

Introduction to JobMon: A Job Performance Monitoring tool at NHR@KIT

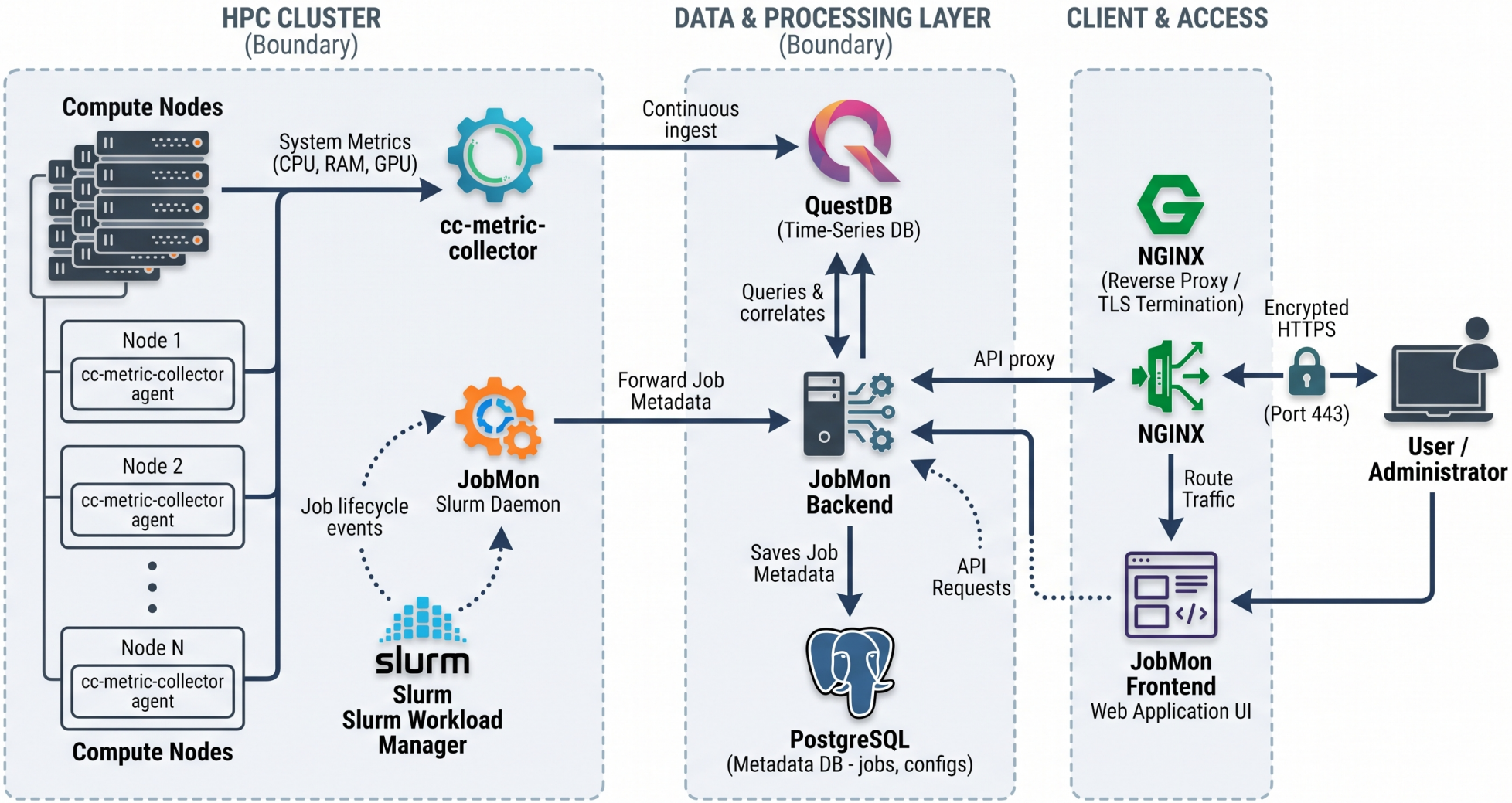
Begatim Bytyqi | 21. October 2026

Scientific Computing Center

What should you be able to answer after this course?

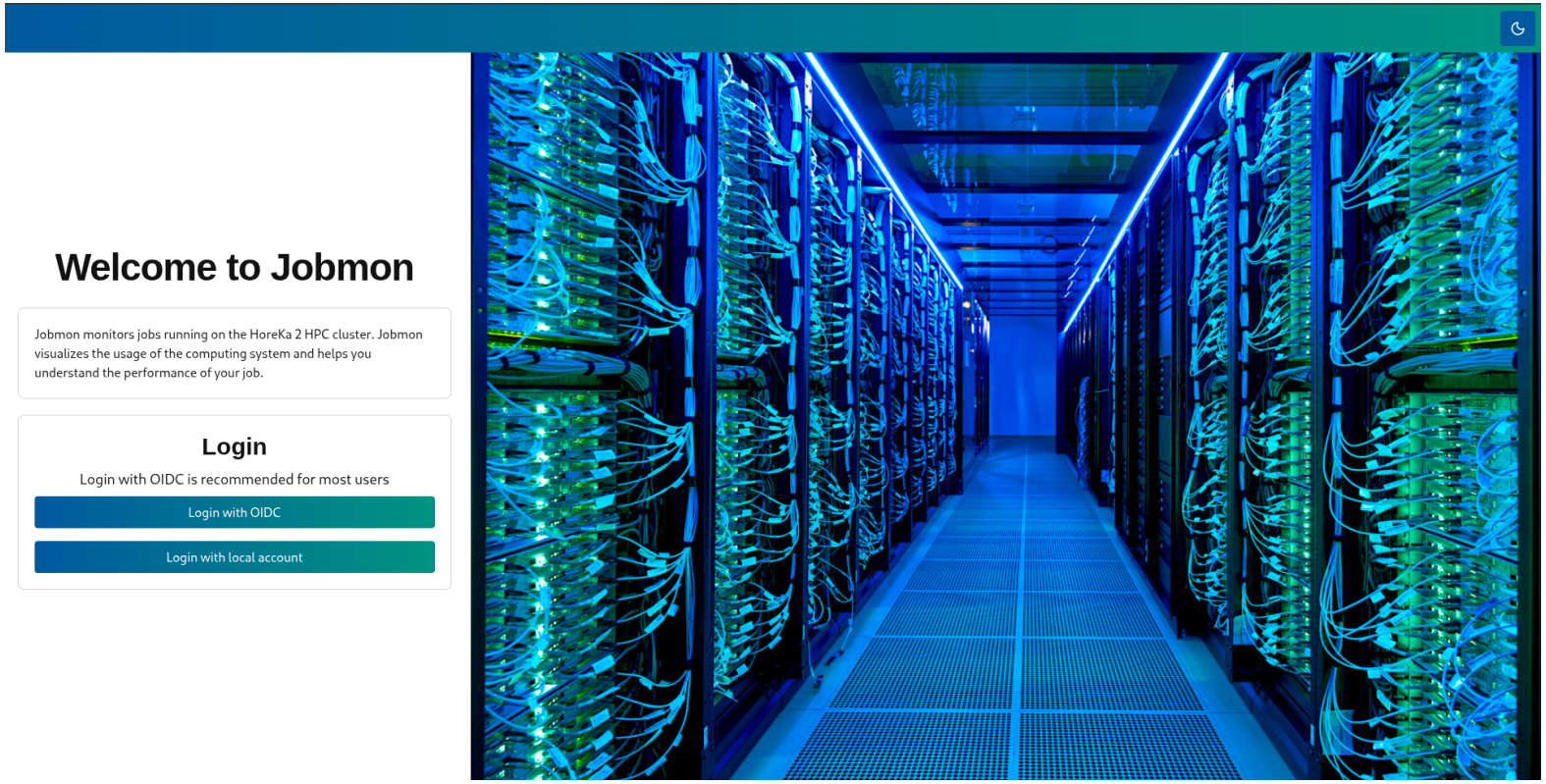
- What does my application do over time?
- Does the job look compute-, memory-, communication-, or I/O-bound?
- Am I actually using all the resources I allocated?
- Do all nodes, CPU sockets, or GPUs behave similarly, is there any imbalance?
- Which metrics should I inspect first?
- How to get the most out of JobMon.

JobMon Architecture Overview

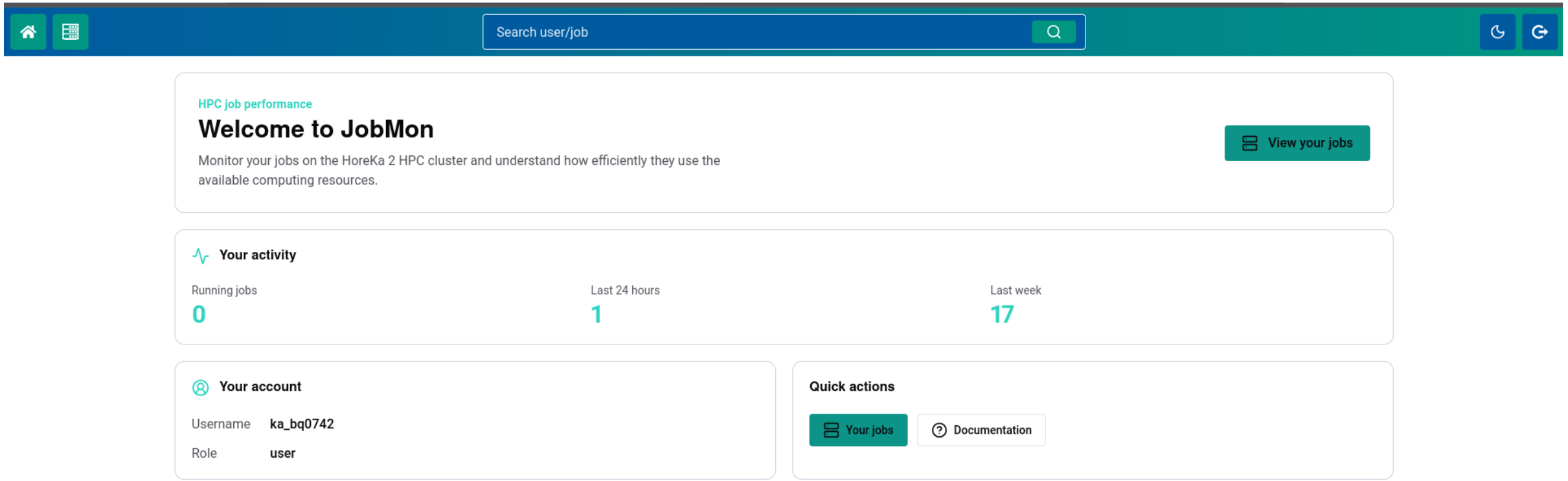


Accessing JobMon

- Open JobMon
- Log in using OIDC option
- The same network restrictions as for HoreKa 2 apply
- Connect through the required KIT network or VPN if necessary



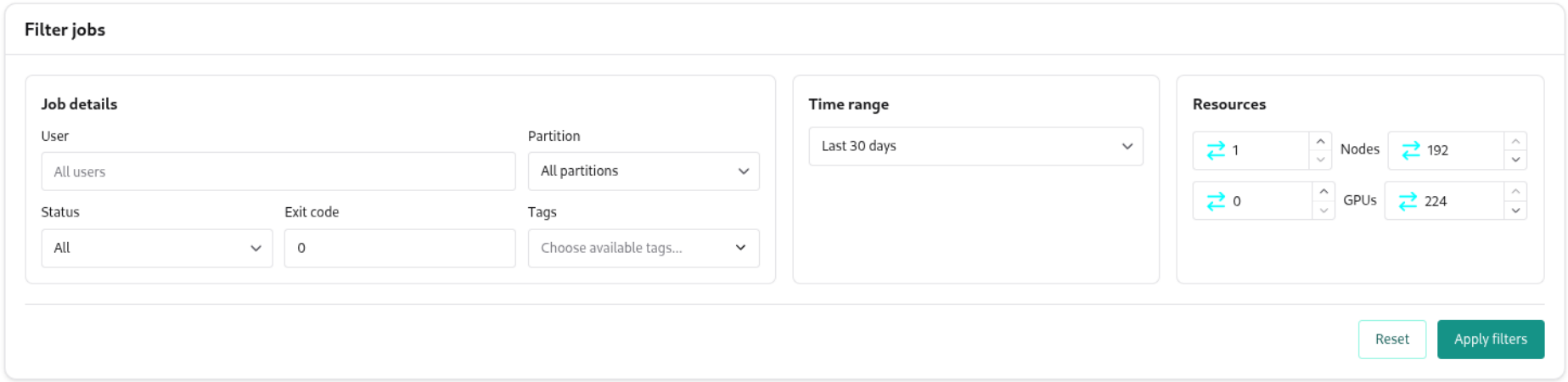
Welcome Page



Finding the job you want to investigate

Useful filters

- Batch partition
- Number of nodes or GPUs
- Job status (Running or Finished)
- Execution time
  - Fixed Time ranges
  - Custom Dates
- Exit code
- Tags



Search Bar

Use the search bar to quickly view the job of your interest.

Jobs with shared resourcess

- Compute nodes by default are shared.
- All metrics shown by JobMon describe activity on the node or device, no accounting information is stored.
- To disable resource sharing submit the job with:

```
#SBATCH -exclusive
```

- A non-exclusive job can still be analyzed given no other job was running on the same resources at the same time.

Check JobMon

When JobMon finds no overlapping jobs, the job is marked with:

No concurrent Jobs detected

Job visibility

Who can see a job?

- By default, a job is visible only to:
  - the job owner
  - JobMon administrators
- Other users cannot access the job directly.

Sharing a job

- Job owners and administrators can create and remove guest links.
- The link opens the job in JobMon's guest view.
- No ownership of the original job is required.

Demo Application Workloads

Categories

- *Compute Bound*
  - Performance is limited by floating point performance
  - Benchmark *DGEMM* performs a matrix matrix multiplication
- *Memory bound*
  - Performance is limited by memory bandwidth
  - Benchmark *Stream* / *BabelStream* performs vector computations
  - Benchmark *HPCG* performs the conjugate gradients method with a sparse matrix
- *Communication bound*
  - Performance is limited by interconnect bandwidth or latency
  - Benchmark *OSU Micro-Benchmarks* performs MPI point to point communication
- *IO bound*
  - Performance is limited by filesystem throughput, metadata operation rate or I/O operation rate.
  - Benchmark *IOR* performs parallel IO operations

A typicall JobMon diagnostic workflow

1. Check job metadata and allocated resources.
2. Use the overview / spider plot for initial assessment.
3. Inspect the overall time evolution of the job.
4. Select an interesting time interval.
5. Check compute activity first.
6. Correlate it with memory, interconnect, and filesystem activity.
7. Check whether nodes, sockets, or GPUs behave similarly.
8. Form a hypothesis and identify one metric that could confirm or contradict it.
9. Change one thing and compare with another run.

Spider plot

Where should I investigate first?

- High-level summary of overall job behaviour
- Combines metrics from CPU, GPU, memory, filesystem, and Infiniband
- Useful for choosing the *first* metric category to inspect
- Based on aggregated values (in case of multi-node jobs), so execution phases can be hidden

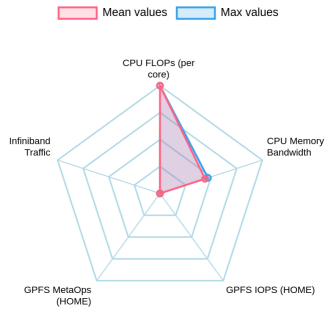
Exercise 1: What do these jobs look like?

Question

For each plot, identify the dominant resource signal and explain the evidence for your choice.
Tip: Check the list of benchmarks used.

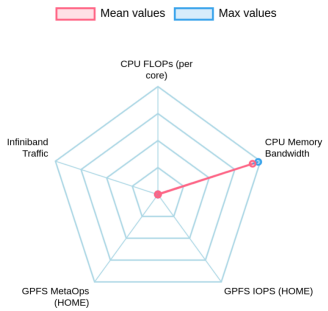
Job A

Id: **3015837**
User: bq0742 (hk-project-scs)
Partition: cpuonly
Name:
Nodes: 1
Start: 27.3.2025, 16:25:13
End: 27.3.2025, 17:21:28
Exit code: 0



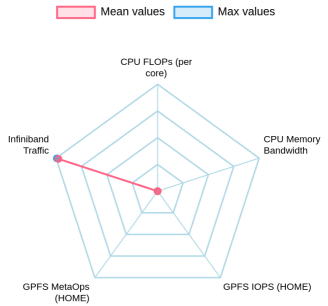
Job B

Id: **3015838**
User: bq0742 (hk-project-scs)
Partition: cpuonly
Name:
Nodes: 1
Start: 27.3.2025, 16:25:13
End: 27.3.2025, 17:27:08
Exit code: 0



Job C

Id: **3019847**
User: bq0742 (hk-project-scs)
Partition: cpuonly
Name:
Nodes: 2
Start: 28.3.2025, 15:30:33
End: 28.3.2025, 16:28:00
Exit code: 0



Exercise 1: Solution

Job A — DGEMM

Strong compute signal. Memory activity is also present because data must still be supplied to the processor.

Job B — STREAM

Strong memory-bandwidth signal with comparatively less emphasis on floating-point throughput.

Job C — OMB

Strong InfiniBand signal, consistent with a communication-focused workload.

Exercise 2: Diagnose with spider plot

Use the spider plot to check if all resources where used

Job **33475**

User: ka_bq0742 (ka_scc)

Partition: gpu-h100

Name: gpu.dgemm

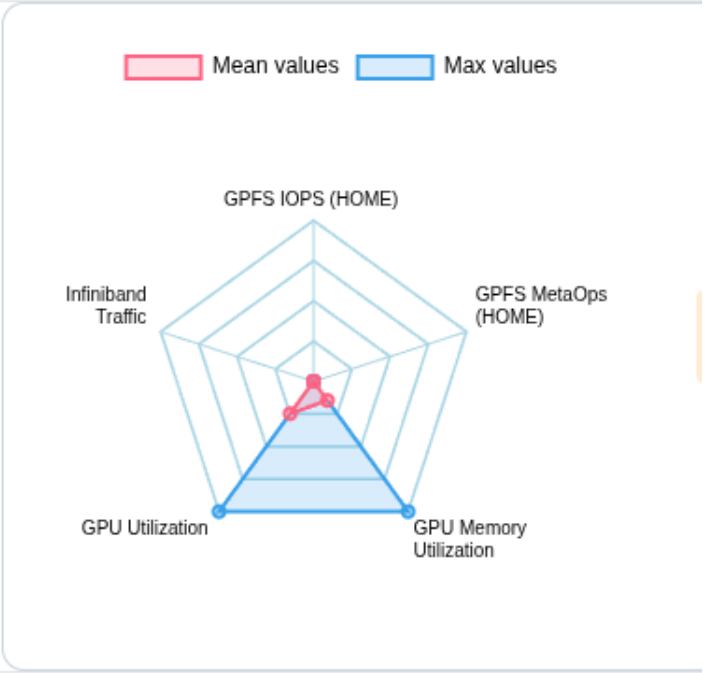
Nodes: 1 GPUs: 4

Start: 9/28/2026, 3:00:14 PM

End: 9/28/2026, 3:57:27 PM

Exit code: 0

No concurrent jobs detected



Question

What additional information do you need before deciding whether this job is using its allocation efficiently?

Exercise 2: Allocation context matters

- A device can be highly utilized while the overall allocation is still inefficient.
- Before interpreting utilization, check how many CPUs, GPUs, and nodes were actually requested.
- For multi-device jobs, compare individual devices rather than relying only on averages.
- For shared resources, interpret node-level metrics cautiously because activity may not belong exclusively to one job.
- Check whether JobMon detected concurrent jobs on the allocated resources.

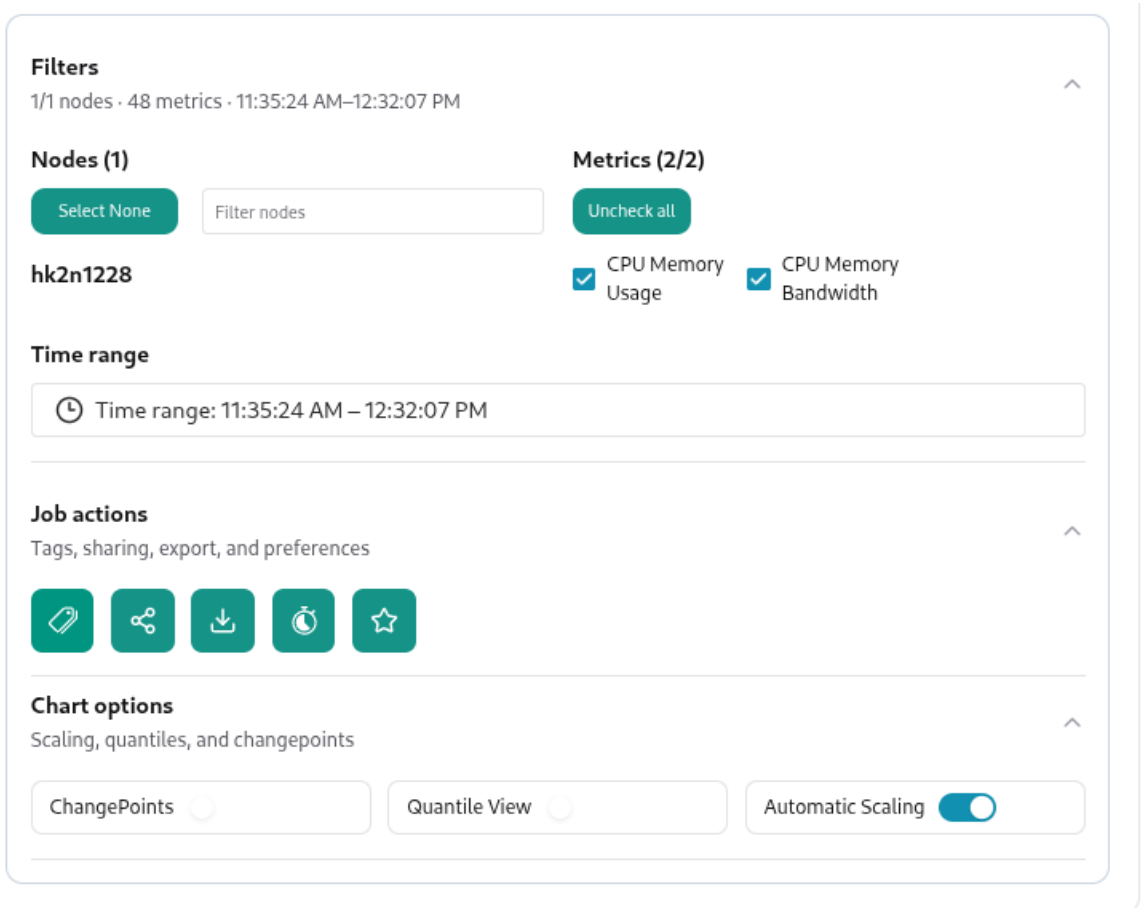
Per-job view

Job view control options

- Select nodes
- Select a time range
- Select metrics
- Adjust the sample rate for long jobs
- Apply per-node aggregation
- Use tags for comparisons

Advanced toggles

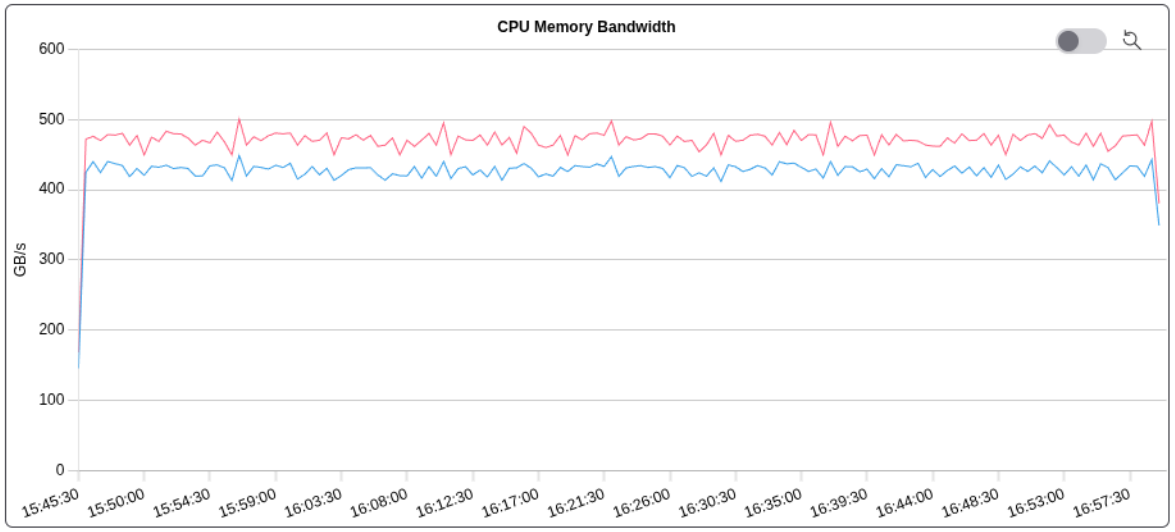
- Automatic scaling
- Quantile view
- Changepoints



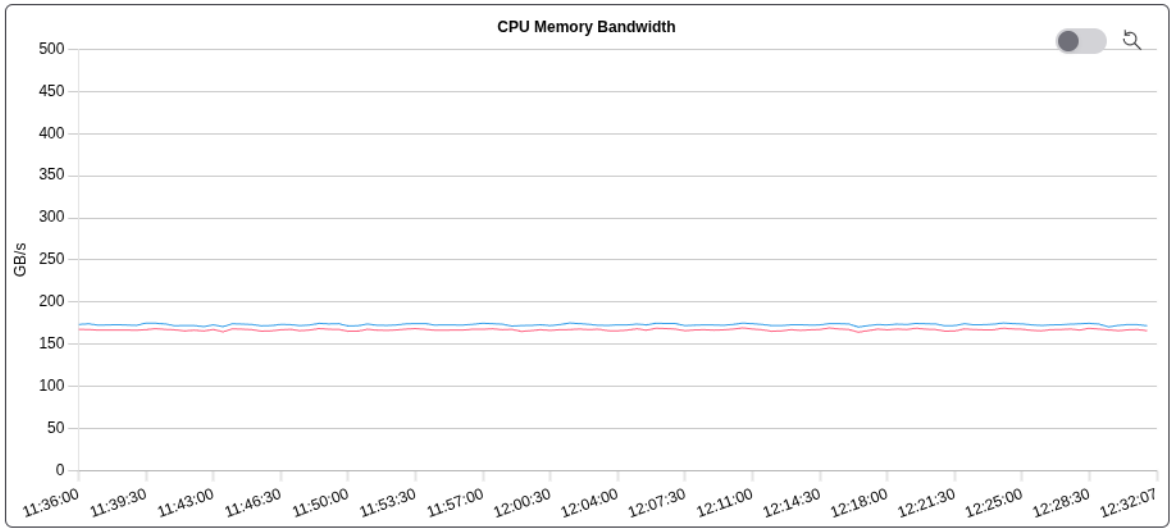
STREAM vs. DGEMM

Memory bandwidth

STREAM



DGEMM



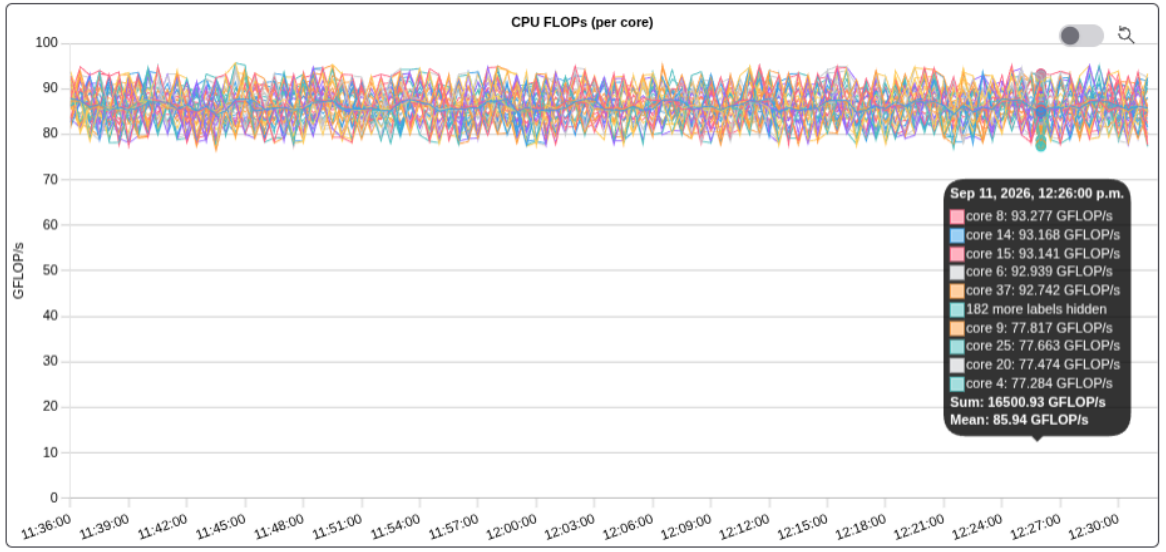
Take-away

STREAM puts sustained pressure on the memory subsystem; DGEMM shows a different memory-access pattern because its primary work is compute-intensive.

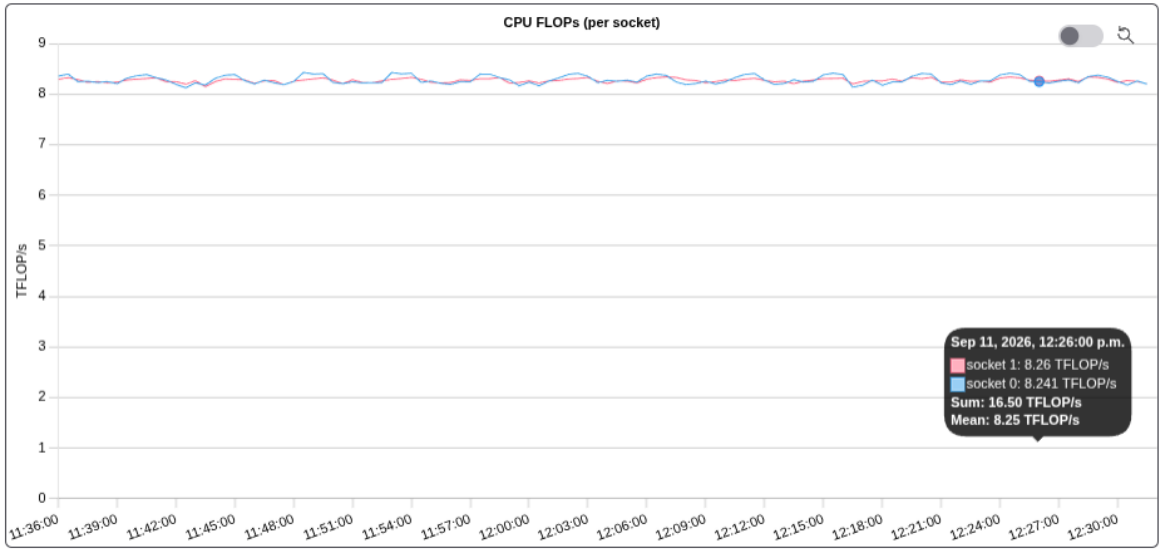
Is work distributed evenly?

DGEMM

Per core



Per socket



- Per-core aggregation helps expose imbalance between CPU cores.
- Per-socket aggregation helps compare sockets and NUMA domains at a higher level.
- The total work does not change; the aggregation changes what imbalance is visible.

Roofline plot

Compute or Memory Bound

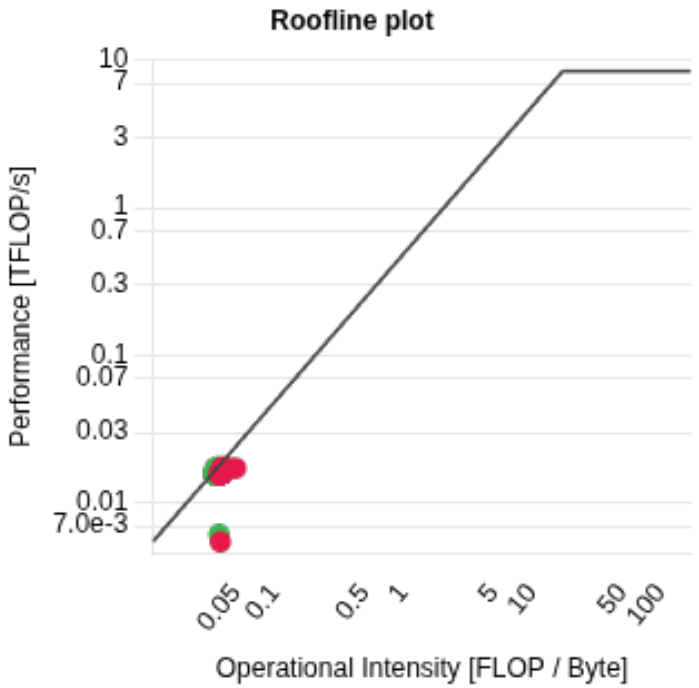
Roofline model

- y-axis: Floating-point operations per second (FLOP/s)
- x-axis: Arithmetic intensity (FLOP/byte)
- Low arithmetic intensity: attainable performance is typically constrained by memory bandwidth
- High arithmetic intensity: attainable performance is typically constrained by processor peak performance

Roofline plot

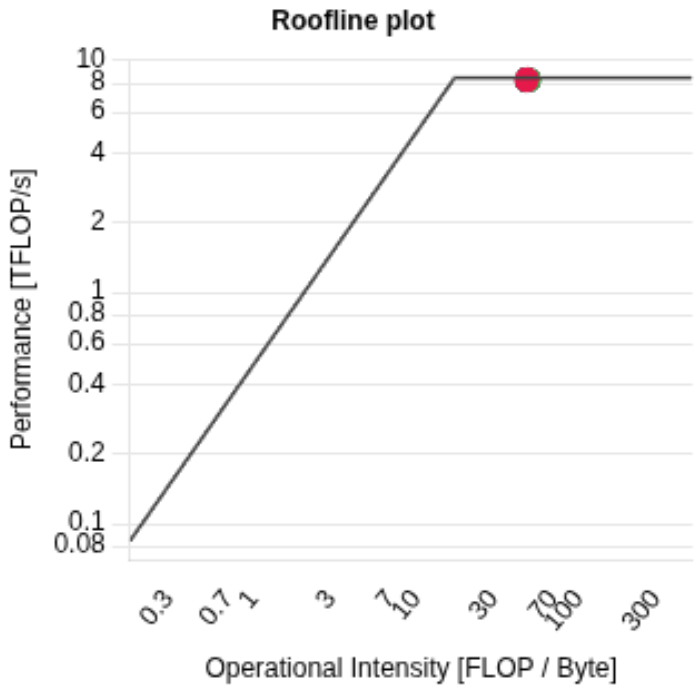
STREAM vs. DGEMM

STREAM



Low arithmetic intensity ⇒ memory-bandwidth regime

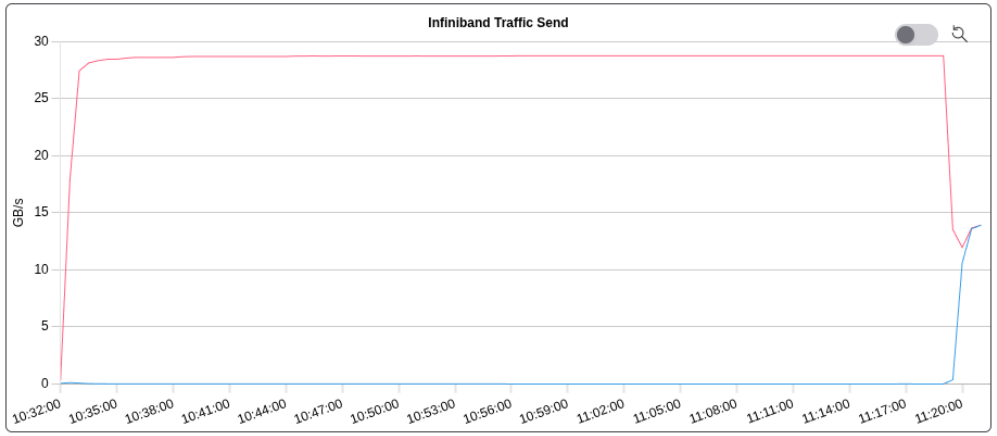
DGEMM

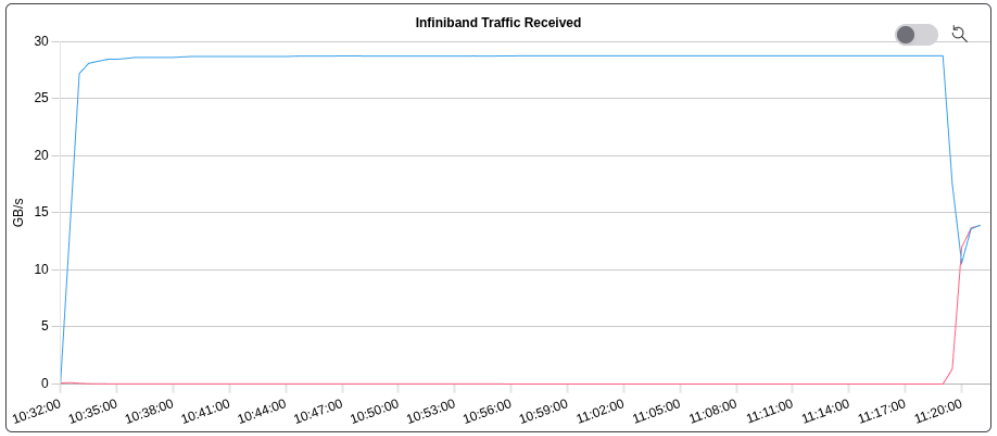


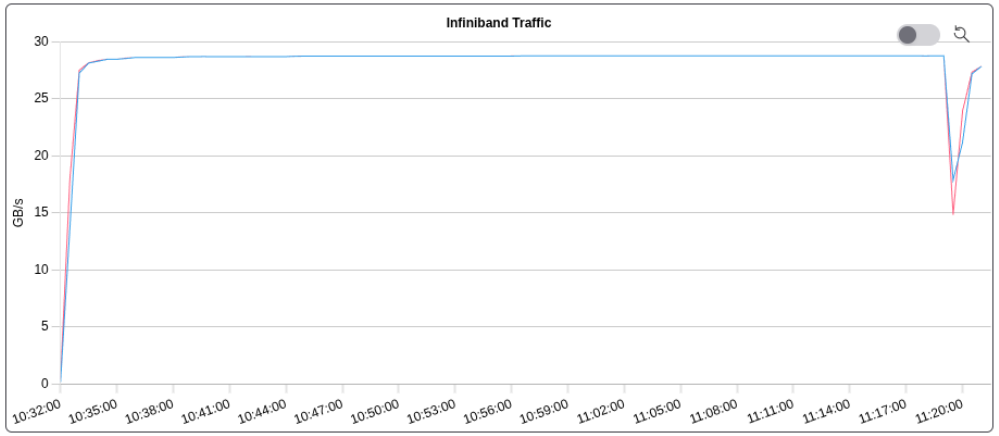
High arithmetic intensity ⇒ compute-performance regime

OSU Benchmark, nodes with separate NICs

InfiniBand Metrics

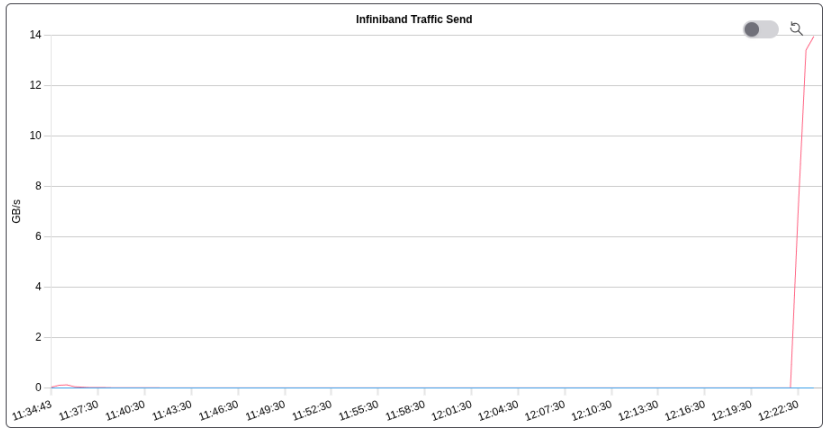


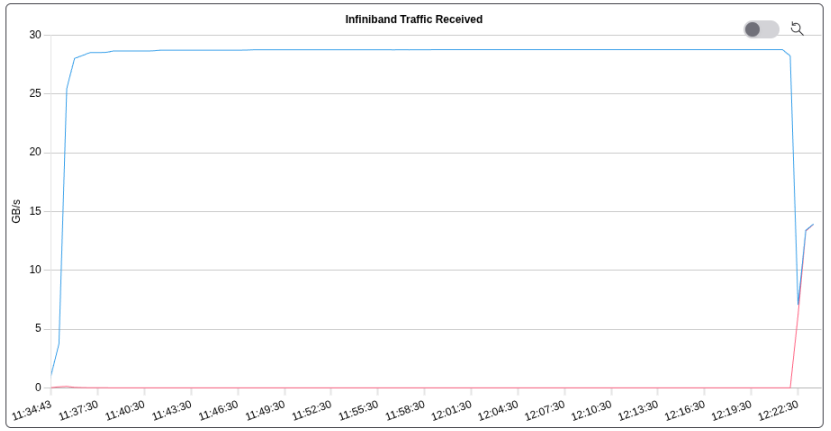


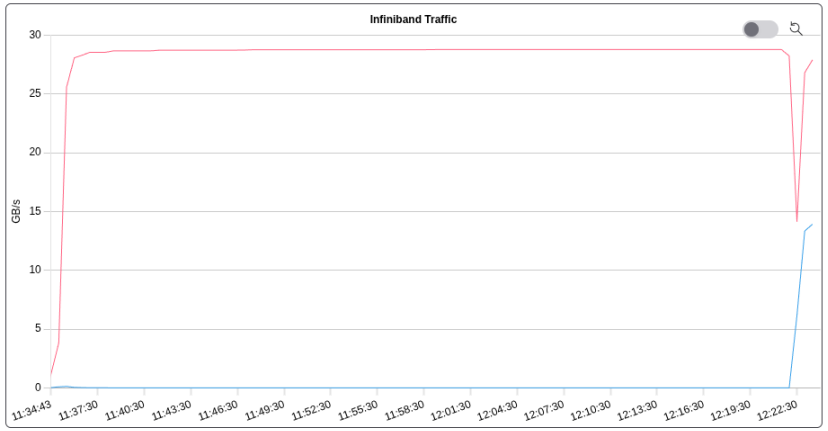


OSU Benchmark, nodes with shared NICs

InfiniBand Metrics

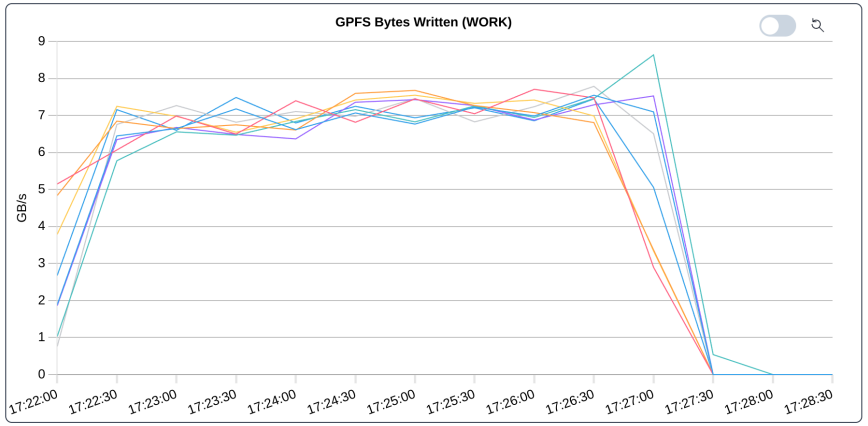


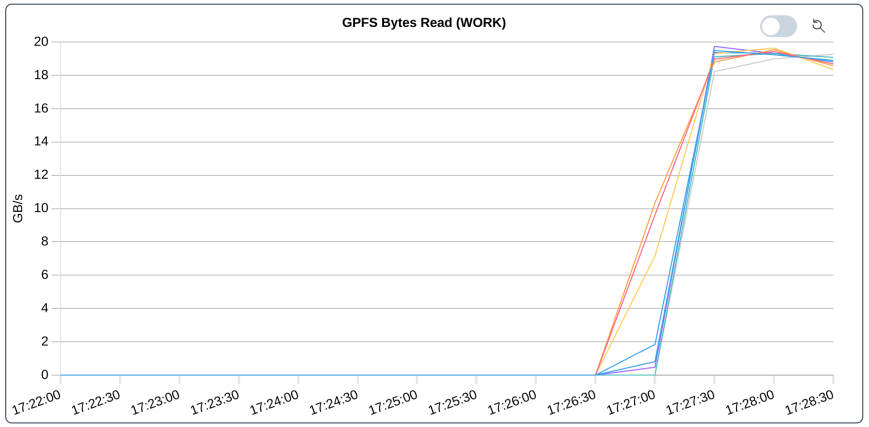


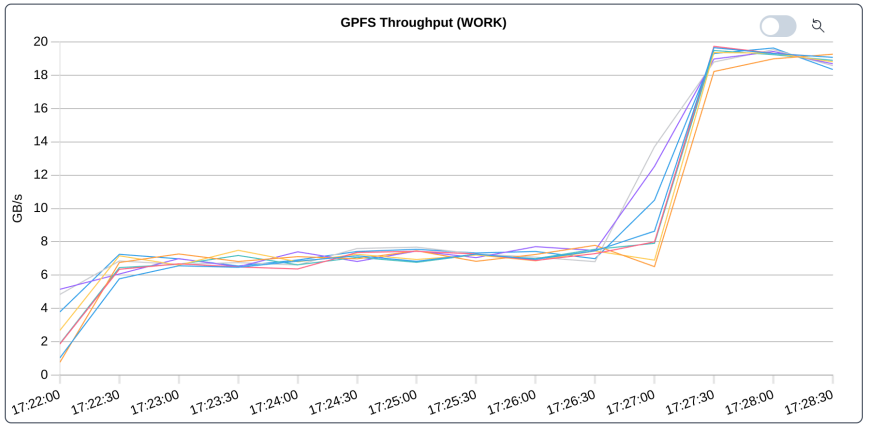


IOR Benchmark

What do read/write phases look like?







Think in timelines, not isolated metrics

Initialization $\rightarrow$ Compute $\rightarrow$ Communication $\rightarrow$ Compute $\rightarrow$ Checkpoint $\rightarrow$ Compute

- Whole-job averages can hide short but important phases.
- A high value matters differently if it lasts 30 seconds or 30 minutes.
- Correlating signals over the same time interval is more informative than reading one plot in isolation.

Question

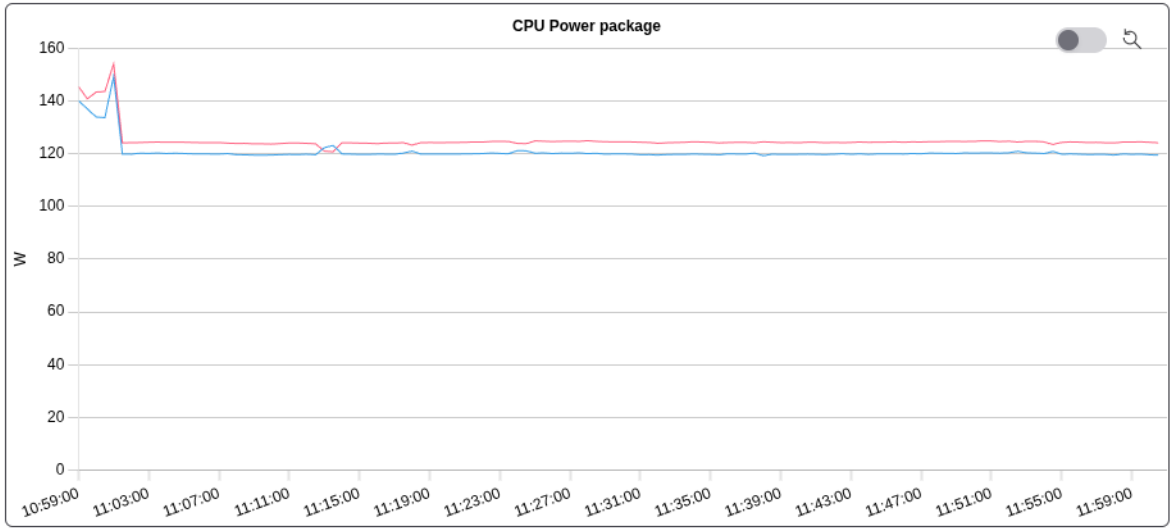
When compute activity drops, what else increases at the same time?

Exercise 3: What phases can you identify?

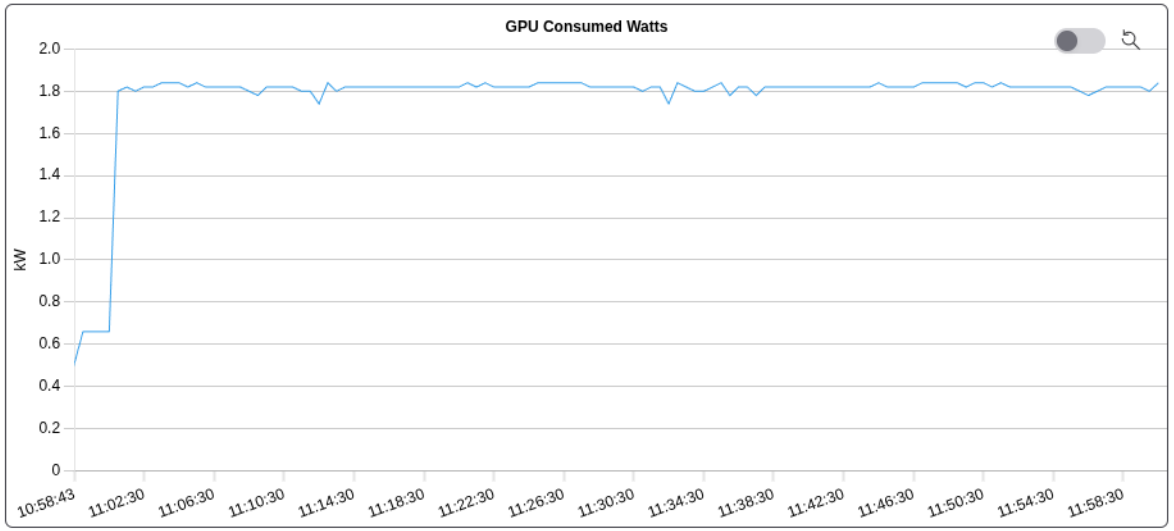
Question

Without knowing how the application is implemented, where do you think execution moves from CPU-heavy work to GPU-heavy work?

CPU package power



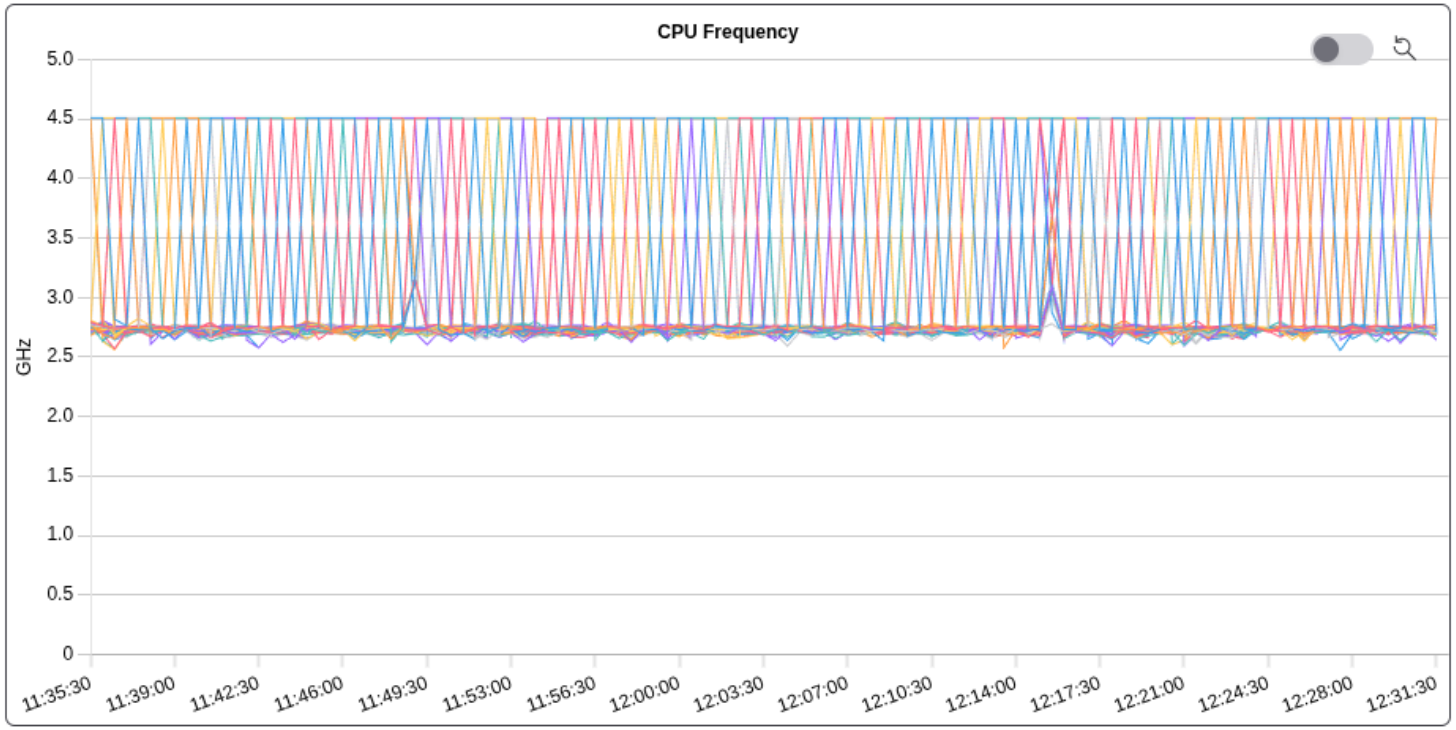
GPU power



Exercise 3: CPU preparation → GPU computation

- The GPU-HPCG example contains a preparation phase executed on CPUs.
- CPU package power is higher during this preparation phase.
- GPU power rises once the GPU computation phase begins.

Problem: Too many lines to interpret

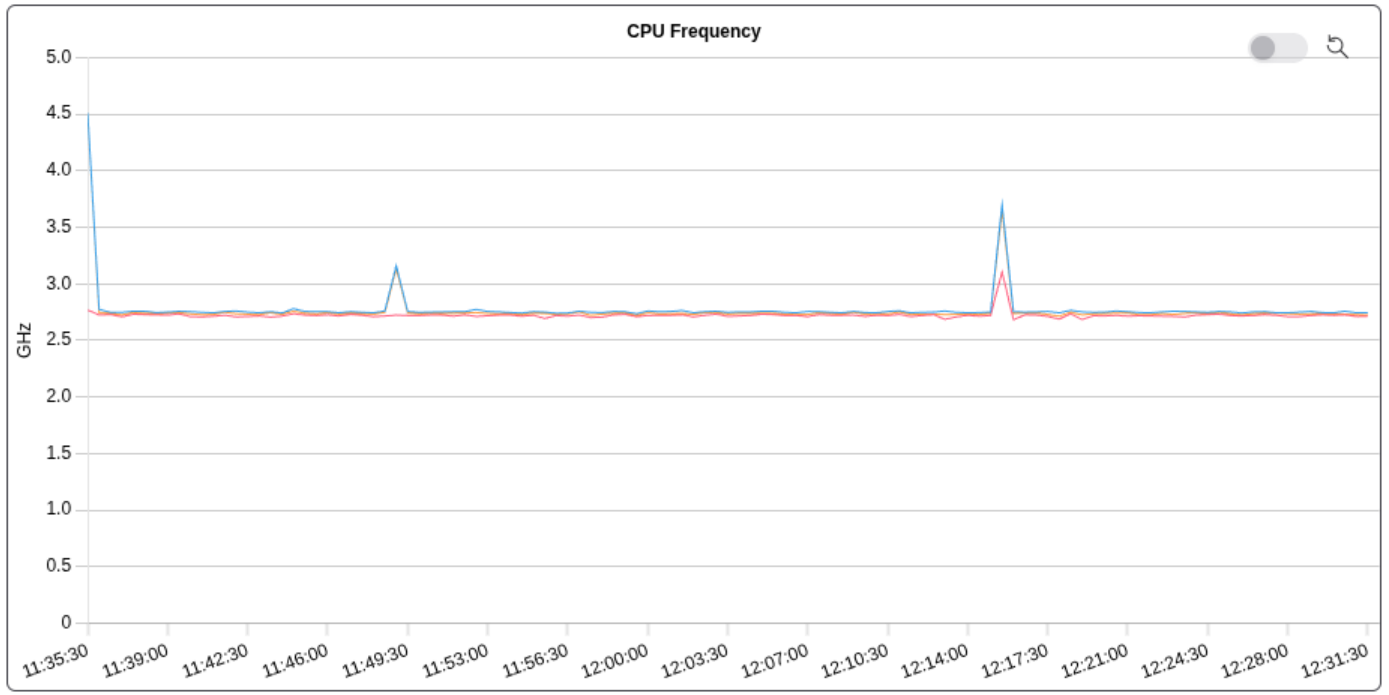


Question

Can you quickly tell whether CPU frequencies are tightly clustered or widely spread?

Quantiles

Summarize behavior without hiding variation

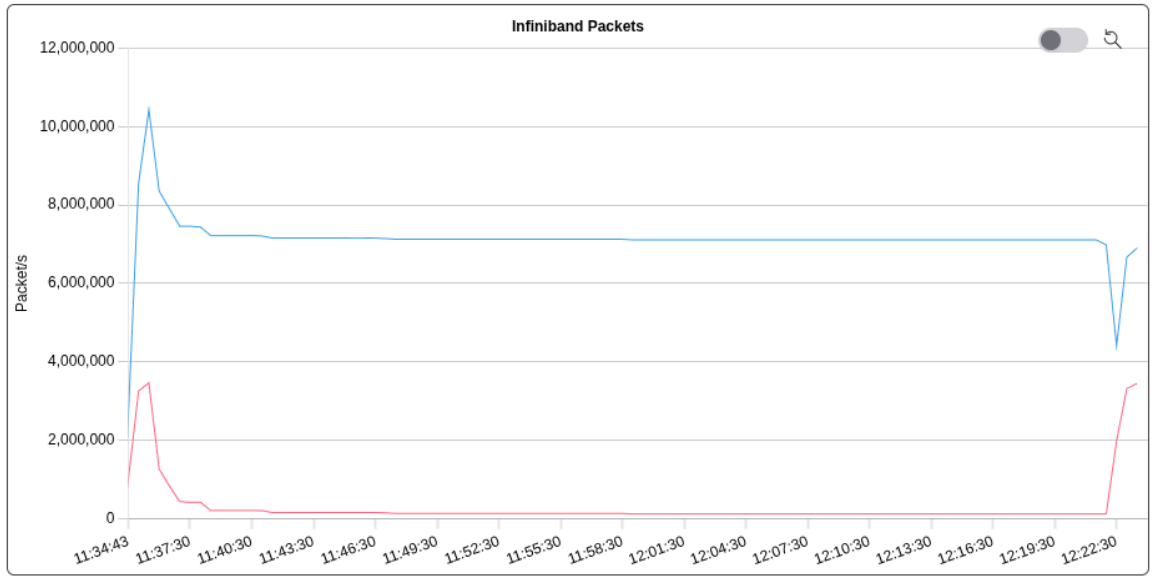


- 25% / 50% / 75% quantiles summarize the distribution.
- The median shows the typical value.
- The distance between lower and upper quantiles shows the spread.

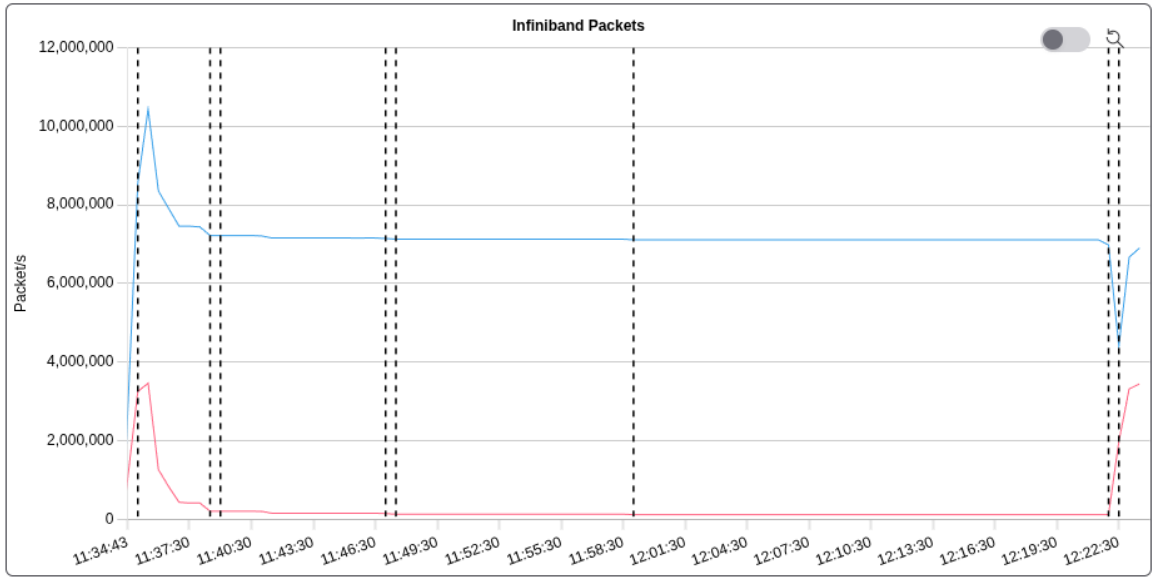
Changepoints

Locating changes in behavior

Changepoints off



Changepoints on



- OMB increases message size, creating distinct performance regimes.
- Changepoints mark transitions between different metric behaviours.

Other JobMon features

- Custom multi-node aggregation functions (e.g. average, sum, maximum)
- Live view of running jobs (time window of 10 minutes)
- Export job data
  - CSV
  - SQLite database
- Tags for grouping related jobs
- Share Jobs with others
- API Endpoints available
- Automatic Job Analysis:
  - Changepoint analyzer - compute changepoints
  - Tag generation analyzers:
    - I/O analyzer - check if I/O metric values exceed certain thresholds
    - Load balance analyzer - check if some of the CPU threads are staying idle
  - Metric stats analyzer - Compute mean/max values for some of the preselected metrics
  - Other analyzers to be added
- Admin only:
  - Cluster wide metrics
  - Cluster wide statistics

Why high level diagnosis is not enough?

- A high metric value is evidence, not automatically a bottleneck diagnosis.
- Whole-job averages can hide short but important phases.
- Correlation between metrics does not by itself establish causation.
- System-level metrics cannot identify the responsible source-code line.
- Very short jobs may not provide enough samples for a meaningful time-series analysis.

JobMon workflow summary

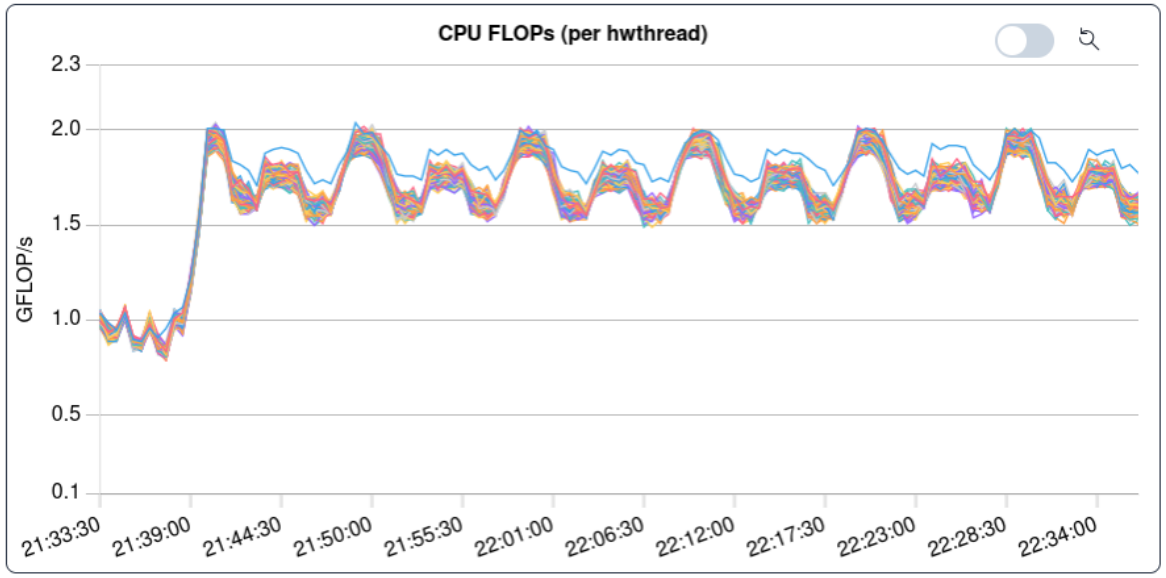
Observe $\rightarrow$ Correlate $\rightarrow$ Hypothesize $\rightarrow$ Change $\rightarrow$ Compare

1. Start with allocation and metadata.
2. Use the spider plot for diagnosis.
3. Read the job as a timeline.
4. Correlate compute with memory, communication, and I/O.
5. Check balance across resources.
6. Compare against another run.

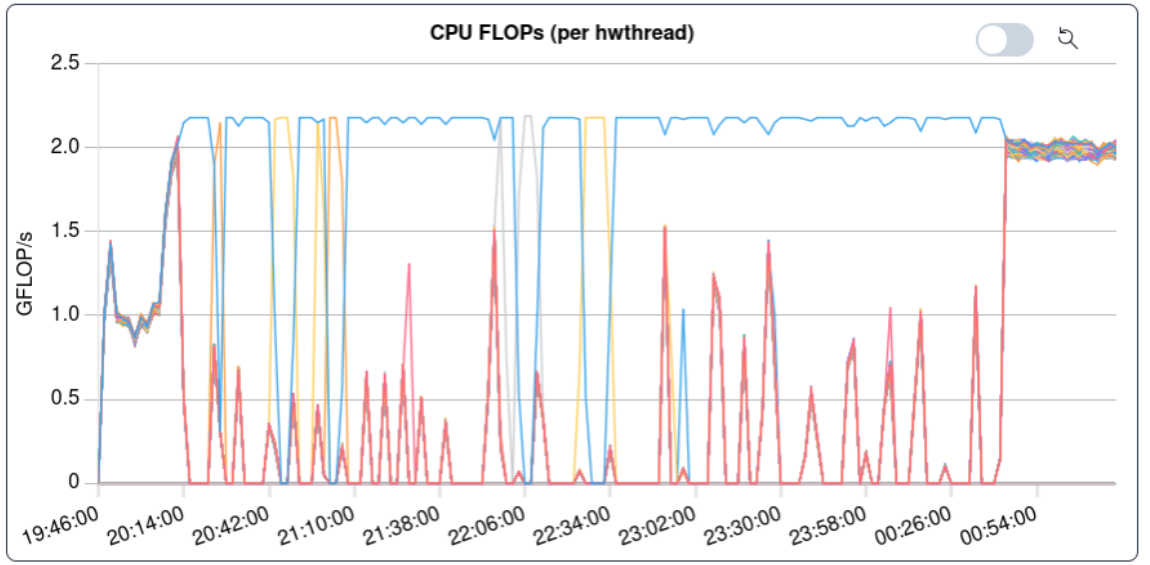
JobMon in Practice: PMFRG Julia Code

Investigating inconsistent performance

Fast Run



Slow Run



- **Problem:** Runtime varied by a factor of 2–3 between otherwise comparable runs.
- **Observation:** JobMon revealed differences in resource utilization between fast and slow runs.
- **Root cause:** Bug in the parallelization of the main computation loop.

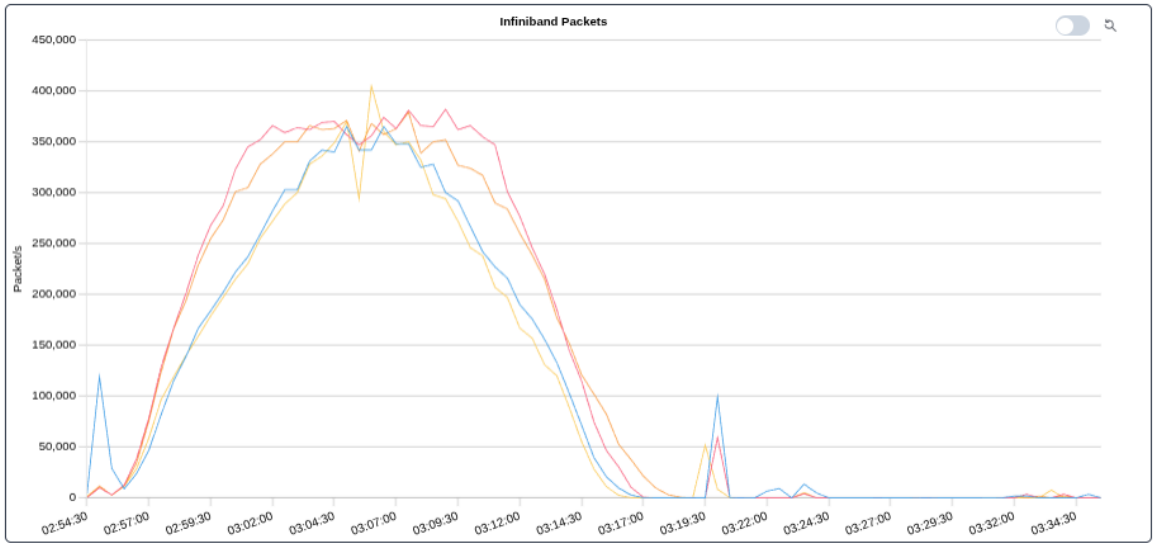
JobMon in Practice: IFOS3D

Identifying a bug in Hessian accumulation

Before Fix



After Fix



- **Problem:** Significant runtime variations between comparable runs.
- **Observation:** JobMon revealed unexpected behavior during Hessian accumulation.
- **Root cause:** Bug in the Hessian accumulation loop across MPI ranks.

Final challenge: Investigate your own jobs in case you have any

Keep in mind the requirements

Question

Use the workflow, not intuition.

1. What resources were allocated?
2. What does the overview suggest?
3. Which time interval is interesting?
4. What happens to compute activity there?
5. Do memory, interconnect, or filesystem signals change at the same time?
6. Are all nodes/devices behaving similarly?
7. State one hypothesis and one metric that could falsify it.

Resources and questions

Reference material

- User documentation: NHR@KIT JobMon documentation
- JobMon: hk2-jobmon.scc.kit.edu

Questions?