My Jobs are Slow? What's Wrong?

Robert Schade

NHR/GCS AI Q&A Mini Tutorial

Paderborn University, Germany
Paderborn Center for Parallel Computing

NATIONALES HOCHLEISTUNGS RECHNEN

Paderborn Center for Parallel Computing

Plan for Today

Situation: You need to do some calculations (e.g. training, inference, data generation,...) with an existing program/framework on an HPC system. You have a setup for a compute job that runs without obvious errors.

Not in scope for today:

- Code/framework-specific options/issues
- Code modifications/programming
- In-depth performance profiling and engineering

For in-depth events/workshops/... please have a look at

- the HPC Calendar: https://veranstaltungen.hpc-in-deutschland.de/en-us/
- NHR HPC events https://www.nhr-verein.de/events-trainings/all-events-trainings/
- GCS HPC events https://www.gauss-centre.eu/trainingsworkshops

Plan for Today

Situation:

You need to do some calculations (e.g. training, inference, data generation,...) with an existing program/framework on an HPC system.
You have a setup for a compute job that runs without obvious errors.

Practical Guide/Steps:

1. Check if there is a known technical issue with the system
2. How to measure performance?
3. Find out what to expect
4. Determine and understand the baseline performance
5. Checks and fixes for common problems
   - Job allocation, threading, pinning
   - bad/old libraries/framworks

Disclaimer:

I will be explaining most things with examples/screenshots/tools on our HPC systems in Paderborn (NHR center PC2, HPC systems Noctua 2 and Otus).

Other centers have the same or similar tools and possibilities available.

Consult the system documentation or contact the HPC support for details.

The content is geared towards beginners.

Prerequisite: Define what you want to do

Example today: batched offline LLM Inference for tagging a corpus of text snippets

Number of text snippets = several millions

Tokens per text snippet = ~120 avg., max 10000

To keep is simple:

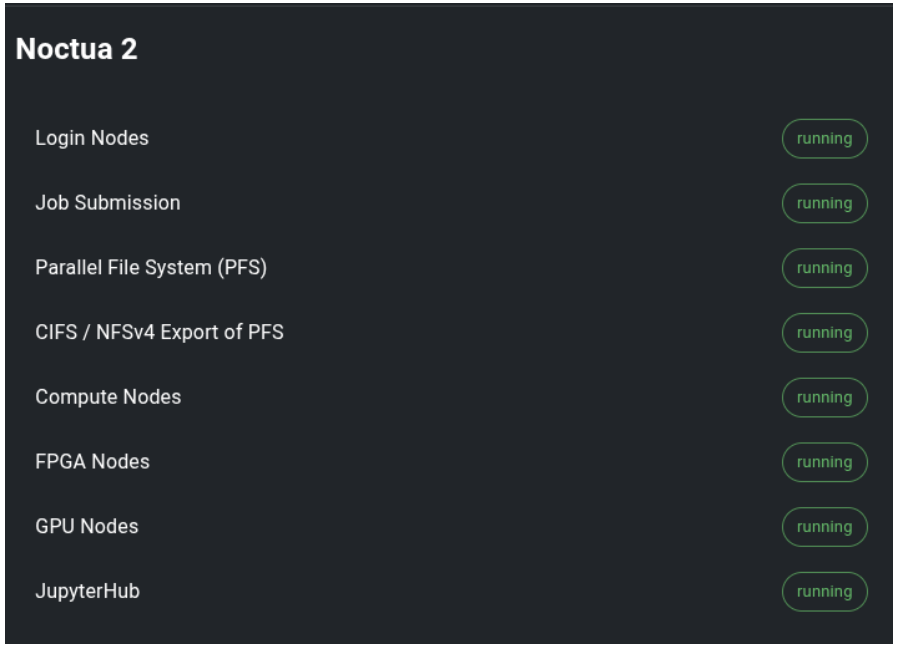
- Inference with vllm
- using a single GPU
- model Meta Llama-3.1-8B-instruct

Note: This is just a constructed simple example, in reality one would do it differently, i.e. different model,...

1. Check if there are known technical issues with the system

Check with our system status at (e.g. https://portal.pc2.uni-paderborn.de/)

- Look out for known issues that could affect your jobs:
- Issues with
  - **the (parallel) file system** could affect jobs with file IO
  - **electrical power/colling** could affect the CPU and GPU performance
  - **the interconnect** (e.g. Infiniband) could affect the communication performance in jobs



2. How to Measure Performance

What is an easily measurable performance number in your case?

- Total wall-time of a job
  - Probably be very long, potentially very wasteful
- Wall-time per iteration or for a set of iterations/part of the data set
  - Need to check if all iterations take the same time, or use the first N iterations as an alternative
- Some performance number your code outputs
  - Depends on the code/situation/use case
  - Sometime hard to find out what the performance number actually means

Important: choose so that an test run doesn't take too long (~ a few minutes)

2. How to Measure Performance

Be careful about different phases of the job, e.g., intialization/read-in/preprocessing/write-out

-> Wall times of a job for a part of the dataset can be misleading

-> Be careful about which part of the job you time.

Example:

With **one** GPU:

With **two** GPUs:

With **four** GPUs:

Using 4x the resources only gets you much less than 4 times the speedup.

(Assuming good scaling with number of GPUs. There are also ways to accelerate/parallelize IO depending on the workload/use case.)

2. How to Measure Performance

What is an easily measurable performance metric in your case?

Example today: batched offline LLM Inference with VLLM

- Total wall-time of a complete job
  - can be many days or weeks
- Wall-time per iteration or for a set of iterations/part of the data set
  - Probably misleading due to initialization
- Some performance number your code outputs
  - Inference: e.g. time-per-output-token (TPOT), time-to-first-token (TTFT)
  - Training: e.g. time per training iteration

-> choose here the **throughput of texts processed per second**

3. Find out what to expect

Which runtime do you expect and why?

Possible information sources:

1. Experience with similar calculations on other systems
   - Can be misleading sometimes
2. Results of benchmarks
   - Often results for benchmarks for a related calculation and related hardware can be found
   - Or one runs the benchmark
3. Theoretical Estimates
   - most reliable if done properly
   - In-depth knowledge about algorithms, implementation, and relevant bottlenecks is required

Important questions?

- Which parameters of your input have the greatest influence on the runtime?

3. Find out what to expect

Example today: batched offline LLM Inference with VLLM

Parameters that influence performance most:

- Size of model, Quantization/data type
- GPU type
- Number of concurrent requests, request type/length, batch size,...

	model	prec	GPU type	Requests	Performance Information
Our Example	Llama-3.1-8b-instruct	BF16	NVIDIA A100-40GB SXM4	our dataset	
https://blog.vllm.ai/2024/09/05/perf-update.html	Llama-3-8b	BF16	NVIDIA A100	Prefill-heavy dataset (~462 input and 16 output token)	TPOT ~ 500 ms, ~28 Req/s, ~13400 t/s
https://blog.vllm.ai/2024/09/05/perf-update.html	Llama-3-8b	BF16	NVIDIA H100	Prefill-heavy dataset	TPOT ~ 230 ms, ~60 req/s, ~ 28700 t/s
https://www.databasemart.com/blog/vllm-gpu-benchmark-a100-80gb	DeepSeek-R1-Distill-LLama-8b	BF16	NVIDIA A100-80GB	(100 input token, 600 output token, 50 requests)	~ 5800 t/s

4. Determine and understand the baseline performance

Console/log output

Problem: In a compute job, the output to the standard output is buffered
Workaround: `python -u …` for unbuffered output

Note: you can also redirect the standard output with either

- `python3 -u tag.py > baseline.log`

or

- `#SBATCH –output baseline.log`

4. Determine and understand the baseline performance

Monitoring the job live

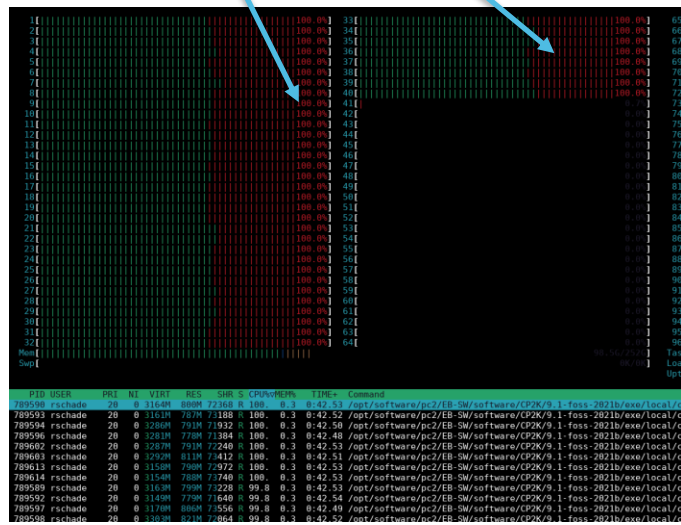
On most HPC systems you can ssh to the nodes where you have a job running.

- CPU-usage: `htop (htop -u [user name])`
  - Top CPU bars
    - green is user time (actual work, good)
    - red is system time (waiting for system calls and IO, bad)

Watch out for:

- high system time of processes (e.g. due to slow IO)
  - RCHAR (read bytes), WCHAR (write bytes), SYSCR (read sys calls), SYSCW (write sys calls)
- how many threads/processes are spawned?
  - Check threads in htop (processes are in white, threads in green)
  - Check that OMP_NUM_THREADS and --cpus-per-task (need to be equal)
- Are the processes/threads properly distributed to the allocated cores?
  - PROCESSOR
  - Check that each MPI-rank/tasks is bound to as many cpu-cores as threads are requested

Significant system time on the cpu-cores of the job!

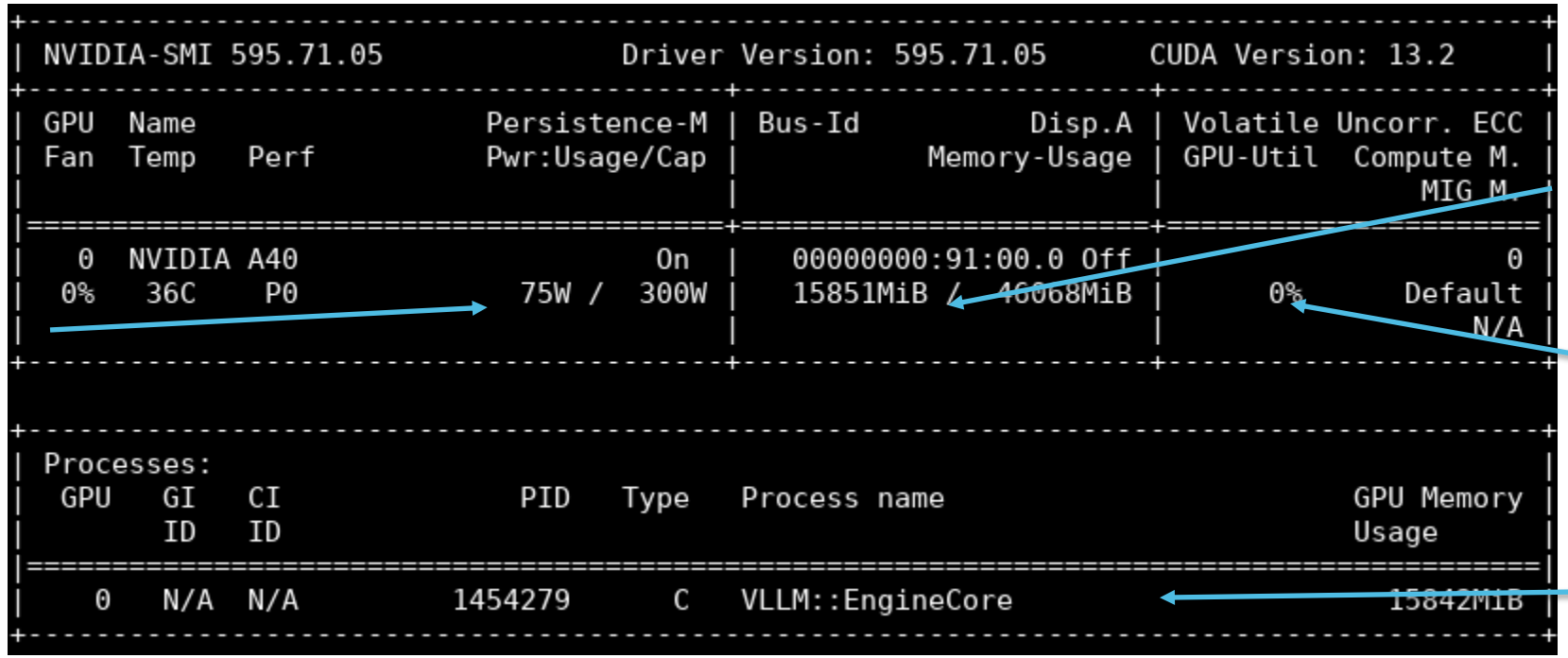


4. Determine and understand the baseline performance

Monitoring the job live

On most HPC systems you can ssh to the nodes where you have a job running.

- CPU-usage: `htop (htop –u [user name])`
- GPU-usage: `nvidia-smi, rocm-smi`



Power usage of GPU

memory usage of GPU

GPU utilization (which portion of time is a kernel active)

Processes with a GPU context

4. Determine and understand the baseline performance

Monitoring the job live

On most HPC systems you can ssh to the nodes where you have a job running.

- CPU-usage: `htop (htop -u [user name])`
- GPU-usage: `nvidia-smi, rocm-smi`

Watch out for:

- low GPU utilization (should be >~80%)
- unequal distribution of load to GPUs
- low GPU power usage (low power usage indicates overall low GPU usage)
- high GPU memory usage (danger of job failing with out-of-memory errors)
- number of processes per GPU (should be 1)

4. Determine and understand the baseline performance

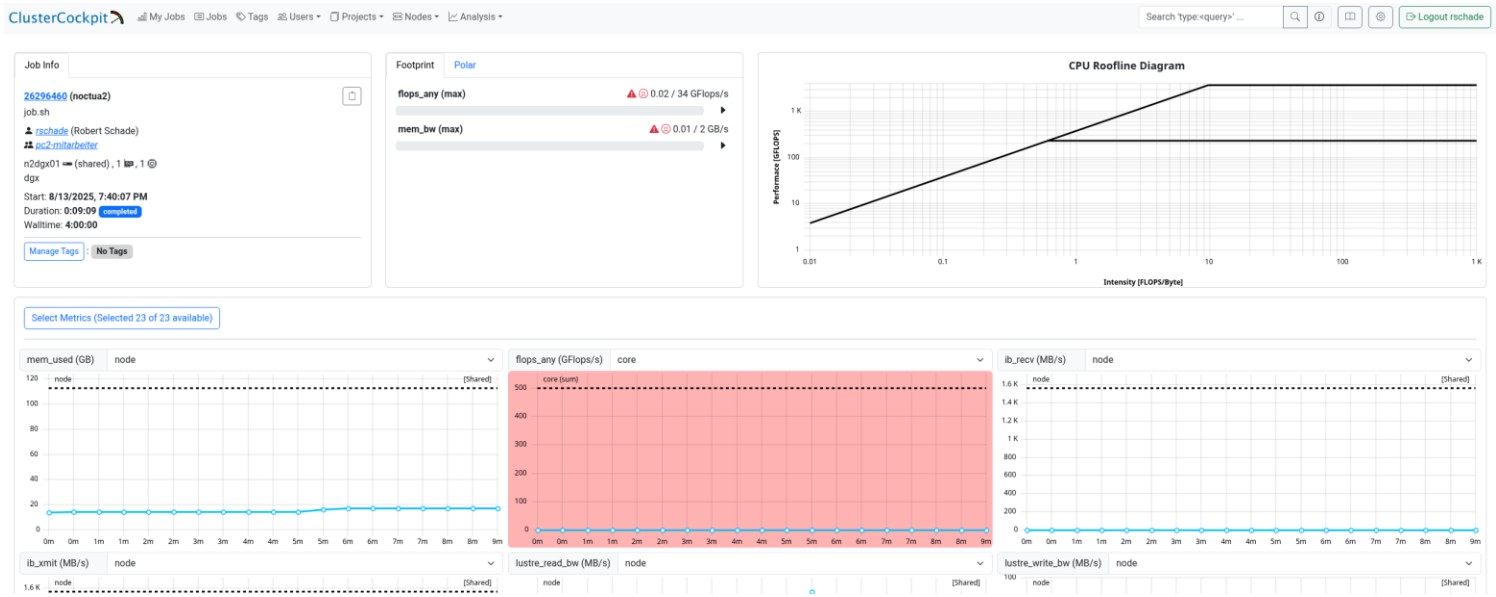
Investigating the job in a job-monitoring system

Most HPC centers offer a job-monitoring system (e.g. Cluster Cockpit, PIKA, LLview)

- automatically records certain helpful metrics during the execution of a compute job
- accessible via web interfaces

Include a variety of metrics:

- CPU usage/throughput/clocks
- Memory usage/bandwidths
- IO metrics
- Communication/Interconnect metrics
- GPU metrics



4. Determine and understand the baseline performance

Profiling Tools

Examples:

- Python profiler
- GPU-profiler, e.g. Nsight Systems

For in-depth events/workshops/... please have a look at

- the HPC Calendar: https://veranstaltungen.hpc-in-deutschland.de/en-us/
- NHR HPC events https://www.nhr-verein.de/events-trainings/all-events-trainings/
- GCS HPC events https://www.gauss-centre.eu/trainingsworkshops

4. Determine and understand the baseline performance

A few test job with different sizes:

- Insights about size-independent overheads (e.g. initialization)

Example:

Information from vllm output

#texts	Job wall time/s	inference time/s	Input Token/s	Output token/s	Job ID
1000	130	23.1	9526.83	1964.43	26296832
5000	295	99.7	10412.06	2124.64	26296833
10000	544	203.2	10116.39	2052.71	26296834
20000	1054	401.2	10186.52	2071.89	26296845

Large discrepancy between job time and actual inference time!

Similar to the estimated results of ~13000 t/s!

Make note of job IDs so that you can later find the jobs again easily

Possible reasons: (de)initialization overheads, IO

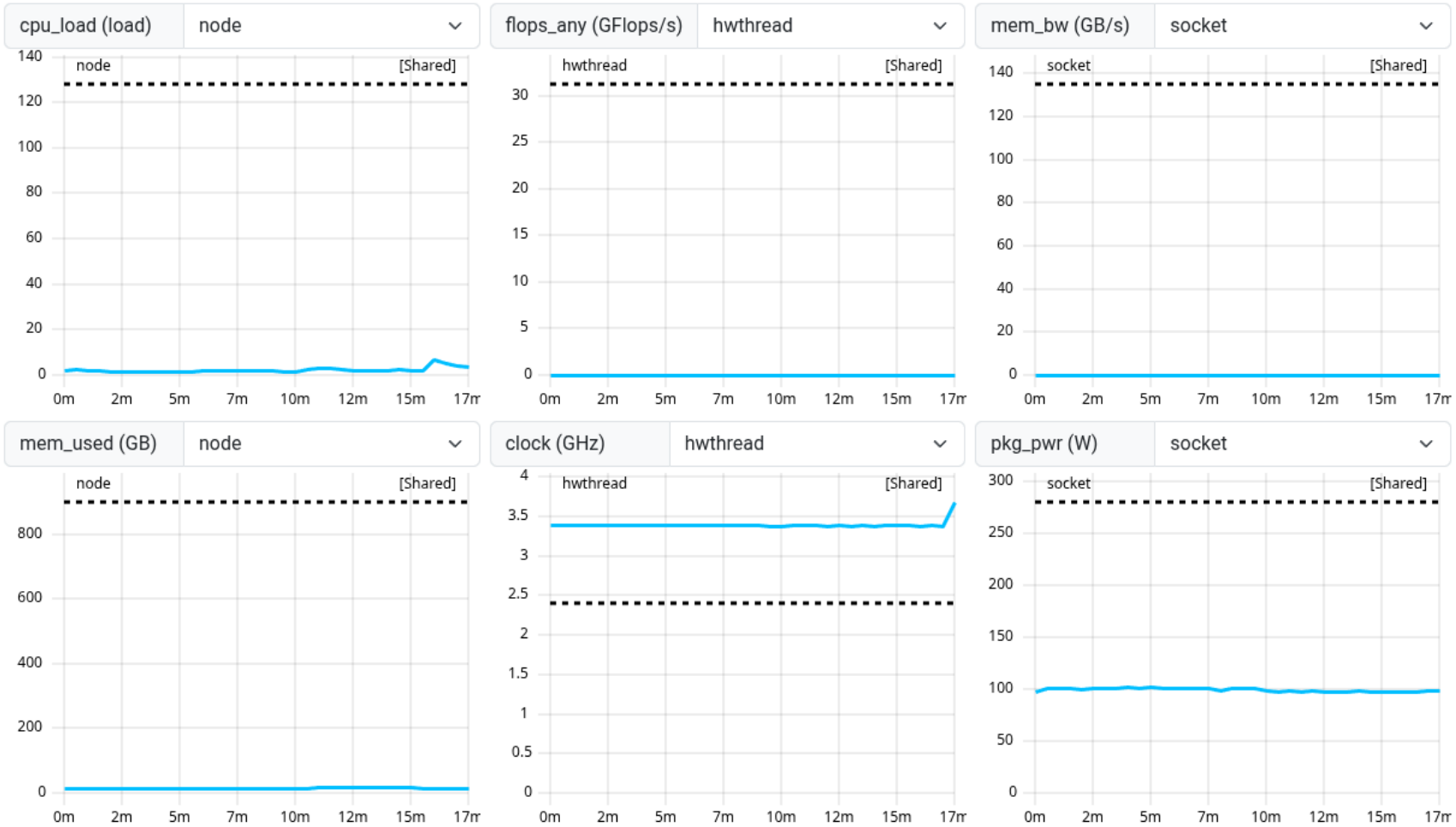
5. Checks for common problems

- job allocation: Do you have the resources allocated that you wanted to allocate?
- In the job:
  - Initialization overhead
  - number and pinning of processes/threads,...
  - GPU usage: utilization, memory usage, processes using the GPU,...
  - File IO: metadata operations (e.g. open/close/....), IO bandwidth,...

Demo with example jobs

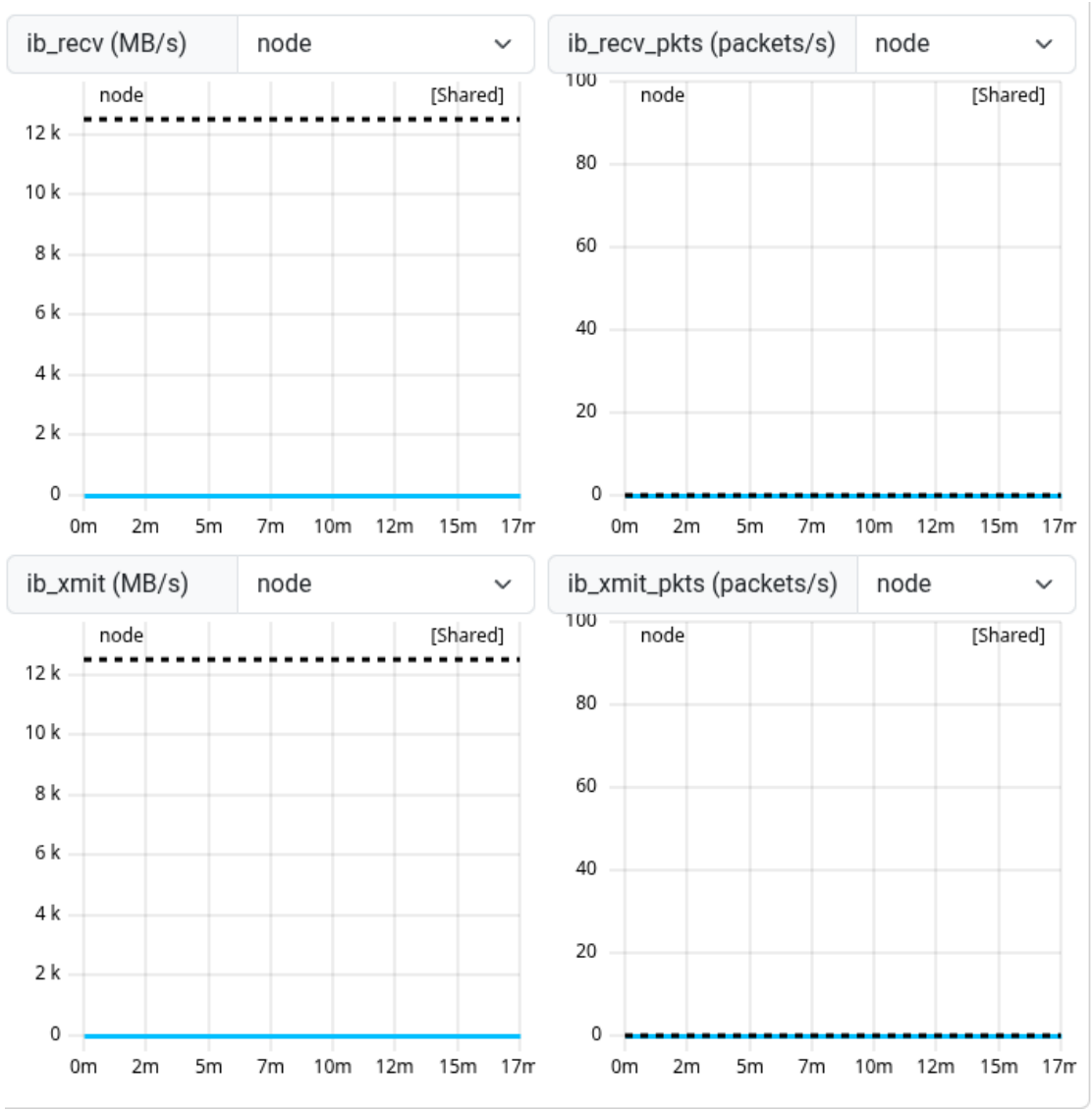
5. Checks for common problems: CPU metrics

CPU load can indicate threading issues



-> every thing is low, because CPU isn't supposed to do much/anything

5. Checks for common problems: Interconnect metrics



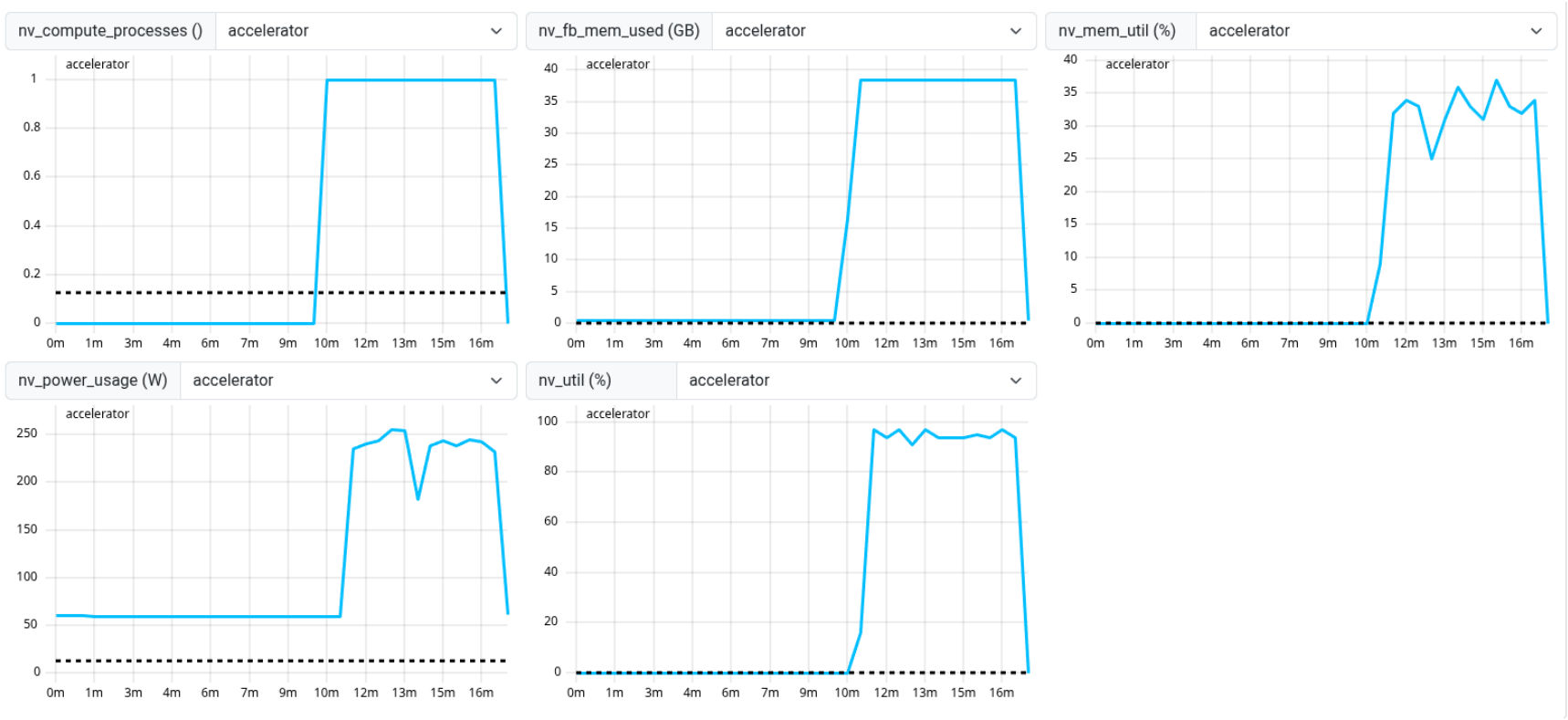
-> no relevant communication via interconnect (infiniband) to other nodes because we are only using one

5. Checks for common problems: GPU metrics

nv_compute_processes:
processes using the GPU,
should be 1

nv_fb_mem_used:
GPU memory usage

nv_mem_util:
GPU memory interface utilization,
should be > 20%



nv_power_usage:
GPU power usage

nv_util:
GPU utilization, should be > 80%

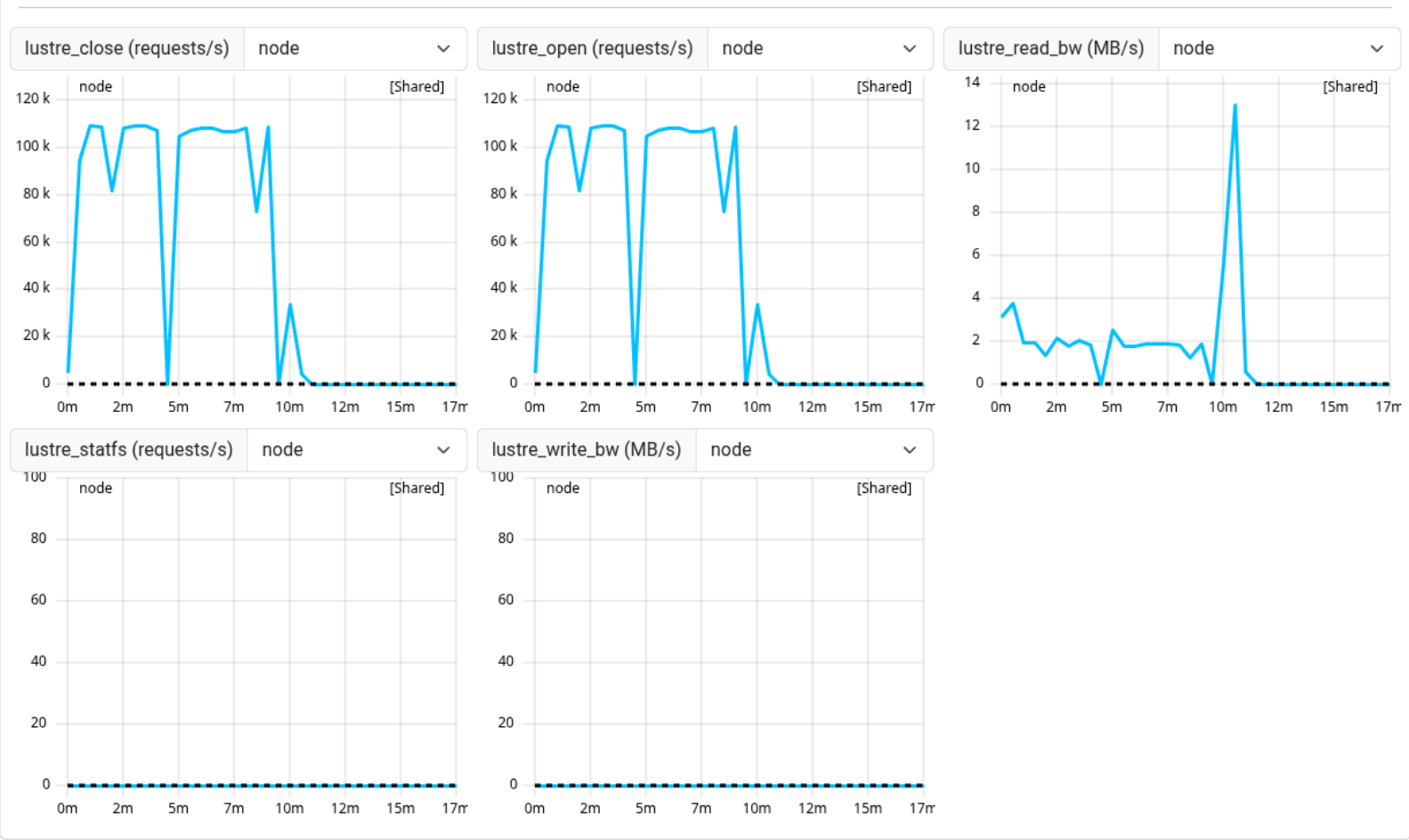
-> GPU is idle during the first part of job !!!!

5. Checks for common problems: IO metrics

File closings

File opens

read bandwidth



File stats (e.g. ls)

Write bandwidth

-> the first part of the job has heavy file IO!!!

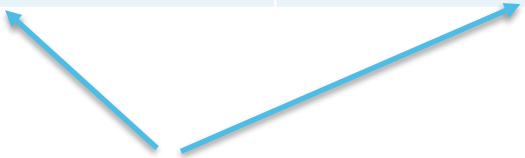
5. Checks for common problems

Example:

- Long readin times: improve file IO
- CPU core load>1, many threads present:
  Allocating more than one CPU-core and setting thread number

```
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
```

#texts	Job wall time [s]	inference time/s	Input token/s	Output token/s
1000	130 -> 82	23.1 -> 22.0	9526.83 -> 10242.64	1964.43 -> 2066.22
5000	295 -> 162	99.7 -> 99.0	10412.06 -> 10357.16	2124.64 -> 2111.01
10000	544 -> 269	203.2 -> 202.5	10116.39 -> 10188.12	2052.71 -> 2063.30
20000	1054 -> 466	401.2 -> 400.3	10186.52 -> 10215.64	2071.89 -> 2076.34



Only constant overhead (~60s) left (vllm initialization, e.g., model loading)

Summary

Practical Guide/Steps:

1. Check if there is a known technical issue with the system
2. How to measure performance?
3. Find out what to expect
4. Determine and understand the baseline performance
5. Checks for common problems