IO for Large Language Models and secured AI workflows

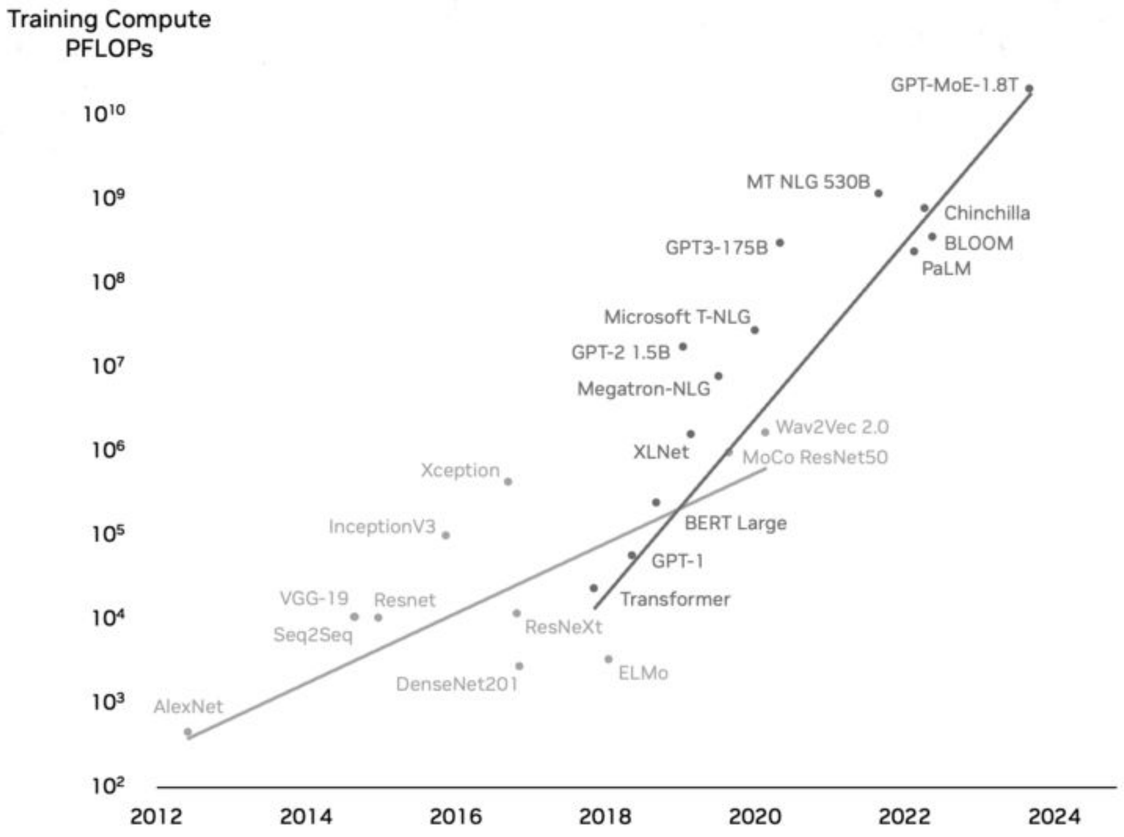
Aurelien Degremont, Nathan Dauchy

LUG'24 - May 7-8, 2024

Introduction

- Generative AI has become well known since ChatGPT chatbot appeared at the end of 2022.
- Large Language Model (LLM) applications are at the core of such capabilities.
- Models have been getting bigger and more complex for several years.

- Compute, Storage, and IO patterns for LLM training are closer to “HPC” than single node ML AI or inference workloads.
- Need for large scale performance platforms with performance storage.
- Need to integrate in more secure environments while sustaining scalability.



Our compute platform

A smaller version of the Eos DGX AI supercomputer

- H100 SuperPOD Deployment
  - #9 on Nov’23 TOP500 (121.40 PF)
  - 576 NVIDIA DGX systems, each with eight H100 GPUs
  - Quantum-2 NDR InfiniBand networking, with separate Compute (8-rail) and Storage fabrics (2-rail)
- Supports wide community of users
  - Early hardware QA
  - Scale testing and CI for software
  - Large-scale AI training
- Our system replicates all the key hardware, design features, and network topologies for ongoing research and testing.

(It's just a little smaller ☺️)

The attached storage platform

DDN EXAScaler Lustre configuration used for these experiments

- 12 DDN AI400X2 appliances. Each with:
  - 24x NVMe drives
  - 8x 200Gb/s IB links
  - 4x Lustre Server VMs
  - 2x MDTs (24 total in the FS, using DNE with automatic round-robin to depth of 4)
  - 8x OSTs (96 total in the FS, using PFL default striping)
- Lustre 2.14.0 with many additional patches, on clients and servers
- PCC-RO available with user flag per Slurm job
- Project Quotas
- Subdirectory Mounts
  - helps with dataset management and access controls
- Kerberos for enhanced security
  - *(more on this later!)*

~1TB/s peak bandwidth

(NOT the bottleneck for our example runs!)

LLM setup with Megatron-LM

Open Source LLM framework

- We are focusing on a training workload.
- We use the Megatron-LM to run our LLM model (https://github.com/NVIDIA/Megatron-LM).
- The previous LLM model presented in the past (LUG2021) has 13B parameters.
- We extended our work with a much larger model and dataset (340B, 8T[1]) to see how I/O evolves in the latest setup.
- Model configuration run was 340B, with:
  - tensor parallelism, pipeline parallelism and data parallelism allowing scalability to the 10k GPU range.

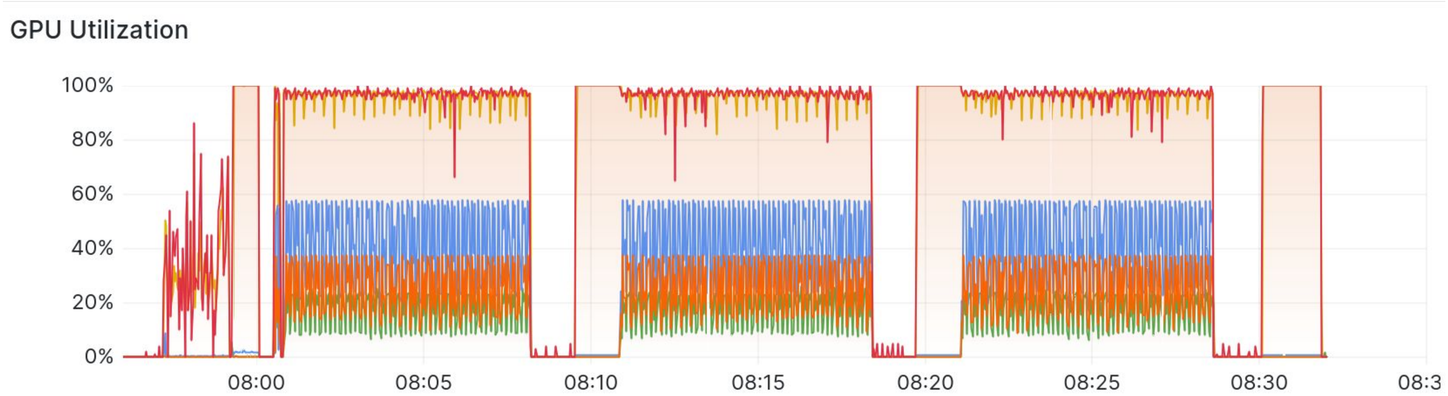
[1] Parmar, Jupinder, et al. "Nemotron-4 15B Technical Report." *arXiv preprint arXiv:2402.16819* (2024).

Workload patterns

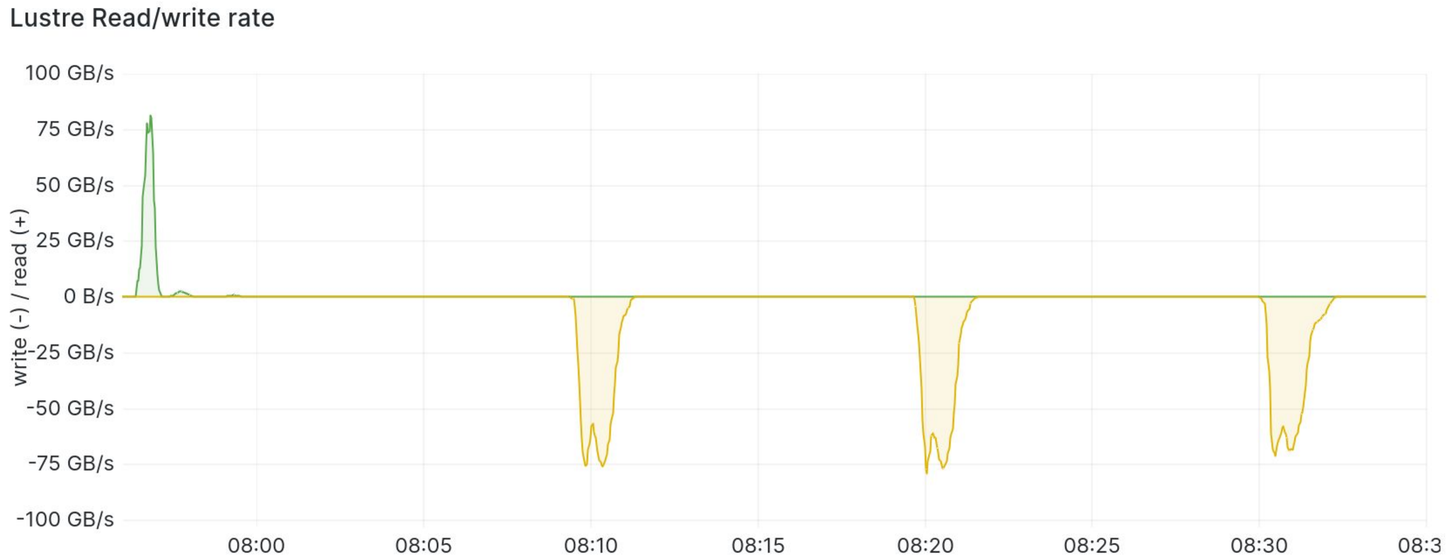
Iterative processing cycle

- Startup phase, followed by sequences of compute iterations, with periodic checkpointing

COMPUTE



I/O



High level pattern is similar to typical HPC checkpointing.

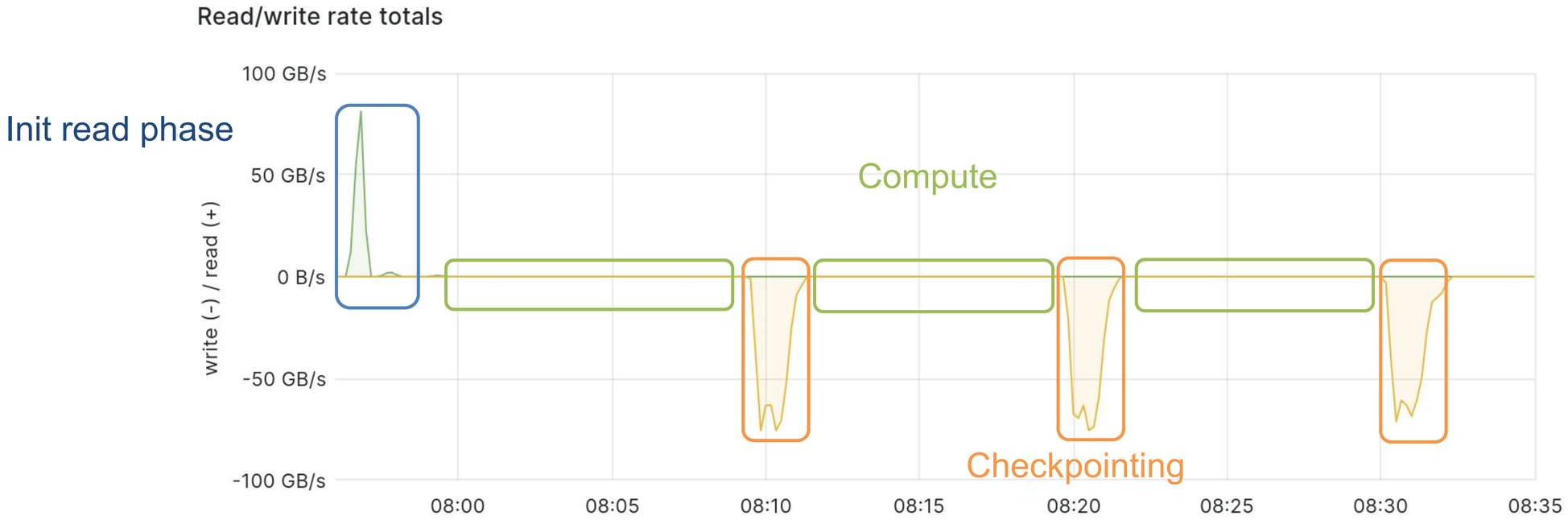
Minimal read at each iteration.

I/O patterns

3 distinct phases

- Initialization read phase – only once
- Compute phase – iterative GPU processing
- Checkpoint write phase – every N compute iterations
  - Note: uses buffered IO

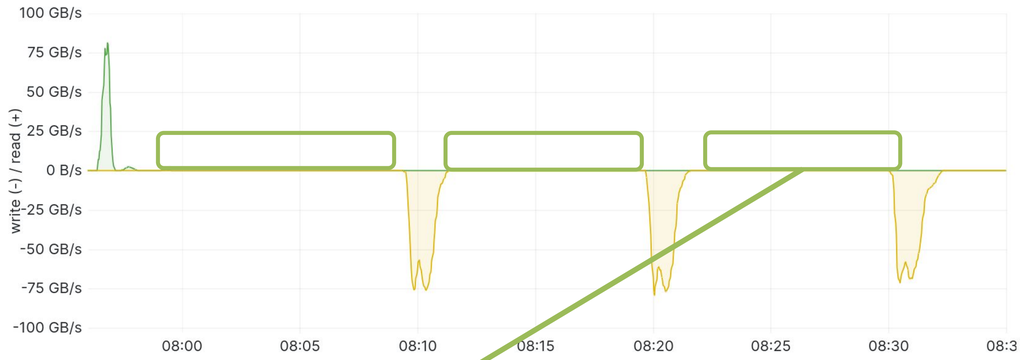
Typical traditional HPC application pattern, with sub-optimal checkpointing that could be improved.



Focus on reads

Zoom in on initialization and **compute** iterations

- Very low **read** during compute phase: ~3MB/s
- I/O sizes are small: < 4KB
- The aggregate read volume scales linearly as node count increases.

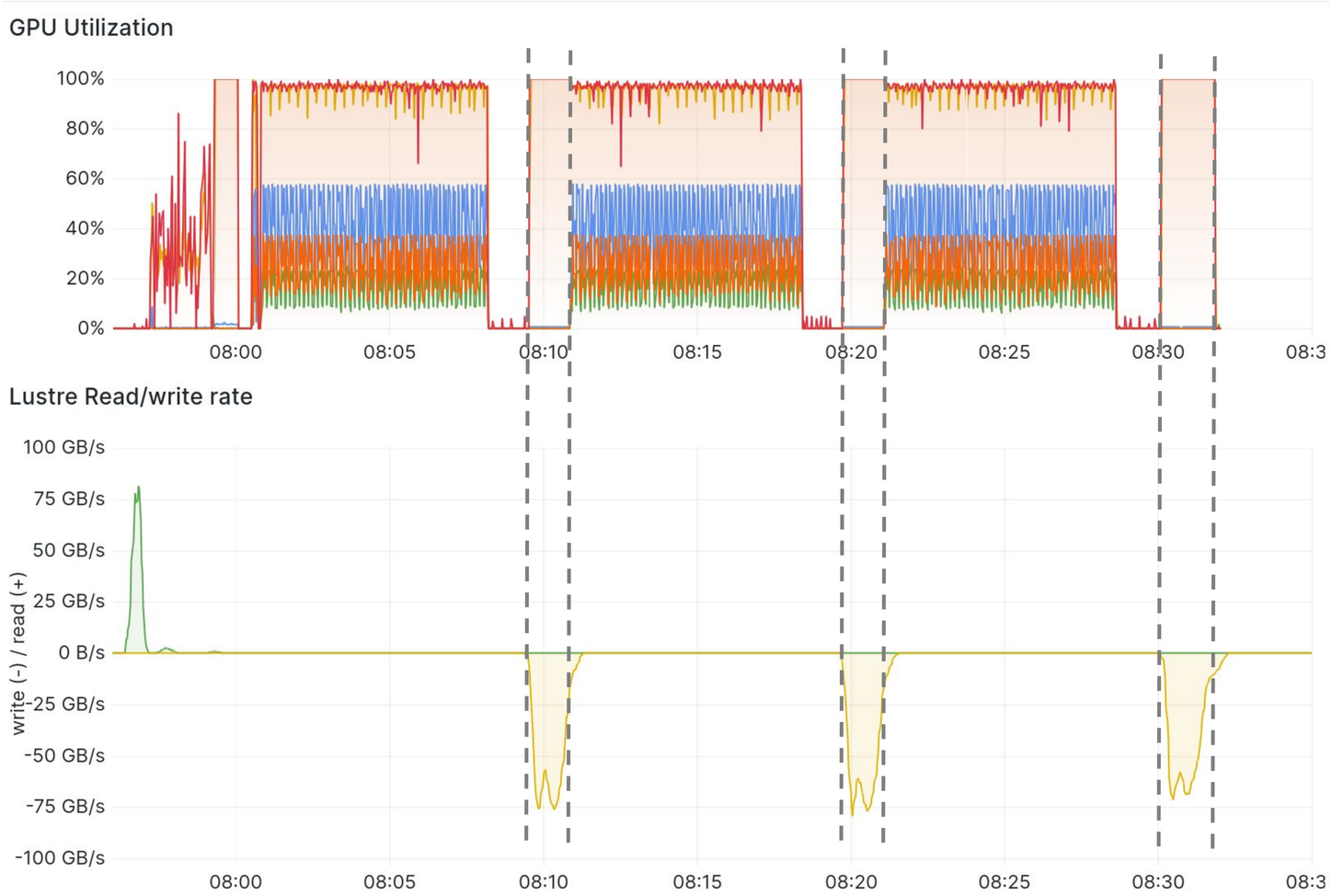


Zoom in shows ongoing dataset ingest.

Focus on writes

The checkpoint challenge

- In this current implementation, synchronous checkpointing is stalling job progress. The GPUs are busy waiting.

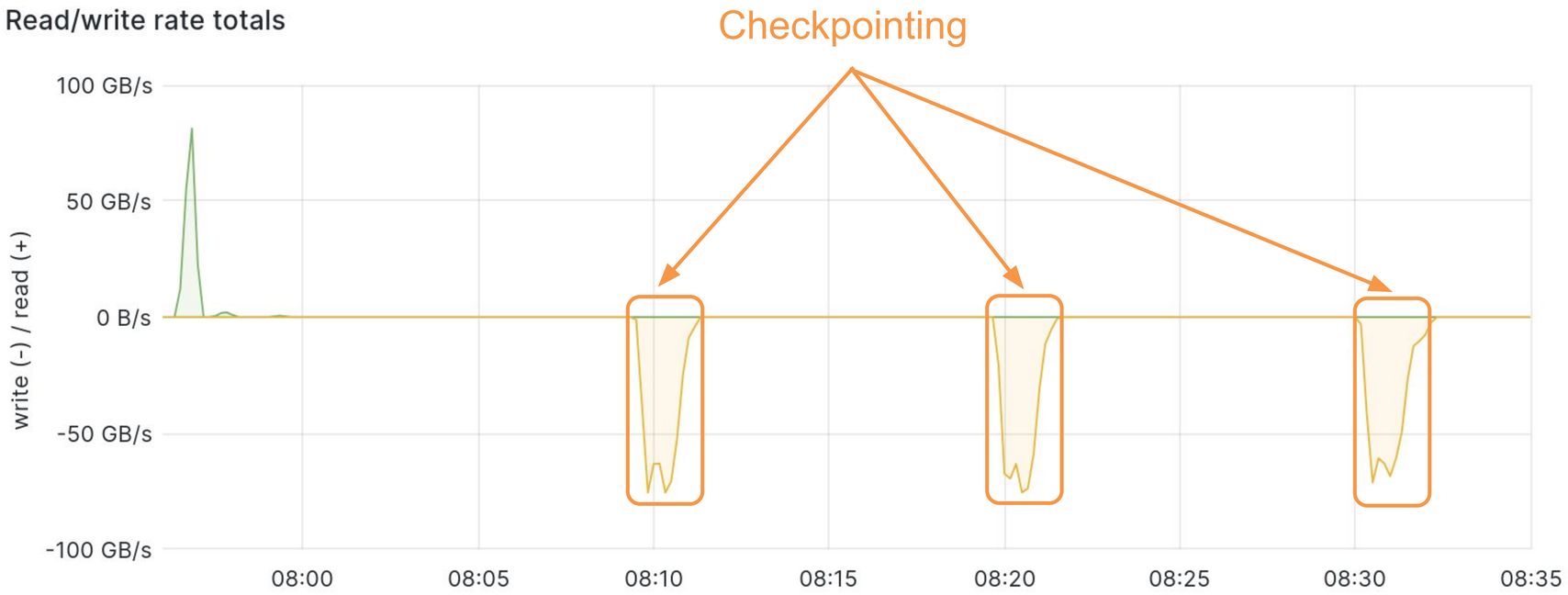


Checkpoint details

The shape of our example model-parallel test

- Checkpoint size is 4.3 TiB
- Checkpoint is done by few nodes (model parallelism). Spike at 75 GB/s
  - Each client is writing at ~6 GB/s (out of 93 GB/s available write capability)
- Checkpointing lasts for 90 sec

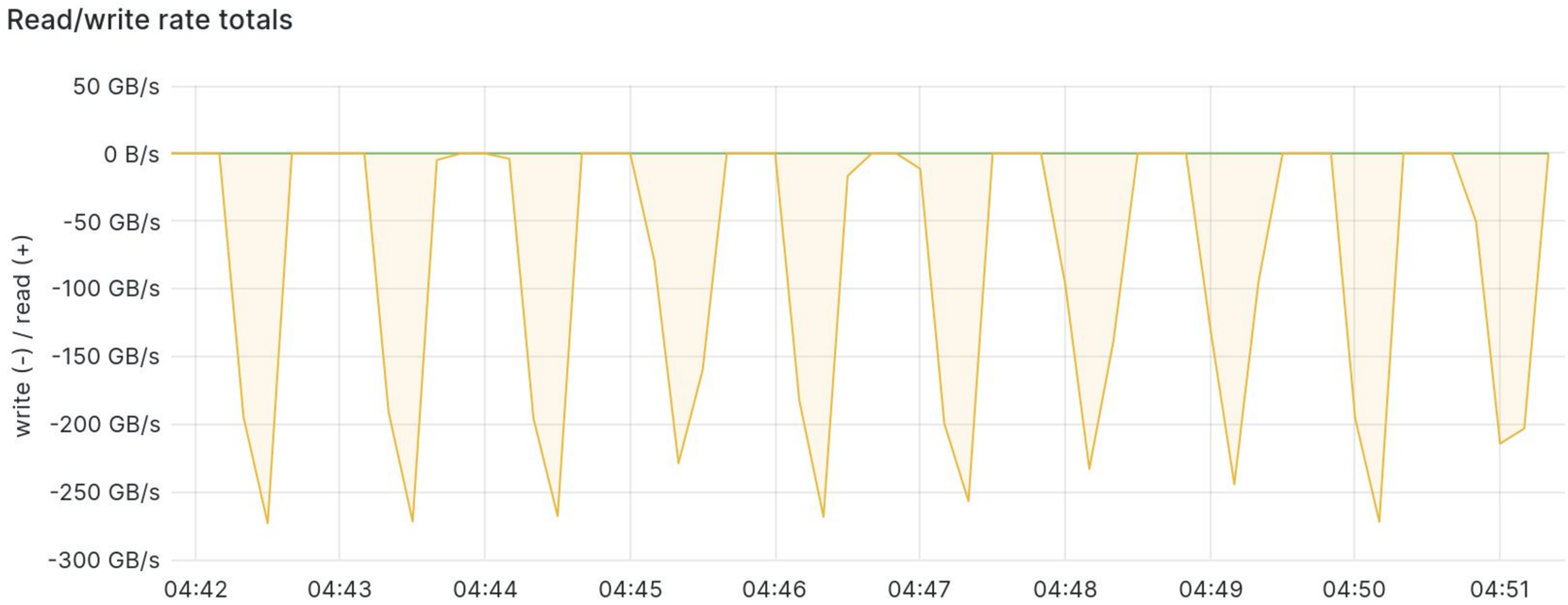
In this version of the model, peak IO scales with model parallelism and NOT with node count.



Opportunities for efficiency improvement

What would parallel checkpointing look like?

- Simulating checkpoints artificially, we can run fully parallel version of the workload's write IO
- Below is an example of 10 checkpoints on 48 nodes
- Each checkpoint lasting 16s, **peak at 275 GB/s** (~4x speedup)

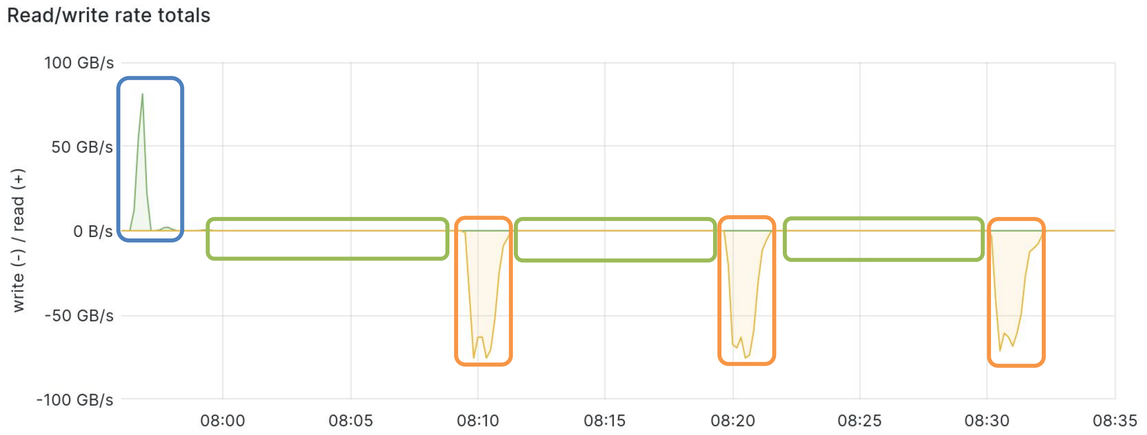


Here, peak IO scales with model size AND with node count.

Potentially more benefits from looking into checkpointing asynchronously.

Takeaways
for LLMs

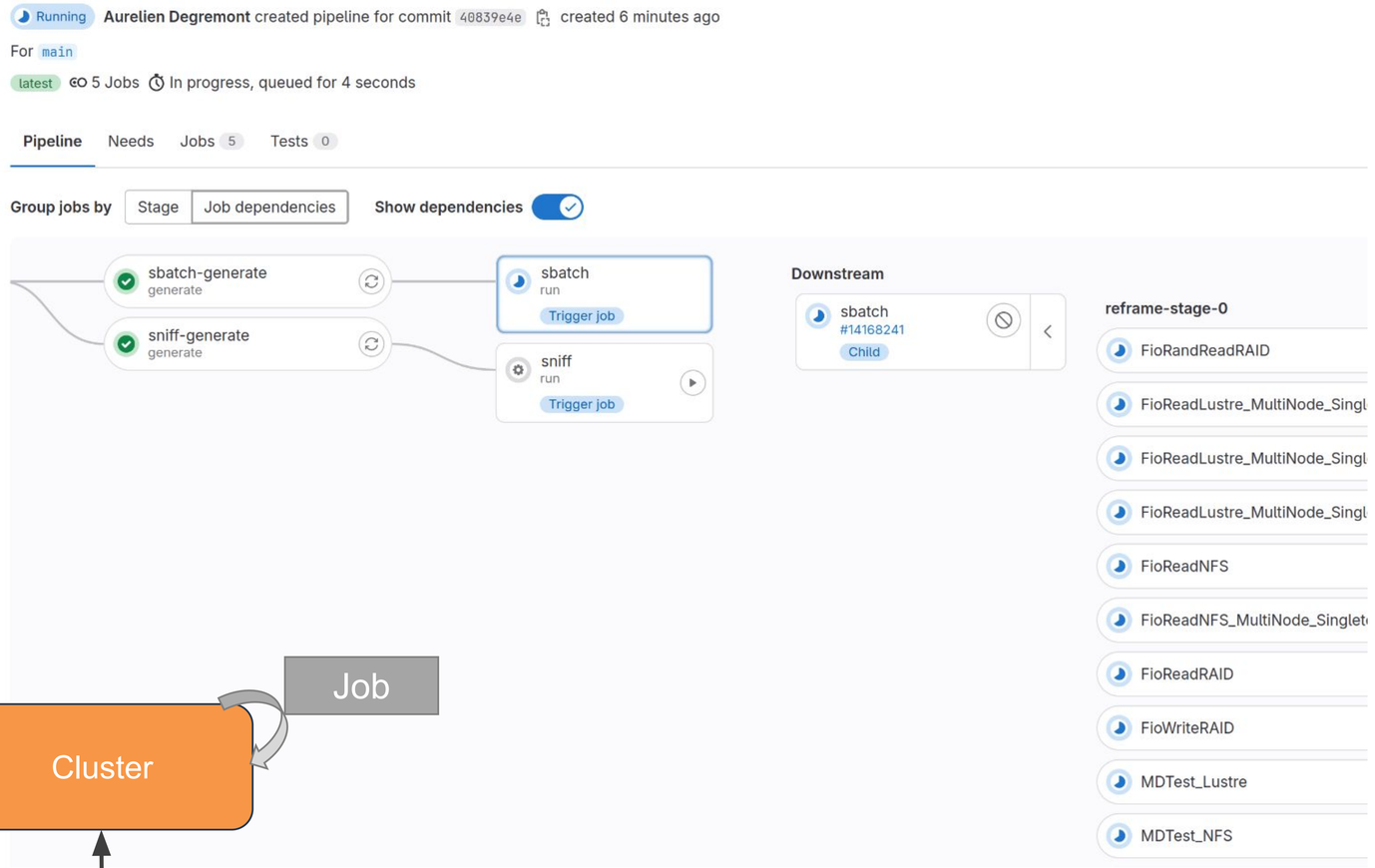
- Very low read bandwidth observed during compute phase → FS capability should focus on small IOs (or other workloads).
- Checkpoint scalability can enable large write peaks → demand more from the file system.
- Future checkpoint improvements could change the IO pattern again → rely on efficient concurrency from CPUs and network fabrics.

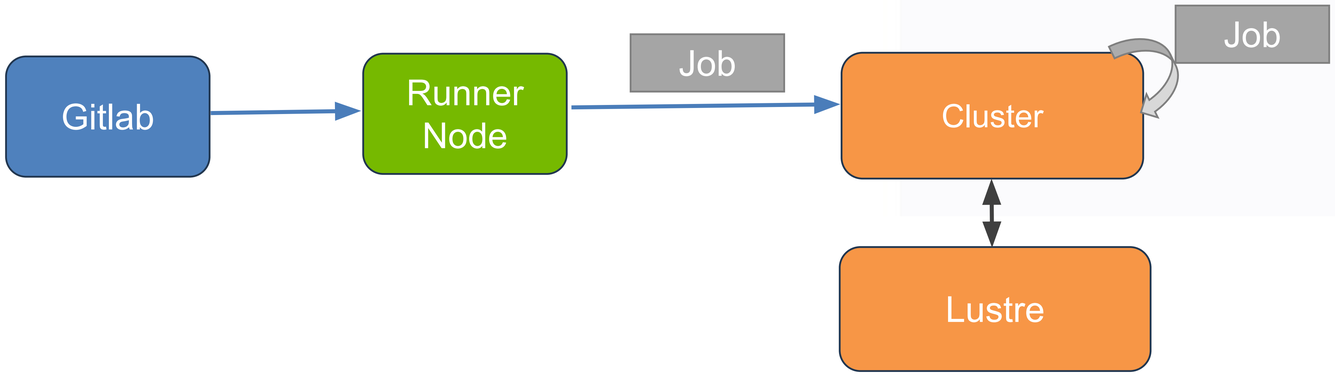


Workflow context

Running through a CI for reproducibility

- Continuous Integration pipelines run a large portion of the jobs on our system.
- CI is great for:
  - System performance regression testing.
  - Developer application regression testing.
  - User ad-hoc test runs.
- Setup and authentication require careful attention.

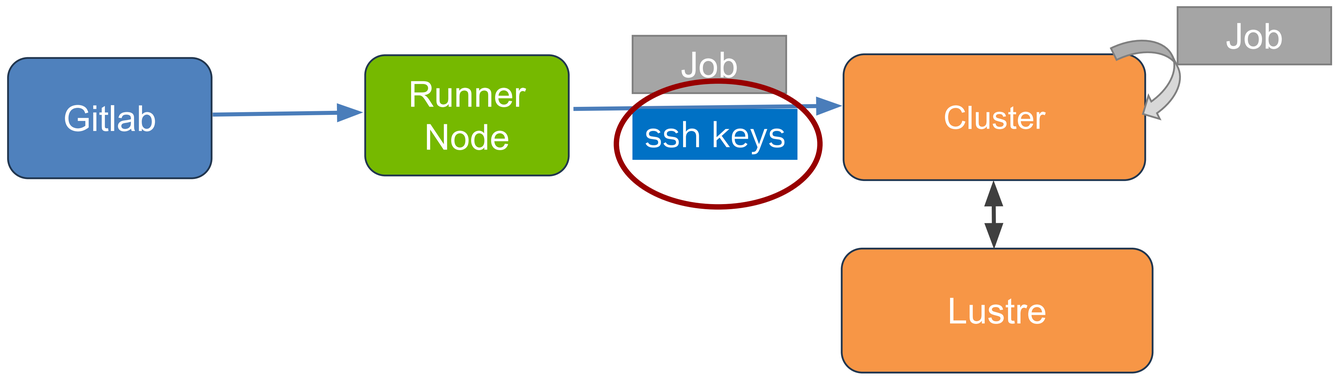




Challenges with CI

Manageability and security

- Running jobs through a CI environment is often done using shared accounts with poor traceability.
- Using SSH keys between runner node and cluster does not provide authentication and revocability.
- Shared CI accounts encourage unnecessarily open filesystem permissions.



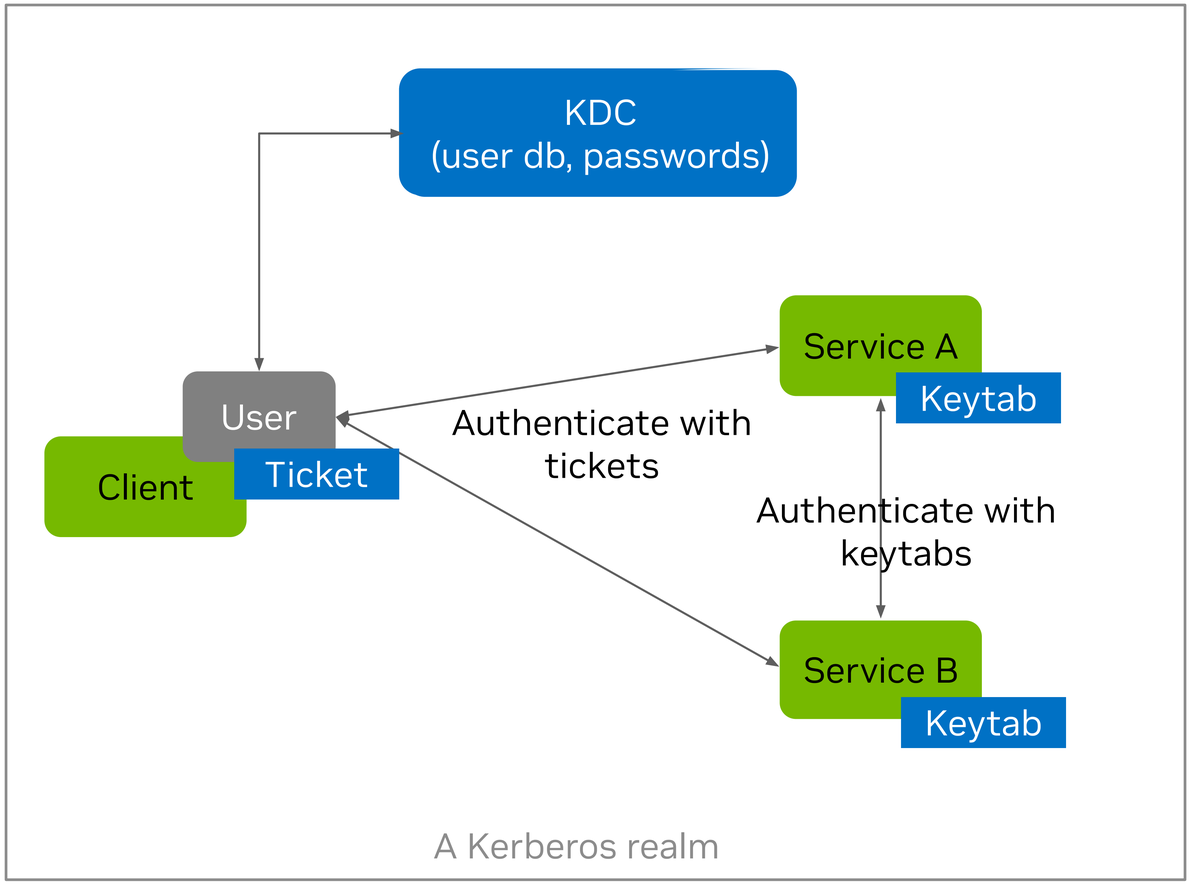
We want real and individual user accounts to be used.

But first we need strong authentication on the cluster itself…

Kerberos framework for strong authentication

What it is? How does it work?

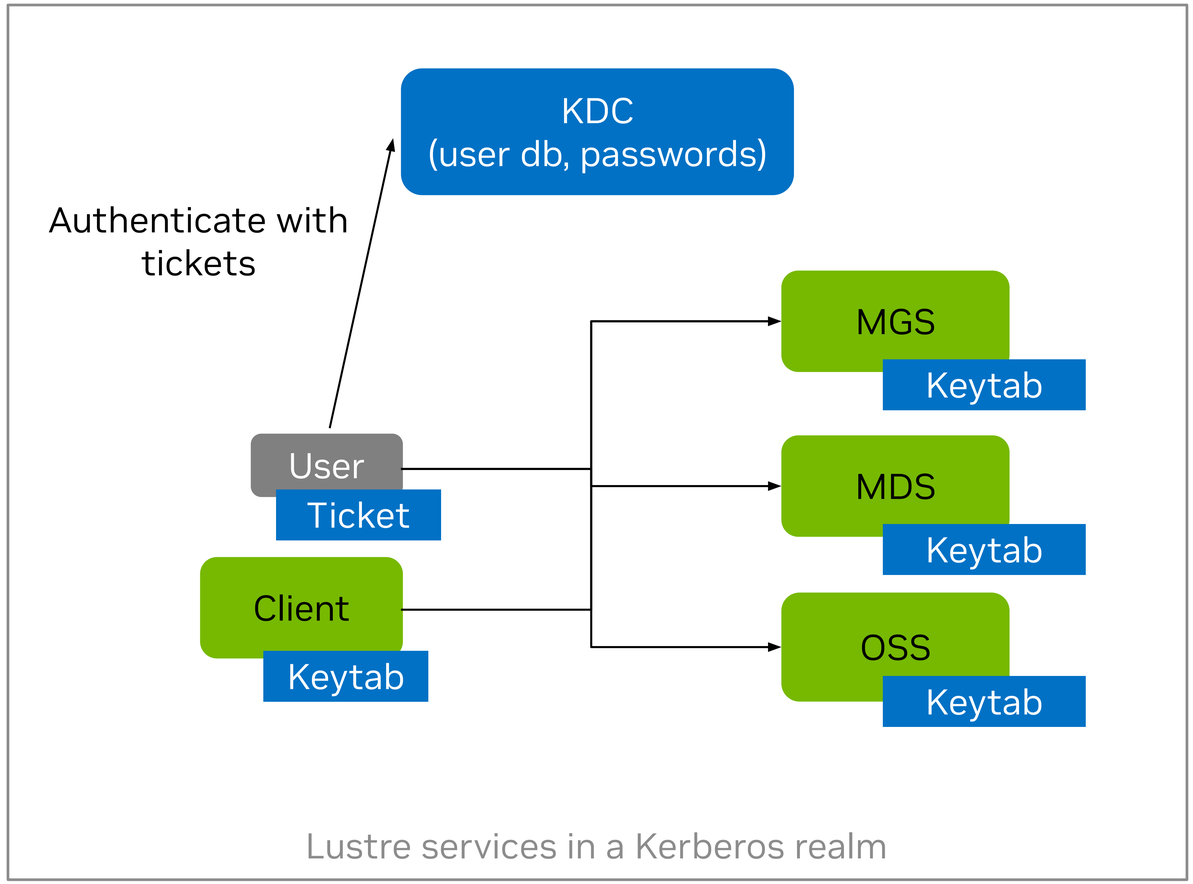
- Well-established authentication protocol
- Supports modern cipher suites (eg: AES and SHA2)
- Authentication is centralized in a KDC
- Kerberos manages:
  - Identities like users or services
  - Credentials (password or keytabs)
  - Domains, named *realms*
- Benefits
  - Keep user and password in one place
  - Single Sign On capability
  - Each service and host has its own keytabs
  - Servers and clients can authenticate each other (not just one-way trust)



Securing the environment

Improving CI and Lustre security with Kerberos

- There is an increasing need for security and integration in computing environments.
- Kerberos provides a way to improve security, limiting wide access permissions from clients.
- **More traditionally, Kerberos is used to control access to cluster login nodes.**
  - **But we are deploying to most services, notably to authenticate storage access.**
- By default, Lustre trusts any mounting clients and UID/GID from them.
  - Kerberos adds protection against a compromised host.
- Lustre has supported Kerberos for a long time.
  - But needed some refreshing.
  - Working with the community on bugfixes and improvements.

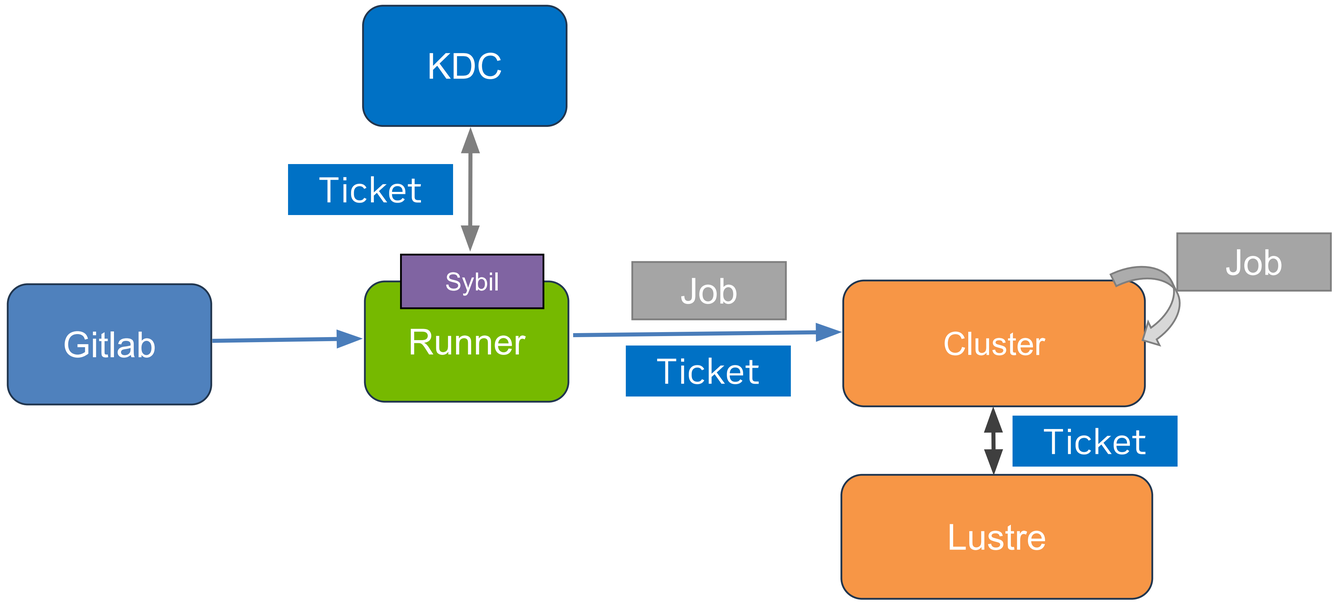


Lustre services in a Kerberos realm

Continuous Integration in a secured environment

Putting it all together - Kerberos, Lustre, CI

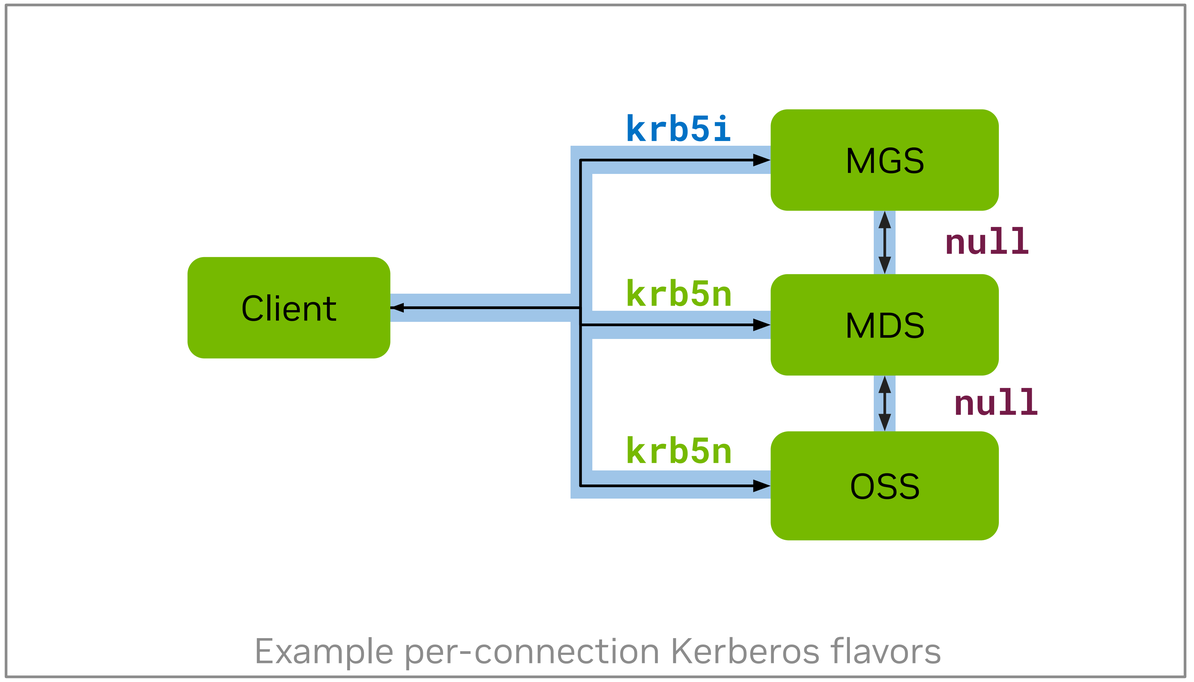
- System access and Lustre access requires a Kerberos ticket.
- Each service and host has its own keytabs.
  - Clients too
  - Servers and clients can authenticate each other at connection
  - MDS validates users using their ticket
- Relying on the open source project Sybil to impersonate user
  - Sybil fetches a user ticket that is used to access the cluster, and thus Lustre filesystem
  - https://github.com/NVIDIA/sybil



Digging deeper into Lustre and Kerberos

Choice of Kerberos flavors

- Kerberos is enabled at connection level, and different flavors exist:
  - Default:
    - `null` No kerberos
  - Authentication:
    - `krb5n` Authentication only
  - Integrity (checksumming):
    - `krb5a` +Header message integrity
    - `krb5i` +Bulk data integrity
  - Privacy (encryption):
    - `krb5p` +Message privacy (encrypted)



Example per-connection Kerberos flavors

- **But performance impact should be considered.**

More security and data integrity comes at the cost of greater RPC processing resource requirements.

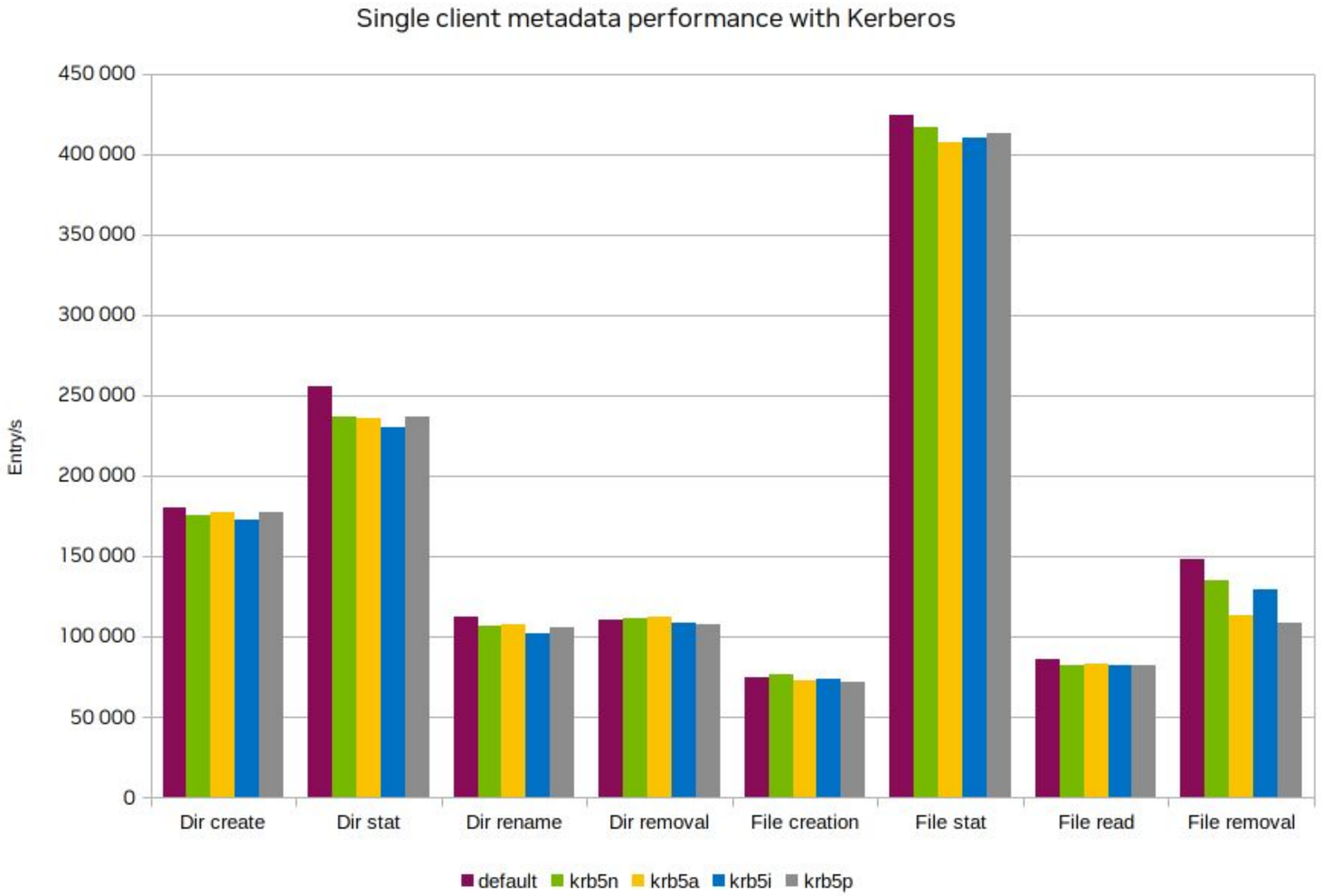
Kerberos flavor metadata performance

Limited impact

- mdtest benchmark run from 2 clients
  - 2x Xeon 8480C 56c
  - 2 TB RAM
  - 2x NDR 400 Gb/s

- Performance impact is limited.
- Results vary in a range of 4-10%.
- Except for *File removal*, where the difference is going up to 25%.

Checksumming or encrypting small metadata RPC is not very expensive.

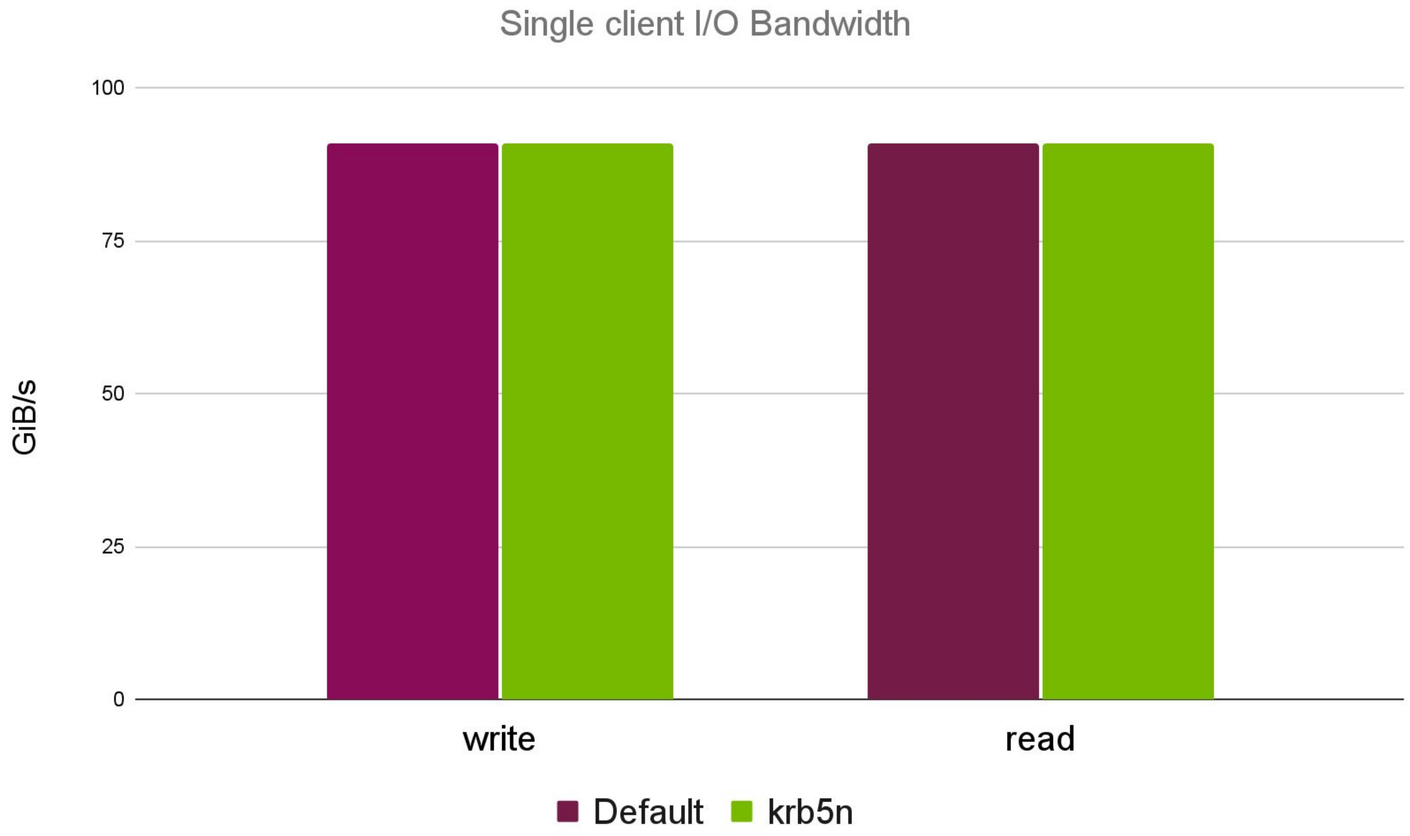


krb5n performance for single client bandwidth

No impact for authentication mode (krb5n)

- Single client I/O bandwidth (fio test run from a DGX H100 client) has no performance impact: **91 GiB/s**
- OST RPC does not authenticate users, so enabling krb5n does not impact the I/O path.

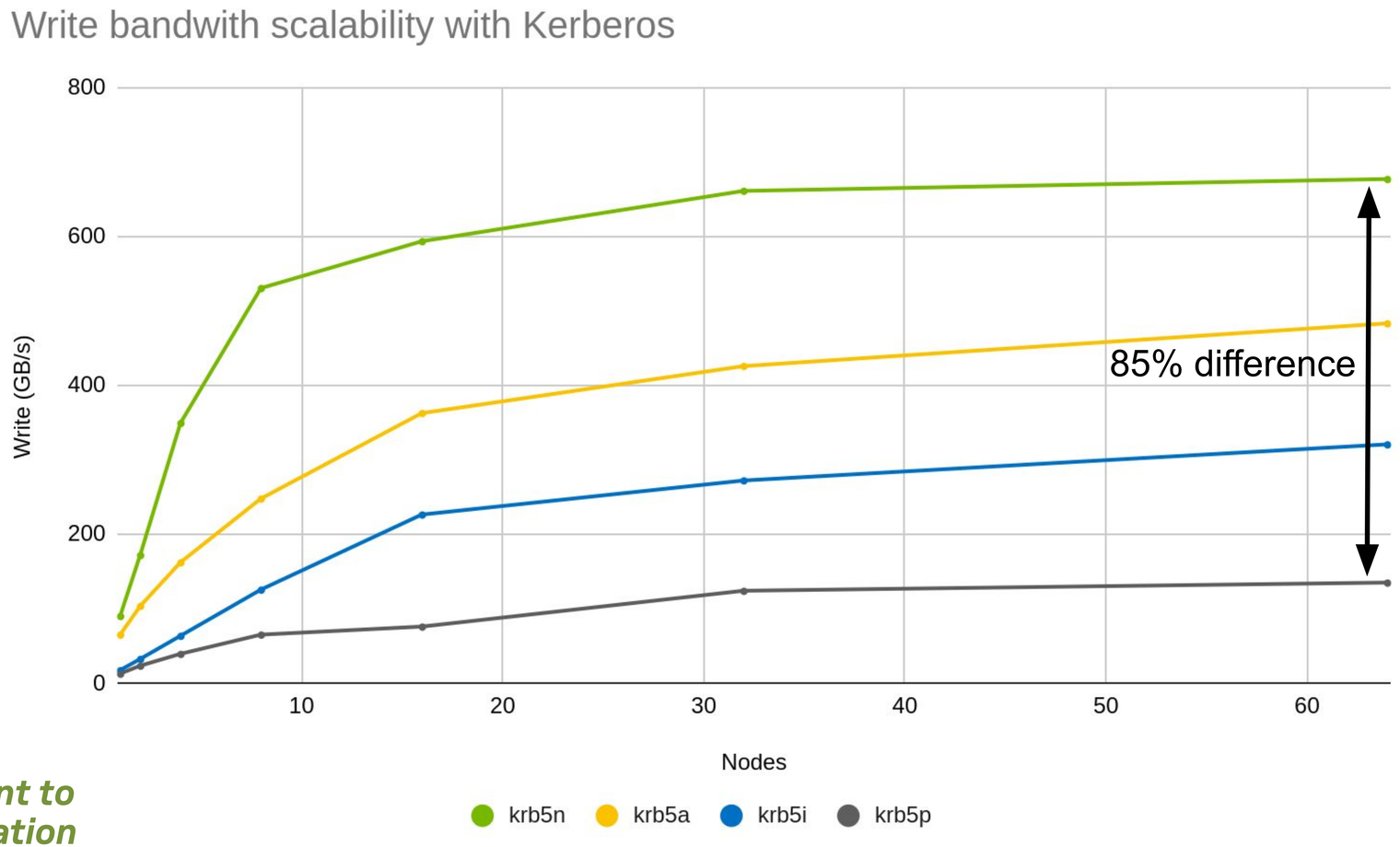
Negligible performance impact for large streaming I/O. No drawback to enable krb5n for OSTs.



Kerberos flavor scalability

Large impact of checksumming and encryption

- Up to 64 clients I/O bandwidth, with ior
  - Up to **85% performance drop**, compared to krb5n
- Checksumming and encryption is consuming lots of CPU in crypto functions. Algorithm depends on the Kerberos key cipher-suite.
  - Tested ciphers were AES and SHA1
- Investigation needed to see if improvements are possible with better implementation or different crypto algorithms.



Performance impact is too important to enable more than krb5n (authentication only) for OSTs.

Conclusion

Using Lustre to support Large Language Model IO and secured AI workflows

- IO profiling of an example open LLM framework shows low read usage and increasing checkpointing performance.
- Security can be improved with Kerberos for Lustre at limited performance impact for both metadata and bulk IO.
- Application and filesystem performance can be monitored efficiently through a CI without sacrificing security.

Thank you

- We would like to thank:
  - *Seonmyeong Bak* at NVIDIA for all the help to collect and decipher all the LLM data
  - *Sébastien Buisson, Andreas Dilger, Peter Jones* for the support and fast patch delivery


NVIDIA
Questions?