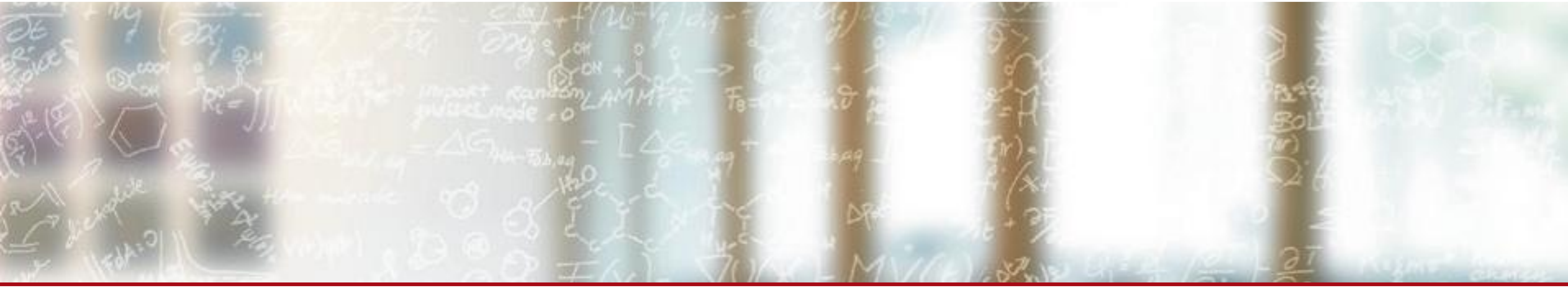


Sharing is Caring: Tackling Node Sharing Challenges at CUG sites

CUG 2025 BoF
May 5th, 2025

SCAN ME

If you haven’t yet answered the ***survey*** *on* ***node sharing****, there’s still time…*
We can update the results live.

Why have this BoF?

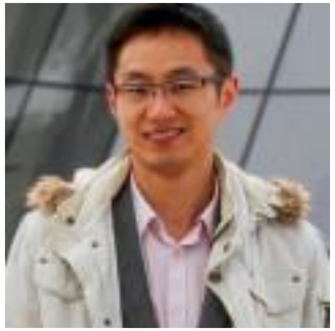
- Nodes are getting crazy powerful and crazy expensive
  - Estimated “real” cost of GH node hour is about $3-4
- “HPC” workflows are changing
  - Need to support interactivity, notebooks, bursty workloads, high-throughput computing, CI/CD pipelines, mixed workloads…
- There are various challenges to node sharing
  - Technical challenges
    - CPU- and memory contention, performance variability
    - Security, isolation, node cleanup
    - Scheduling complexity
  - Business challenges
    - Fine-grained billing and accounting
  - User experience
    - Concerns about performance degradation or security/isolation

SCAN ME

Our panelists

Cristian di Pietrantonio

Supercomputing Applications Specialist at Pawsey



Pengfei Ding

Data Science Workflow Architect at NERSC

Tim Wickberg

CTO of SchedMD

Tim Robinson

HPC Platform Service Manager at CSCS

SCAN ME

NERSC, Pawsey, CSCS and SchedMD perspectives

Node Sharing at NERSC

NERSC

CUG 2025 - **Sharing is Caring: Tackling Node-Sharing Challenges at CUG Sites**

Pengfei Ding
May 5th 2025

Node sharing at NERSC

- How do we do node sharing at NERSC at the moment?
- The mechanism of node sharing is via **Shared QoS** in Slurm:
- We **do not expose partitions to users**;
  - Partitions work behind the scenes of QoSes;
  - Users use **QoS and Constraints** to select the queue and CPU/GPU nodes.
- CPU node sharing:
  - Each CPU node each has 2 AMD EPYC 7763 (64-core, 128-thread) CPUs, 512 GB memory;
  - CPU nodes can **be sliced up by any number of cores, or portions of memory;** number of cores and memory will be matched up, and the charge factor is the higher ratio of these two relative to the whole node;
- GPU node sharing:
  - Each GPU node each has 4 Nvidia A100 GPUs, and one AMD EPYC 7763 CPU, 256 GB memory;
  - GPU node can only **be partitioned by number of GPUs requested (1, 2, 3),** and the number of cores and memory will be allocated proportional to the number of GPUs.

“Sharing” in computational systems (I)

- "Sharing” has been sitting at the center stage of computational systems from > 60 years ago…
- The concept of Operating System was partially (mainly?) driven by the need of **timing sharing**:
    - Dartmouth Time-Sharing System (1963);
    - Apollo Guided System (cooperative multitasking, non-preemptive multitasking, processes yield themselves) (1966);
    - Unix, Linux (preemptive multitasking).
- Sharing on the Operating System side has been mainly focused on sharing CPU time between processes and was done primarily by the process scheduler; later when multi-core CPUs become a norm, came the resource management (cgroups and numactl etc).
- Workload manager (batch scheduler) **primarily focused on sharing nodes, or portions of a node** (in the unit of at least one CPU core) within a cluster(s) (i.e. no processes-level context-switching).

“Sharing” in computational systems (II)

- The horizon of sharing in HPCs is changing, one driver is more powerful GPUs:
  - Some applications **cannot use a whole GPU 100%** throughout the duration of a job;
  - Partition single GPUs, or put multiple processes on single GPUs becomes necessary;
  - Toolchains enabling this are: Nvidia Multi-instance GPU (MIG) and Multi-Process Service; AMD GPUs have somewhat equivalent features.

  - **Flashing back memories of OS support for multi-core CPUs and multi-threaded processes.**

“Sharing” in computational systems (III)

- In resemblance, it is interesting to look at the evolution of the Linux scheduler:
  - It is **hard** to implement;
  - The scheduler **evolved** a lot, largely driven by **hardware capabilities**;
  - Linux schedulers:
    - *Completely Fair Scheduler* (non-deterministic, issues for real-time processes);
    - Preemptive scheduler;
    - Real-time scheduler (in mainline kernel after >20 years development, involved changes you wouldn’t think about in the first place).
- Things to look ahead – resource management for GPUs and NICs
  - Integration with 1) **Operating Systems**; and 2) **Workload Managers**.

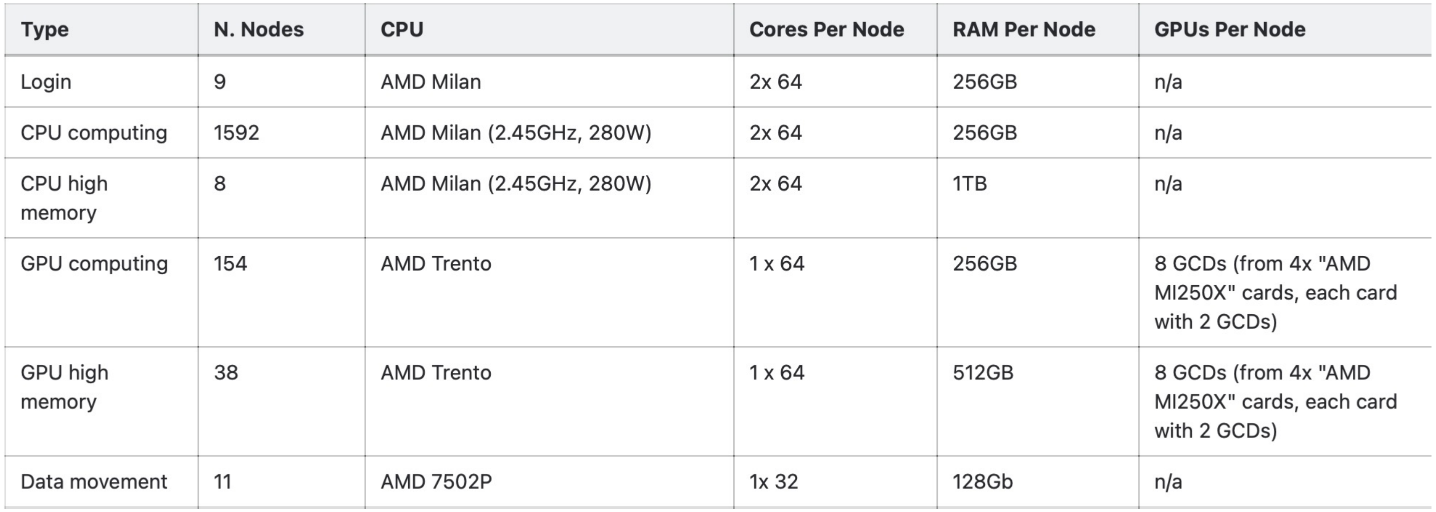
Pawsey's node sharing approach

Sharing is Caring: Tackling Node-Sharing Challenges at CUG Sites

Setonix configuration overview

Why node sharing?

All nodes of Setonix are in node sharing mode. We want to maximise the use of resources, especially GPU nodes. We do not have many nodes, hence suboptimal usage is unacceptable. Furthermore, job packing can be cumbersome and difficult to implement on non-trivial cases. Priority is still given to bigger jobs.



Type	N. Nodes	CPU	Cores Per Node	RAM Per Node	GPUs Per Node
Login	9	AMD Milan	2x 64	256GB	n/a
CPU computing	1592	AMD Milan (2.45GHz, 280W)	2x 64	256GB	n/a
CPU high memory	8	AMD Milan (2.45GHz, 280W)	2x 64	1TB	n/a
GPU computing	154	AMD Trento	1 x 64	256GB	8 GCDs (from 4x "AMD MI250X" cards, each card with 2 GCDs)
GPU high memory	38	AMD Trento	1 x 64	512GB	8 GCDs (from 4x "AMD MI250X" cards, each card with 2 GCDs)
Data movement	11	AMD 7502P	1x 32	128Gb	n/a

Table 1. Setonix node partitions.

Fractional node allocation

On Setonix we allow fractional node allocations.

Fractional node allocation = Max(
Requested Core Fraction,
Requested Memory Fraction,
Requested GPU Fraction)

- Through various mechanisms, an equivalence is established across consumable resources.
- Allows fair sharing of node resources.
- Useful to define a meaningful SU billing mechanism (https://arxiv.org/pdf/2110.09987).

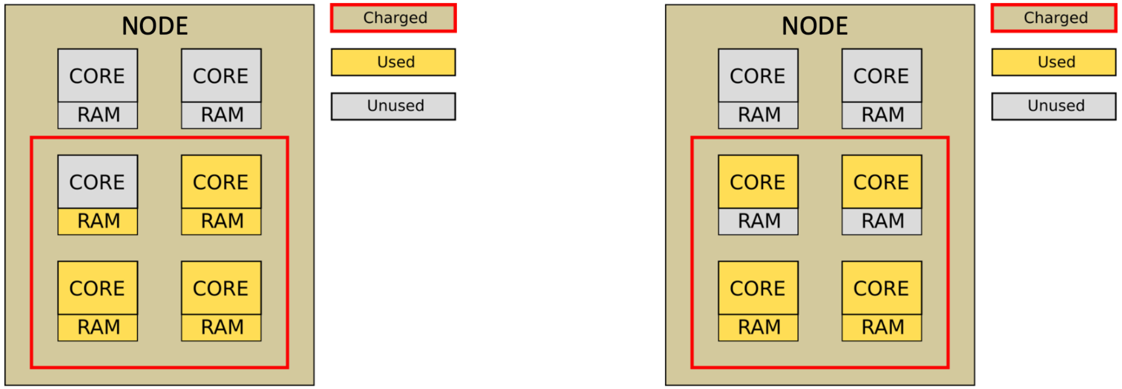


Figure 1. Fractional node allocation on Setonix. On the left, job requests 2/3 of memory and ½ of CPU cores. It will be charged for 2/3 of the node. On the right, the job requests 2/3 of CPU cores and 1/3 of memory. Will still be charged for 2/3 of the node.

Node sharing with SLURM, SLURM plugins, and CLI Filter

SLURM support

- Uses cgroups to make sure jobs are constrained to requested resources (CPU cores, memory, GPUs).

CLI Filter

It is a LUA script, installed on every node, executed before every job to constrain resource requests by users.

- On *GPU nodes*,
  - forbids users to request anything other than GPUs on GPU nodes.
  - CPU cores are determined using information in gres.conf (see next slide)
  - memory is determined by SLURM using `DefMemPerGPU`.

- On *CPU nodes*,
  - we mainly rely on `DefMemPerCPU and MaxMemPerCPU.`
  - `--mem-per-cpu` must be properly defined irrespective of hyperthreading being enabled or not.
  - `--mem-per-cpu` is necessary for Slurm to allocate corresponding number of CPUs.

Tmpfs SLURM plugin

The tmpfs plug in, now bundled with Slurm, is used to clean up /tmp and allocate space on node-local NVMe disks, moiunted on /nvme. Both /tmp and /nvme are private and automatically cleaned up, through overlays.

NVMe disk usage is not included in billing calculations.

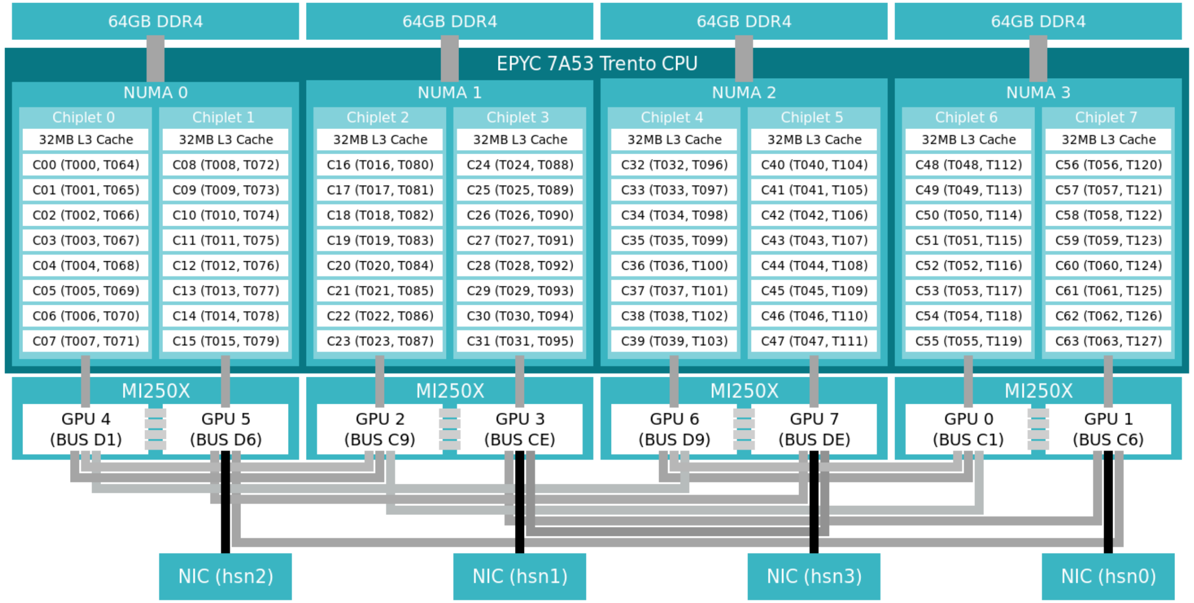
SLURM CLI Filter sample

```
if not is_gpu_partition(partition) and not is_excepted_partition(partition) then
        -- Non-gpu partition path: compute correct mem-per-cpu value from available memory and threads-per-core option
        -- if memory has not been reqested explicitly


        if not has_explicit_mem_request then
            local pinfo = get_partition_info(partition)
            if pinfo == nil then return slurm_error("unable to retrieve partition information") end

            local mem_per_hw_thread = math.floor(tonumber(pinfo.DefMemPerCPU))

            if is_node_exclusive or has_all_mem_request then
                local hw_threads_per_node = math.floor(tonumber(pinfo.TotalCPUs)/tonumber(pinfo.TotalNodes))
                options['mem'] = math.floor(mem_per_hw_thread * hw_threads_per_node)
            else
                local mem_scale = 1
                if tonumber(options['threads-per-core']) == 1 then mem_scale = 2 end


                options['mem-per-cpu'] = mem_per_hw_thread * mem_scale
            end
        end
```

https://github.com/PawseySC/pawsey-slurm-plugin-configuration

GPU node configuration

In the gres.conf file we enforce GPU-CPU affinity by associating a MI250X GCD with the closest NUMA region and subset of 8 cores.



EPYC 7A53 Trento CPU

64GB DDR4		64GB DDR4		64GB DDR4		64GB DDR4	
NUMA 0		NUMA 1		NUMA 2		NUMA 3	
Chiplet 0	Chiplet 1	Chiplet 2	Chiplet 3	Chiplet 4	Chiplet 5	Chiplet 6	Chiplet 7
32MB L3 Cache	32MB L3 Cache	32MB L3 Cache	32MB L3 Cache	32MB L3 Cache	32MB L3 Cache	32MB L3 Cache	32MB L3 Cache
C00 (T000, T064)	C08 (T008, T072)	C16 (T016, T080)	C24 (T024, T088)	C32 (T032, T096)	C40 (T040, T104)	C48 (T048, T112)	C56 (T056, T120)
C01 (T001, T065)	C09 (T009, T073)	C17 (T017, T081)	C25 (T025, T089)	C33 (T033, T097)	C41 (T041, T105)	C49 (T049, T113)	C57 (T057, T121)
C02 (T002, T066)	C10 (T010, T074)	C18 (T018, T082)	C26 (T026, T090)	C34 (T034, T098)	C42 (T042, T106)	C50 (T050, T114)	C58 (T058, T122)
C03 (T003, T067)	C11 (T011, T075)	C19 (T019, T083)	C27 (T027, T091)	C35 (T035, T099)	C43 (T043, T107)	C51 (T051, T115)	C59 (T059, T123)
C04 (T004, T068)	C12 (T012, T076)	C20 (T020, T084)	C28 (T028, T092)	C36 (T036, T100)	C44 (T044, T108)	C52 (T052, T116)	C60 (T060, T124)
C05 (T005, T069)	C13 (T013, T077)	C21 (T021, T085)	C29 (T029, T093)	C37 (T037, T101)	C45 (T045, T109)	C53 (T053, T117)	C61 (T061, T125)
C06 (T006, T070)	C14 (T014, T078)	C22 (T022, T086)	C30 (T030, T094)	C38 (T038, T102)	C46 (T046, T110)	C54 (T054, T118)	C62 (T062, T126)
C07 (T007, T071)	C15 (T015, T079)	C23 (T023, T087)	C31 (T031, T095)	C39 (T039, T103)	C47 (T047, T111)	C55 (T055, T119)	C63 (T063, T127)
MI250X		MI250X		MI250X		MI250X	
GPU 4 (BUS D1)	GPU 5 (BUS D6)	GPU 2 (BUS C9)	GPU 3 (BUS CE)	GPU 6 (BUS D9)	GPU 7 (BUS DE)	GPU 0 (BUS C1)	GPU 1 (BUS C6)
	NIC (hsn2)		NIC (hsn1)		NIC (hsn3)		NIC (hsn0)

```
# /etc/slurm/gres.conf
NodeName=nid[...] Name=gpu Flags=amd_gpu_env File=/dev/dri/renderD128 Cores=[48-55]
NodeName=nid[...] Name=gpu Flags=amd_gpu_env File=/dev/dri/renderD129 Cores=[56-63]
NodeName=nid[...] Name=gpu Flags=amd_gpu_env File=/dev/dri/renderD130 Cores=[16-23]
NodeName=nid[...] Name=gpu Flags=amd_gpu_env File=/dev/dri/renderD131 Cores=[24-31]
NodeName=nid[...] Name=gpu Flags=amd_gpu_env File=/dev/dri/renderD132 Cores=[0-7]
NodeName=nid[...] Name=gpu Flags=amd_gpu_env File=/dev/dri/renderD133 Cores=[8-15]
NodeName=nid[...] Name=gpu Flags=amd_gpu_env File=/dev/dri/renderD134 Cores=[32-39]
NodeName=nid[...] Name=gpu Flags=amd_gpu_env File=/dev/dri/renderD135 Cores=[40-47]
```

Accounting and challenges

The fractional node allocation nicely extends the *core-hour* SU model. Memory and GPU resources are expressed in terms of equivalent number of cores, scaled by a given multiplier for GPU nodes.

More information at https://arxiv.org/pdf/2110.09987

Challenges:

- accurately measure the energy consumption for individual jobs on a shared node.
- Resource contention on I/O and Networking, but also on OS resources such as loop devices (containers).
- We do not use/suggest hyperthreading, but people can override default setting.

Best Practices for Shared Nodes in Slurm

Tim Wickberg

Best Practices for Shared Nodes in Slurm

- Two broad areas to consider
  - Scheduling Policy
  - Node Resource Management / Enforcement

Scheduling Policy

Scheduling Policy

- DefMemPerCPU
  - Without these set, and without users specifying --mem, then whole node will be allocated
  - DefMemPerGPU needed if users use --gpu
- Controls on what workloads may share a node
  - MCS may be used to only permit jobs from the same account, group, user, or arbitrary label
  - The "--exclusive=user" option on the job can also provide job-by-job control
- Avoid idle GPUs by reserving CPUs that can only be used by GPU jobs
  - RestrictedCoresPerGPU on the Node
  - Or MaxCPUsPerNode on the Partition
    - Requires GPU vs CPU Partitions

Usage Charges

- TRESBillingWeights can be used to control "charges" for fair-share based on different ratios of cpu, mem, gpu, ...
  - Otherwise cpu time is the default
  - Stored as the "billing" TRES in the accounting database
  - One limitation - this is a partition level setting, cannot be set at the node level currently
- PriorityWeightTRES
  - Similar but independent formula, used to calculate the job size priority component
- PriorityFlags=MAX_TRES or MAX_TRES_GRES
  - Bill on the highest component used, or highest cpu/mem plus all gres (GPU) use

Job Validation

- Two plugin interfaces designed for site-specific business logic
  - CliFilterPlugins
    - Runs as the user before job is submitted
      - But a sophisticated user can bypass
  - JobSubmit
    - Runs in slurmctld
    - Cannot be bypassed
    - But needs to be quick
      - And should avoid interacting with external systems

What is a socket?

- Slurm's model of a node is Board, Socket, Core, Thread
- Socket is the "best" split point for workloads
  - Board is mostly ignored - was designed for old-school NUMA systems like SGI UV
- SlurmdParameters=l3cache_as_socket may be useful on CCX machines
  - E.g., Frontier runs with this enabled

Node Resource Management

Node Resource Management

- proctrack/cgroup and task/cgroup enabled
    - Ensure jobs are restricted to allocated CPUs, Memory, GPUs
- pam_slurm_adopt
    - pam_slurm plugin insufficient as SSH won't be confined to the allocated resources
- job_container/tmpfs
    - Creates separate filesystem namespaces for each job
        - Usually isolating common scratch locations
        - Usually /tmp, /dev/shm
    - Can manage bind mounts - not just adding new tmpfs mounts over existing paths
        - Can set quota on directory on local disk, then bind mount that into the expected scratch location
    - Some sites even using this to start automounters
    - Strongly recommended
        - Avoids trying to build complex Epilog scripts to clean up scratch spaces

UnkillableStepProgram / UnkillableStepTimeout

- Try to recover from stuck processes automatically
- Most common from hung filesystem processes
- GPU issues can also potentially trigger this

GPU Sharing

- NVIDIA dominates the discussion here
  - AMD and Intel not as common
    - Less refined software stacks
- Still not as common
  - Difficult to handle enforcement correctly
  - GPU models for resource management are much coarser than main Linux system
    - Tied to specific hardware, and likely hiding behind proprietary APIs

GPU Sharing

- Three approaches
    - NVIDIA MIG
        - Hardware-driven partitioning
        - Can be changed dynamically
            - Slurm doesn't support managing this though, only statically partitioned MIG
    - NVIDIA MPS
        - Software enforced, percentage allocated to different workloads
    - "Shards"
        - Slurm-specific, works on any device
            - Just a way to track soft divisions of one or more GPUs in the node
            - Requires jobs to cooperate

Caveats

Current Limitations

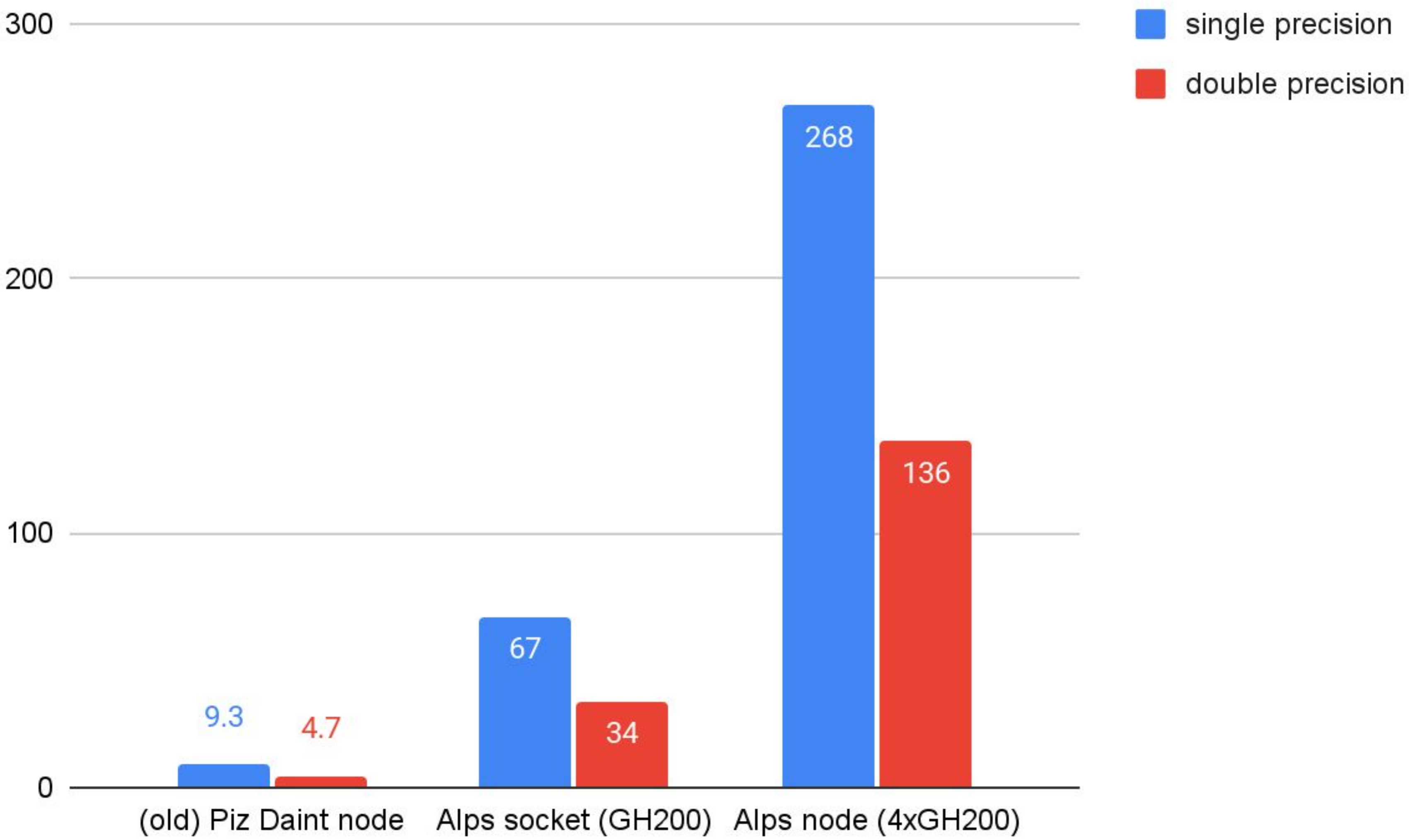
- Depending on your security policy, shared access may come with some concerns from workloads being able to view details from each other
  - Process names (scripts, executable names, file names may be sensitive)
- Contention between applications for shared resources
  - Filesystem contention is one of the biggest headaches
  - Network contention less common, but possible
    - More likely when running multi-node GPU jobs alongside multi-node CPU-only jobs
  - Memory bandwidth
  - Thermal limiting from CPUs impacting other cores

Future Directions

- Working to expand namespace support beyond the filesystem
  - Such as PID, User, Network, IPC

CSCS’s perspective

Why do we need to start sharing at CSCS



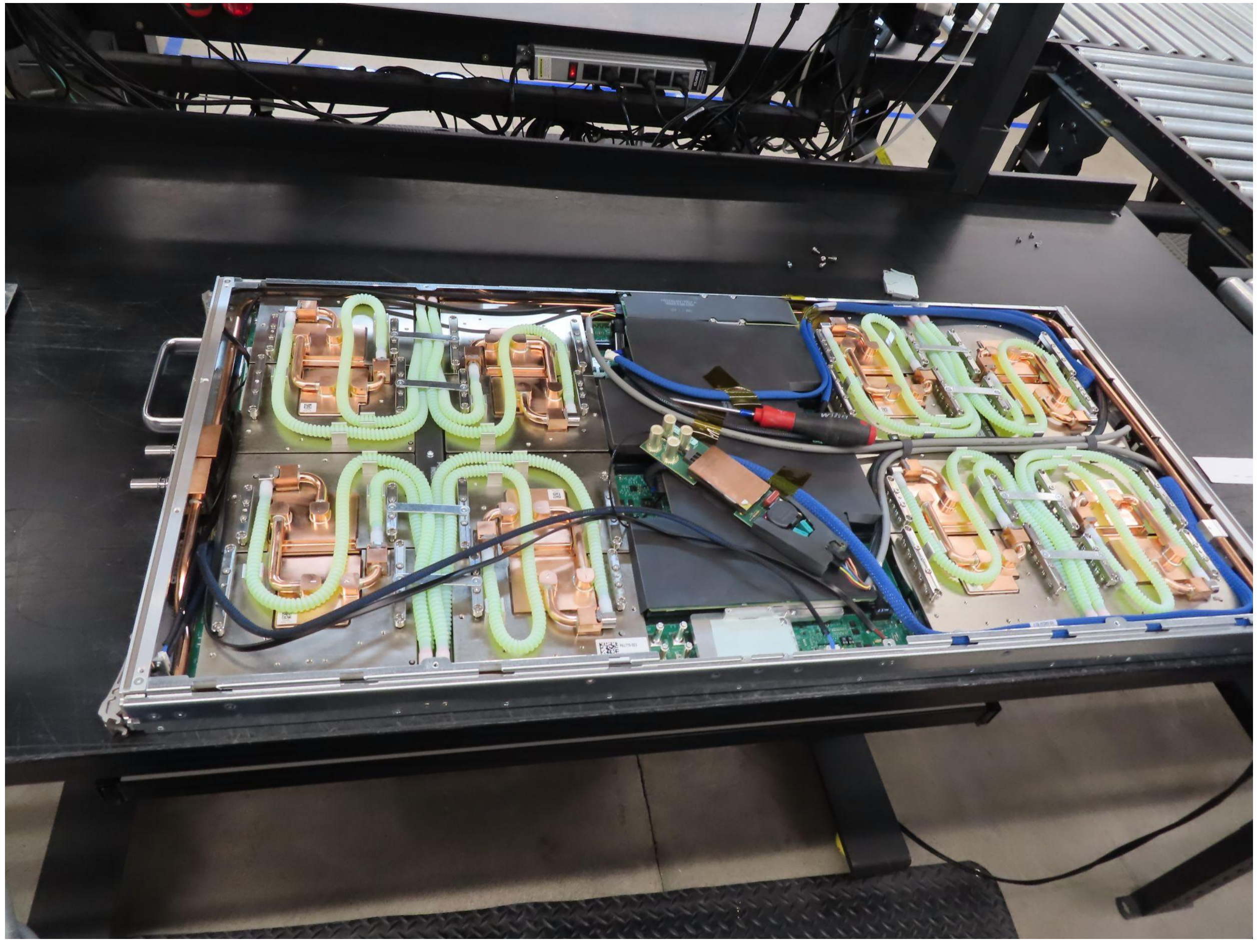
Blanca Peak Nodes

Each Blanca Peak (EX254n) node:

- 4x GH modules
- 2x Sawtooth cards
  - 4x Slingshot 200 Gb NICs

Each GH200 module:

- 72 core ARM Neoverse v2 CPU with 128 GB LPDDR
- H100 with 96 GB HBM2e
- TDP = 800W



What are we doing right now? (Spoiler alert)

- We allocate in whole nodes!

We need to support a bunch of workloads

- Jupyter notebooks
- Debug partition
- Interactive development and testing
- CI/CD pipelines
- Data science and AI workflows
- High throughput computing
- HPC codes that can’t make efficient use of all four GPUs
- CPU-heavy codes
- …

What we are currently testing

- Allocating by GH200 module, i.e. four jobs per node
- Need to ensure locality of CPUs and memory
  - Use cgroups to enforce CPU, memory and GPU affinity

```
# cgroup.conf

ConstrainCores=yes
ConstrainRAMSpace=yes
ConstrainDevices=yes
```

```
# gres.conf

Name=gpu File=/dev/nvidia0 Cores=0-71
Name=gpu File=/dev/nvidia1 Cores=72-143
Name=gpu File=/dev/nvidia2 Cores=144-215
Name=gpu File=/dev/nvidia3 Cores=216-287
```

```
# slurm.conf

ProctrackType=proctrack/cgroup
TaskPlugin=task/affinity,task/cgroup
```

  - Partition

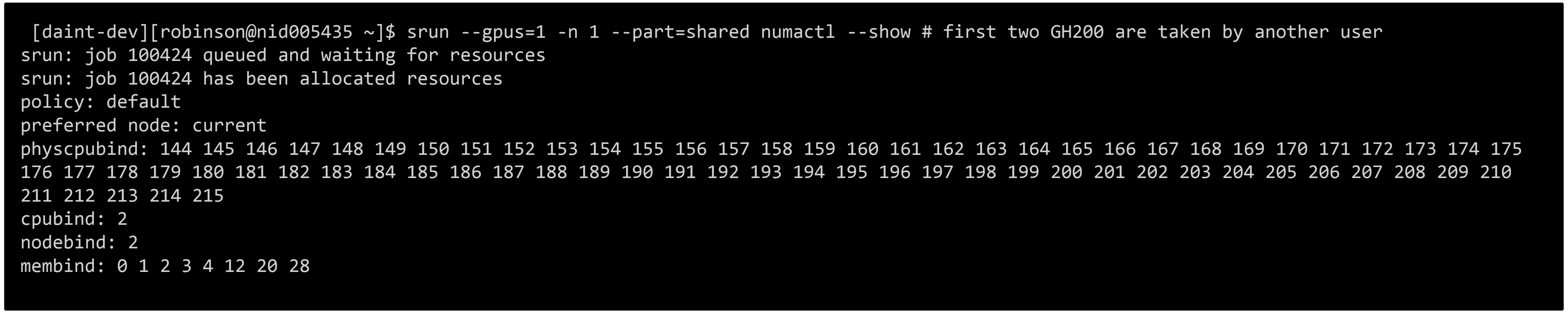
```
SelectTypeParameters=CR_Socket_Memory DefCpuPerGPU=72 DefMemPerGPU=115000
```

  - Nodes

```
Sockets=4 CoresPerSocket=72 ThreadsPerCore=1 RealMemory=460000 Gres=gpu:4
```

Works, but affinity?

- Users simply request number of GPUs and get the right amount of Grace
- May need further tuning of `RealMemory` and `DefMemPerGPU` (Grace & Hopper)
- `numactl` suggests memory isn’t fully localized



```
 [daint-dev][robinson@nid005435 ~]$ srun --gpus=1 -n 1 --part=shared numactl --show # first two GH200 are taken by another user
srun: job 100424 queued and waiting for resources
srun: job 100424 has been allocated resources
policy: default
preferred node: current
physcpubind: 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175
176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210
211 212 213 214 215
cpubind: 2
nodebind: 2
membind: 0 1 2 3 4 12 20 28
```

  - ○ Memory limits might be sufficient to ensure locality (?)

What we are planning next

- NVIDIA Multi-Instance GPU (MIG)
  - Partition GPU into up to seven instances (configurable)
  - Instances are isolated (compute, cache, memory)
  - Different from time slicing: workloads run in parallel
- Set by `nvidia-smi`
  - Should not require a GPU reset
- But… it crashes our nodes 🤦
  - CAST opened - now with NVIDIA
  - Anyone else using MIG? (on GH?)
- What about Run:ai?
  - AI workload and GPU orchestration (Kubernetes)
  - Purchase by NVIDIA, completed in Dec 2024 (open-sourced the scheduler part)

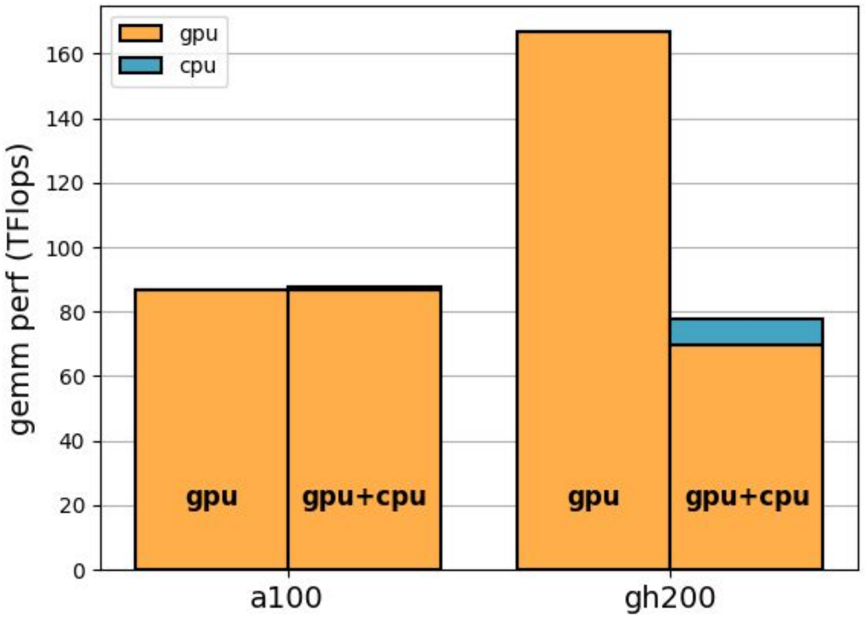
Sharing a Grace Hopper module?

- GH200 has a decent CPU
  - 72 Arm Neoverse V2 cores (4×128-bit SIMD units per core)
  - 128 GB of LPDDR of memory
  - >2 Teraflops in DGEMM
- GPU codes often throw everything onto the GPU
- So let’s co-locate applications on a single GH200…!

…but…

Sharing a Grace Hopper module?

- Power sloshing (“power steering”)
  - Dynamic power redistribution between CPU & GPU
  - Power envelope of the GH module is fixed (~800 W)
  - If CPU is idle / low load the power budget moves to GPU (clocks go up!) **and vice versa**
  - **Good** for a mixed workload e.g. CPU-heavy initialization/pre-proc then GPU training
  - Potentially **troublesome** for co-locating different applications from different users
  - CPU first behaviour: OS-level tasks, thermal controllers -> Grace has a minimum power floor



Using 64 Grace cores **halves H100 performance!**

🔥 GPU Burn 🔥

gpu-burn Public

GPU Burn is a popular tool for measuring GPU Flops

- Runs GEMM back-to-back on the GPU for a duration and measures average performance

To investigate the tight integration of GH200 we want to understand:

- Both CPU and GPU performance
- Both CPU and GPU power consumption
- Whether workloads on CPU affect GPU workloads, and vice versa?

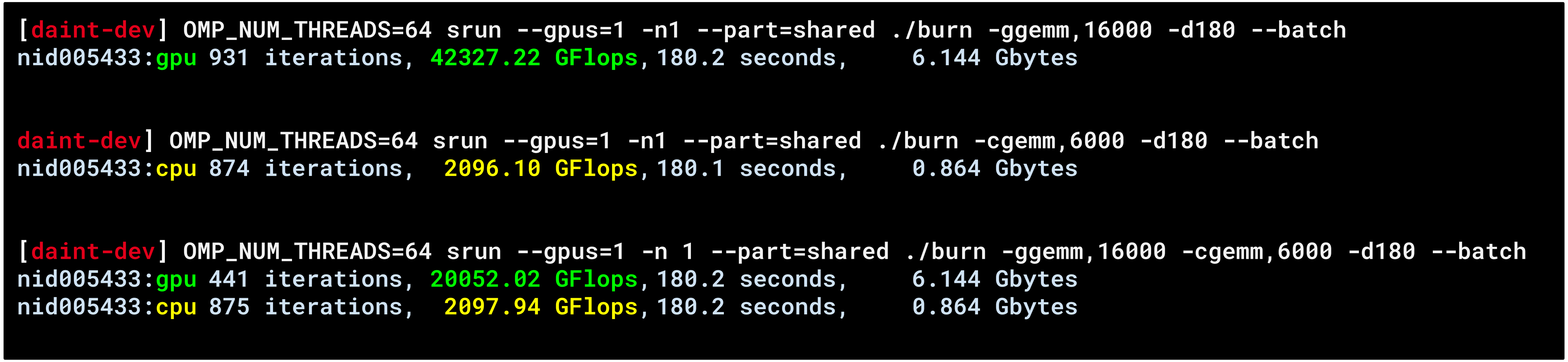
This requires the ability to:

- Run workloads on CPU and GPU concurrently
- Configure the type of workload (one of GEMM, STREAM, NONE) on each

🔥 NodeBurn 🔥

Colleagues at CSCS developed **nodeburn**

- GEMM and STREAM can be configured for both CPU and GPU
- pm_counters are supported
- Batch mode for executing on many nodes at the same time
- github.com/eth-cscs/node-burn



```
[daint-dev] OMP_NUM_THREADS=64 srun --gpus=1 -n1 --part=shared ./burn -ggemm,16000 -d180 --batch
nid005433:gpu 931 iterations, 42327.22 GFlops,180.2 seconds,    6.144 Gbytes


daint-dev] OMP_NUM_THREADS=64 srun --gpus=1 -n1 --part=shared ./burn -cgemm,6000 -d180 --batch
nid005433:cpu 874 iterations,  2096.10 GFlops,180.1 seconds,    0.864 Gbytes


[daint-dev] OMP_NUM_THREADS=64 srun --gpus=1 -n 1 --part=shared ./burn -ggemm,16000 -cgemm,6000 -d180 --batch
nid005433:gpu 441 iterations, 20052.02 GFlops,180.2 seconds,    6.144 Gbytes
nid005433:cpu 875 iterations,  2097.94 GFlops,180.2 seconds,    0.864 Gbytes
```

Additional challenges

- Accounting / billing is easy when done by node
  - We’d like to do fine-grained accounting
  - CPUs, GPUs (and parts thereof), memory, network, I/O, power…
  - Are the tools ready?
- GPU partitioning
  - Static partitioning: not yet working
  - Dynamical partitioning: being investigated (Run:ai / k8s)

Sharing is caring!

- Proposal: Share our collective knowledge on the CUG Slack (**#node-sharing**)
  - Q&A on any topics related to node sharing
  - Handy prolog or epilog scripts (e.g. node cleanup, /dev/shm, TMPDIRs)
  - Slurm / PBS configurations
  - Accounting practices
  - Gotchas
  - …

Survey and Q&A

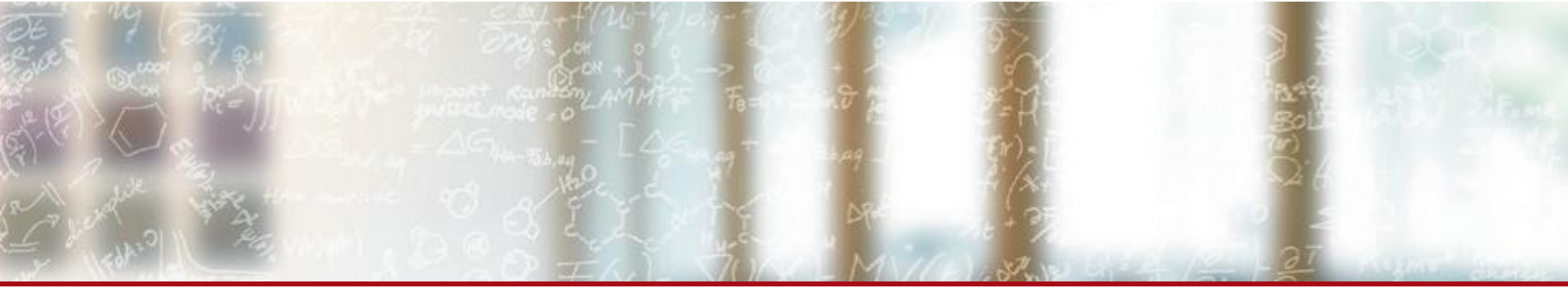
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

pawsey

SCHEDMD
The Slurm Company

NERSC

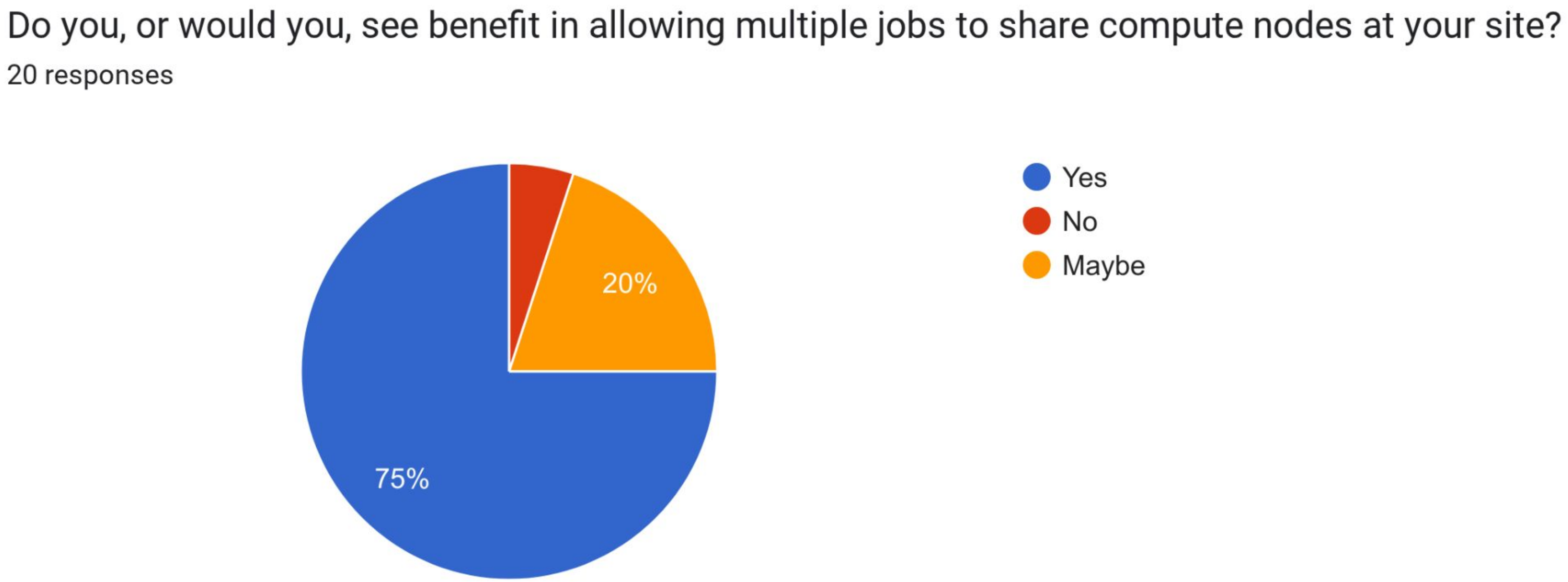
ETH zürich



Survey time!

CUG 2025 BoF
May 5th, 2025

Benefit in sharing nodes?



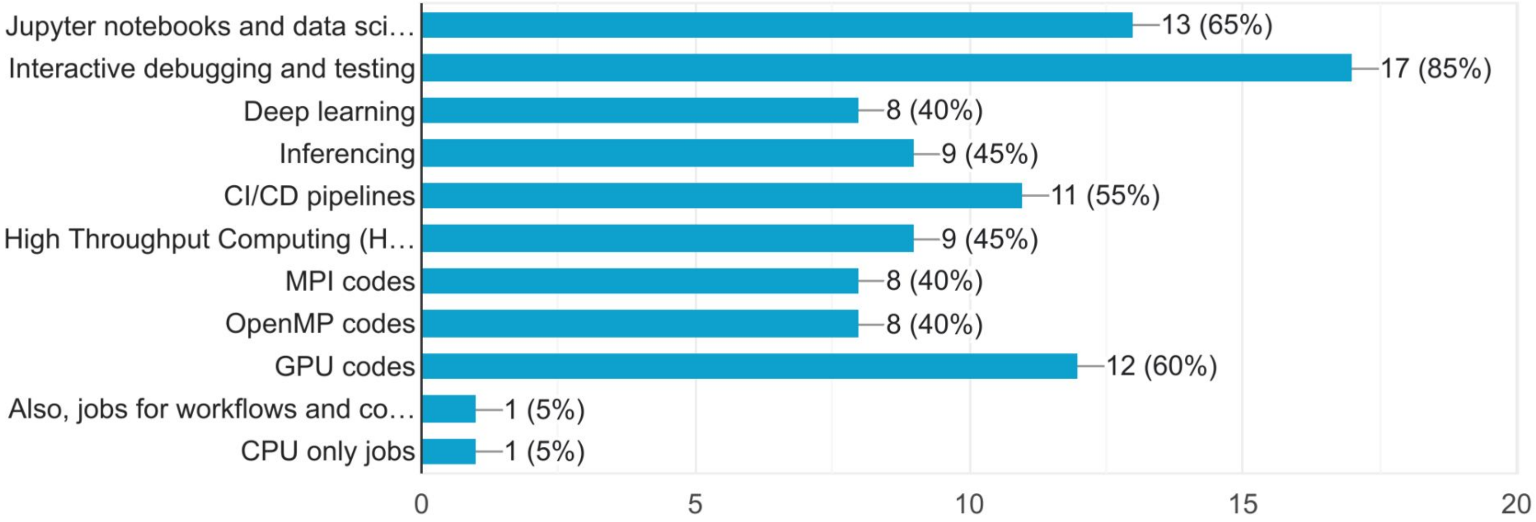
Do you, or would you, see benefit in allowing multiple jobs to share compute nodes at your site?

20 responses

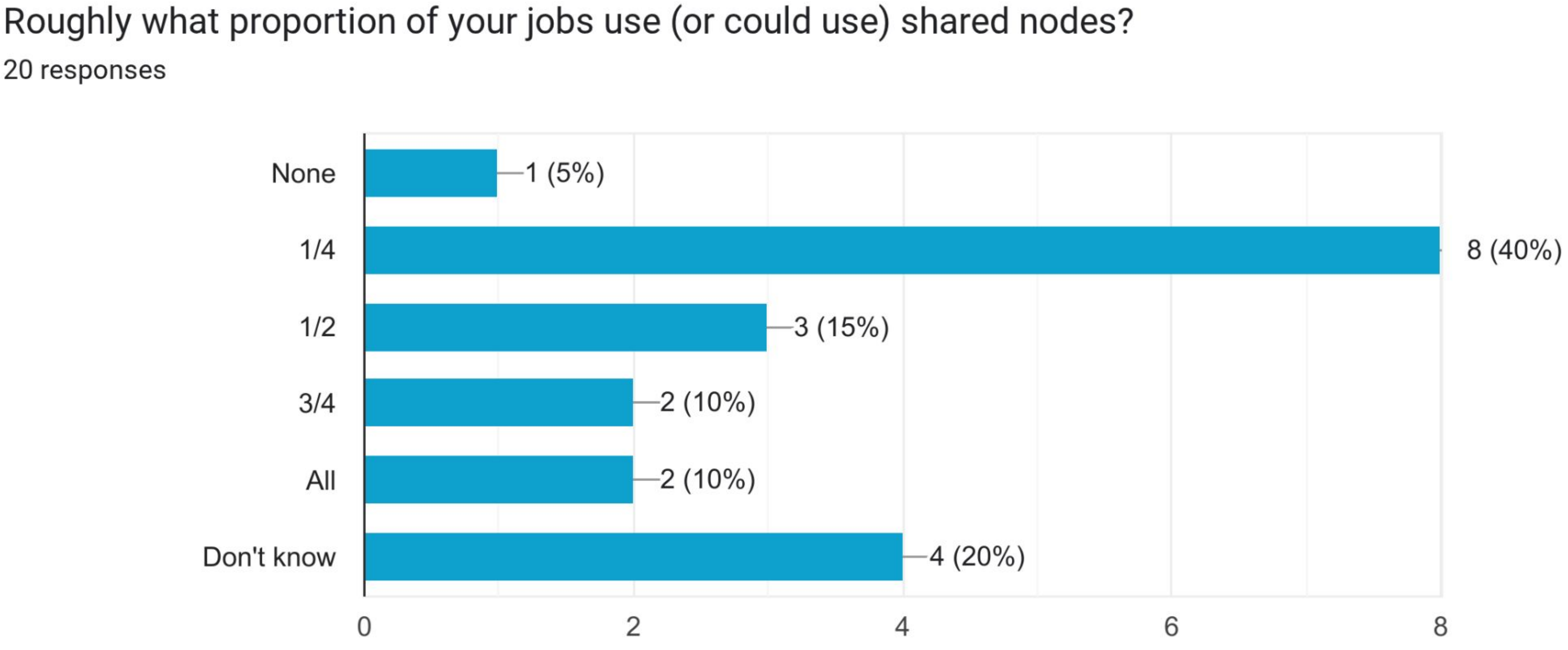
Workloads that could benefit

Which workloads do you think are or could be suited to node sharing at your site?

20 responses



Impact of node sharing



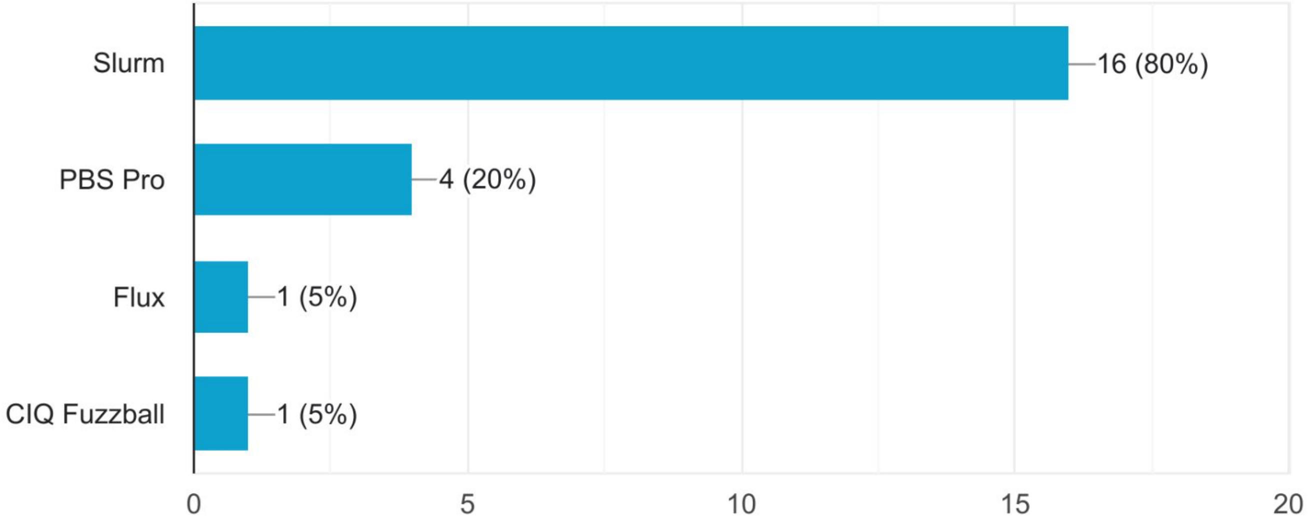
Roughly what proportion of your jobs use (or could use) shared nodes?

20 responses

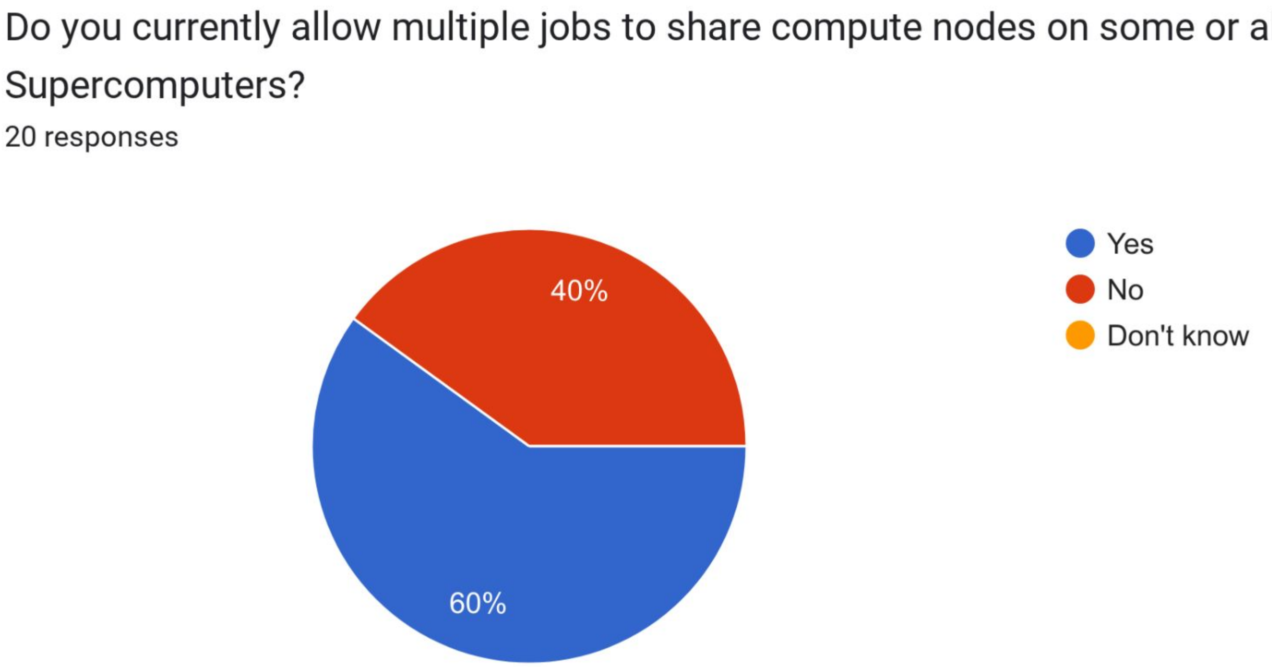
Workload managers

Which workload manager(s) do you use on your HPE Cray Supercomputers?

20 responses



State of the game



Do you currently allow multiple jobs to share compute nodes on some or all of your HPE Cray Supercomputers?

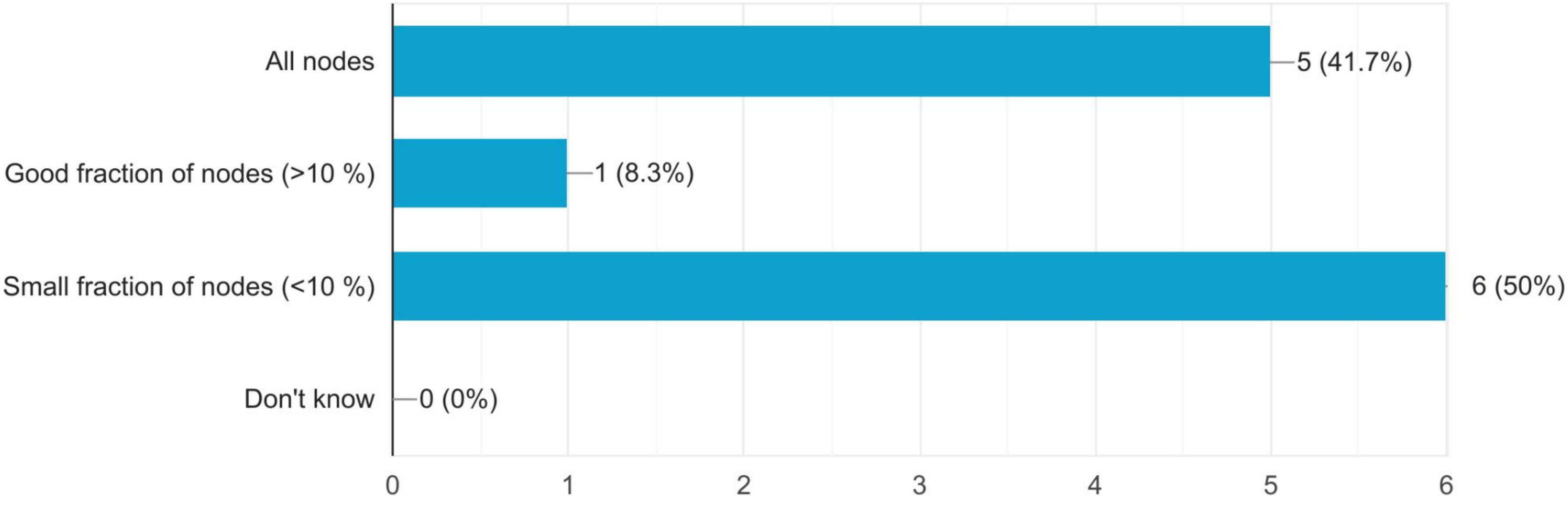
20 responses

The following questions were answered by those who share nodes…

Node sharing at your site

On roughly what fraction of your total nodes do you have node sharing enabled?

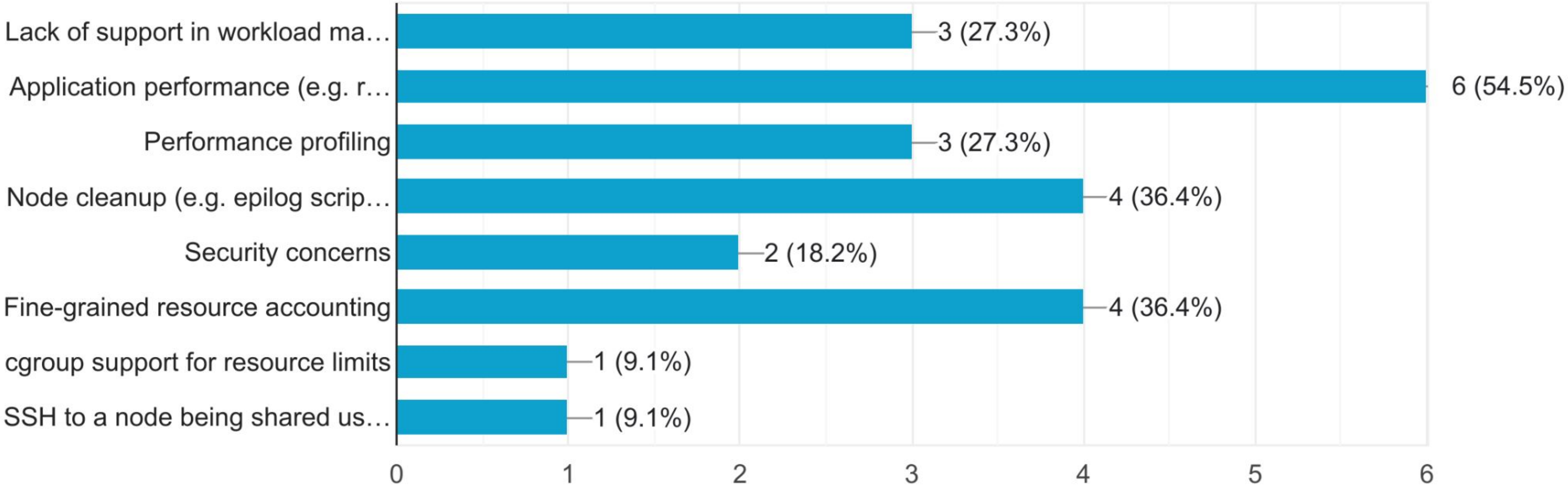
12 responses



Challenges to node sharing

What are the biggest challenges you experience(d) with respect to node sharing?

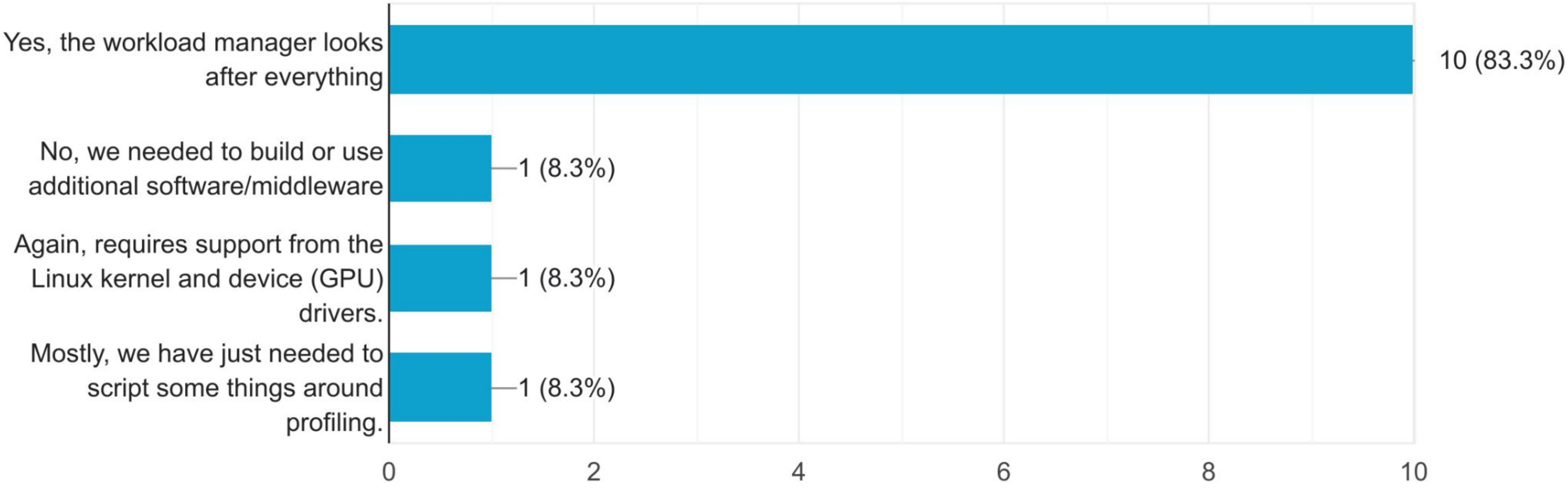
11 responses



Node sharing management

Is the node sharing managed by your workload manager alone? (Specify in "Other")

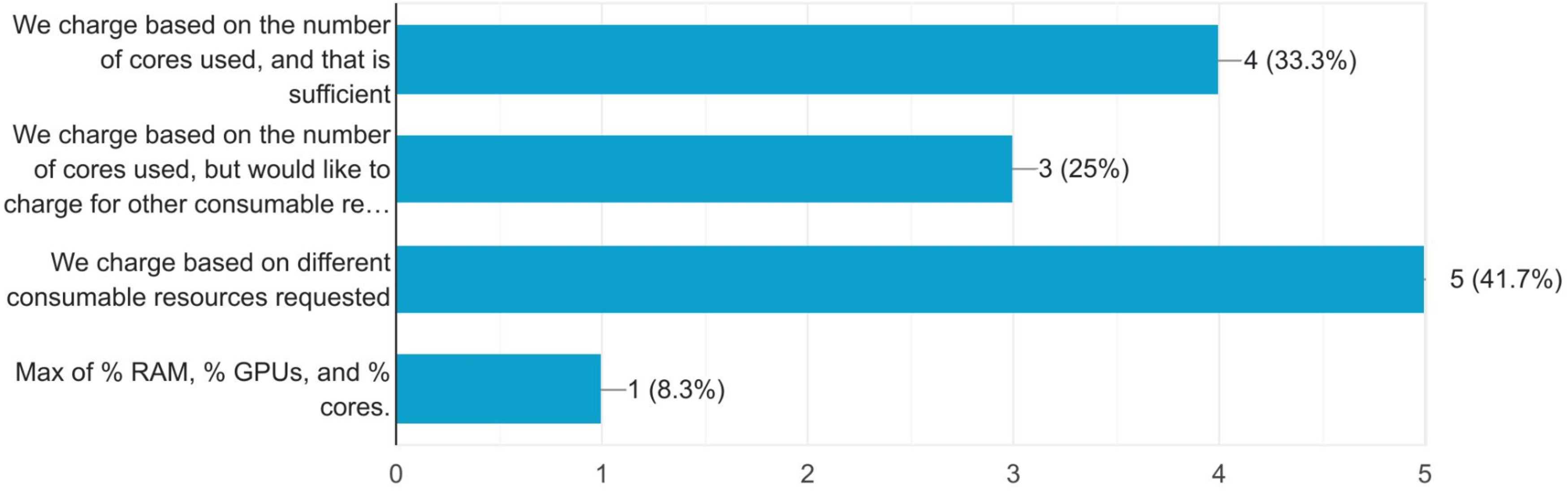
12 responses



Accounting / charging

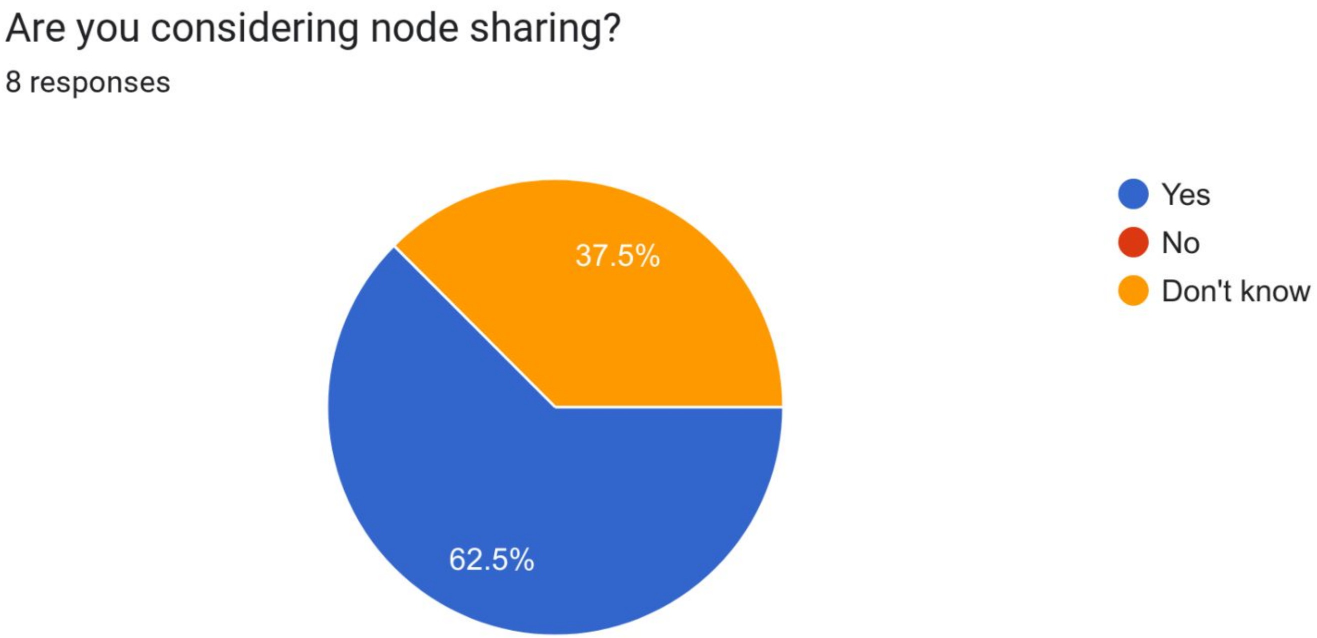
On shared nodes, do you charge jobs only for the number of CPU cores requested, or also based on other consumable resource requests, e.g. memory?

12 responses



The following questions were answered by those who don’t currently share nodes

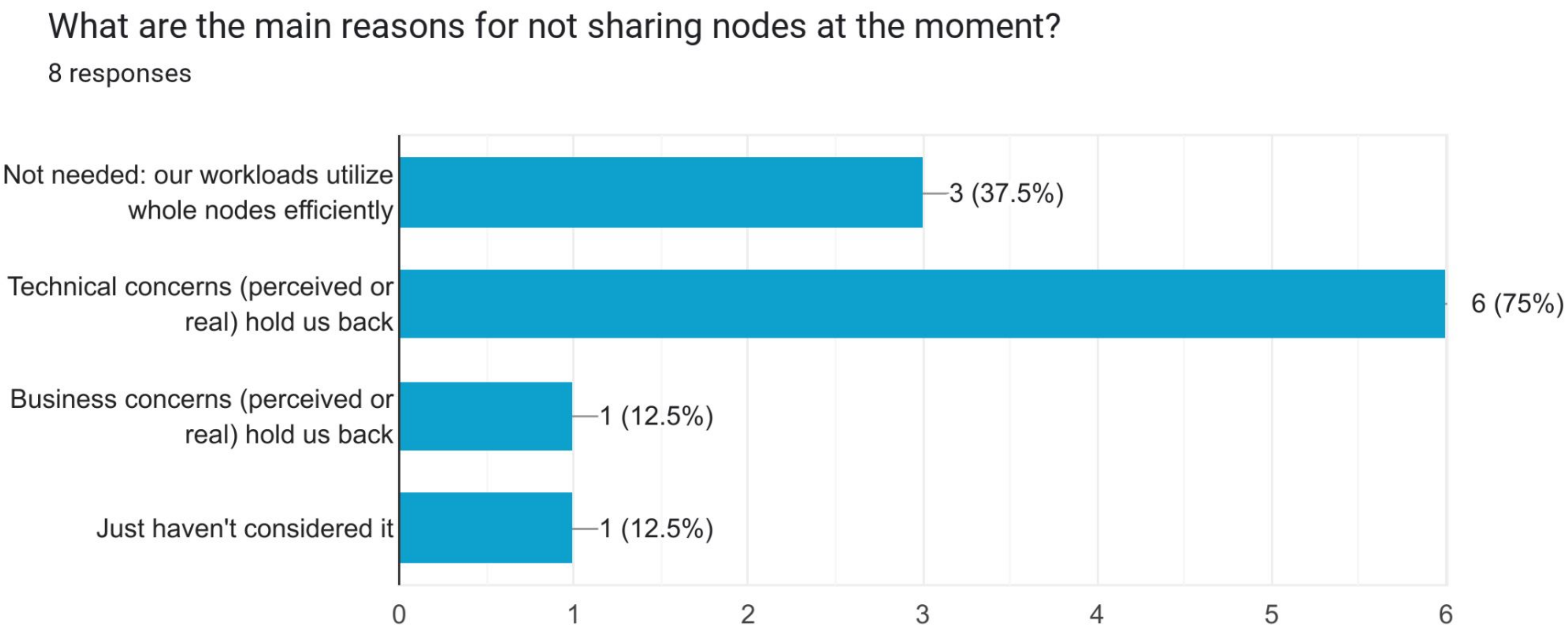
Future plans



Are you considering node sharing?

8 responses

Why not?



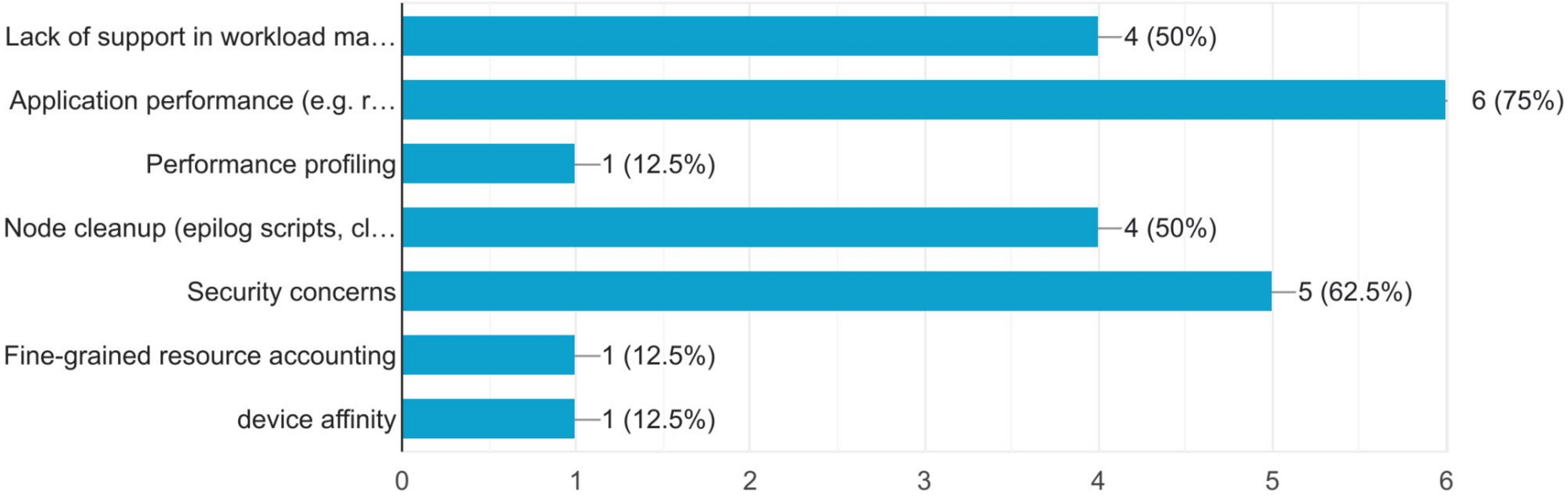
What are the main reasons for not sharing nodes at the moment?

8 responses

Perceived challenges

What challenges do you perceive with respect to node sharing?

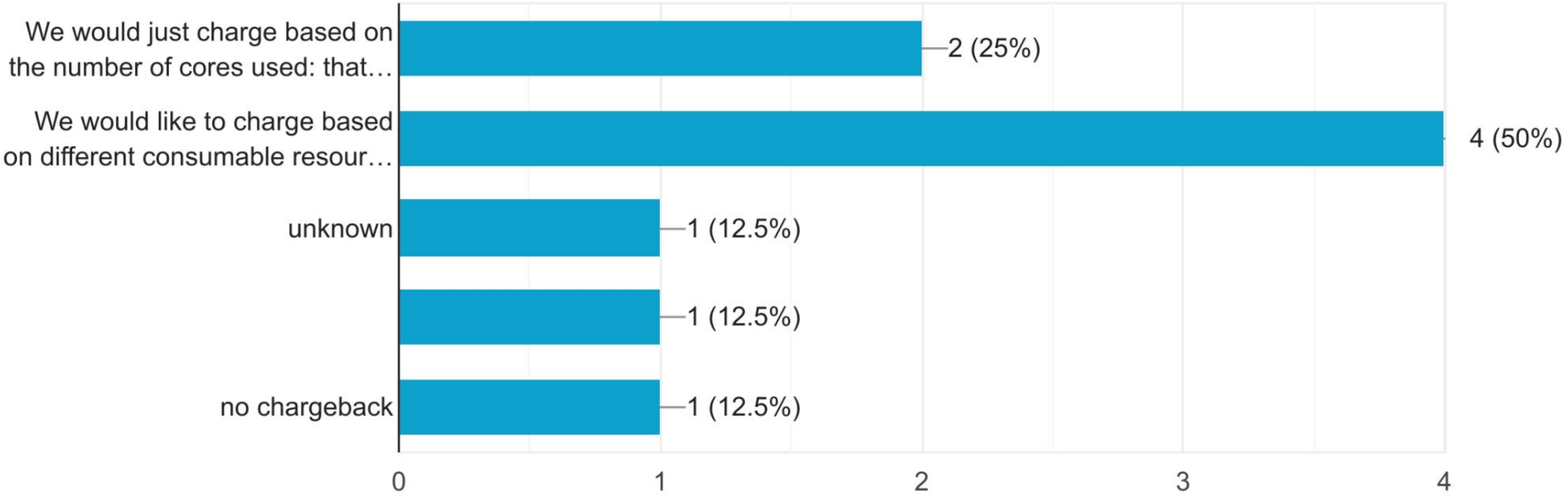
8 responses



Accounting / charging

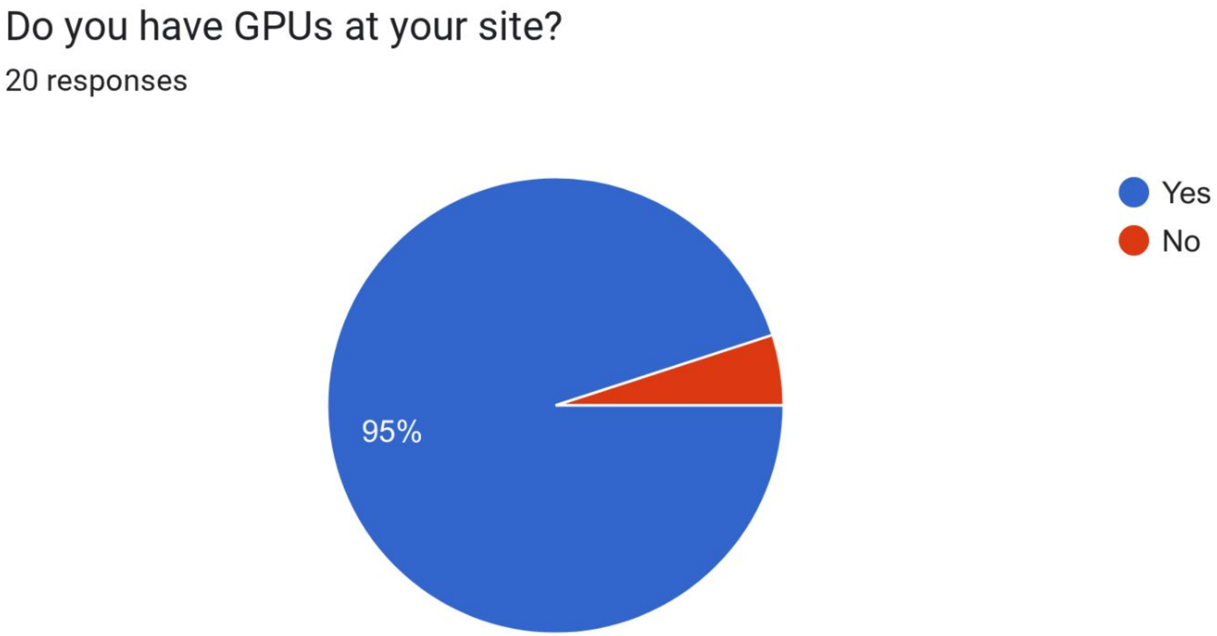
If you were to share nodes, would you charge jobs only for the number of CPU cores requested, or also based on other consumable resource requests, e.g. memory?

8 responses



The following questions are about GPUs

GPU-based nodes



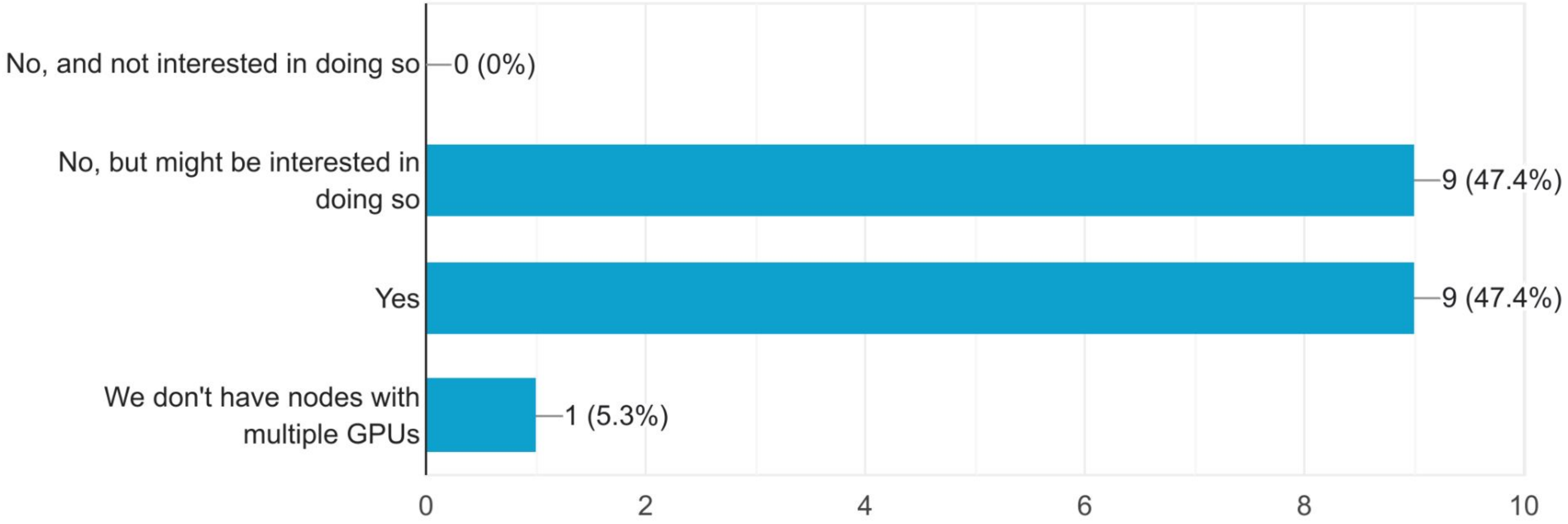
Do you have GPUs at your site?

20 responses

Sharing of Multi-GPU nodes

If you have nodes with multiple GPUs, do you share the nodes in a way such that each job gets a dedicated GPU?

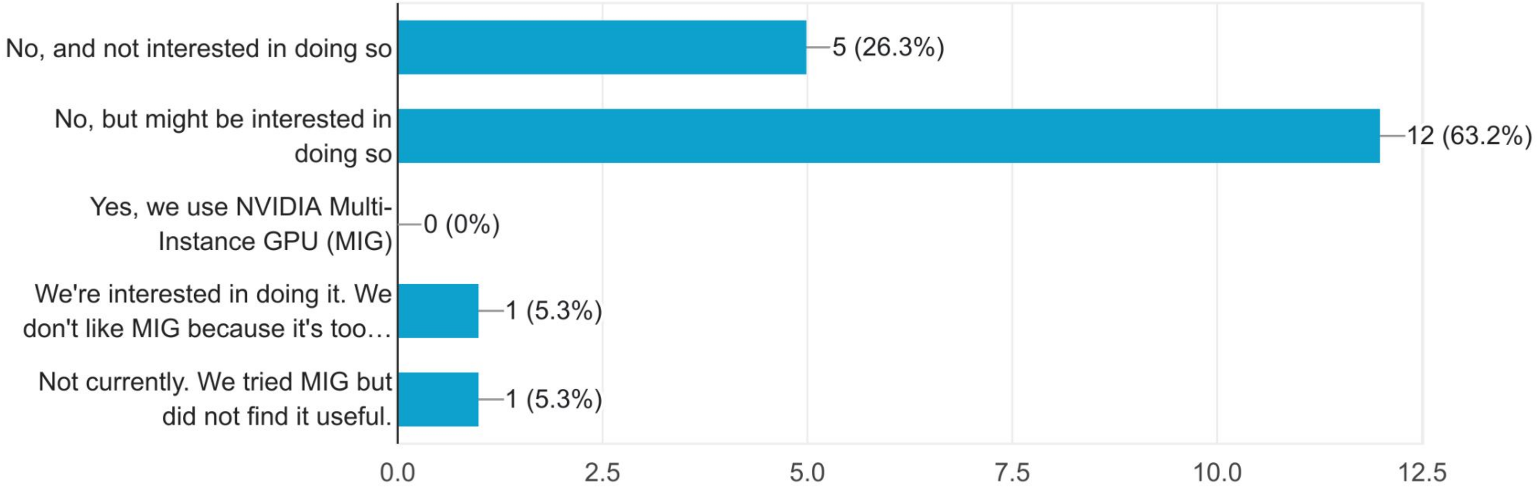
19 responses



Sharing of individual GPUs

Do you partition individual GPUs so that multiple jobs can use the same GPU? e.g. through NVIDIA Multi-Instance GPU (MIG) or similar means for AMD or Intel (Please specify in "Other")

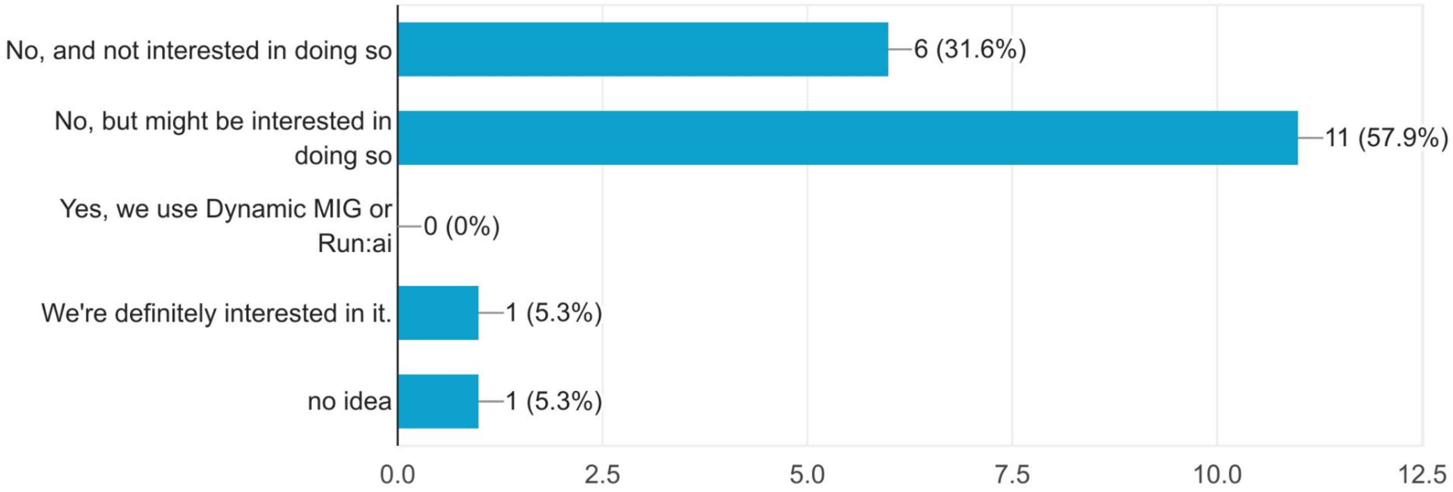
19 responses



Dynamic sharing of GPUs

Do you enable dynamic partitioning of individual GPUs, e.g. through Dynamic MIG / Run:ai or similar means for AMD or Intel (Please specify in "Other")

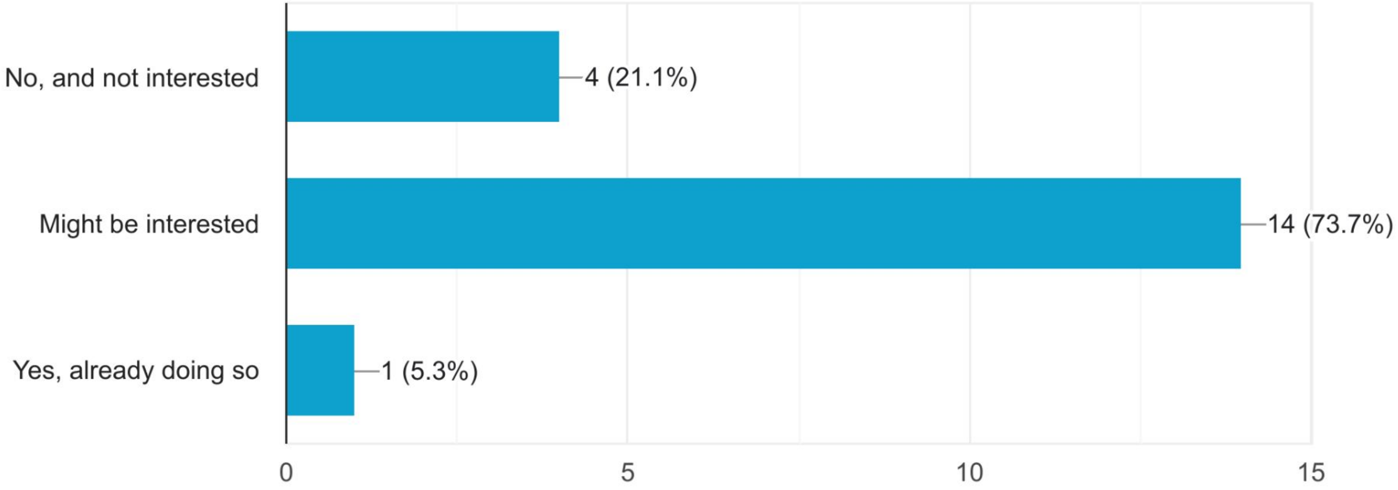
19 responses



Co-locating applications

Have you considered co-locating different applications on a node to use different resources more efficiently? e.g. running a GPU-heavy code and a CPU...ribe further in "Other" if you are already doing so)

19 responses


No, and not interested
Might be interested
Yes, already doing so
4 (21.1%)
14 (73.7%)
1 (5.3%)