Characterizing Cooperative I/O Caching for Data Loading at Scale on Aurora

When Bounded Coordination Cost Overtakes Contention in Metadata-Intensive Startup I/O

Anthony Cardia†
Elmhurst University
Elmhurst IL United States
acardia@protonmail.com

Kaushik Velusamy
Argonne National Laboratory
Lemont IL United States
kaushik.v@anl.gov

Kevin A. Brown
Argonne National Laboratory
Lemont IL United States
kabrown@anl.gov

ABSTRACT

Deep-learning jobs on exascale systems begin by reading identical files from a shared parallel filesystem, forcing thousands of processes to issue near-identical requests at once. This simultaneous access contends for the filesystem and slows data loading; a cooperative cache can serve redundant reads locally instead. But caching adds its own overhead, so it is not always worthwhile — the open question is *when* it pays off. We characterize Copper, a cooperative read-caching filesystem, against direct Lustre reads for two data-loading workloads on Aurora, to identify the conditions where caching wins.

CCS CONCEPTS

• Information systems → Storage management; Distributed storage.

• Computer systems organization → Distributed architectures.

• Computing methodologies → Distributed computing methodologies.

KEYWORDS

cooperative caching, parallel I/O, metadata contention, deep learning I/O, HPC storage, Lustre, data loading at scale

ACM Reference format:

Anthony Cardia, Kaushik Velusamy, and Kevin A. Brown. 2026. Characterizing Cooperative I/O Caching for Data Loading at Scale on Aurora. In *Proceedings of SC26.* ACM, New York, NY, USA, 2 pages.

1 Problem

Supercomputer time is expensive, so time spent waiting on startup I/O is time not spent computing science. At scale, thousands of ranks reading the same files create a metadata-and-read stampede: the storage system bottlenecks as every process requests the same data at once, competing with every other job on the machine.

Cooperative caching offers a way out. Copper reads each file once per node and serves the redundant reads locally, so they need not all reach the filesystem. But this is not free — the cache adds per-access overhead and a one-time coordination cost, so it can be slower than reading directly when contention is low. The real problem, then, is not contention itself but knowing *when* cooperative caching is worth its cost: for which workloads and at what scale does Copper beat direct Lustre reads?

2 Approach

To answer this, we measure the two factors that should govern the trade-off: contention, which grows with node count, and access pattern, which is set by a workload's file sizes and counts. We benchmark Copper — a cooperative caching layer serving a node-local cache through a FUSE mount [7] — against direct Lustre reads, from 2 to 2048 nodes, on two workloads with contrasting access patterns: PyTorch startup (many small files) and ResNet-50 dataset loading (fewer, larger files). Each is measured cold (first read) and warm (cached re-read), separating the caching benefit from raw read cost. Comparing the same workloads with and without Copper across scale isolates when the cache's benefit exceeds its overhead.

3 Results

PyTorch startup is metadata-bound: syscall tracing shows an ~8:1 ratio of metadata operations (stat/open) to reads across ~24,000 files with a median size of 2.5 KB. The cost is finding and opening many small files, not moving bytes. Scaled by ranks, a 512-node startup issues roughly 150 million file operations against the metadata servers.

Against this workload, Copper shows the predicted crossover. At small scale its per-file overhead makes it slower, but its cost stays flat as nodes are added while direct reads degrade. At 2048 nodes (24,576 ranks), startup through Copper is ~3× faster cold and

~11× faster warm than reading directly, and the advantage grows with scale.

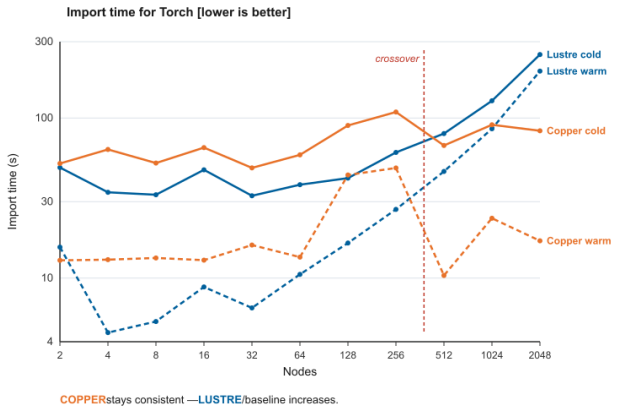


Figure 1: PyTorch startup time versus node count, with and without Copper (log scale). Copper's cost stays roughly flat while direct Lustre reads climb with contention; the lines cross and diverge, reaching ~3× cold and ~11× warm gaps at 2048 nodes.

4 Mechanism

The benefit is bounded by a one-time coordination cost. Node registration stays flat (~0.1s) at every scale, and first-RPC overlay-tree formation (built on the Mochi framework [5]) grows only gracefully, from ~10s at small scale to ~23s at 2048. Registration is local and independent of job size, and the tree branches hierarchically rather than fanning out flat. This asymmetry makes the crossover inevitable at scale. By collapsing the redundant fraction of those ~150M operations, Copper removes load from the shared metadata servers, easing contention for co-running jobs as well.

5 Scope, Generalization, and Limitations

Not all data loading benefits. On the Places365 dataset, GPU compute bounds throughput, so caching relieves an I/O bottleneck that is not the limiting one. We include it as characterization rather than a training speedup: cold populates the cache and warm re-reads are served locally, with warm throughput scaling near-linearly (2,403 to 16,826 images/s, 1–8 nodes) and far exceeding the OS page cache (~2.3×). It usefully delimits where caching helps.

This differs from bulk-write parallel I/O middleware such as ADIOS; the target here is the metadata-dominated small-file read pattern of framework startup. Because the mechanism depends on redundant-read contention rather than any Aurora-specific feature, the effect should intensify on future systems with denser accelerator nodes, where more compute per node drives more simultaneous I/O against one shared filesystem. A synthetic file-size sweep to map the crossover as a predictive guideline is in progress. Repetitions at the highest node counts are limited; firming these margins is ongoing.

6 Conclusion

We identify a crossover in metadata-intensive startup I/O where caching's bounded cost overtakes contention-bound direct reads. Future work includes completing the synthetic file-size sweep, developing a parameterized analytical model of the crossover, and exploring scheduler-integrated preloading and topology-aware cooperative sets. This identifies both a concrete benefit for data loading at scale and, through the load removed from shared storage, a potential benefit to the facility as a whole.

ACKNOWLEDGMENTS

This work was performed at the Argonne Leadership Computing Facility, a U.S. Department of Energy Office of Science user facility. The author thanks Kaushik Velusamy and Kevin A. Brown for their guidance, and acknowledges the ALCF for access to Aurora. The author also acknowledges and thanks the WCCPE of Elmhurst University, and Tim and Robert Jans for their funding of the JANS internship program.

During the preparation of this work, the author used Claude (Anthropic) to assist with drafting and refining the text of the abstract and Sections 1–6. After using this tool, the author reviewed and edited the content as needed and takes full responsibility for the content of the publication.

REFERENCES

[1] Argonne Leadership Computing Facility. 2024. Copper: A cooperative caching layer for HPC. Retrieved from https://github.com/argonne-lcf/copper

[2] Argonne Leadership Computing Facility. 2024. dl_scaling: Deep learning scaling benchmarks. Retrieved from https://github.com/argonne-lcf/dl_scaling/tree/main/resnet50

[3] Intel Corporation. 2021. oneCCL Environment Variables. Developer Guide and Reference. Retrieved from https://www.intel.com/content/www/us/en/docs/oneccl/developer-guide-reference/2021-9/environment-variables.html

[4] Anthony Cardia. 2026. copper-benchmarking: Benchmarking scripts for cooperative I/O caching on Aurora. Retrieved from https://github.com/AnthonyMontego/copper-benchmarking

[5] Robert B. Ross, George Amvrosiadis, Philip Carns, Charles D. Cranor, Matthieu Dorier, Kevin Harms, Greg Ganger, Garth Gibson, Samuel K. Gutierrez, Robert Latham, Bob Robey, Dana Robinson, Bradley Settlemyer, Galen Shipman, Shane Snyder, Jerome Soumagne, and Qing Zheng. 2020. Mochi: Composing Data Services for High-Performance Computing Environments. J. Comput. Sci. Technol. 35, 1 (2020), 121-144. https://doi.org/10.1007/s11390-020-9802-0

[6] Philip Schwan. 2003. Lustre: Building a File System for 1,000-node Clusters. In Proceedings of the 2003 Linux Symposium. 380-386.

[7] Bharath Kumar Reddy Vangoor, Vasily Tarasov, and Erez Zadok. 2017. To FUSE or Not to FUSE: Performance of User-Space File Systems. In Proceedings of the 15th USENIX Conference on File and Storage Technologies (FAST '17). USENIX Association, 59-72.

[8] Anthropic. 2026. Claude. https://www.anthropic.com/claude