SESSION 2 · HPSF 2026

Scan to Connect

DevOps for Scientific Software

Performance & Scale at the Edge

Pavan Madduri

Senior Platform Engineer @ W.W.Grainger | CNCF Kubestronaut

Productivity, Performance, and the HPC Pipeline

March 16–20, 2026 · Chicago, IL

The Handoff

Connecting Platform to Practitioner

**"We built the GitOps Platform.
Now, how do we run MPI and GPUs on it?"**

✅ Session 1 — GitOps Platform

- ArgoCD, Crossplane, KEDA
- Infrastructure as Code
- Fleet management at scale

⏭ Session 2 — This Talk

- HPC containers & MPI
- GPU scheduling & FinOps
- Hardware-level observability

Shifting focus from Infrastructure-as-Code to high-concurrency scientific workloads

PILLAR 1

HPC-Optimized Containers

The Container Overhead Myth

CONCEPT

Goal: < 1% performance overhead for compute-heavy workloads

🔌 Host Networking

Bypass virtual networking overhead
Direct NIC access for MPI traffic

⚡ SR-IOV

Hardware-level network virtualization
Near bare-metal InfiniBand performance

📌 CPU Pinning

Dedicated cores, no context switching
NUMA-aware memory allocation

Building the HPC Container

CODE

CUDA base images with exact version pinning

```
FROM nvidia/cuda:12.2.0-devel-ubuntu22.04
WORKDIR /climate-model

# Install dependencies
RUN apt-get update && apt-get install -y \
    python3 python3-pip \
    libnetcdf-dev libhdf5-dev \
    gfortran \
    && rm -rf /var/lib/apt/lists/*

# Install Python scientific stack
RUN pip3 install --no-cache-dir \
    numpy==1.24.3 \
    xarray==2023.5.0 \
    netCDF4==1.6.4

# Copy custom Fortran modules
COPY fortran_modules/ ./fortran_modules/
RUN cd fortran_modules && make install

# Copy model code
COPY model/ ./model/
ENTRYPOINT ["python3", "model/run_simulation.py"]
```

PILLAR 2

Advanced Orchestration

The MPI Scheduling Problem

CONCEPT

The default Kubernetes Scheduler was never designed for tightly-coupled MPI workloads

❌ Default K8s Scheduling

× Pod-by-pod scheduling decisions

× Resource deadlocks: Job A holds 3 of 4 GPUs, Job B holds 1 — both stuck

× Idle GPU waste while pods wait for siblings to be scheduled

× No concept of a "job" — only pods

✅ What MPI Needs

✓ All N pods scheduled together

✓ Synchronized start at the exact same millisecond

✓ Dedicated GPU allocation per rank with NCCL topology awareness

✓ Atomic job scheduling — all or nothing

PILLAR 2

Advanced Orchestration

The Solution: Gang Scheduling

IMPLEMENTATION

Volcano / Kueue: All-or-nothing scheduling

```yaml
apiVersion: batch.volcano.sh/v1alpha1
kind: Job
metadata:
    name: mpi-climate-sim
spec:
    minAvailable: 8     # Gang: all 8 or none
    schedulerName: volcano
    plugins:
        ssh: []
        svc: []
    tasks:
        - replicas: 8
          name: mpi-worker
          template:
              spec:
                  containers:
                      - name: climate-worker
                        image: registry/climate:v2.1
                        resources:
                            limits:
                                nvidia.com/gpu: "1"
                        command:
                            - mpirun
                            - --np 8
                            - ./run_simulation
```

🎯 Key Guarantee

minAvailable: 8 ensures all pods start simultaneously — no partial scheduling, no deadlocks, no wasted GPUs

✅ Result

Synchronized start across all pods at the exact same millisecond

PILLAR 3

Predictive FinOps

The Cost of Bursty Workloads

CONCEPT

Reactive scaling provisions too late — jobs queue while waiting for compute

❌ Reactive Scaling

- × Wait for pod pending ⟶ then provision
- × 5-10 min delay for GPU node startup
- × Jobs queued during peak submission
- × Over-provision 'just in case' = waste

✅ Proactive Scaling

- ✓ Time-series based prediction
- ✓ Pre-warm nodes before peak hours
- ✓ Spot + On-Demand mix for cost
- ✓ Scale-to-zero during quiet periods

Moving to proactive, time-series based scaling

PILLAR 3

Predictive FinOps

Predictive FinOps with Karpenter

CODE

Targeting p3.2xlarge GPU instances with Spot/On-Demand mix

```yaml
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
    name: scientific-gpu
spec:
    template:
        spec:
            nodeClassRef:
                group: karpenter.k8s.aws
                kind: EC2NodeClass
                name: gpu-optimized
            requirements:
                - key: karpenter.sh/capacity-type
                  operator: In
                  values: ["spot", "on-demand"]
                - key: node.kubernetes.io/instance-type
                  operator: In
                  values: ["p3.2xlarge", "p3.8xlarge",
"p3.16xlarge"]
    limits:
        cpu: "1000"
        memory: 1000Gi
    disruption:
        consolidationPolicy: WhenUnderutilized
        expireAfter: 720h
```

💰 **34% Cost Reduction**

How It Works

→ Spot for fault-tolerant batch (checkpointed simulation)

→ On-Demand for latency-sensitive training

→ Auto-consolidation reclaims underutilized nodes

→ Scale to zero overnight

Low Sensitivity

Deep Hardware Observability

CONCEPT + CODE

Are you actually using the GPUs you requested?

```
# prometheus.yml — GPU Metrics
scrape_configs:
  - job_name: 'gpu-metrics'
    static_configs:
      - targets: ['dcgm-exporter:9400']
    relabel_configs:
      - source_labels: [__address__]
        target_label: instance

  # Key DCGM metrics exposed:
  # DCGM_FI_DEV_GPU_UTIL      → GPU core %
  # DCGM_FI_DEV_MEM_COPY_UTIL → Memory bandwidth %
  # DCGM_FI_DEV_GPU_TEMP      → Temperature °C
  # DCGM_FI_PROF_PIPE_FP32    → FP32 utilization
```

What You Discover

- → Per-GPU utilization %
- → Memory bandwidth usage
- → Temperature & throttling
- → FP32 / FP16 pipe activity

PILLAR 4

Deep Observability

The $50k Discovery

IMPACT

Per-GPU utilization, memory bandwidth, and temperature tracking

🔍 The Discovery

30% of ML training jobs used only 1 of 8 requested GPUs

Researchers requested 8× GPUs 'just in case' — but code only utilized a single device

$50,000	30%	8→1
/month saved	jobs over-provisioned	GPU right-sizing

Optimized batch sizes & resource requests → actionable savings

The Complete Loop

Infrastructure as Code + DevOps = High-Performance Science

< 1%

Overhead

Container performance vs bare metal

100%

Reproducibility

Pinned versions, immutable images

34%

Cost Reduction

Spot + consolidation with Karpenter

$50k

Monthly Savings

Hardware observability reveals waste

Pillar 1 HPC Containers → **Pillar 2** Gang Scheduling → **Pillar 3** FinOps → **Pillar 4** Observability

Each pillar reinforces the next — containers ⟶ scheduling ⟶ cost ⟶ visibility

Thank You!

Questions & Discussion

Pavan Madduri

Senior Platform Engineer @ W.W.Grainger | CNCF Kubestronaut

🔗 LinkedIn: linkedin.com/in/pavanmadduri

📝 Blog: pavanmadduri.wordpress.com

💻 GitHub: github.com/pmady

Resources & Further Reading

→ github.com/volcano-sh/volcano — Gang scheduling

→ karpenter.sh — Just-in-time node provisioning

→ github.com/NVIDIA/dcgm-exporter — GPU metrics

#HPSFcon #DevOps #ScientificComputing #CloudNative

Scan for Slides & Research Papers

CNCF Kubestronaut — All 5 K8s Certs

KCNA | CKA | CKAD | CKS | KCSA