Doing More with Slurm
Advanced Capabilities

Shawn Hoopes, Director – Training Services
shawn@schedmd.com
What is Slurm…

- Policy-driven, open source, fault-tolerant, and highly scalable workload management and job scheduling system

- Some Key Functions
  - Allocates exclusive and/or non-exclusive access to resources to users for some duration of time for a workload
  - Provides a framework for starting, executing, and monitoring work on the set of allocated nodes
  - Arbitrates contention for resources by managing a queue of pending work
  - Enforces customized workload policies to grant and/or restrict access to compute resources
What is Slurm...

- **Features Details**
  - Straight-forward batch and serial job submission methods
  - Easy to administer
  - Plug-in infrastructure
  - Very highly scalable
  - Secure and fault-tolerant
  - Flexible priority and fairshare policies
  - Powerful database integration for job detail tracking, reporting, and policy enforcement, as well as job script storage and QOS definitions
  - Policy-driven preemption methods
What is Slurm...

- Some Advanced Features
  - NSS Slurm, pam_slurm_adopt, scrontab
  - Configless Slurm
  - Job dependencies
  - Heterogenous job submission
  - MPI Support via srun
  - Cgroup v1 and v2 support
  - Detailed cpu-binding options
  - Job Profiling
  - Node Sets and Dynamic Node provisioning
<table>
<thead>
<tr>
<th>Rank</th>
<th>System</th>
<th>Cores</th>
<th>Rpeak</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Frontier - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE</td>
<td>8,730,112</td>
<td>1,685.65 PFlop/s</td>
</tr>
<tr>
<td>2</td>
<td>Supercomputer Fugaku - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu</td>
<td>7,630,848</td>
<td>537.21 PFlop/s</td>
</tr>
<tr>
<td>3</td>
<td>LUMI - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE</td>
<td>1,110,144</td>
<td>214.35 PFlop/s</td>
</tr>
<tr>
<td>4</td>
<td>Summit - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM</td>
<td>2,414,592</td>
<td>200.79 PFlop/s</td>
</tr>
<tr>
<td>5</td>
<td>Sierra - IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM / NVIDIA / Mellanox</td>
<td>1,572,480</td>
<td>125.71 PFlop/s</td>
</tr>
<tr>
<td>6</td>
<td>Sunway TaihuLight - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway, NRCPC</td>
<td>10,649,600</td>
<td>125.44 PFlop/s</td>
</tr>
<tr>
<td>7</td>
<td>Perlmutter - HPE Cray EX235n, AMD EPYC 7763 64C 2.45GHz, NVIDIA A100 SXM4 40 GB, Slingshot-10, HPE</td>
<td>761,856</td>
<td>93.75 PFlop/s</td>
</tr>
<tr>
<td>8</td>
<td>Selene - NVIDIA DGX A100, AMD EPYC 7742 64C 2.25GHz, NVIDIA A100, Mellanox HDR Infiniband, Nvidia</td>
<td>555,520</td>
<td>79.22 PFlop/s</td>
</tr>
<tr>
<td>9</td>
<td>Tianhe-2A - TH-IVB-FEP Cluster, Intel Xeon E5-2692v2 12C 2.2GHz, TH Express-2, Matrix-2000, NUDT</td>
<td>4,981,760</td>
<td>100.68 PFlop/s</td>
</tr>
<tr>
<td>10</td>
<td>Adastra - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE</td>
<td>319,072</td>
<td>61.61 PFlop/s</td>
</tr>
</tbody>
</table>
But what is SchedMD?

- Maintainers and Supporters of Slurm
  - Only organization providing level-3 support
  - Training
  - Consultation
  - Custom Development
Industry Trends

- GPUs - AI Workloads
- Hybrid Cloud
- AI Tooling Integration

- Manufacturing & EDA
- Healthcare & Lifesciences
- Financial Services & Insurance
- Energy
- Government
- Academic
GPU Scheduling for AI Workloads
Fine-Grained GPU Control

All options apply to salloc, sbatch and srun commands

- **--cpus-per-gpu** = CPUs required per allocated GPU
- **-G/--gpus** = GPU count across entire job allocation
- **--gpu-bind** = Task/GPU binding option
- **--gpu-freq** = Specify GPU and memory frequency
- **--gpus-per-node** = Works like “--gres=gpu:” option today
- **--gpus-per-socket** = GPUs per allocated socket
- **--gpus-per-task** = GPUs per spawned task
- **--mem-per-gpu** = Memory per allocated GPU
Examples of Use

$ sbatch --ntasks=16 --gpus-per-task=2 my.bash

$ sbatch --ntasks=8 --ntasks-per-socket=2 --gpus-per-socket=k80:1 my.bash

$ sbatch --gpus=16 --gpu-bind=closest --nodes=2 my.bash

$ sbatch --gpus=k80:8,a100:2 --nodes=1 my.bash
Configuring GPUs

- GPUs fall under the Generic Resource (GRES) plugin
  - Node-specific resources

- Requires definition in slurm.conf and gres.conf on node

- GRES can be associated with specific device files (e.g. specific GPUs)

- GPUs can be autodetected with NVML or RSMI libraries

- Sets CUDA_VISIBLE_DEVICES environment variable for the job
Restricting Devices with Cgroups

- Uses the `devices` subsystem
  - `devices.allow` and `devices.deny` control access to devices
  - All devices in `gres.conf` that the job does not request are added to `devices.deny` so the job can’t use them
- Must be a Unix device file. Cgroups restrict devices based on major/minor number, not file path (/dev/nvidia0)
- GPUs are the most common use case, but any Unix device file can be restricted with cgroups
NVIDIA MIG Support

- Configured like regular GPUs in Slurm
- Fully supported by task/cgroup and --gpu-bind
- AutoDetect support
- Make it work with CUDA_VISIBLE_DEVICES
- MIGs must be manually partitioned outside of Slurm beforehand via nvidia-smi
Hybrid Cloud Autoscaling
Hybrid Cloud
Cloud Enablement

- Power Saving module
  - Requires 3 parameters to enable
    - ResumeProgram
    - SuspendProgram
    - SuspendTime (Either global or Partition)
  - Other important parameters
    - ResumeTimeout
    - SuspendTimeout
Power State Transition - Resume

Node State
- IDLE
- POWERED_DOWN ~

Job State
- Configuring
- Running
- Completing

- ALLOCATED / MIXED
- POWERING_UP #
- ALLOCATED / MIXED
Power State Transition - Suspend
What about the Data?

- Most common question - How do we get my data from onprem to cloud?
- Previous best option - mini-workflow w/ job dependency

Stage-in job > Application job > Stage-out job

- Benefit: easy to increase the number of nodes involved in moving the data
New Option: Lua Burst Buffer plugin

- Originally developed for Cray Datawarp
  - Intermediate storage - in between slow long-term storage and the fast memory on compute nodes
- Asynchronously calls an external script to not interfere with the scheduler
- Generalized this function so you don’t need Cray Datawarp or actual hardware “burst buffers” or Cray’s API
- Good for Data movement or provisioning cloud nodes
  - Anything you think you want to do while the job is pending (or at other job states)
Asynchronous “stages”

- **Stage in** - called before the job is scheduled, job state == pending
  - Best time for Cloud data staging
- **Pre run** - called after the job is scheduled, job state == running + configuring
  - Job not actually running yet
- **Stage out** - called after the job completes, job state == stage out
  - Job cannot be purged until this is done
- **Teardown** - called after stage out, job state == complete
AI Tooling Integration: Enter the REST API
New Integration Requirements
What is Slurm REST API

Client

JSON/YAML

GET
POST
PUT
DELETE

Server

HTTP

Client sends a request. (NOT srun, sbatch, salloc)

Server sends a response.

HTTP Method
slurmrestd

A tool that runs inside of the Slurm perimeter that will translate JSON/YAML requests into Slurm RPC requests
Slurm REST API Architecture (rest_auth/jwt)

AuthAltTypes Perimeter - JWT authentication

Munge Perimeter

- slurmctld
- slurmdbd
- slurmd

cluster network

slurmrestd

client
Slurm REST API Architecture (rest_auth/jwt + Proxy)

AuthAltTypes Perimeter
Munge Perimeter

- slurmctld
- slurmdbd
- slurmd

slurmrestd

Authenticated HTTP proxy

Site Authentication Perimeter

Authenticated client

Site Authentication Server

TLS wrapped
JSON/YAML output

- Slurmrestd uses content (a.k.a. openapi) plugins. These plugins have been made global to allow other parts of Slurm to be able to dump JSON/YAML output.
- New output formatting (limited to these binaries only):
  - `sacct --json` or `sacct --yaml`
  - `sinfo --json` or `squeue --yaml`
  - `squeue --json` or `squeue --yaml`
- Output is always same format of latest version of slurmrestd output.
  - Formatting arguments are ignored for JSON or YAML output as it is expected that clients can easily pick and choose what they want.
```json
$ sinfo --json
{
  "meta": {
    "plugin": {
      "type": "openapi/v0.0.37",
      "name": "Slurm OpenAPI v0.0.37"
    },
    "Slurm": {
      "version": {
        "major": 22,
        "minor": 5
      },
      "release": "21.08.6"
    },
    "errors": [],
    "nodes": [
      {
        "architecture": "x86_64",
        "burstbuffer_network_address": "",
        "boards": 1,
        "boot_time": 1646380817,
        "comment": "",
        "cores": 6,
        "cpu_binding": 0,
        "cpu_load": 64,
        "extra": "",
        "free_memory": 3208,
        "cpus": 12,
        "last_busy": 1646430364,
        "features": "",
        "active_features": "",
        "gres": "",
        "gres_drained": "N/A",
        "gres_used": "scratch:0",
        "mcs_label": "",
        "name": "node00",
        "next_state_after_reboot": "invalid",
        "address": "node00",
        "hostname": "node00",
        "state": "idle",
        "state_flags": [],
        "next_state_after_reboot_flags": [],
        "operating_system": "Linux 5.4.0-100-generic #113-Ubuntu SMP Thu Feb 3 18:43:29 UTC 2022",
        "owner": null,
        "partitions": [
          "debug"
        ],
        "port": 6818,
        "real_memory": 31856,
        "reason": "",
        "reason_changed_at": 0,
        "reason_set_by_user": null,
        "slurmd_start_time": 1646430151,
        "sockets": 1,
        "threads": 2,
        "temporary_disk": 0,
        "weight": 1,
        "tres": "cpu=12,mem=31856M,billing=12",
        "slurmd_version": "22.05.0-0pre1",
        "alloc_memory": 0,
        "alloc_cpus": 0,
        "idle_cpus": 12,
        "tres_used": null,
        "tres_weighted": 0.0
      }
    ]
  }
}```
How To Get There with Slurm
Large Energy Company

- Using their scheduler for many years
  - Can’t just flip a switch and go to production
- Massive scale - multiple international sites, nodes and workloads
- Many integrations required

3-4 Months to Production
Three Migration Steps

● Admin/User education
  ○ Training - Help admins identify the commonalities and learn the Slurm way
  ○ Wrappers - a bridge to migration not a crutch
    ■ LSF, Grid Engine - command and submission
    ■ PBS - command, submission, environment variables, #PBS scripts

● Policy replication
  ○ Reevaluate policies
    ■ Are we continuing to produce technical debt due to “doing things how we’ve always done them?”
  ○ Optimizing for scale and throughput - 1 million jobs/day
    ■ Some Financial sites doing up to 15 million/day

● Tooling integration
  ○ Most time consuming of the journey
Questions?

Thank You

schedmd.com

http://slurm.schedmd.com

shawn@schedmd.com