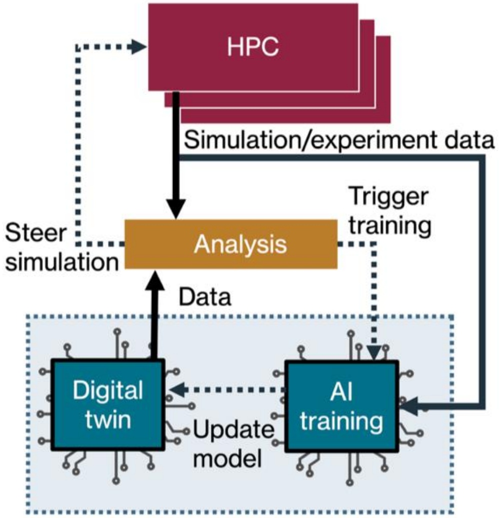
Data

AI analysis

Guide the simulation

AI training

Data (files or NRT)



HPC

Simulation/experiment data

Steer simulation

Analysis

Trigger training

Data

Digital twin

Update model

AI training

Digital Twin

Brewer et al. "AI-coupled HPC workflow applications, middleware and performance." ***arXiv:2406.14315*** **(2024)**

How to Couple HPC Simulations and AI/ML?

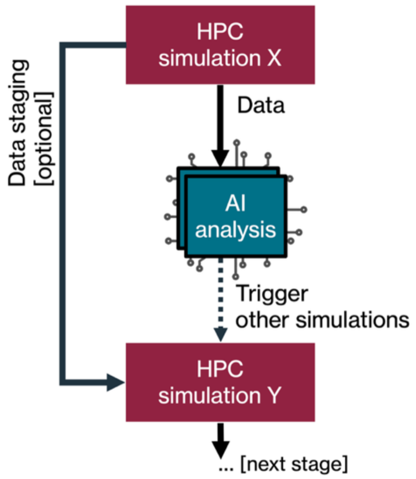
Type of Coupling

□ ML inference only

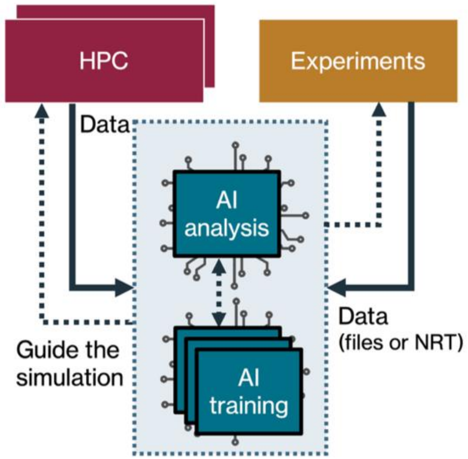
□ ML training only

□ Both training and inference

Inference Only



Training and Inference



Programming Languages and Programming Models

□ Simulation and ML components often written in different languages

□ AI/ML relies heavily on vendor libraries (stuck with vendor language or programming model)

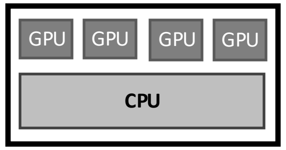
□ Separate or shared parallelism strategies

- ➢ Shared vs. separate MPI communicators
- ➢ Domain decomposition vs. batch or model parallelism

Brewer et al. "AI-coupled HPC workflow applications, middleware and performance." ***arXiv:2406.14315*** **(2024)**

How to Couple HPC Simulations and AI/ML?

Heterogeneous HPC node

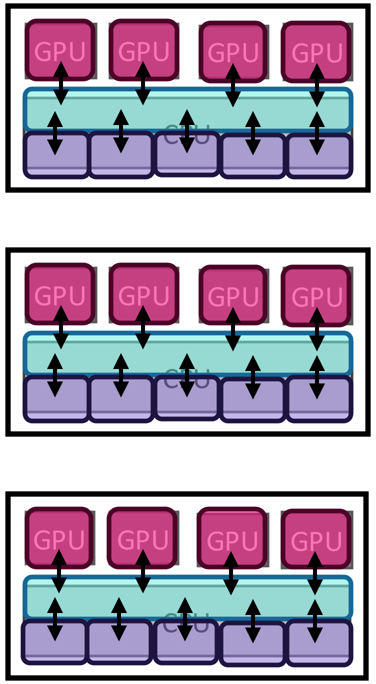


Simulation 🟪
Database 🟦
ML component 🟥
Data transfer ↔

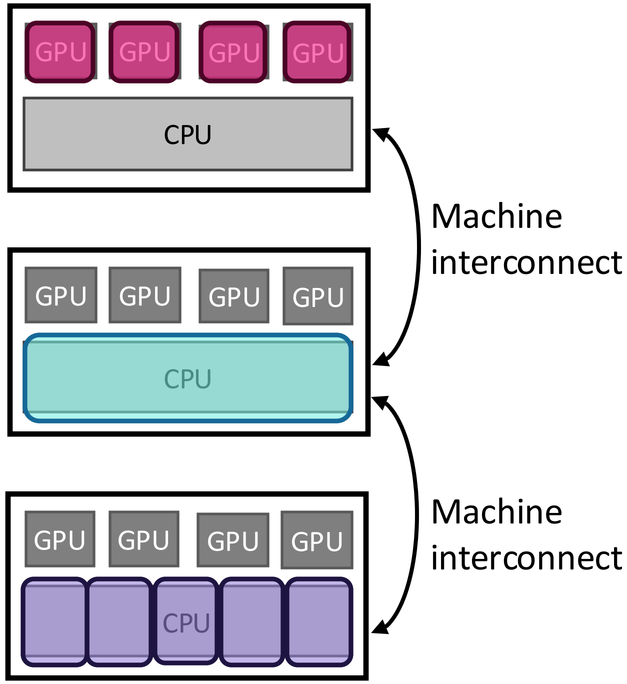
Physical Proximity

- ❑ Colocation: components share the same nodes
- ❑ Node-level clustering: components use different nodes on the same system
- ❑ Multi-system: components are run on separate specialized systems

Colocated Deployment



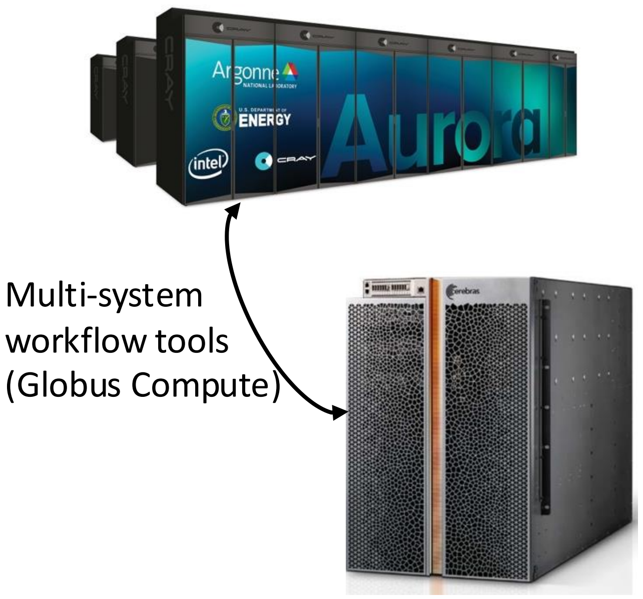
Node Clustered Deployment



Machine interconnect

Machine interconnect

System Clustered Deployment



Multi-system workflow tools (Globus Compute)

How to Couple HPC Simulations and AI/ML?

Data Access

❑ Coupling simulation and ML requires frequent data sharing/transfer between components

❑ Direct: components share same memory space (may allow for zero-copy data transfer)

❑ Indirect: components use distinct logical memory (requires data copy and may require data transfer)

Data Staging or Streaming

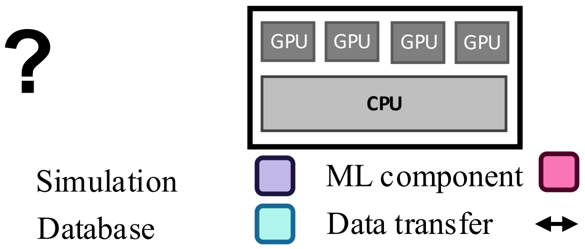
❑ Staging: data is staged in memory or on disk (can reduce idle time but increases number of transfers)

❑ Streaming: data is streamed directly between components (can increase idle time due to synchronization)

Childs et al., "A terminology for in situ visualization and analysis systems", Intl. Journal of High Performance Computing Applications, 2020

How to Couple HPC Simulations and AI/ML?

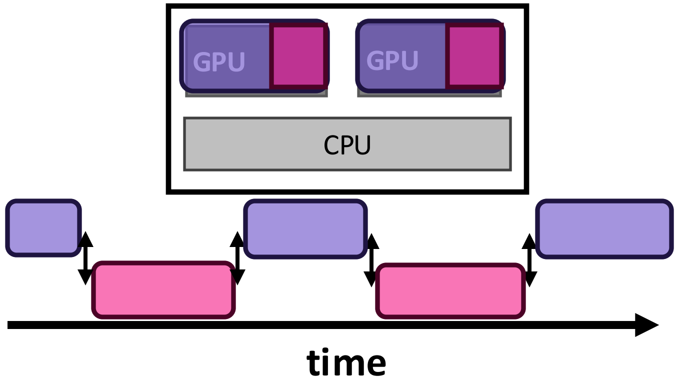
Heterogeneous HPC node



Execution Management

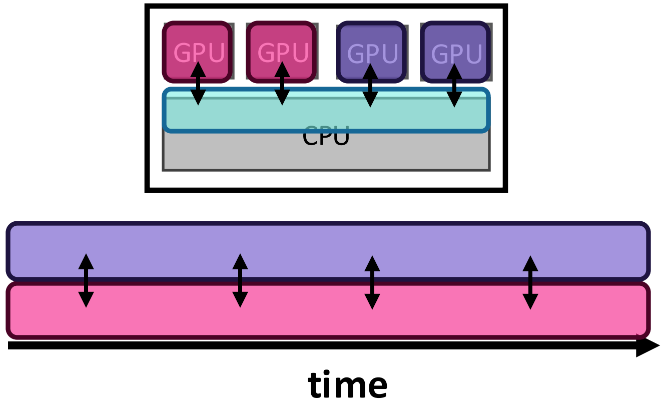
- ❑ Time division (tight coupling)
  - ➢ Components run on same compute resources (may even use same processes)
  - ➢ Staggered in time, execution of one component halts the other
  - ➢ May allow for direct memory access and no data copy/transfer
  - ➢ Idle time of individual components may be significant

Time Division: Same Compute Hardware



- ❑ Space division (loose coupling)
  - ➢ Components run on separate compute resources
  - ➢ Concurrent in time, components run simultaneously
  - ➢ Minimal idle time of components for fast data copy/transfer
  - ➢ Usually requires indirect memory access with data copy/transfer

Space Division: Separate Compute Hardware



Childs et al., “A terminology for in situ visualization and analysis systems”, Intl. Journal of High Performance Computing Applications, 2020

How to Couple HPC Simulations and AI/ML?

❑ Coupling simulation and AI/ML can be a complex space to navigate

❑ Implementation choices can vary significantly depending on ML task and application needs

❑ This talk will cover common tools and approaches supported at ALCF

Overview

❑ Motivation and methods for coupling AI and simulation

❑ Tools and examples on ALCF systems:

- ➢ LibTorch for performant inference
- ➢ ADISO2 for I/O and data streaming
- ➢ In-memory distributed data-stores with SmartSim and DragonHPC
- ➢ Use of node-local memory/storage

❑ Summary and best practices

Software Tools for Simulation-AI Coupled Workflows

❑ Software tools can be grouped by their capabilities

PyTorch SMARTSIM DragonHPC

Task Launching or Kernel Execution

Does the tool enable execution of ML kernels on heterogeneous hardware?

Does the tool allow launching different components with fine-grained control on process placement?

SMARTSIM DragonHPC ADIOS

Data Management

Does the tool provide ways to share, stream, and manage data across components?

SMARTSIM DragonHPC

Persistence Across Jobs

Does the tool support execution across multiple jobs (e.g., by storing checkpoints)?

Software Tools for Simulation-AI Coupled Workflows

❑ Software tools can be grouped by their capabilities or by execution management approach

PyTorch ADIOS

Tight Coupling

Embedding Python or ML framework libraries into C, C++, Fortran simulation codes (e.g., PythonFOAM, TensorFlowFOAM, HONEE, and many others)

SMARTSIM DragonHPC ADIOS

Loose Coupling

Workflow managers and client libraries which enable composition of workflows with multiple components, concurrency of execution, and data sharing/streaming

PyTorch C++ API for Tightly-Coupled Inference

- ❑ PyTorch publicly exposes portions of the C++ API to provide support for
  - ➢ Tensor and mathematical operations through Aten
  - ➢ Automatic differentiation through Autograd
  - ➢ High-level API for training and inference of models, including TorchScript JIT compiled models
- ❑ Support for CPU, CUDA and Intel devices
  - ➢ On Aurora, need Intel Extension for Torch to use Intel GPU until PyTorch 2.8 and for possible optimizations
- ❑ Ideally suited for performant inference from traditional simulation codes
- ❑ Available on Polaris and Aurora through ML Frameworks modules
- ❑ Documentation available for Polaris and Aurora with CMake build instructions and more examples

PyTorch C++ API for Tightly-Coupled Inference

```
#include <torch/torch.h>
#include <torch/script.h>

// Select the device: CPU, CUDA or XPU
torch::DeviceType torch_device;
if (torch::cuda::is_available()) {
  torch_device = torch::kCUDA;
} else if (torch::xpu::is_available()) {
  torch_device = torch::kXPU;
} else {
  torch_device = torch::kCPU;
}

// Load jit-traced model and offload to device
torch::jit::script::Module model;
model = torch::jit::load(argv[1]);
model.to(torch::Device(torch_device));

// Create input tensor with specified dtype and device
auto options = torch::TensorOptions().dtype(torch::kFloat32).device(torch_device);
torch::Tensor input_tensor = torch::rand({1,3,224,224}, options);

// Perform inference
torch::NoGradGuard no_grad; // equivalent to with torch.no_grad()
torch::Tensor model_output = model.forward({input_tensor}).toTensor();
```

PyTorch C++ API for Tightly-Coupled Inference

- ❑ Support for direct memory access on CPU and GPU through torch::from_blob() (zero-copy view of data)
- ❑ On Aurora using SYCL

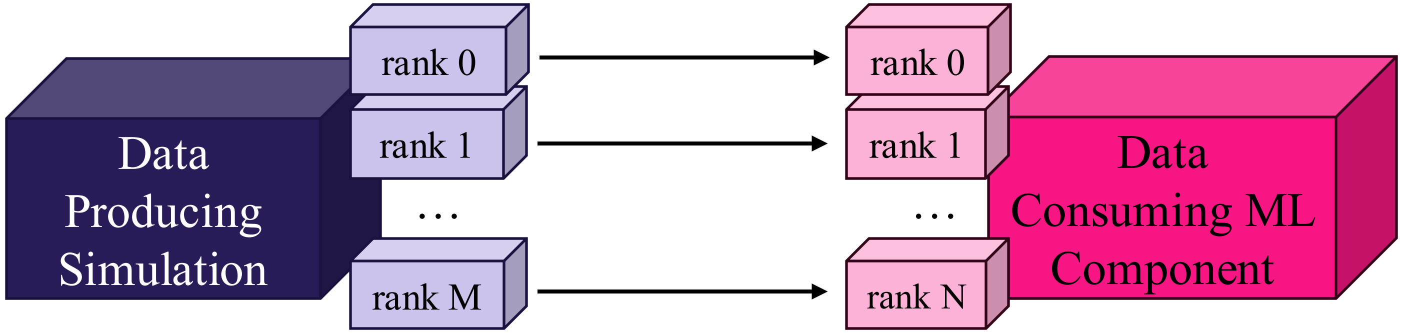
```
// Allocate inputs on device
sycl::queue Q;
Q = sycl::queue(sycl::gpu_selector_v);
float *d_inputs = sycl::malloc_device<float>(INPUTS_SIZE, Q);
Q.fill(d_inputs, 1.0f, INPUTS_SIZE);
Q.wait();

// Create zero-copy view of inputs
auto options = torch::TensorOptions().dtype(torch::kFloat32).device(torch::kXPU);
torch::Tensor input_tensor = torch::from_blob(d_inputs,{N_BATCH,N_FEATURES},options);

// Perform inference (always creates a new tensor)
torch::NoGradGuard no_grad;
torch::Tensor model_output = model.forward({input_tensor}).toTensor();

// Copy outputs to SYCL array
float *d_outputs = sycl::malloc_device<float>(OUTPUTS_SIZE, Q);
auto output_tensor_ptr = model_output.contiguous().data_ptr();
Q.memcpy((void *) d_outputs, (void *) output_tensor_ptr, OUTPUTS_SIZE*sizeof(float));
Q.wait();
```

ADIOS2 for I/O and Data Streaming

- ❑ ADIOS2 is a framework for I/O and streaming of scientific data
- ❑ Developed under U.S. DOE Exascale Computing Project and mostly maintained at OLCF
- ❑ APIs for traditional file system I/O and various data transport approaches (streaming, staging, zero-copy)
- ❑ Provide data streaming through Sustainable Staging Transport (SST) engine
- ❑ Support for C, C++, Fortran, and Python codes
- ❑ Modules available on both Polaris and Aurora, see our Documentation

ADIOS2: Data Streaming with SST

- ❑ Classic data streaming architecture from producer to consumer (or multiple consumers)
- ❑ Designed for HPC with support for RDMA and MPI*, but with fallback to Wide Area Network over sockets
- ❑ Supports MxN model, producer and consumer ranks don’t have to match
- ❑ SST engine ideally suited for workflows with
  - ➢ Single data producer and uni-directional data flow
  - ➢ Low data-reuse (user must implement own data management otherwise)
  - ➢ Fast and scalable data transfers needs across interconnect



***MPI transport layer currently hangs on Aurora**

ADIOS2: Data Streaming with SST

- ❑ SST transport engine offers synchronous and asynchronous streaming

```
adios2::ADIOS adios(comm);
adios2::IO sstIO = adios.DeclareIO("solutionStream");
sstIO.SetEngine("Sst");
adios2::Params params;

if (mode == "sync") {
  params["RendezvousReaderCount"] = "1";
  params["QueueFullPolicy"] = "Block";
  params["QueueLimit"] = "1";

} else if (mode == "async") {
  params["RendezvousReaderCount"] = ”0";
  params["QueueFullPolicy"] = "Discard";
  params["QueueLimit"] = "3";
  params["ReserveQueueLimit"] = "0";
}

params["DataTransport"] = “RDMA”;
params["OpenTimeoutSecs"] = "600"; // useful if consumer takes long to open reader stream
sstIO.SetParameters(params);
```

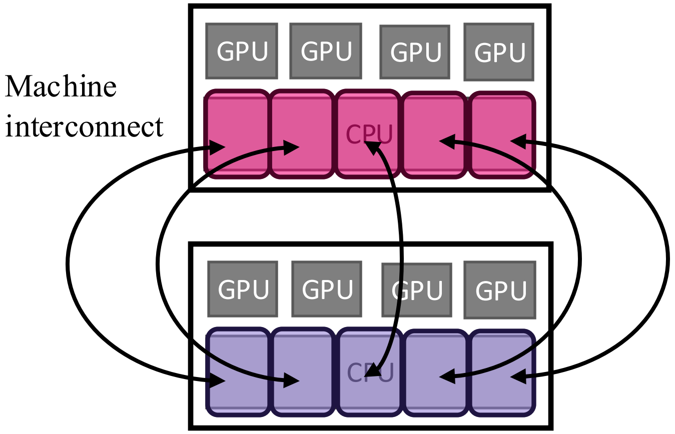
Sync mode

Producer waits for consumer to open stream
Producer is blocked if more than one step in queue
Useful if sequential execution of components wanted
Idle time on producer can be significant for slow consumer

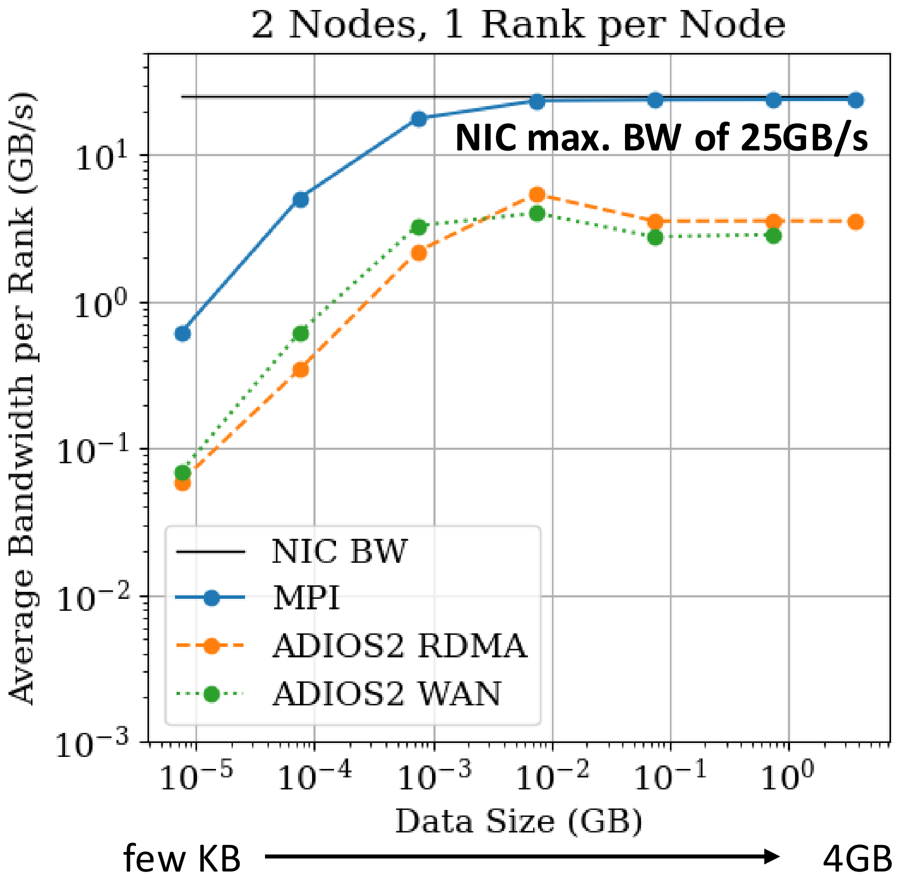
Async mode

Producer proceeds even if consumer not available to read
Producer discards writing step if queue is full and continues
Enables concurrent execution of components with minimal idle time
Steps are discarded with a slow consumer

ADIOS2: Data Streaming with SST

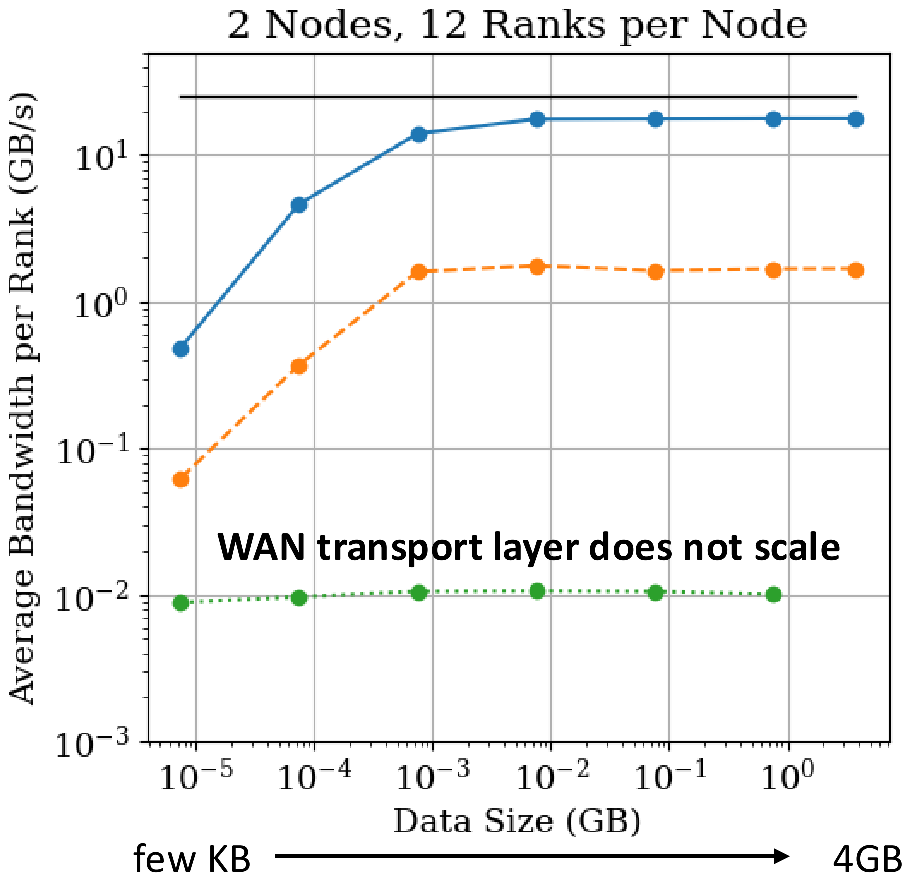


- ❑ SST streaming performance on Aurora with single producer, single consumer clustered example
- ❑ Comparing to MPI send/recv benchmark



NIC max. BW of 25GB/s

few KB → 4GB

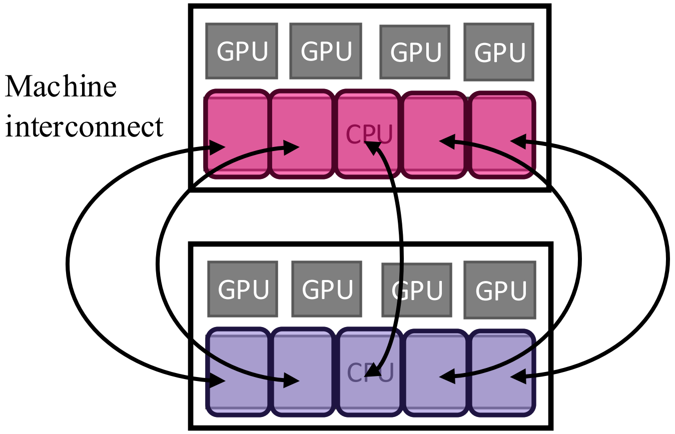


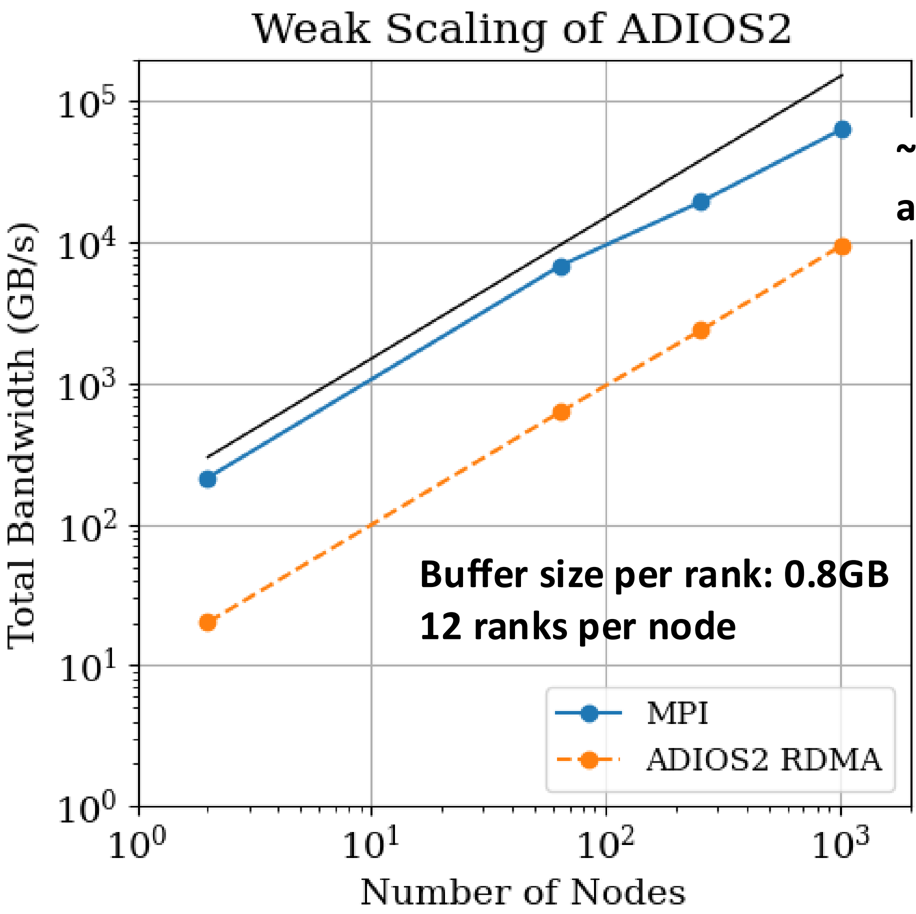
WAN transport layer does not scale

few KB → 4GB

ADIOS2: Data Streaming with SST

- ❑ SST streaming performance on Aurora with single producer, single consumer clustered example



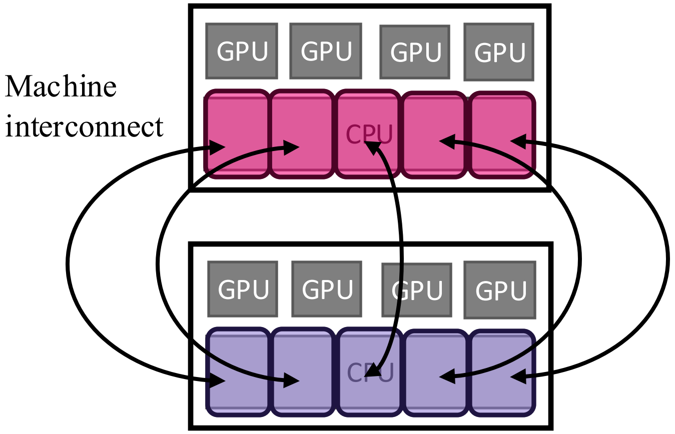


Buffer size per rank: 0.8GB
12 ranks per node

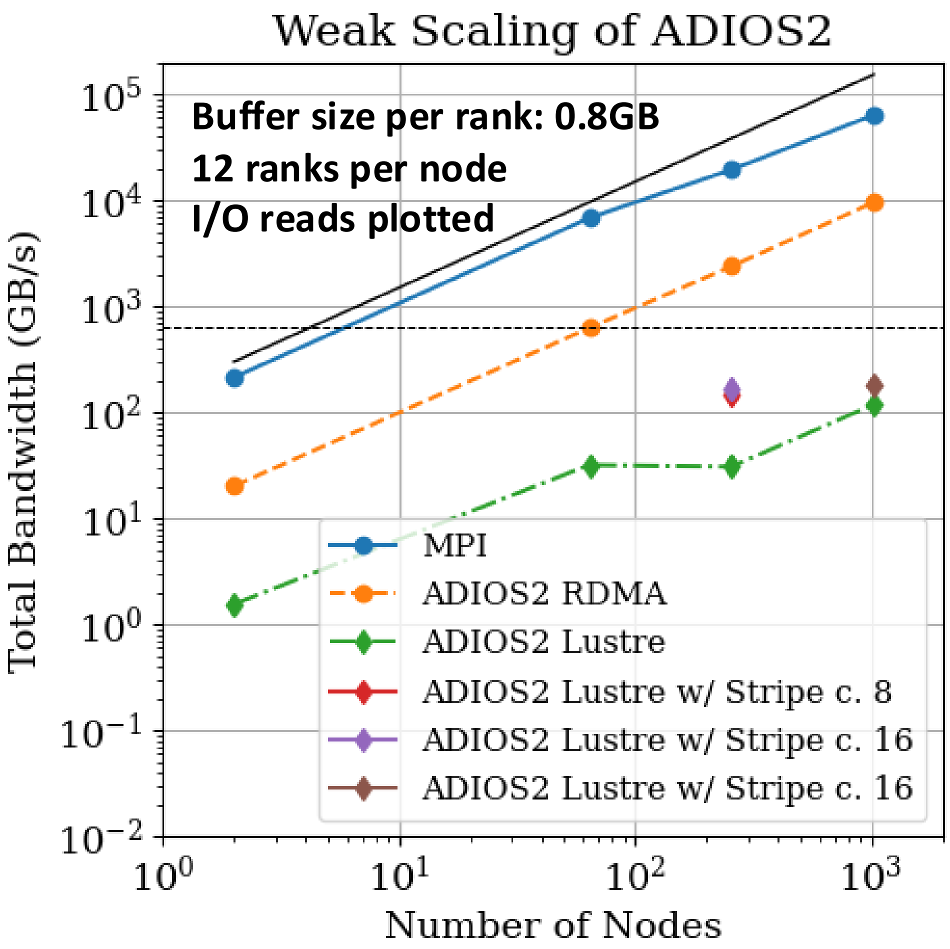
~5TB of total data at 1024 nodes

- ➢ SST streaming with RDMA is a scalable solution
- ➢ Expect similar performance to MPI send/recv benchmark with SST MPI transport and MPMD launch

ADIOS2: SST vs. I/O Performance



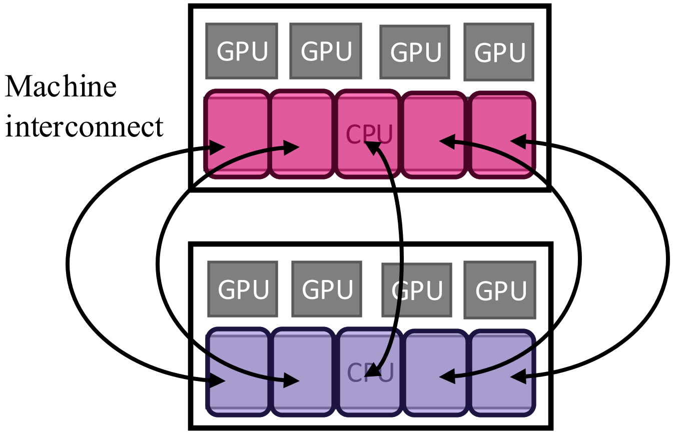
❑ Comparison of SST RDMA to ADIOS2 I/O with BP5 format

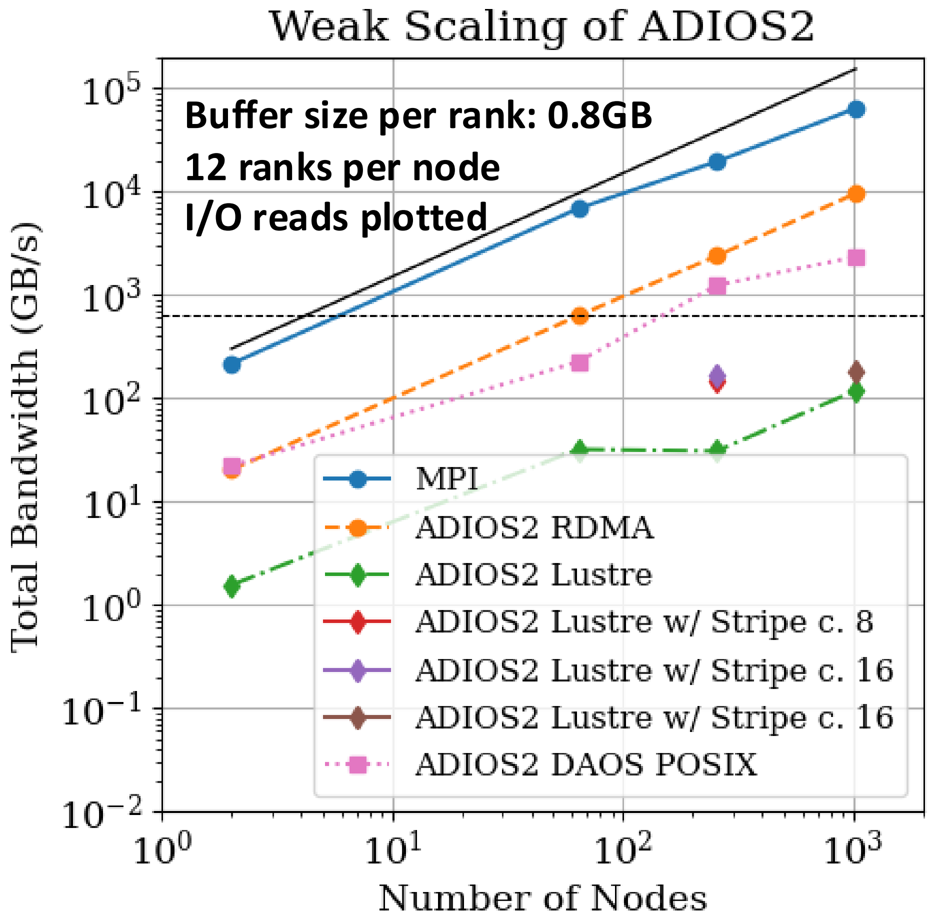


- ➢ SST streaming offers significant performance compared to Lustre I/O
- ➢ Increasing striping count can help with Lustre I/O

ADIOS2: SST vs. I/O Performance

- Comparison of SST RDMA to ADIOS2 I/O with BP5 format
- Using DAOS with POSIX container


Machine interconnect

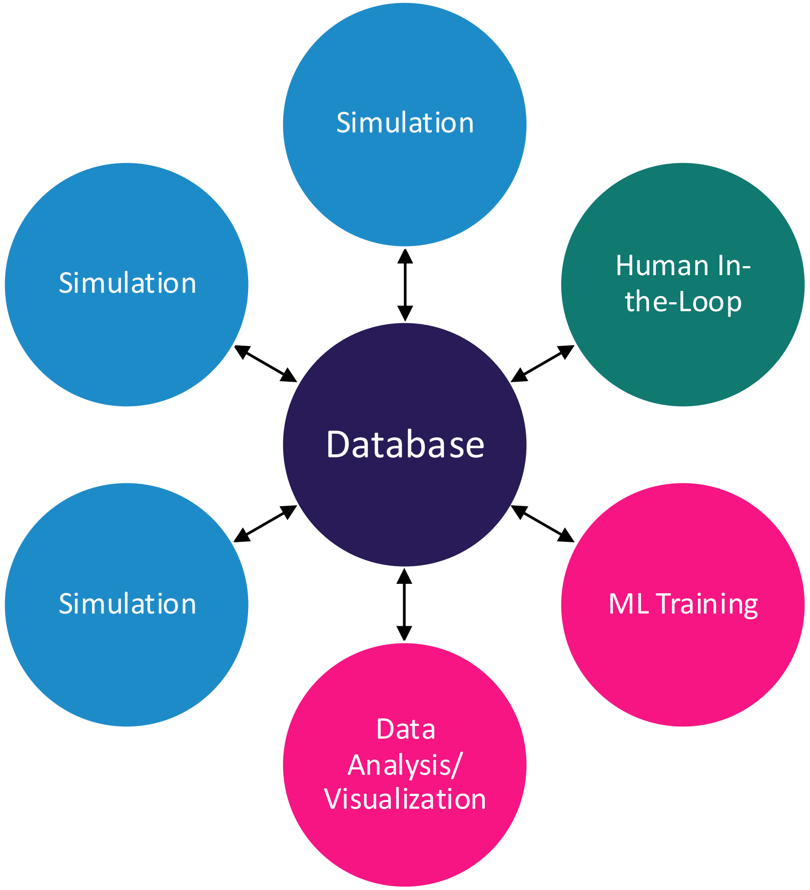

Weak Scaling of ADIOS2

Buffer size per rank: 0.8GB
12 ranks per node
I/O reads plotted

- ADIOS I/O with DAOS significantly improves performance over Lustre
- DAOS is great solution to combine performance with persistent storage of data transferred

SmartSim: Orchestrating Complex Workflows

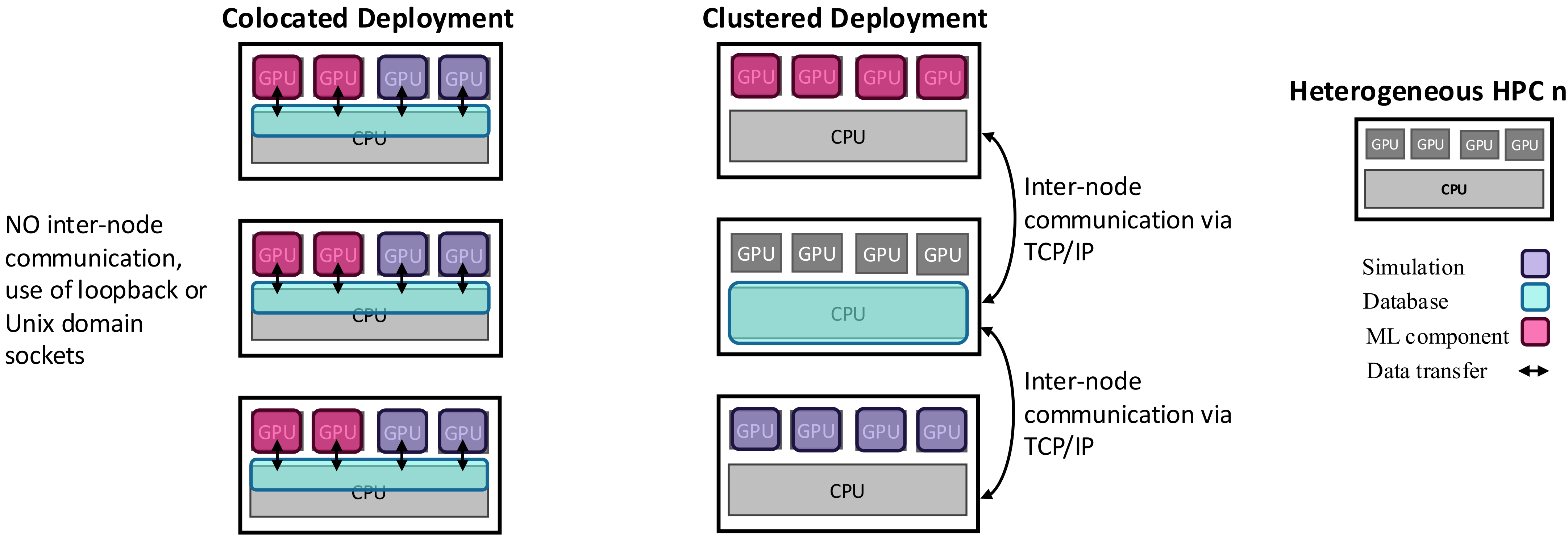
- ❑ Open source tool developed by HPE designed integrate traditional HPC simulations with ML
- ❑ SmartSim Infrastructure library (IL)
  - ➢ Python API to manage workflow and launch HPC applications
  - ➢ Leverages distributed in-memory Redis database for staging and sharing data across components
- ❑ SmartRedis client library
  - ➢ Python, C, C++, Fortran APIs to connect applications to database
  - ➢ APIs for easy data management and model inference
- ❑ Hub and spoke model make SmartSim ideal for
  - ➢ Complex workflows with multiple producers and consumers
  - ➢ Need for data reuse or long-term staging
  - ➢ User interactivity and dynamism (can attach and detach components)
- ❑ See ALCF documentation to install SmartSim on Polaris and Aurora



SmartSim: Orchestrating Complex Workflows

❑ SmartSim allows for clustered or colocated deployment of components

❑ Fine-grained control of process placement with CPU and GPU binding



Colocated Deployment

NO inter-node communication, use of loopback or Unix domain sockets

Clustered Deployment

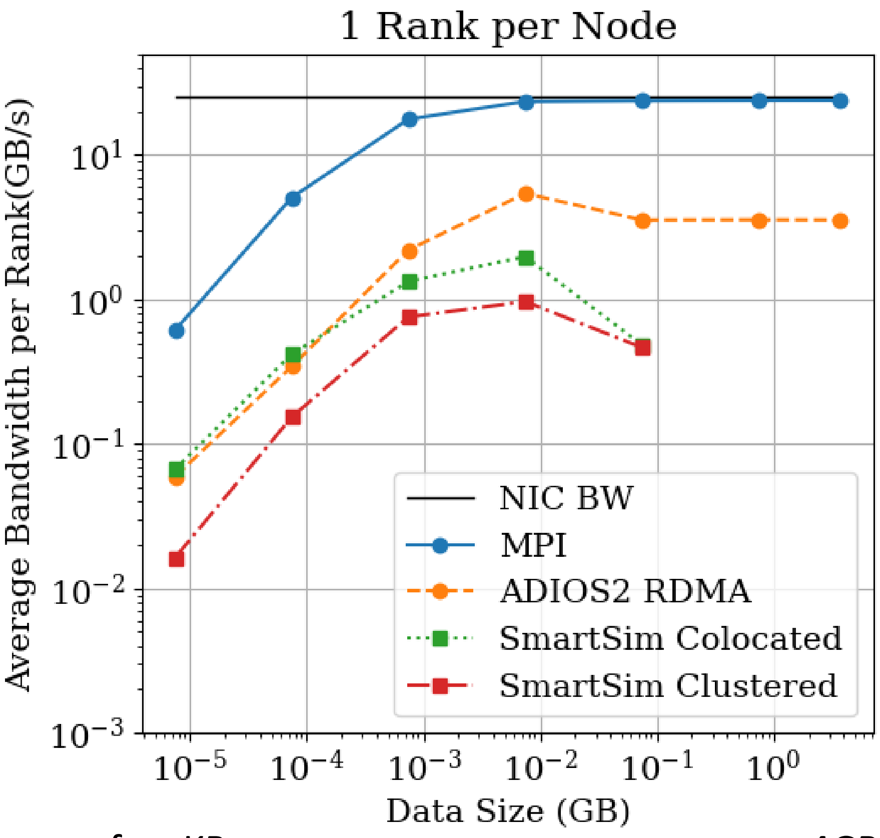
Inter-node communication via TCP/IP

Inter-node communication via TCP/IP

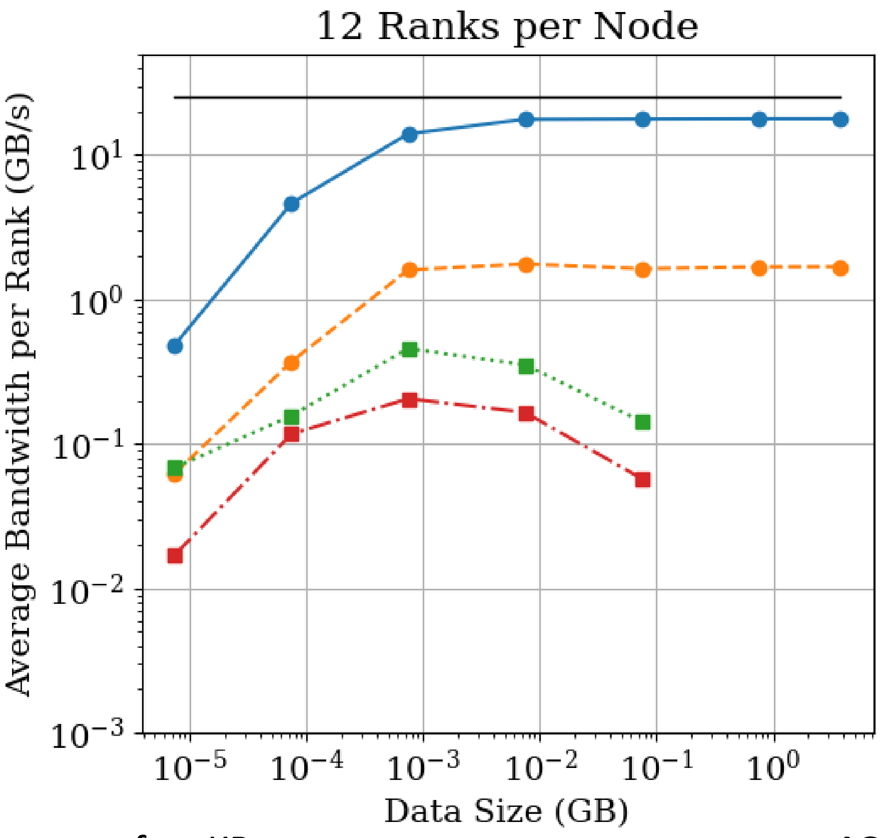
Heterogeneous HPC node

SmartSim: Orchestrating Complex Workflows

❑ SmartSim single producer, single consumer example on Aurora



few KB → 4GB

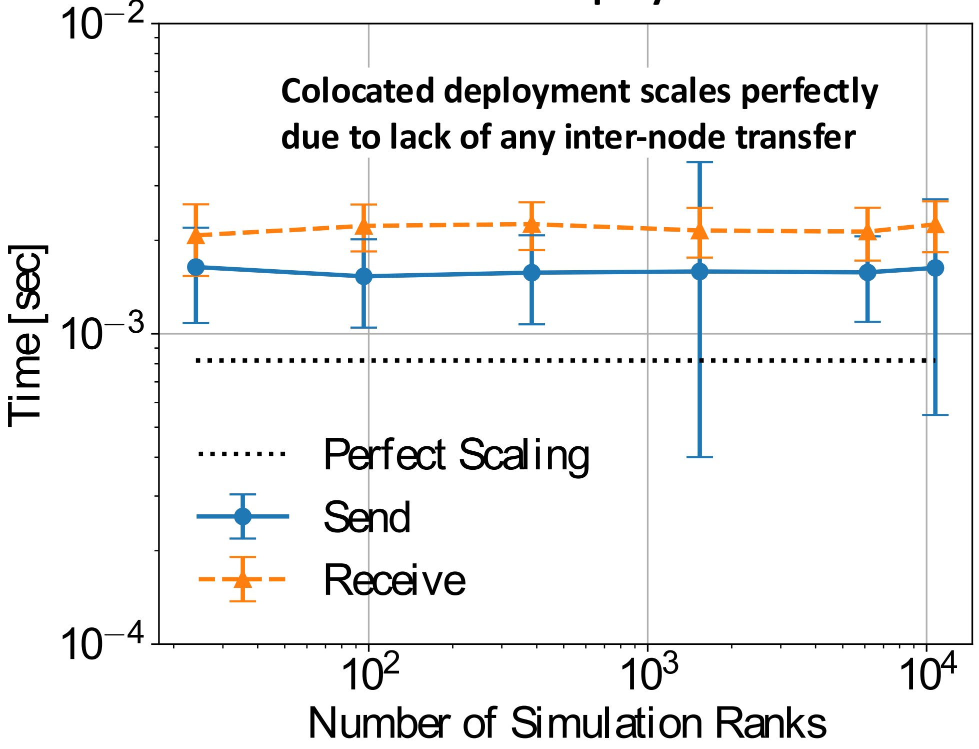


few KB → 4GB

SmartSim: Orchestrating Complex Workflows

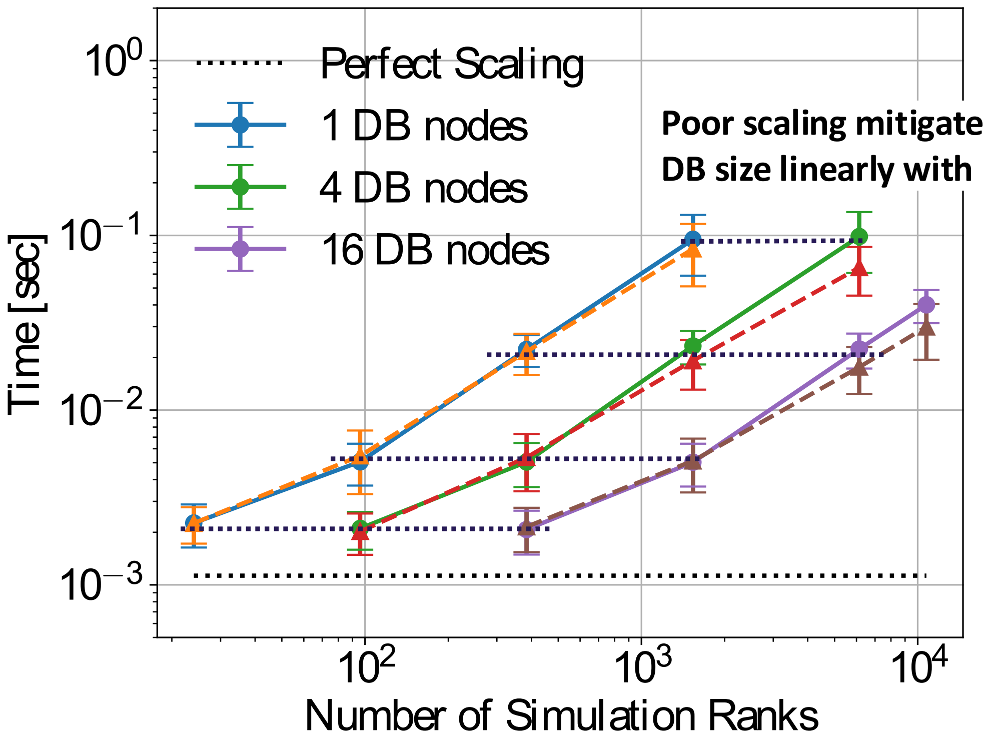
- ❑ SmartSim single producer, single consumer example
- ❑ Weak scaling on Polaris up to 448 nodes with 4 ranks per node
- ❑ Similar results for colocated deployment observed up to 2048 Aurora nodes

Colocated Deployment



Colocated deployment scales perfectly due to lack of any inter-node transfer

Clustered Deployment



Poor scaling mitigated by increasing DB size linearly with number of clients

Balin et al., "In Situ Framework for Coupling Simulation and Machine Learning with Application to CFD". arXiv:2306.12900 (2023)

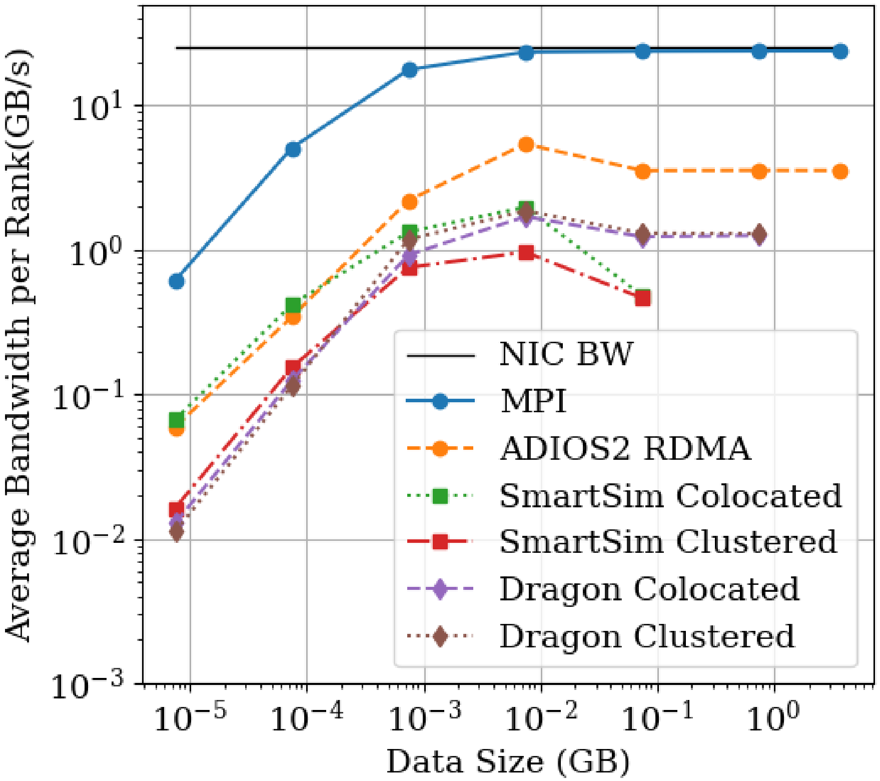
DragonHPC: Scalable Python for HPC

- ❑ DragonHPC is a composable distributed run-time for managing processes, memory, and data at scale through high-performance communication objects
- ❑ Open-source project developed by HPE
- ❑ Key features:
  - ➢ Multi-node extension to Python multiprocessing (mp.Process, mp.Pool, mp.Queue, …)
  - ➢ Managed memory through sharded dictionary objects (DDict) available from Python and C++ clients
  - ➢ Parallel process launching, including PMI enabled for MPI applications with fine-grained control of CPU/GPU affinity
  - ➢ TCP and RDMA transport agents for inter-node communication
  - ➢ Live visualization of telemetry data, including hardware utilization and custom performance metrics
- ❑ DragonHPC is ideal for
  - ➢ Extending Python workflows from local machine to HPC clusters
  - ➢ Complex workflows with multiple producers and consumers
  - ➢ Need for data reuse or long-term staging thanks to DDict
  - ➢ Improved inter-node performance over SmartSim

DragonHPC: Scalable Python for HPC

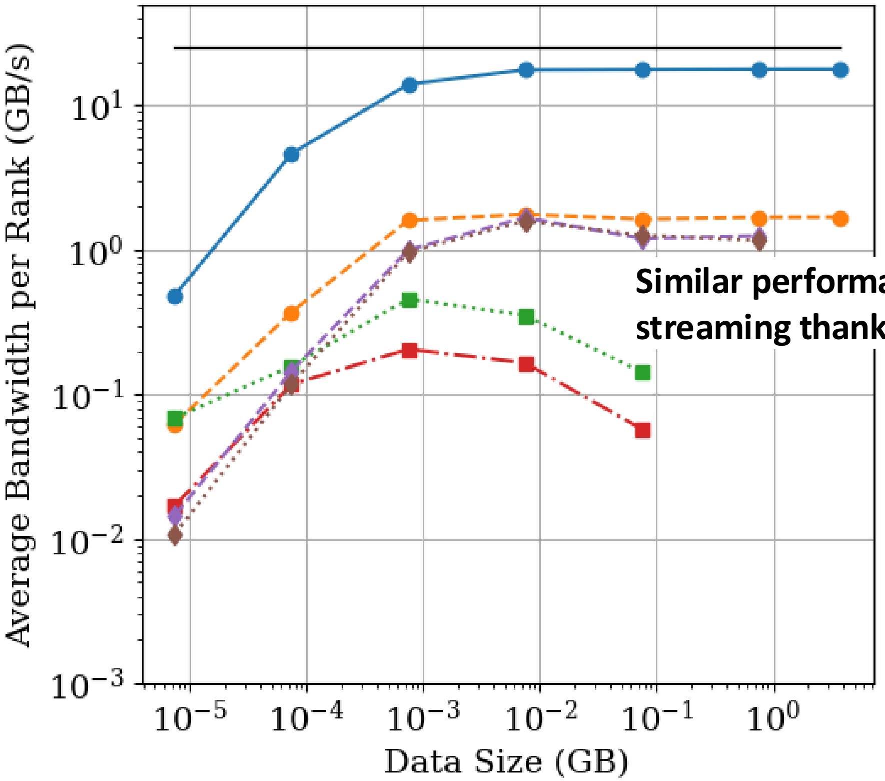
❑ DragonHPC single producer, single consumer example (colocated and clustered deployment with DDict)

1 Rank per Node



few KB → 4GB

12 Ranks per Node

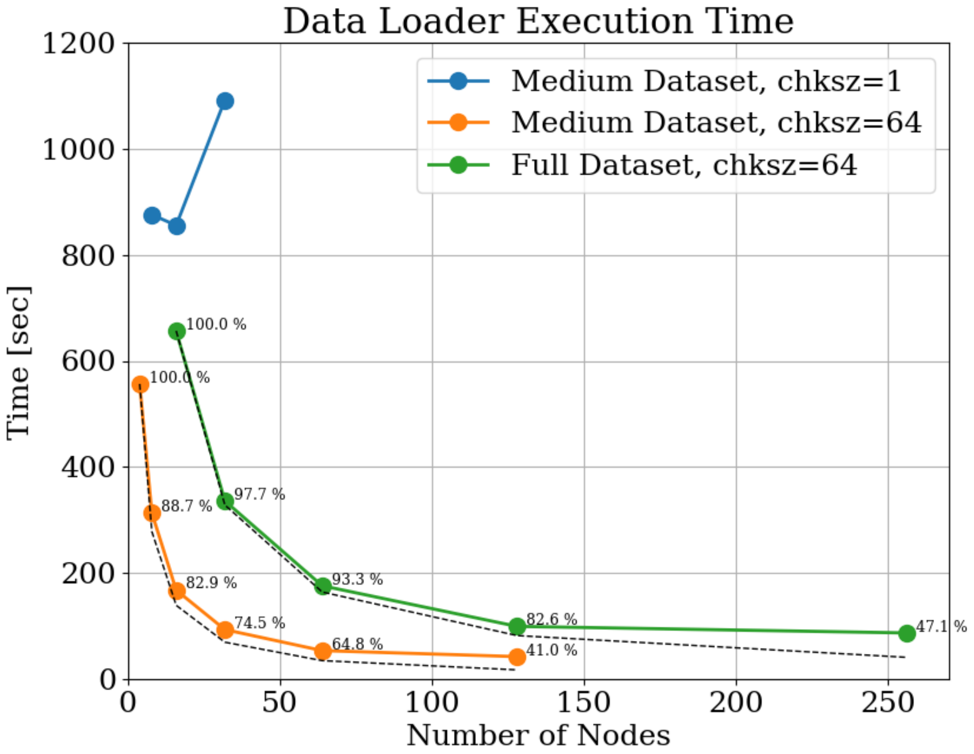


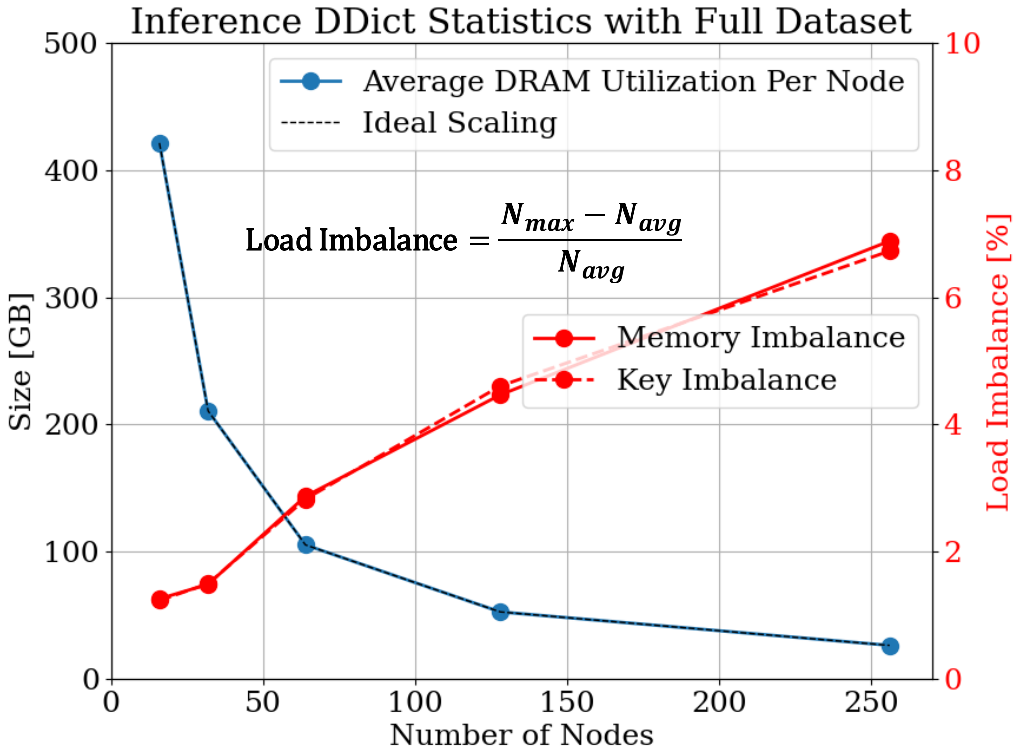
few KB → 4GB

Similar performance to ADIOS2 SST streaming thanks to RDMA transport

DragonHPC: Scalable Python for HPC

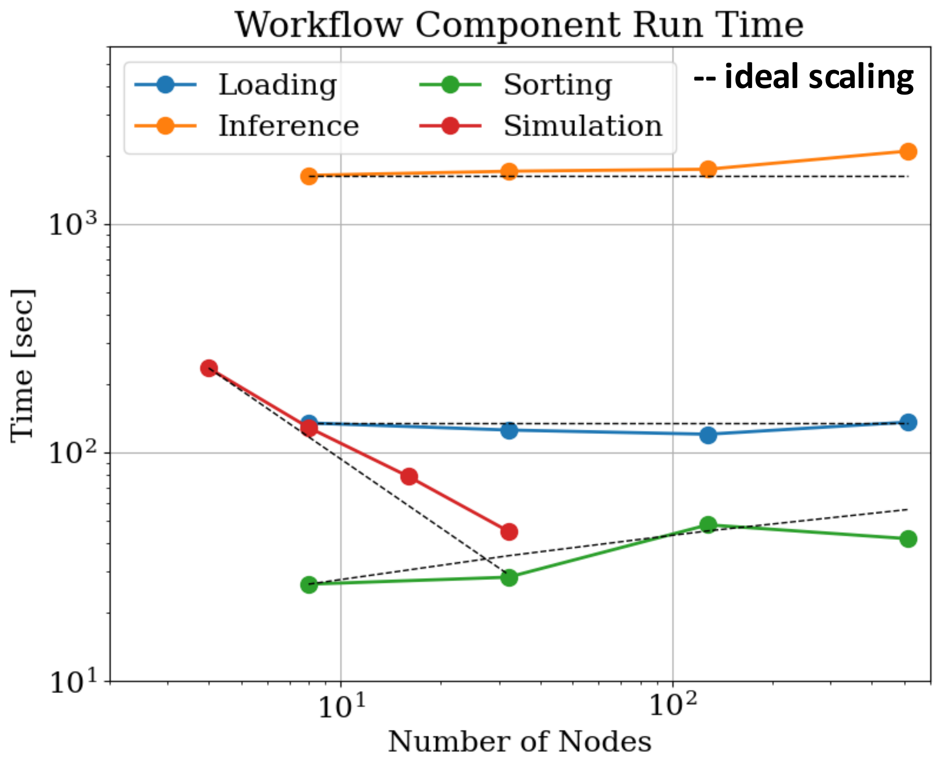
- ❑ Data loading of large datasets into distributed DDict with multi-node mp.Pool
  - ➢ Medium dataset: 1.5TB and 131,072 files
  - ➢ Full dataset: 6TB and 500,354 files



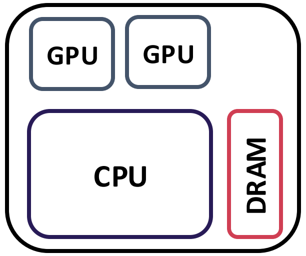


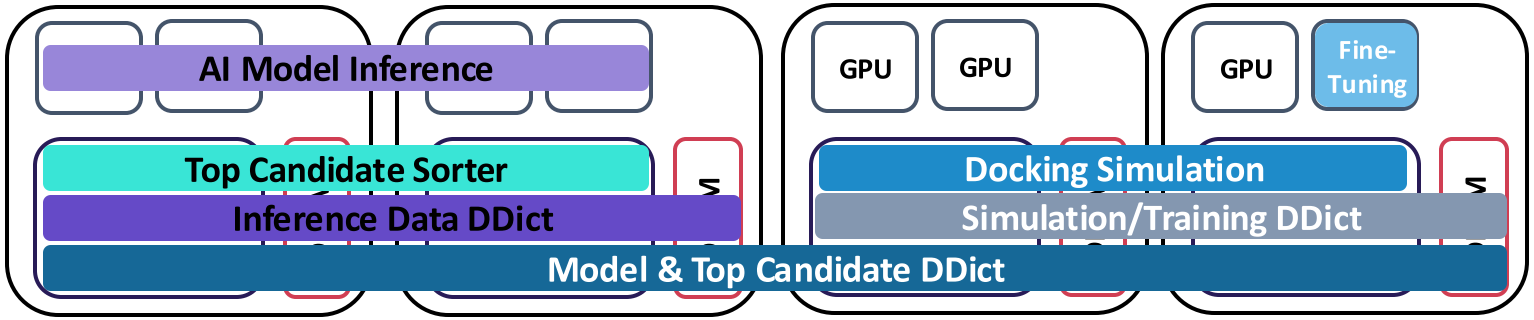
DragonHPC: Scalable Python for HPC

- ❑ Scalable workflow for fine-tuning AI surrogates for drug discovery
  - ➢ Screen large compound dataset with AI model
  - ➢ Identify top candidate based on docking score
  - ➢ Perform docking simulations of candidate compounds
  - ➢ Fine-tune AI model
- ❑ Concurrently deploy components and DDict to enable data staging and re-use across iterations
- ❑ Colocate data with compute processes
- ❑ Workflow deployed up to 544 Aurora nodes



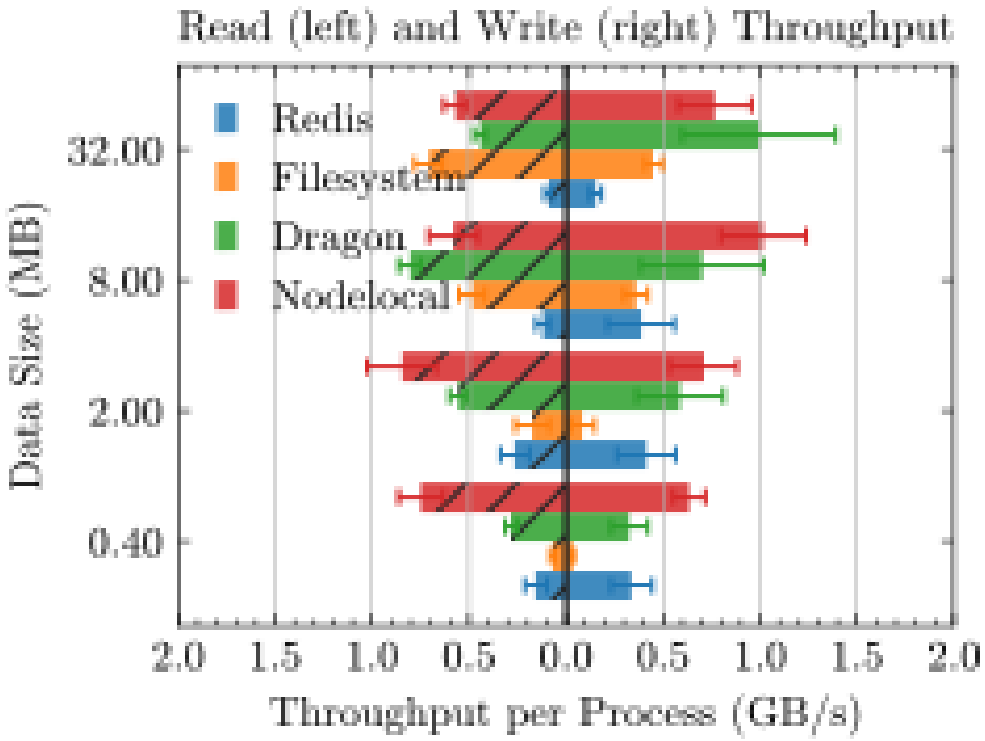
Example of HPC system heterogeneous node





Node-Local Storage for Colocated Solutions

- ❑ Workflows benefiting from colocated deployment solutions can use node-local storage or memory
- ❑ On Aurora, only DRAM is available through /tmp
- ❑ On Polaris, can use pair of 1.6TB SSDs through /local/scratch or DRAM through /tmp
- ❑ Provides fast I/O (especially small data sizes) and simple implementation



Overview

- ❑ Motivation and methods for coupling AI and simulation
- ❑ Tools and examples on ALCF systems:
  - ➢ LibTorch for performant inference
  - ➢ ADISO2 for I/O and data streaming
  - ➢ In-memory distributed data-stores with SmartSim and DragonHPC
  - ➢ Use of node-local memory/storage
- ❑ Summary and best practices

Summary

- ❑ Coupling AI with traditional HPC simulation can accelerate science campaigns, but deployment and integration can be challenging
- ❑ ALCF users can leverage various approaches and software tools to best fit their needs
- ❑ User guide to sim-AI coupled workflows
  - ➢ Fast inference? Tight-coupling and direct memory access with PyTorch C++ API
  - ➢ Uni-directional data flow between large scale producer and consumer(s)? Streaming with ADIOS SST
  - ➢ Combine data sharing during online workflow with persistent storage? Leverage DAOS on Aurora
  - ➢ Complex, multi-component workflows with multiple producers and consumers? Use SmartSim or DragonHPC
  - ➢ Staging of large datasets with data reuse? Leverage SmartSim database and DragonHPC Distributed Dictionary
  - ➢ Need integration with LLMs? Check out our Inference Endpoints!

Best Practices for Sim-AI Workflows on HPC Systems

- ❑ Limit I/O and and parrel file system usage by streaming data across components
  - ➢ Remember DAOS on Aurora can be a performant solution!
- ❑ Ensure high data-reuse by staging relevant data in memory
- ❑ Ensure good resource utilization by leveraging available CPU & GPU in colocated deployments and exploiting concurrency of tasks when possible
  - ➢ Make use of CPUs, especially on Aurora!
- ❑ Achieve good scalability by reducing data transfer across network with colocation of data and compute or using fast streaming solutions

Questions?

For further questions or issues, please reach out to ALCF support at support@alcf.anl.gov

This research used resources of the Argonne Leadership Computing Facility (ALCF), which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357.

User Guide to Sim-AI Coupled Workflows

❑ Fast inference? Tight-coupling and direct memory access with PyTorch C++ API

❑ Uni-directional data flow between large scale producer and consumer(s)? Streaming with ADIOS SST

❑ Combine data sharing during online workflow with persistent storage? Leverage DAOS on Aurora

❑ Complex, multi-component workflows with multiple producers and consumers? Use SmartSim or DragonHPC

❑ Staging of large datasets with data reuse? Leverage SmartSim database and DragonHPC Distributed Dictionary

❑ Need support for multiple programming languages? Build workflow with SmartSim/SmartRedis

❑ Can leverage Python? Utilize DragonHPC

❑ Need GPU-direct data transfer? Use ADIOS2

❑ Need scalability to 1000s of nodes? Colocated deployment of components with SmartSim or DragonHPC

❑ Need integration with LLMs? Check out our Inference Endpoints!