Benchmark Specification: Multi-Objective Comparison of HPC Schedulers

Problem Definition

This project aims to benchmark and compare multiple HPC schedulers across different hardware and software environments. The ultimate goal is to determine which scheduler configurations are best suited for various workload categories found in high-performance computing (HPC).

Formally, given a benchmark $F : (A, B, C) \rightarrow (\alpha, \beta, \gamma, \delta)$ composed of a virtual cluster A that's running a workload C on a given a Scheduler B and output a set of metrics $O = \{\alpha, \beta, \gamma, \delta\}$. We'll explore and analyze the related pareto space. We seek to identify a set of solutions (the Pareto frontier) where improving one metric necessarily degrades at least one other. In the context of cluster scheduling, we balance metrics, recognizing that different workloads may prioritize different objectives.

Input Space Definition

The following configurations will be used for the benchmark:

1. Hardware Topologies A
    - Defines the virtual machine configurations and cluster architectures
    - Includes node counts, CPU/GPU resources, memory capacities, and network characteristics
    - Represents the underlying infrastructure on which schedulers will be evaluated
2. Scheduler Systems B
    - Defines the resource management and job scheduling technologies under evaluation
    - Consists of three primary systems: SLURM, Kubernetes+Volcano, and Flux Framework
    - Represents different approaches to workload coordination and resource allocation in HPC environments
3. Workload Types C
    - Characterizes the computational jobs and applications to be scheduled
    - Encompasses standard HPC benchmarks that stress different system components
    - Represents typical scientific and technical computing scenarios found in production environments

Hardware Topologies A

- Defines the virtual machine configurations and cluster architectures

- Includes node counts, CPU/GPU resources, memory capacities, and network characteristics
- Represents the underlying infrastructure on which schedulers will be evaluated
- Configure each environment to match specific hardware profiles (1,2,4,8 cpus, 1,2,4,8 gpus, 4,8,16,32 Gb Ram, etc…).

Each topology configuration specifies:

- Number and type of head nodes
- Number and type of compute nodes
- Network configuration
- Storage configuration
- Resource allocations (CPU, memory, disk)

Scheduler Systems B

The following scheduler and resource management systems will be evaluated:

1. SLURM : b_1
2. Kubernetes + Volcano : b_2
3. Flux Framework : b_3

Workload Types C

The benchmark F will be composed of the elements including, but not limited to :

- HPL (High Performance Linpack): Stresses CPU floating-point capabilities
- HPCG (High Performance Conjugate Gradient): Exercises memory hierarchy and interconnect

Output Space Definition

The following metrics will be used for evaluation:

1. Resource Utilization (α)

- Measures the extent to which available resources (e.g., CPU, memory, GPU, network bandwidth) are employed.
- Higher utilization indicates more efficient use of resources, while underutilization suggests waste.

2. Fairness (β)

- Assesses the equitable distribution of resources among users or workloads.

- Finish-time fairness, for instance, compares completion time in a shared environment versus alone, ensuring no single workload monopolizes resources.

3. Job Completion Time (γ)

- Encompasses the entire duration required for a job to finish—queueing, execution, and teardown.
- Particularly important in multi-node HPC environments, where parallel processing can drastically reduce total completion time.

4. Elasticity (δ)

- Evaluates how well the scheduler can adapt to fluctuating workloads by scaling resources up or down.
- Efficient elasticity ensures workload demands are met while preventing resource bottlenecks or waste.

Research Question

> How do SLURM, Kubernetes+Volcano, and Flux Framework schedulers compare in terms of resource utilization (α), fairness (β), job completion time (γ), and elasticity (δ) across varying HPC workloads and hardware configurations in the sense of Pareto?

Hardware Resource

The benchmark F will be executed on `rhodey`, a server in Lawrence Berkeley National Laboratory that has the same topology of a CPU Perlmutter node. The virtual clusters A will be configured for the schedulers B and run the benchmarks C.

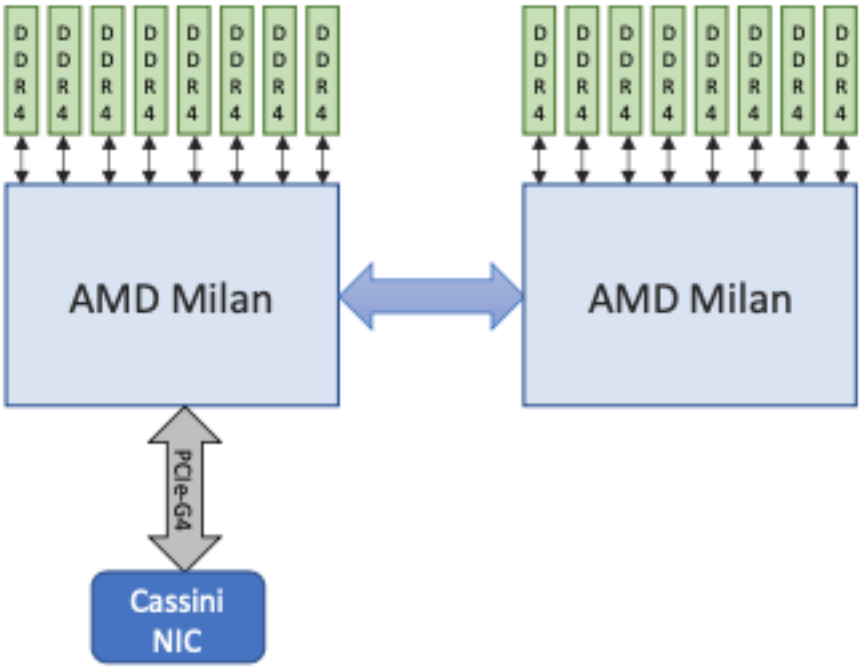

Figure 1: Perlmutter CPU nodes

- 2x AMD EPYC 7763 (Milan) CPUs
- 64 cores per CPU
- AVX2 instruction set
- 512 GB of DDR4 memory total
- 204.8 GB/s memory bandwidth per CPU
- 1x HPE Slingshot 11 NIC
- PCIe 4.0 NIC-CPU connection
- 39.2 GFlops per core
- 2.51 TFlops per socket
- 4 NUMA domains per socket (NPS=4)

Methodology

1. **Environment Deployment**

- Provision the virtual test cluster according to match specific hardware profiles A
- Install and configure the chosen schedulers B

2. **Execution & Metrics Collection**

- Run the suite of workloads C on each scheduler B and each hardware configuration A.
- Collect data for resource utilization (α), fairness (β), job completion time (γ), elasticity (δ)

3. **Pareto Analysis**

- For each tuple (A, B, C) plot performance across multiple metrics in a Pareto frontier representation.
- Identify trade-offs (e.g., improved throughput vs. potentially reduced fairness).
- Compare scheduling strategies and highlight performance bottlenecks or advantages.
- Provide guidelines on choosing the best scheduler for specific workload types and hardware configurations.

Architecture

In order to set up the benchmark environment, we need to provision A, configure B, and run C automatically. To define the complex configuration (A, B, C) we will use Hydra. To provision VMs with the topology A we will use OpenTofu. To install and configure the schedulers B we will use Ansible.

The infrastructure provisioning component follows a layered architecture:

- Configuration Layer: Hydra-based configuration system for defining infrastructure parameters
- Orchestration Layer: Python modules that interpret configurations and manage the provisioning workflow
- Provisioning Layer: Terraform modules that handle the actual resource creation
- Resource Layer: The virtualized infrastructure created on the target platform

Requirements

Functional requirements

- Automated Provisioning: The system must be able to automatically provision virtual clusters with different topologies without manual intervention.
- Topology Flexibility: Support for multiple predefined cluster topologies (small, medium, large) with varying numbers of head and compute nodes.
- Resource Configuration: Ability to specify CPU, memory, disk, and network configurations for each node type.
- Network Configuration: Creation of isolated networks for each cluster with appropriate routing and DNS.

- Idempotency: Ensure provisioning operations are idempotent to support retries and incremental changes.
- Cleanup: Provide mechanisms to completely tear down provisioned resources after experiments.
- Speed: Intermediate images should be used to speed up the provisioning process, allowing for quick reconfiguration and testing of different setups.

Non-functional requirements Reliability: The provisioning process should be robust against transient failures. Observability: Provide detailed logging and status information during the provisioning process.

Configuration System

The infrastructure configuration uses Hydra to manage complex, hierarchical configurations. In order to be configurable, the configuration should define composable dataclasses.

One for the whole topology A, One for a compute node, One for a resource (RAM, CPU, Accelerator, Storage), one for a Network

For the schedulers B, the configuration should have a Scheduler, Scheduler configuration

Orchestration Layer

The orchestration Layer should be able to perform a provisioning workflow and gather relevant configuration parameter between the provisioning of VMs and the configuration of schedulers.

Here's the provisioning workflow:

1. Initialization: Parse and validate the selected topology configuration
2. Resource Planning: Determine the resources needed and verify availability
3. Head Node Deployment: Provision and configure head node(s)
4. Compute Node Deployment: Provision and configure compute nodes
5. Verification: Validate that all resources are correctly provisioned
6. Output Generation: Generate structured output for the configuration phase