STORAGE DEVELOPER CONFERENCE
SDC 21
BY Developers FOR Developers
Virtual Conference
September 28-29, 2021
A SNIA Event

From NASD to DeltaFS
CMU and Los Alamos's Effort in Building Large-Scale Filesystem Metadata

Qing Zheng, Scientist, Los Alamos National Laboratory

Atomic Variables in C

- Declared with the _Atomic type qualifier
  - E.g.: "`_Atomic int a;`"
- **Sequentially consistent**: all memory operations form a global ordering
  - *The latest data written to memory is always used*
- **NOT** always necessary
  - Programs explicitly request it when needed

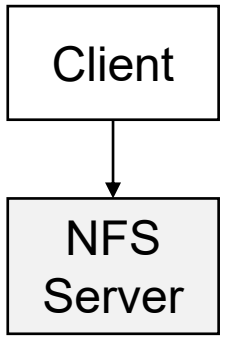
While C Provides Choices, FS Doesn't

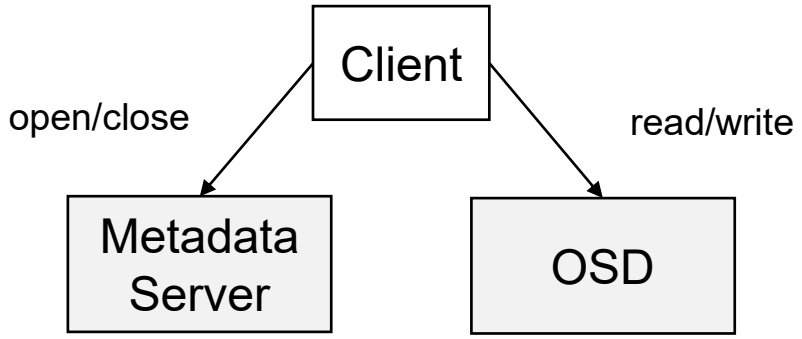
- All FS metadata operations are **_Atomic**
  - *Files created by 1 process are immediately visible to all regardless of user need and increasingly prohibitive performance penalties*
- **DeltaFS** handles this issue
  - 49x faster overall metadata throughput, 52x better resource usage

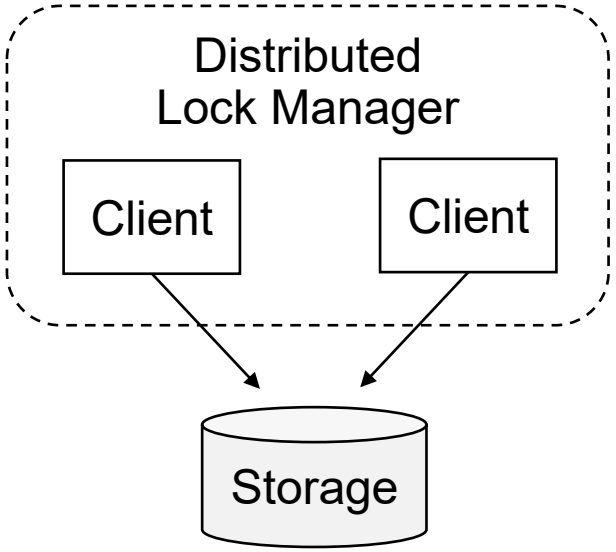
Background

Three Forms of Distributed FS Today

DeltaFS belongs here!







NFS

- Typically used for implementing /home, /project

NASD

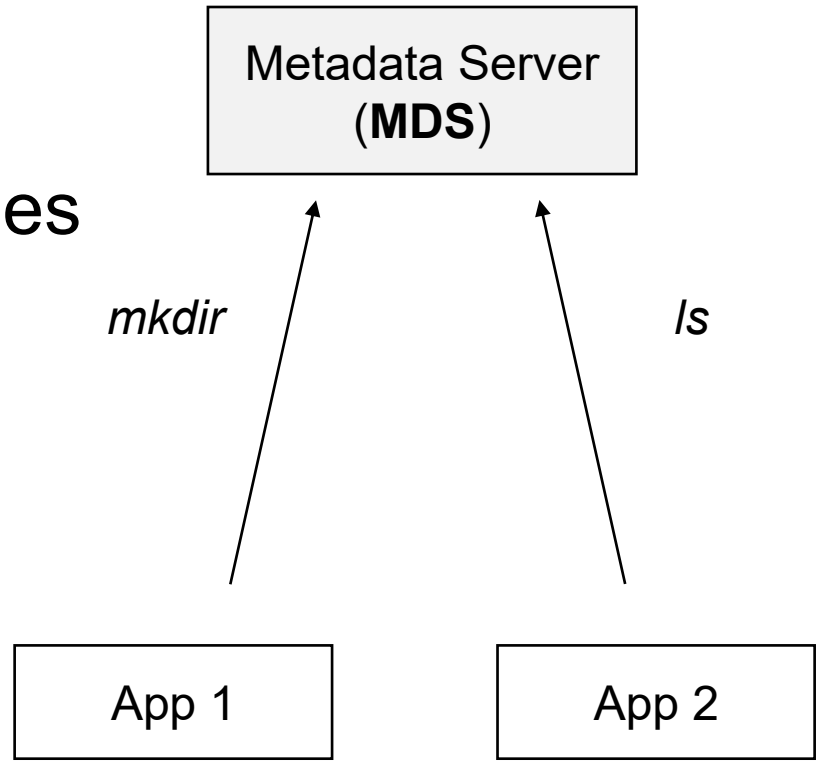
- Lustre, Hadoop FS, Ceph FS
- Separated metadata and data servers

Cluster FS

- Oracle OCFS2, RedHat GFS2, IBM GPFS
- No dedicated metadata managers, moderate scale

Global FS Metadata Synchronization

- All metadata operations are **ordered** by a dedicated metadata server (MDS)
- **Fully consistent** for all access across all nodes
  - As if they were done on a single node
- **Enable interprocess communication:**
  Applications may use the filesystem to achieve synchronization
- **But** MDS often a bottleneck



For example: App 1 creates a dir. App 2 can immediately see it and list it

Current Techniques for Scalable FS Metadata

- **Dynamic namespace partitioning** for load balancing across multiple MDS nodes
  - *Potentially lots of dedicated servers are needed for extreme workload*
- **Latency hiding** through deep metadata write buffering and LSM-Trees
  - *Still a bottleneck when background operations cannot keep up with foreground activities*

DeltaFS's Take on Scalable Metadata

- **NOT** all programs need to read other programs' files
  - Unrelated applications never communicate
- As cluster size grows, synchronization of anything global should be **avoided**
- *What if we build a FS that does not globally serialize all metadata operations?*
  - Applications self-commit their namespace changes as logs
  - Followup jobs selectively merge logs as needed

Logs can be merged in any order. Not all logs need to be merged

DeltaFS

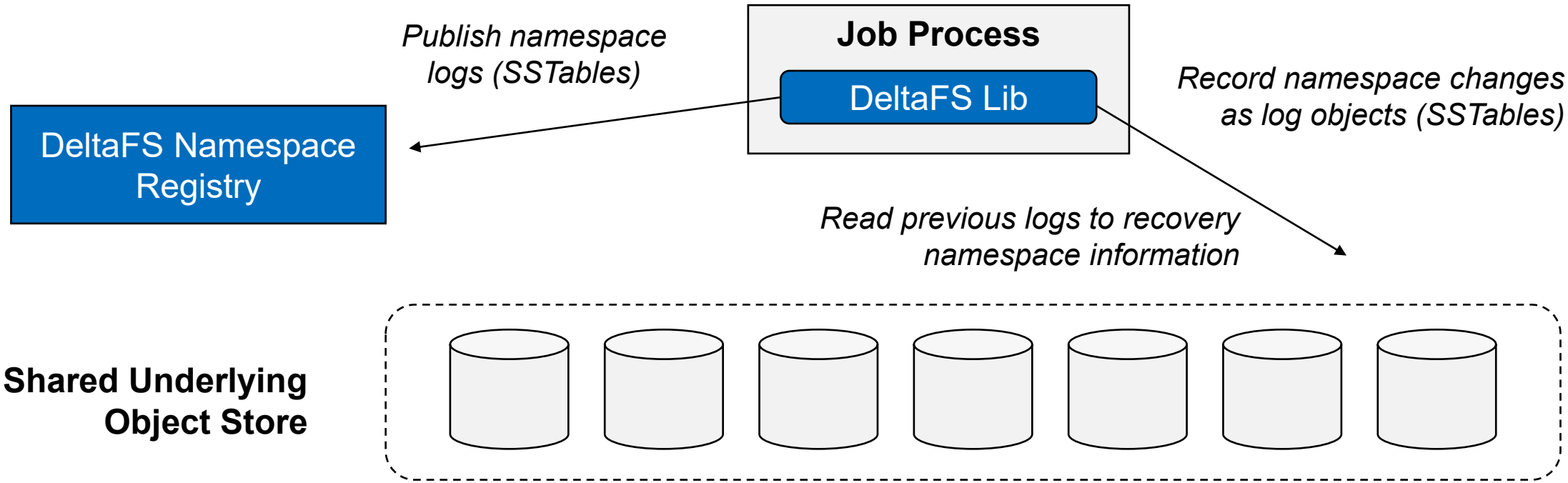
Target Workload: Massively-Parallel Computing

- DeltaFS applications are **batch jobs** that are submitted to run on large-scale computing platforms
  - A job is submitted, waits some time, is scheduled, reads its input, writes output, and exits
  - Input tends to be **static** at the time the job is submitted
  - Output typically **not examined until** after the job is done
- **In other words**:
  - Constant global FS synchronization not necessary

Batch jobs are non-interactive programs. They know what they will read and write.

System Overview

- DeltaFS is a set of **user space libraries and daemons** for scalable FS namespace management



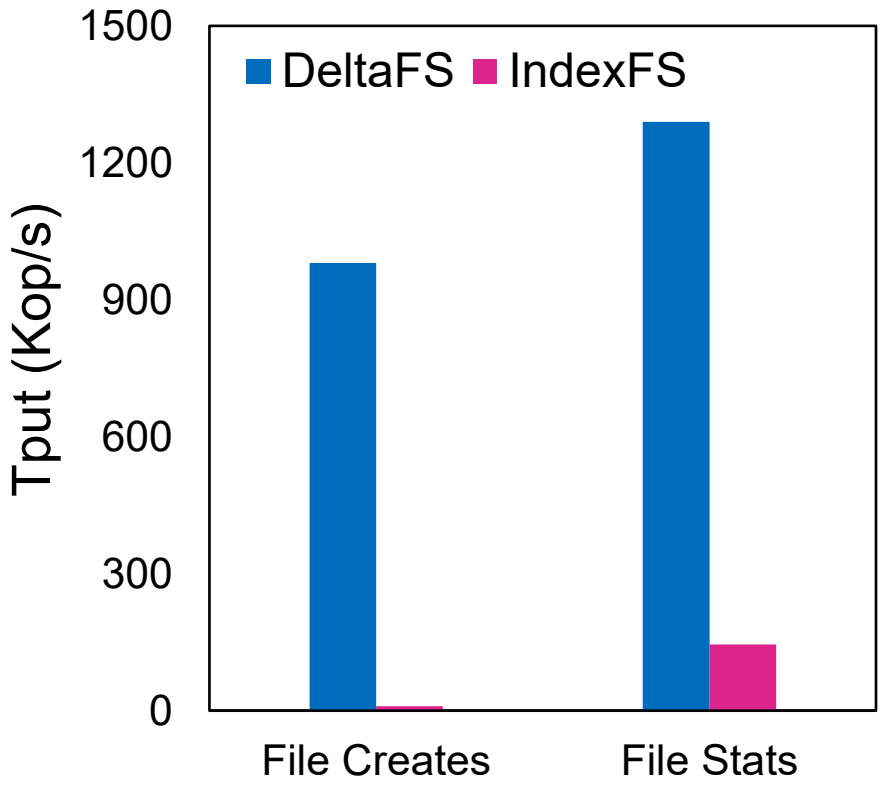
No Dedicated Metadata Servers

- DeltaFS **library code** running inside each job process serves as FS metadata clients and servers
  - When a job consists of multiple processes (i.e., a parallel job), each job process manages a partition of the job's namespace
- **No global namespace**
  - Each job runs on top of logs produced by previous jobs to obtain their output files

Recall that batch jobs know their input and output, making DeltaFS design efficient

High Single Job Performance

- Today's FS immediately serializes every metadata operation in a global namespace
  - Performance **limited** by the dedicated MDS
- DeltaFS distributes metadata processing across job processes
  - Jobs independently execute FS metadata operations with their own CPU cores
  - Performance **NOT limited** by dedicated servers



DeltaFS is up to 98x faster than IndexFS, the current state-of-the-art

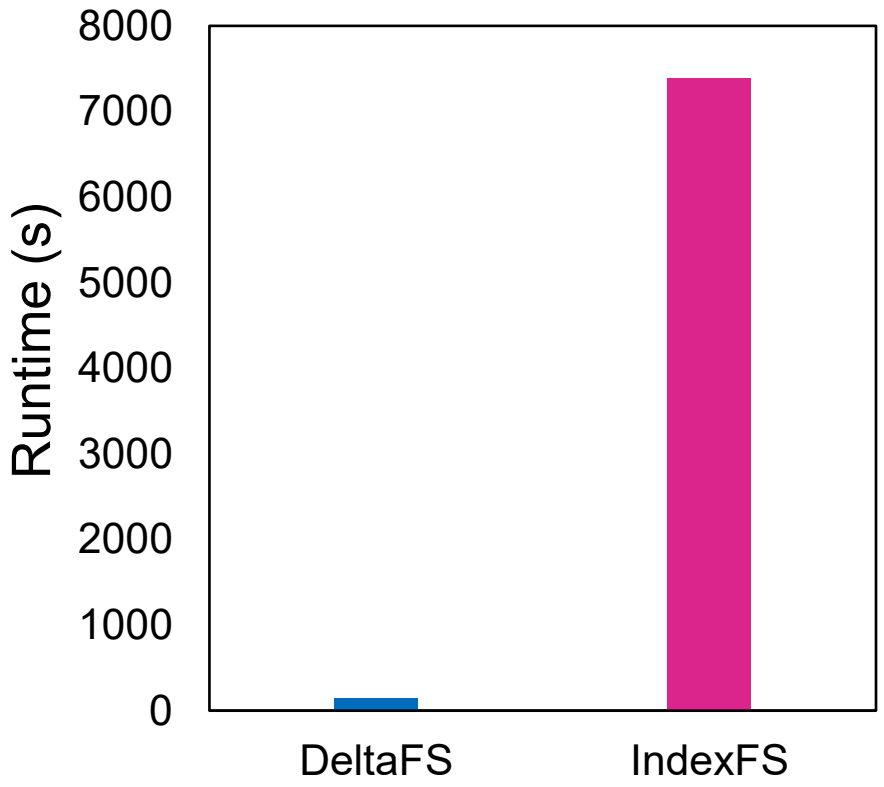
Cross-Job Data Sharing

- **Trivial** when the FS maintains a global namespace and immediately serializes all namespace changes
- DeltaFS requires a job to **explicitly specify logs of previous jobs** for input
- And to **explicitly run log compaction** for fast namespace queries

Recall that logs are published at registry and can be recalled by log names.

High Cross-Job Performance

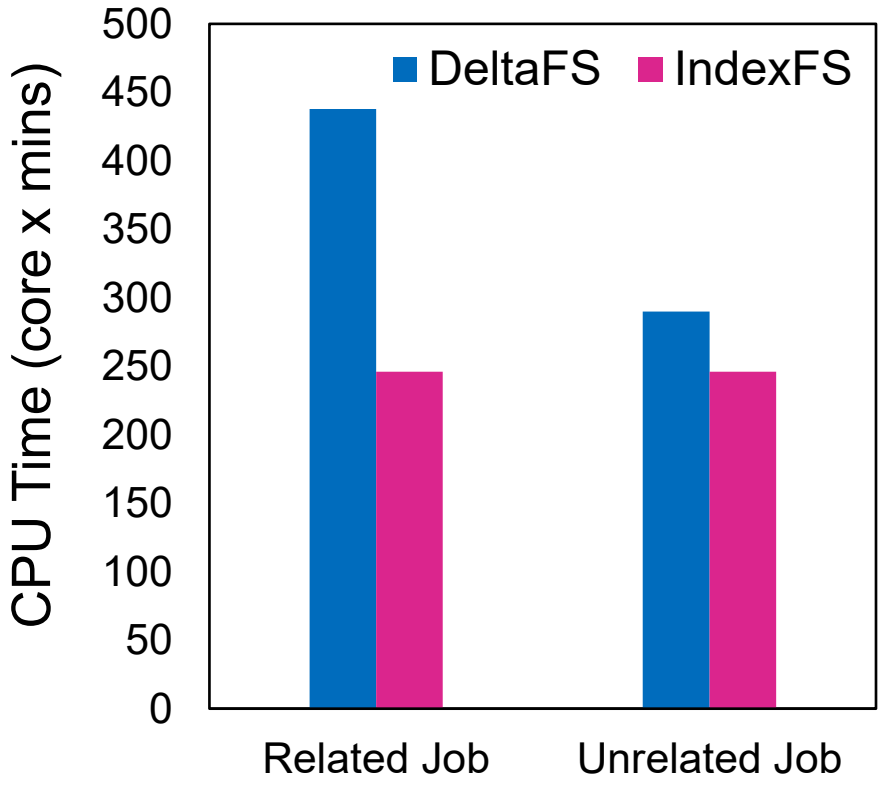
- Even though DeltaFS jobs need to explicitly read and compact logs, **DeltaFS still shows higher performance**
- This is because today's FS relies on dedicated MDS to perform all metadata operations while DeltaFS can leverage client cores to speed up metadata processing (*including log compaction*)



DeltaFS speeds up a 7-stage workflow execution 49x

Cost of No Dedicated Metadata Server

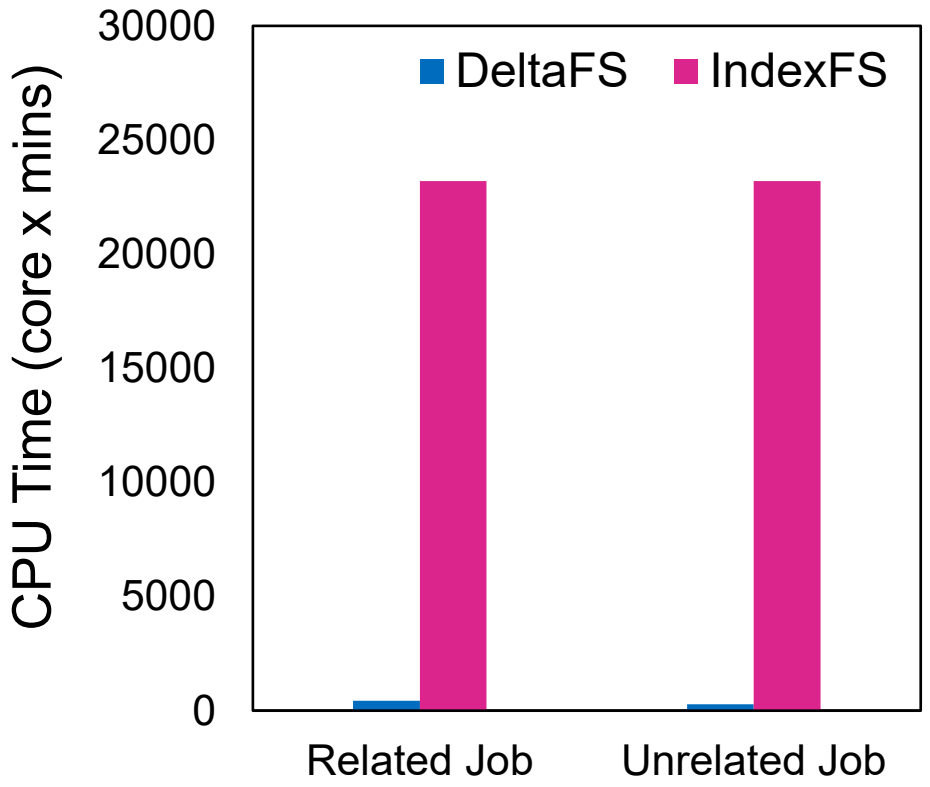
- DeltaFS requires **more CPU resources** for executing the same set of metadata operations (*due to potentially repeated log compaction)*
- Today's FS uses less CPU time despite taking more wall time to finish



DeltaFS consumed up to 2x more compute resources in the form of CPU core minutes

Today's FS Wastes Client Resources

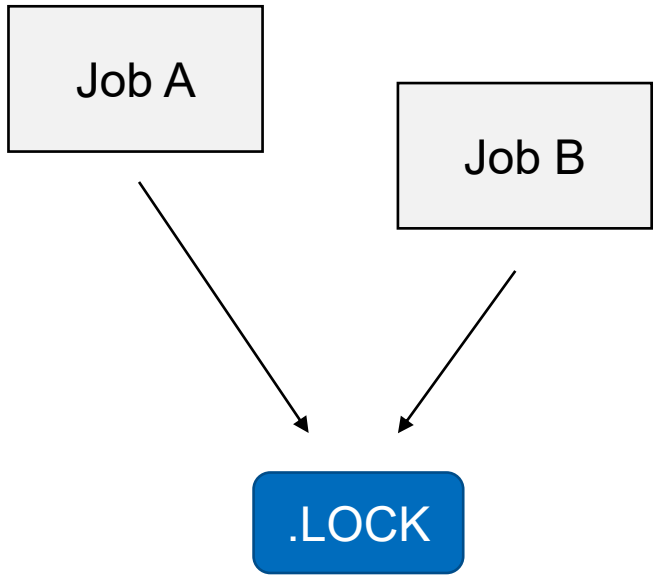
- While today's FS uses less server-side CPU time, it wastes massive client resources **by blocking them** doing nothing
- **DeltaFS** leverages client resources for scalable metadata, resulting in much better overall resource usage (client + server)



DeltaFS improves overall resource usage up to 52x

What If Jobs are Using the FS to Communicate?

- DeltaFS **redirects** them to a mechanism outside the FS (e.g., ZooKeeper) for synchronization
- Users explicitly declare a subdir as "**_Atomic**"
  - *Analogous to C's atomic variables*
- **Key point:** No need to treat all metadata operations as atomic



For example: Jobs A and B use a marker file (.LOCK) for anonymous synchronization

Conclusion

- Strong **scalability** needs strong **decoupling**
  - Existing FS clients communicate too often with their servers
  - Removing servers forces us to rethink on what is necessary
- **DeltaFS** converts FS metadata to per-job log operations
  - Job communicate only when needed
- At exascale and beyond, **one-size-for-all** will continue to lose value
  - Need to try radically different models for shared storage

Thank you!

Please see our DeltaFS papers for more information

DeltaFS Papers and Presentations

- **DeltaFS**: A Scalable No-Ground-Truth Filesystem For Massively-Parallel Computing. In *SC* 21, https://doi.org/10.1145/3458817.3476148
- Streaming Data Reorganization at Scale with **DeltaFS Indexed Massive Directories**, In *ACM TOS* 16/4, https://doi.org/10.1145/3415581
- **Beyond DeltaFS**: Designing Storage Systems to Support HPC and AI Workloads, *SDC* 2019
- **Breaking the Metadata Bottleneck**: the Exascale Filesystem DeltaFS as a LANL and Carnegie Mellon Collaboration, *SDC* 2019
- https://www.nextplatform.com/2021/07/14/will-deltafs-become-the-file-system-of-exascales-future/
- **DeltaFS**: Exascale File Systems Scale Better without Dedicated Servers, In *PDSW* 15, https://doi.org/10.1145/2834976.2834984

Reference

- **NASD**: A cost-effective, high-bandwidth storage architecture, In *ASPLOS* VIII, https://doi.org/10.1145/291069.291029
- **GPFS**: A Shared-Disk File System for Large Computing Clusters. In *FAST* 02, https://dl.acm.org/doi/10.5555/1083323.1083349
- **Lustre**: Building a File System for 1000-Node Clusters. In Annual *Linux Symposium* 03, https://www.kernel.org/doc/ols/2003/ols2003-pages-380-386.pdf
- **OCFS2**: The Oracle Clustered File System, Version 2. In Annual *Linux Symposium* 06, https://www.kernel.org/doc/ols/2006/ols2006v1-pages-289-302.pdf
- The **GFS2** Filesystem. In Annual *Linux Symposium* 07, https://www.kernel.org/doc/ols/2007/ols2007v2-pages-253-260.pdf
- The **Hadoop** Distributed **File System**. In *MSST* 10, https://doi.org/10.1109/MSST.2010.5496972
- **Ceph**: A Scalable, High-performance Distributed File System. In *OSDI* 06, https://dl.acm.org/doi/10.5555/1298455.1298485
- **IndexFS**: Scaling File System Metadata Performance with Stateless Caching and Bulk Insertion. In *SC* 14, https://doi.org/10.1109/SC.2014.25

