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

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

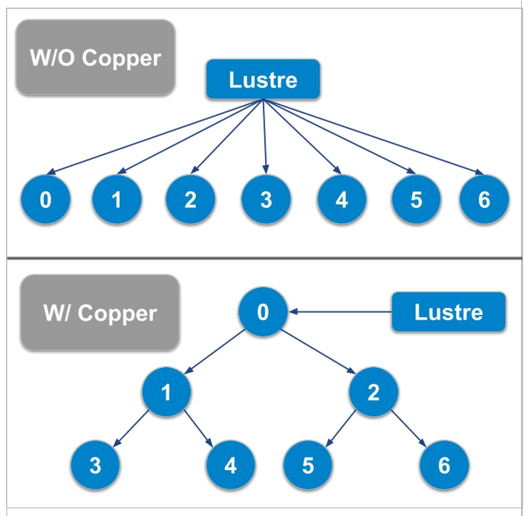
Anthony Cardia[1,2], Kaushik Velusamy[2], Kevin A. Brown[2]
[1] Elmhurst University, [2] Argonne National Laboratory

ABSTRACT

- **COPPER**: Copper is a caching filesystem that centralizes and reduces redundant file accesses.
  - One compute node reads the file and shares to all other nodes in the job.
  - Reduces storage and network contention; ***speeds up applications***.
- **TWO WORKLOADS**: We test this across two workloads on Aurora (**PyTorch startup** and **ResNet-50 image loading**), measuring read cost with and without Copper.
- **PRACTICAL GUIDELINE**: We developed a guideline for when caching's benefit is worth its overhead, given a workload's **file count and size**.

MOTIVATION

- **TIME IS SCIENCE**: Supercomputer time is expensive, so time spent waiting is wasted science for AI or other scientific workloads.
- **TRAFFIC**: The bigger the job the worse the jam at shared storage, because thousands of processes all grab the same files at once.
- **COOPERATE**: Copper is a **cooperative cache**: the first reads fill a node-local cache served through the filesystem.
  - Redundant reads across nodes don't all hit shared storage.
  - It has overhead, so the benefit only appears once enough nodes are competing.



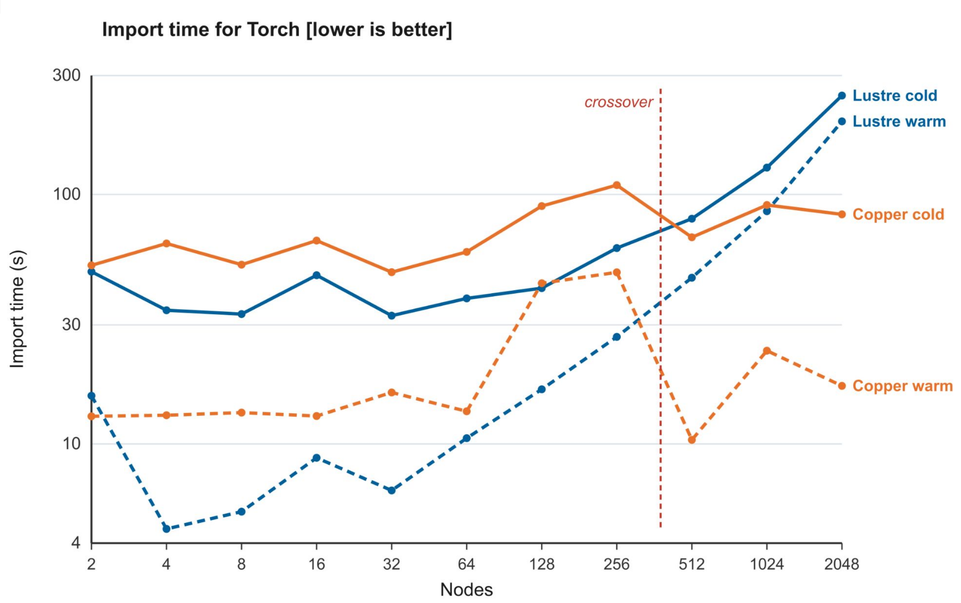
Lustre talks to multiple nodes w/ Copper, omitted for simplicity

METHODS

- **TWO TASKS**: PyTorch startup and image loading, with and without Copper, across many job sizes.
- **COLD vs. WARM**: Each timed **cold (first read)** vs. **warm (re-read)**, isolating the caching benefit from raw read cost.
- **IMPORT**: Custom torch env routed through Copper vs. Lustre.
- **ResNet-50**: Places365 images (~1.8M) through Copper vs. Lustre.
- **PLANNED**: synthetic sweep mapping the crossover against total file requests.

IMPORT TORCH [Many small files]

- Starting PyTorch opens **tens of thousands of tiny files** — the "many small grabs" that reducing redundant file accesses can majorly improve.
- **METADATA-BOUND**: **~8:1** metadata-to-read syscalls (24,412 files, median **2.5 KB**). At 512 nodes, **~150M** file operations hit the metadata servers.
- **SLOW AT FIRST**: Copper is slower (overhead not yet worth it). **At scale:** its flat cost overtakes direct reads, crossover by **~256 nodes**.
- **THE PAYOFF**: at 2048 nodes, Copper **~3× faster cold**, **~11× faster warm**, the gap grows with scale.
  - Coordination cost stays **bounded** (registration ~0.1s, first-RPC ~10–23s).
  - Copper wins here because it caches file lookups locally, sparing the storage system's metadata servers from the same request repeated by every node.



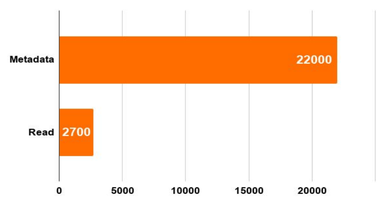
COPPERstays consistent —LUSTRE/baseline increases.

Torch Environment

size range	files	total bytes	% files
<1KB	4564	2.41 MB	18.7%
1-16KB	15251	63.95 MB	62.5%
16-256KB	4193	190.23 MB	17.2%
256KB-1MB	229	127.72 MB	0.9%
1-10MB	91	252.32 MB	0.4%
10-100MB	79	3.51 GB	0.3%
>100MB	5	2.72 GB	0.0%

TOTAL: 24412 files, 6.84 GB

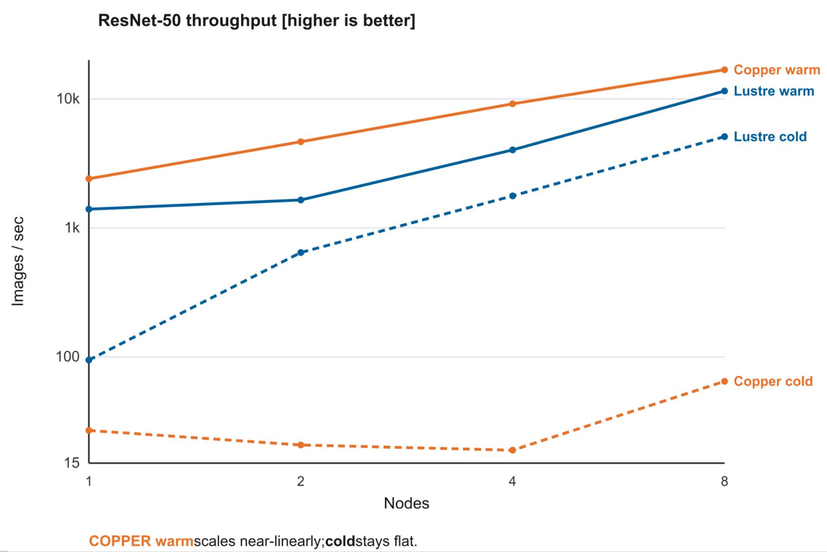
median file size: 2.52 KB



Figures above relate to 1x instance not 512x

ResNet-50 [Many larger files]

- File size of ~150KB per image, 1.8M images.
- Loading images is **slow cold, but far faster warm** through Copper, even at small scale.
- **FIRST PASS**: The first pass **fills the cache**; re-reads are served locally (warm **200–485× faster** than cold).
  - Warm throughput **scales near-linearly** (2,403 → 16,826 img/s, 1→8 nodes) and beats the **OS page cache** (~2.3×).
- **REPETITION**: big cold cost, bigger warm payoff, value grows with **how much you re-read**.
  - Copper's benefit here comes from repetition rather than relieving a metadata jam.
- **CAVEAT**: Measured with a small working set that fits in cache; full-epoch behavior at scale is future work.



COPPER warmscales near-linearly;**cold**stays flat.

CONCLUSIONS & NEXT STEPS

- Copper helps most when many processors repeatedly grab lots of small files in a big job.
- Copper's value isn't peak speed; it's steady, predictable cost that reduces load on shared storage, easing contention for every job on the machine, not just its own.
- Copper benefits small-file, metadata-heavy workloads read redundantly at scale, quantified by total file requests (files × ranks) and file size.
- **PRACTICAL RULE OF THUMB**: reach for Copper when a job has many nodes repeatedly reading many small shared files, and skip it for a few large files read once.
- **NEXT**: more reps to firm the crossover, a synthetic sweep toward a usable guideline, and testing whether the benefit grows on denser next-gen systems where more compute per node intensifies I/O contention.

REFERENCES

- [1] Copper. 2024. https://github.com/argonne-lcf/copper
- [2] dl_scaling. 2024. https://github.com/argonne-lcf/dl_scaling/tree/main/resnet50
- [3] Intel Corporation. 2021. https://www.intel.com/content/www/us/en/docs/oneccl/developer-guide-reference/2021-9/environment-variables.html
- [4] copper-benchmarking. 2026. https://github.com/AnthonyMontego/copper-benchmarking

U.S. DEPARTMENT OF ENERGY
Argonne National Laboratory is a U.S. Department of Energy laboratory managed by UChicago Argonne, LLC.

Acknowledgement: This work was supported in part by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research, under contract number DE-AC02-06CH11357.

REPO

Argonne NATIONAL LABORATORY