Microsoft Azure

Extant Challenges to Efficient HSM Integration with Cloud-deployed Lustre

Ellis Wilson

Agenda

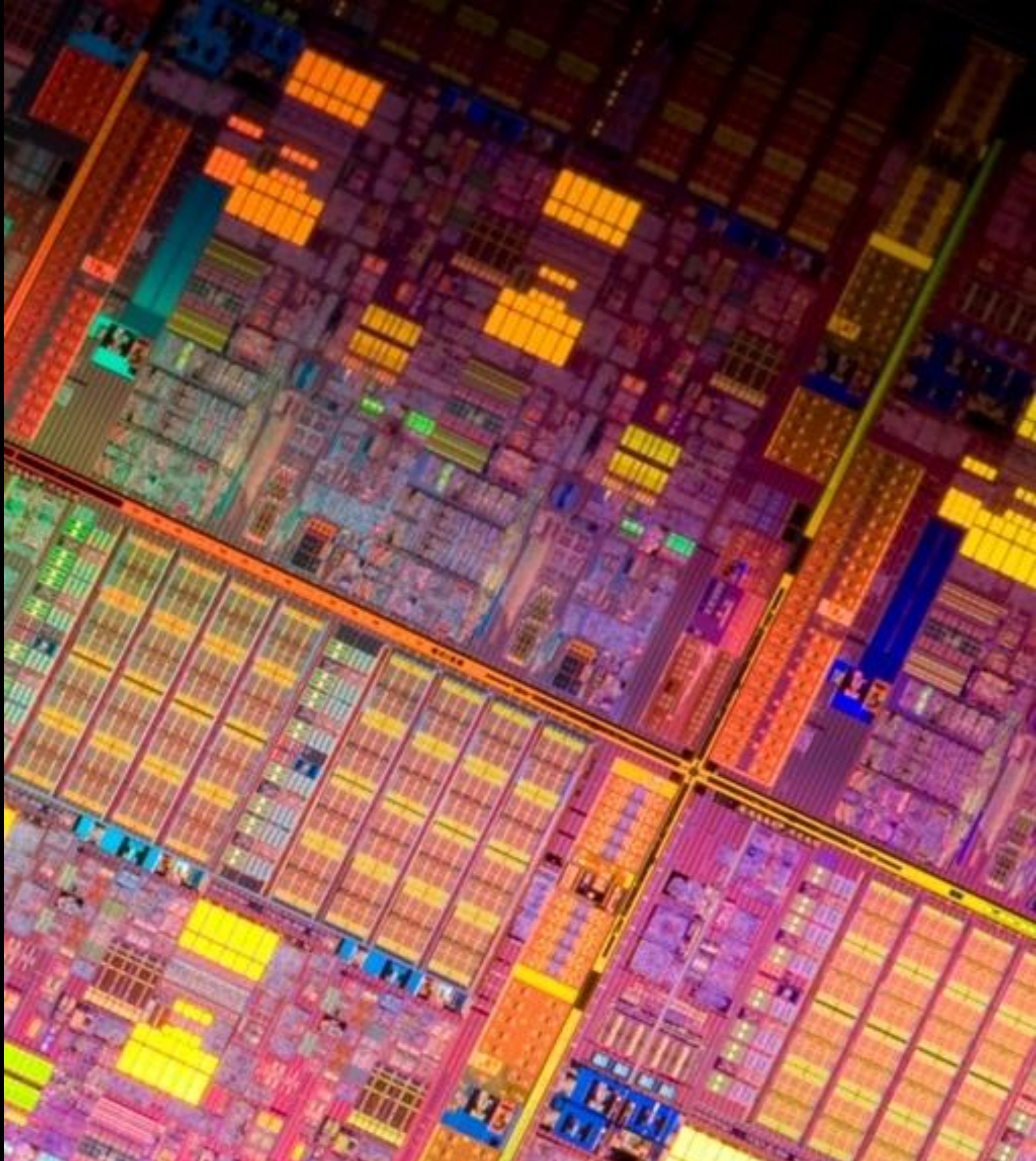
Review of HSM Architecture in Lustre

Typical On-Premise HSM/DR Usage

New Usage Patterns in Cloud-deployed Lustre

Architectural Challenges

Potential Areas of Improvement



Review of HSM Architecture in Lustre

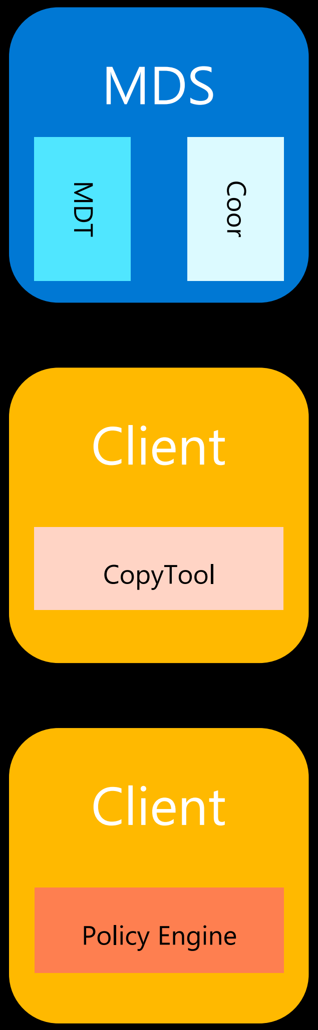
Hierarchical Storage Management (HSM)

A software architecture or framework that facilitates the movement of data between cost- and performance-differentiated storage tiers

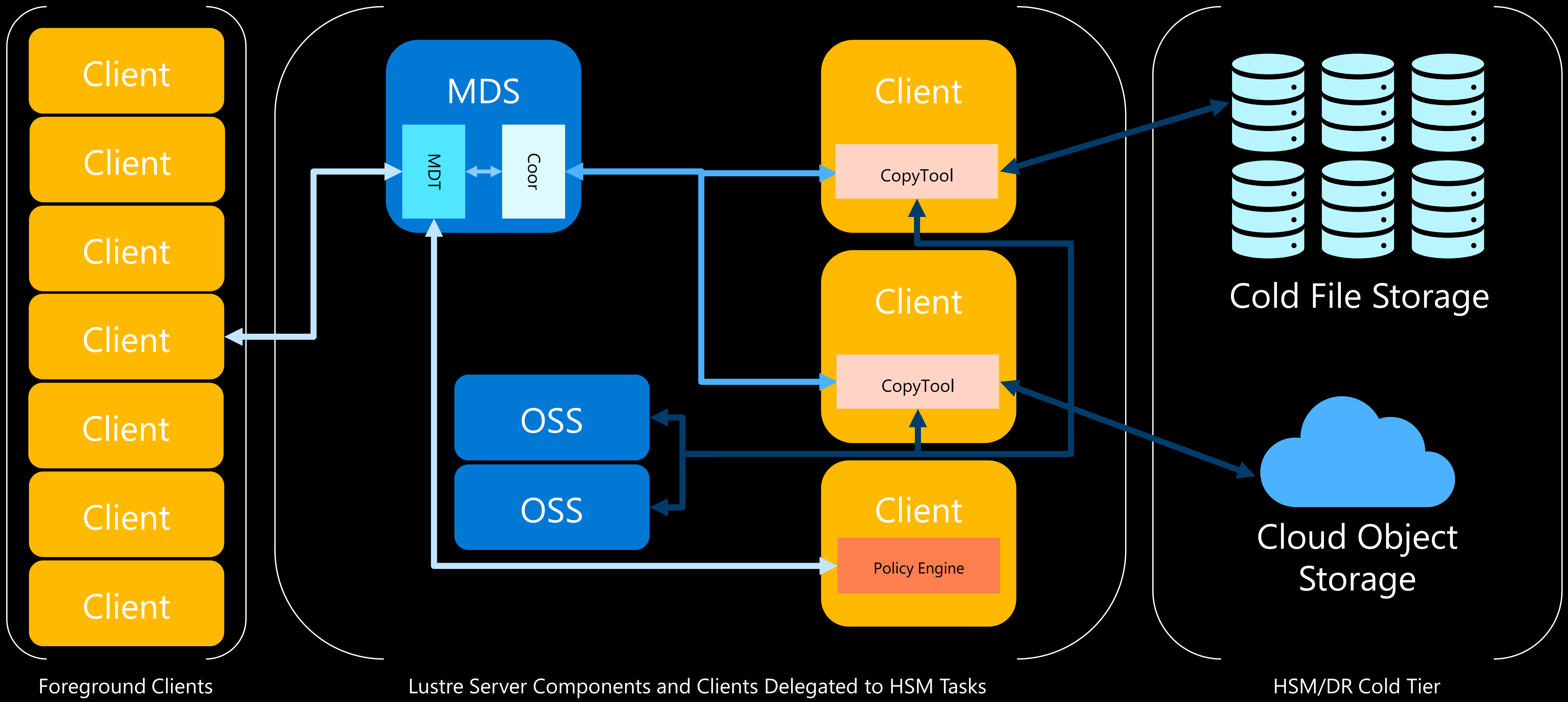
Leadership examples of HSM perform this movement automatically, and present a unified namespace to the end-user regardless of the location of the data

The HSM Components of Lustre

- MDT (MDS node, kernel-space)
  - Tracks file state(s): NONE, EXISTS, DIRTY, RELEASED, ARCHIVED
  - Accepts HSM actions against files, translates these, and sends HSM commands to Coordinator
- HSM Coordinator (MDS node, kernel-space)
  - Receives HSM commands from and communicates status back to MDT
  - Queues HSM work
  - Registers, sends work to, and communicates with CopyTools on work status
- CopyTool (Client node(s), user-space)
  - Registers locally to receive HSM work from Coordinator
  - Data mover that understands how to send/receive data to/from lower tier
- Policy Engine (Client node, user-space) (optional)
  - Initiates HSM commands against files in accordance with user-specified policies



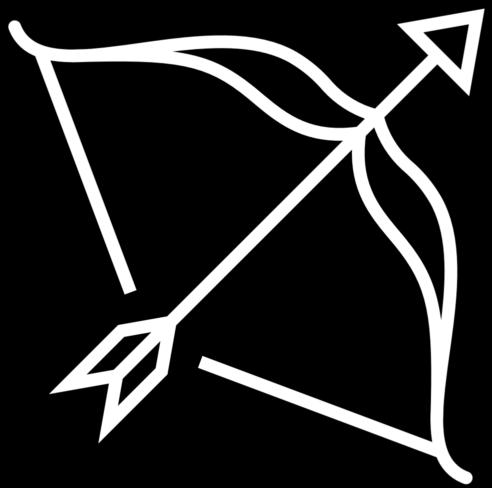
(Simplified) Diagram of HSM Architecture in Lustre



Note on the External Policy Engine

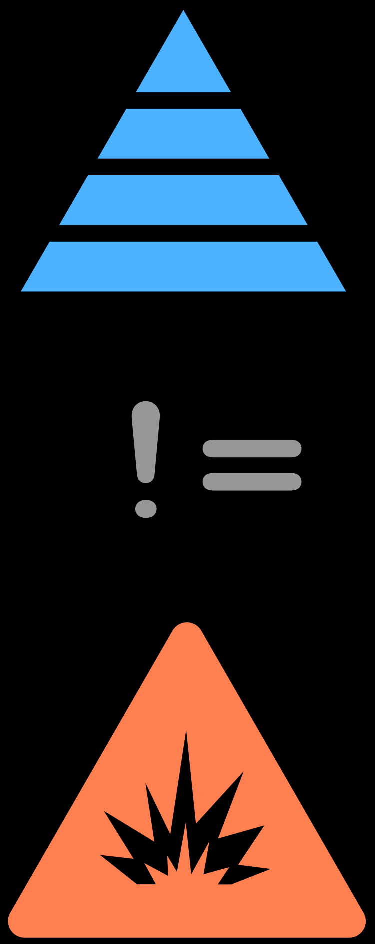
- While optional, without it most operations are manual
  - No component internal to Lustre automatically archives or releases cold data
  - Intractable to manage a very large namespace this way

- Robinhood is the most popular incarnation
  - Tracks full namespace of Lustre in SQL tables
    - Crucial to quickly answering: *Since time T, what has changed?*
    - Utilizes Lustre changelogs to efficiently track changes to system after initial filesystem walk
  - Users can build policies to be carried out on events
    - E.G., archive+release files sufficiently cold when above some capacity threshold



Typical On-Premise HSM/DR Usage

HSM vs DR



- HSM, like RAID, *is not Backup*
  - HSM solves the problem of how to accelerate the working set at the hottest tier of storage
  - Disaster Recovery (DR) solves the problem recovering from partial or total system failure
- On-premise solutions may employ one or both
- Replication targets may vary:
  - Another Lustre cluster (e.g., via lustre_rsync and changelogs)
  - A different filesystem or object storage entirely (e.g., Azure Blob object storage)

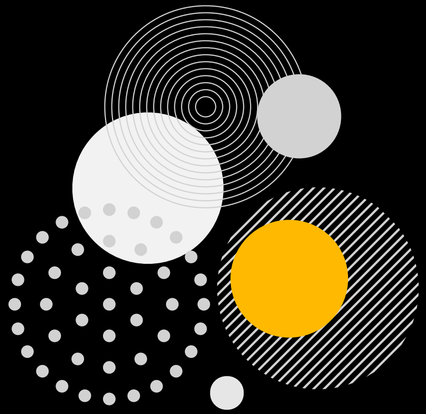
Typical On-Premise HSM/DR Usage

- Lifetime in Years: Storage architecture designed for 3+ years
  - HSM cold storage is deployed with a similar lifespan
  - Storage capacity and performance requirements are dictated by the most demanding jobs
    - Adding storage can be difficult so usually sized entirely up-front
- One-time Initial Standup Cost
  - Deployment/ingest may be very heavy, but should be a one-off
  - If moving between clusters initial Robinhood scan or DR replication may take days to weeks
- Track Changelogs: HSM or DR post-standup just track changes
  - New data is added; old data tends to not change
  - Robinhood and nightly replications (usually) process tractable amounts of churn
- Non-critical Path HSM/DR
  - DR and HSM tasks are rarely on any critical paths for the parallel file system
  - Archive+release of cold files may in fact be purposefully delayed until non-peak hours

New Usage Patterns in Cloud-deployed Lustre

New HSM/DR Patterns for Cloud-Deployed Lustre

- Hyper Transiency
  - Lustre clusters are erected on-demand for days, weeks, or months
  - “Scratch cleanup” achieved by full cluster deletion



- One Becomes Many
  - Trivial to deploy more Lustre filesystems and clients as jobs require
  - Avoids idle compute, network, and storage and side-step I/O contention
  - Resiliency achieved by cloud components rather than FLR or similar
  - Not-so-distant-future: imagine PBS jobs that specify Lustre cluster parameters
- Fully Amorphous Cold Tier
  - HSM “cold storage” is unbounded, unsized object storage
  - Sole sizing to be done recurringly is how much data has value to persist

Architectural Challenges

Performance Challenges from New Use Patterns

- Multiple arms of Lustre HSM enter the critical path:

- *Time-to-Hydrate*
  - On job start would like to standup just the namespace without the data
  - Lustre HSM import works, but is performance-constrained

- *Time-to-Restore*
  - Without explicit user pre-restoration, delays on every accessed file in released state

- *Time-to-Archive*
  - On job completion, the faster the customer can archive results and tear-down the better
  - Robinhood can struggle to keep up with massive changelog generation from ingest+churn

Architectural Impedance Mismatches

- Many aspects of Lustre HSM work great for cloud-deployed Lustre:
  - Perfect match for initial hydration of just metadata
  - Perfect match for erecting a job-sized Lustre FS atop a massive object store
  - Perfect match for solely archiving changed data
- Some architectural mismatches due to differing use-case:
  - *Directories*: Not supported whatsoever in Lustre HSM
  - *Minor file modifications*: CopyTools today operate against the entire file
  - *"Unique" file types*: symlinks, hard-links, sparse files
  - *Striping and Extended Attributes*: Not restorable via import API or tracked for archive
  - *File attribute changes*: May not trigger needs-archive in Lustre HSM
  - *Directory renames*: hierarchical filesystem vs. full-path-indexed object filesystem
- **At the end of Archive, would like a mirror achieved in the Cold Tier**

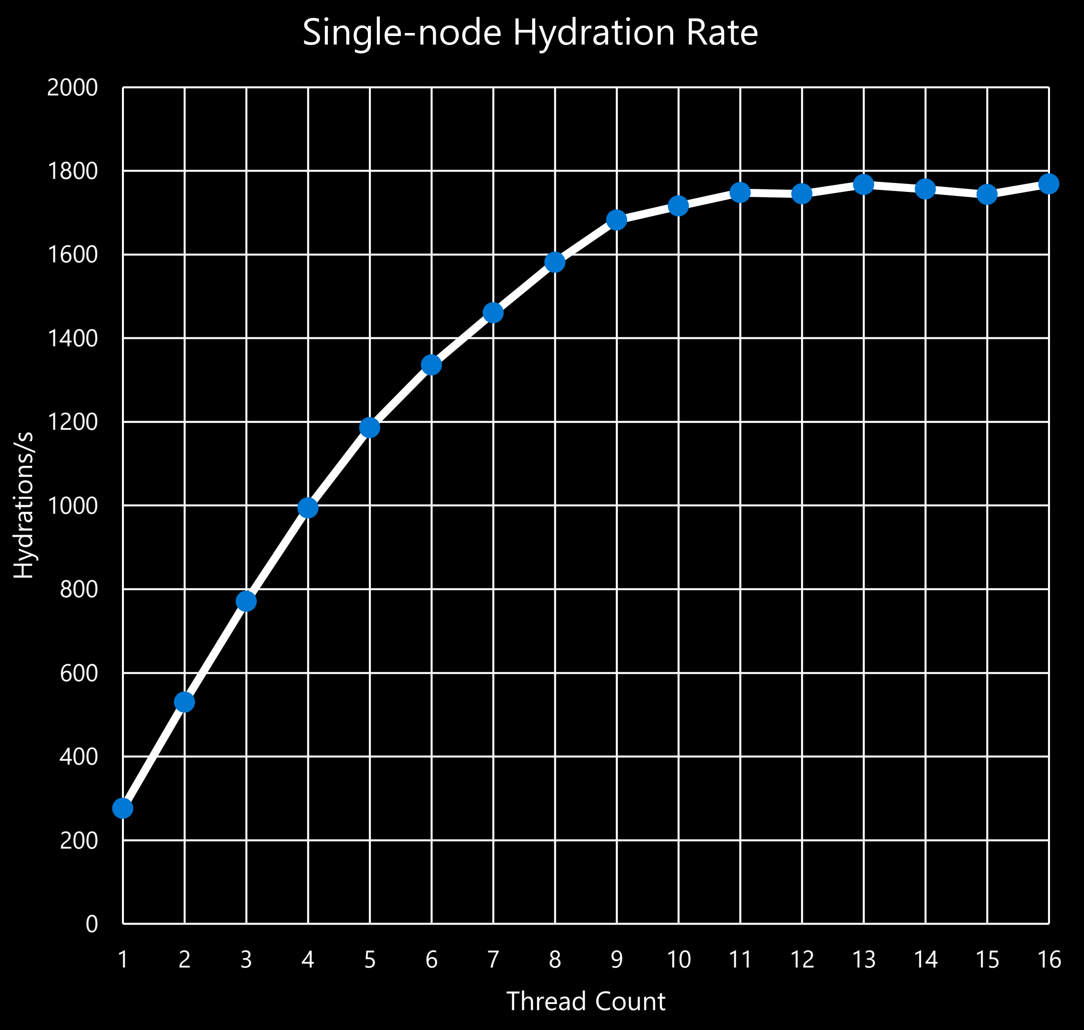
Potential Areas for Improvement

Potential (Tractable) Areas for Improvement

- Metadata Hydration:
  - Batch version of the import API (e.g., single syscall per smallish directory)
  - Reduced overhead checking mounts and taking locks
  - Implement support for setting stripes
- Data Restoration:
  - Restore-ahead – perhaps leverage stat-ahead existing code in a cautious manner
  - Batching improvements to the CopyTool API
- Final Archival:
  - Batching improvements to the CopyTool API
  - Native support for directories in HSM code (reach goal)

Examining Time-to-Hydrate Metadata

- Benchmarked using a very small (~50 LOC) pthread-enabled C program
  - Unique directory per-thread
  - Each thread calls llapi_hsm_import serially against 10,000 files within directory
  - Peaks at roughly 1/4 the rate this node can perform normal file creation
  - Lustre 2.14.0

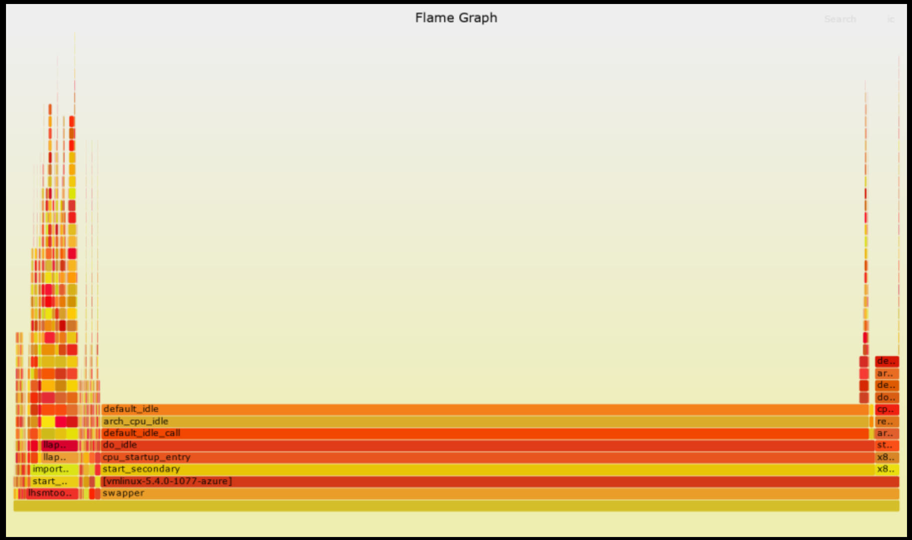


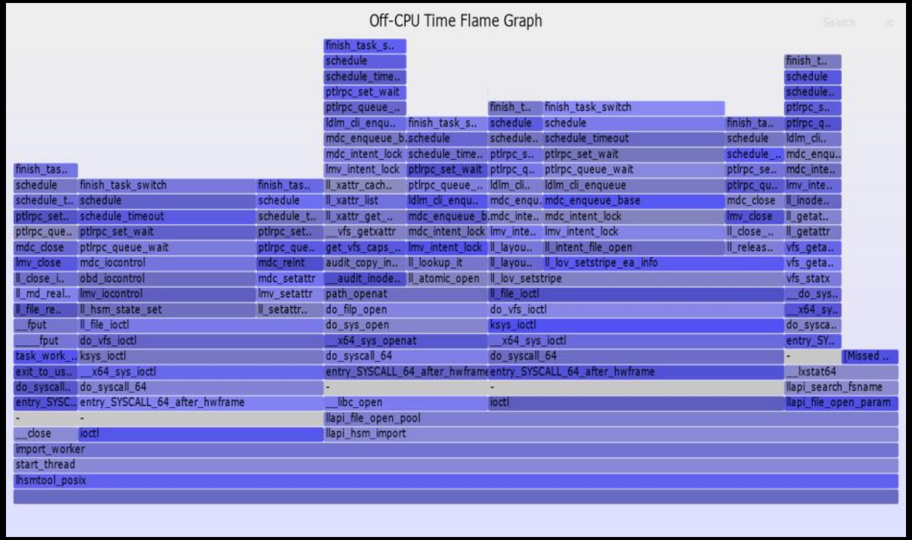
Bottlenecks in llapi_hsm_import

- On/Off CPU flamegraphs shown
  - Benchmark at 16 threads

- On-CPU time is very low
  - 0.62% at 1 thread, 5.9% at 16 threads
- Off-CPU shows many RPCs (6+) required to achieve relatively straightforward task of import
  - Most time spent following ptlrpc_set_wait
  - ll_lov_setstripe consumes 3 and doesn't even end up setting stripe
  - Can improve by bumping max_rpcs/peer_credits, but band-aid

- Craft a new import API that imports in batches and relaxes some guardrails during initial cloud import?





In Summary

- Existing HSM code lays solid foundation for transition to cloud
  - Many applications of Lustre HSM in the cloud are **very** different than on-prem
  - Cloud vendors need to show greater engagement with Lustre community
    - E.g., Discuss with Lustre developers on more substantive changes (e.g., enabling HSM directory change tracking)
- Careful performance analysis required to make this successful
  - Time-to-{Hydrate,Restore,Archive} will need study and optimizations
- Improvements for cloud-deployed Lustre lift on-prem Lustre boats
  - Metadata hydration, data restoration, and archival are critical regardless of location

Microsoft Azure

Thank you!

Questions?