A DEEP REINFORCEMENT AGENT FOR RESOURCE SCHEDULING WITH SUNFISH IN A COMPOSABLE DISAGGREGATED INFRASTRUCTURE

Catherine Appleby, Computer Scientist, Applied Machine Intelligence
Michael Aguilar, EIT, PI, Senior Computer Scientist, HPC Research and Development

Sandia National Laboratories
A Deep Reinforcement Agent for Resource Scheduling with Sunfish in a Composable Disaggregated Infrastructure

Catherine Appleby, Computer Scientist, Applied Machine Intelligence, Sandia National Labs

Michael Aguilar, EIT, PI, Senior Computer Scientist, HPC Research and Development, Sandia National Labs

Christian Pinto, Phil Cayton, Russ Herrell, Richelle Ahlvers, Alex Lovell-Troy

OpenFabrics Alliance Workshop

April, 2024
A Deep Reinforcement Agent for Resource Scheduling with Sunfish in a Composable Disaggregated Infrastructure

1. Why Composable Disaggregated Infrastructure (CDI)
2. Design Considerations for a Composability Manager on a Large-Scale HPC system
3. Sunfish
4. Deep Reinforcement Learning for Resource Allocation
5. Integrations
6. Acknowledgements and Questions
Why Composable Disaggregated Infrastructure (CDI)

Current Beowulf architectures
- Larger HPC systems create a larger potential impact of stranded resources
- Resource limits are fixed for each compute node
- Need to build out components to address all possible types of application codes that the HPC must support
- Hardware failures during run-time can kill running batch applications

CDI
- Mitigate stranded 'wasted' resources
- Dynamically utilize hardware resources such as CPUs, GPUs, and memory
  - Enables better application workload matching to increase application efficiency
- Dynamically apply resources to potentially abate out-of-memory conditions, abate IO thrashing, route around network connection failures, and reduce batch job failures due to hardware failures

Fun fact: 2% of the US's energy consumption is input into Datacenters: https://www.energy.gov/eere/buildings/datacenters-and-servers
Augmenting a Basic Node using CDI and NVMe

If we need more Storage servers to mitigate load issues, we can compose additional servers and automatically add them into the storage pool.
If there is node failure, dynamically swap it out

- We can leave a malfunctioning node behind, allocate another node, and utilize the memory, from the pool, that the other node was using, seamlessly.
Scaling the Control Structure to Very Large Systems

**CDI Control**

- Need to keep track of a huge number of concurrent resources
- Need to keep management and query communications down to a reasonable quantity
- Need to be able to execute timely changes to the HPC system as those changes are requested
Introducing Sunfish

**Application Domain**
- Clients
- Management Layer
- Hardware Layer

**Administration Domain**
- System composition
- Systems update
- Infra management

**CDI Composition Interface**

CDI Database and Tools

**Hardware Interface**

**Hardware Resources**
Introducing Sunfish

Application Domain

- Clients
- Management Layer
- Hardware Layer

Administration Domain

- Infra management
- Systems composition
- Systems update

CDI Composition Interface

Redfish/Swordfish

Redfish/Native Translation

App driven system reconfig

Translation
Introducing Sunfish

Application Domain

Clients

- Systems composition
- Systems update

Infra management

App driven system reconfig

Management Layer

CDI Composition Interface

- Resource Control Operations
- Resource Graph Representation
- Resource Events
- Evaluate Client Requirements
- Composition Policies
- Authorization Block

Resource Graph Representation

Resource Control Operations

Resource Events

Evaluate Client Requirements

Composition Policies

Authorization Block

Graph Reference Database

Client Batch Requestors

Redfish/Swordfish

Redfish/Native Translation

Hardware Layer
Introducing Sunfish

Application Domain
- Systems composition
- Systems update

Administration Domain
- Infra management

Management Layer
- CDI Composition Interface
  - Resource Control Operations
  - Resource Graph Representation
  - Resource Events
  - Evaluate Client Requirements
  - Composition Policies
  - Authorization Block
- DB Entry
- Graph Reference Database
- Redfish/Swordfish

Clients
- Client Batch Requestors

Hardware Layer
- Redfish/Native Translation
Introducing Sunfish

Batch Job

Interactive Job

Container Deployment

Workload Manager

Intelligent Resource Scheduler (IRS)

Resource ‘Broker’

Active

Pending

Free

Clients

Client Batch Requestors

Management Layer

Clients

Client Batch Requestors

Management Layer

Intelligent Resource Scheduler (IRS)
Introducing Sunfish

Proposed Atomic Operation Component

Start → IRS checks ‘Proposal Database’ for available resources → IRS proposes nodes and resources → A locking mechanism is employed to block out resource contenders → Verification is made that the allocation proposal is valid

Composability Manager requests Sunfish Core allocate resources → Sunfish moves the requested resources to a pending state → Sunfish orders hardware agents to execute desired connections → The hardware agents attempt to execute the connections

The hardware agents events record success or failure → Roll everything back, or partial? → SUCCESS

Sunfish is updated to reflect active resources and the Composability Manager is updated → FAILURE
Each batch request for Software Defined Nodes is a single imperative operation.
Do we have a partial success or failure?

Verify all the proposed changes are going to be successful.

Evaluate Client Requirements

Atomic Operation Component
Atomic Operation Component
Atomic Operation Component

Atomic Operation Component
Atomic Operation Component
Atomic Operation Component

Atomic Operation Component
Atomic Operation Component
Atomic Operation Component

Atomic Operation Component
Atomic Operation Component
Atomic Operation Component

Aggregated Verification of a Successful Operation

Introducing Sunfish
Introducing Sunfish

Application Domain

- Systems composition
- Systems update

Infra management

Client Batch Requestors

App driven system reconfig

Management Layer

Graph Reference Database

Message Queue

RESTful API (RF/SF)

Redfish/Swordfish

Resource Control Operations

Resource Graph Representation

Resource Events

Evaluate Client Requirements

Composition Policies

Authorization Block

CDI Composition Interface

Sunfish Services

- Resource Inventory
- RF Tree Management
- Resource Configuration
- Fabric Configuration
- Authentication
- Access Control
- Events and Logs

Redfish/Native Translation

Hardware Layer

REDfish Tree

Database

Resource Control Operations

Operations

Resource Graph

Representation

Resource Events

Evaluate Client Requirements

Composition Policies

Authorization Block

Graph Reference Database

Client Batch Requestors
Sunfish Core Services

Redfish/Swordfish

RESTful API (RF/SF)

Resource Inventory
RF Tree Management
Resource Configuration
Fabric Configuration
Authentication
Access Control
Events and Logs

Redfish Tree
The Sunfish Composability Management Framework

Interface to Sunfish
- Resources
- Paths
- Locality

Interface to Resource Graph Representation

Interface to ML Scheduler
(Input data)

Client Transaction Commands

Client Request

ML Plugin

Janusgraph Training Model Storage

Command Component Execution

Interface to Composer
4-Dimensional Software Defined Node Allocation

Virtual-Cluster Managers

Flux Workload Manager
- Flux Agent
- Flux Engine
- Flux Client

Kubernetes Container Orchestration
- Pre-Enqueue
- Pre-Bind
- Queueing Hint

Intelligent Resource Scheduler

Custom Resource Broker

CDI Composition Interface
- Resource Control Operations
- Resource Graph Representation
- Resource Events
- Evaluate Client Requirements
- Composition Policies
- Authorization Block

Graph Reference Database

DB Entry
ML Plugin: Intelligent Resource Scheduler

**Inputs:**
- Job queue
- Resources
  - Connections
  - Locations
  - Status

**Multi-agent**
- Heirarchical Neural Network
  - Level 1
  - Level 2
- Resource Policy Agent

**Outputs**
- Job assignments (ready and reserved)
- Backfilled jobs
- Resource assignments

**Schedule**
At time t:
- Job 6 is running on:
  - Resource A
  - Resource B
  - Resource C

Resource Pool
ML Plugin: Intelligent Resource Scheduler

Inputs:
- Job queue
- Resources
  - Connections
  - Locations
  - Status

Multi-agent
- Heirarchical Neural Network
  - Level 1
  - Level 2
- Resource Policy Agent

Outputs
- Job assignments (ready and reserved)
- Backfilled jobs
- Resource assignments

Schedule
At time t:
- Job 6 is running on:
  - Resource A
  - Resource B
  - Resource C

Rewards: Queue time, reserved time, runtime, energy usage, etc.
Why Reinforcement Learning?

Heuristics
• Can be fair, but often aren't the most efficient

Optimization Algorithms
• Must be highly tailored to specific machines

Reinforcement Learning
1. Customized rewards function
   1. Prioritize fairness
   2. Penalize undesirable scheduling
2. Machine agnostic
   1. Adapts to changing resources
   2. Adapts to different traffic volumes
   3. Learns a better algorithm over time
3. Potential cons
   1. Prone to job starvation
   2. May need lots of compute/time
Integrating BeeOND, Sunfish, and the Intelligent Resource Scheduler

Hardware execution is performed using Sunfish connected hardware Agents. Management of the HPC System is performed by the Sunfish core services.

A Workload Manager (example Slurm or Flux) allocates nodes and requests hardware Resources as a client to the Composability Manager.

Deep Reinforcement Agent for Scheduling
Integrating BeeOND, Sunfish, and the Intelligent Resource Scheduler

Flow is from left to right, and back
Integrating BeeOND, Sunfish, and our ML Algorithm

The Composability Manager requests the hardware Resources from Sunfish.
Integrating BeeOND, Sunfish, and our ML Algorithm

Sunfish orders the hardware Agents to aggregate fabric routes and endpoints to fulfill a request for NVMeoF.

Events are generated and propagated back through the Composability Manager to the Workload Manager.