MarFS and libNE Utility Development

Presenter: Daniel Perry

Mentors: David Bonnie,
Garrett Ransom

Managed by Triad National Security, LLC for the U.S. Department of Energy's NNSA

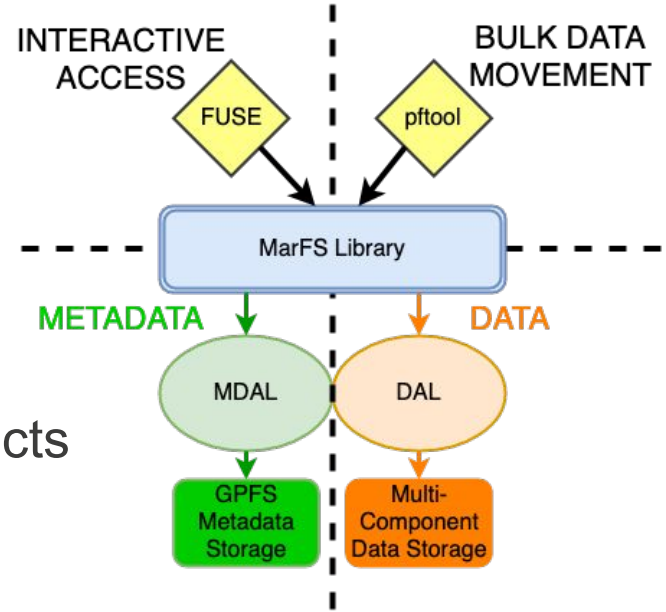
LA-UR-21-27942

Need for MarFS

- Data sets growing faster than archival storage can support
  - Trinity: 2PB of memory / 4PB of flash
  - Crossroads: ~1PB of memory ~10PB of flash
  - HPSS Archive: ~90PB Total, continuously expanding, ~3GB/sec
- Solution: Create intermediary capacity tier (MarFS)
  - Want long-term (months/years), high bandwidth storage
  - Parallel file systems: fast, but limited reliability
  - Tape: reliable, but slow and complicated to design
- Object storage is promising, but has limitations
  - Flat namespace allows for scalability
  - Erasure coding allows for cheaper disk media
  - Machines love object-IDs, people generally don’t
  - Potentially billions $ in applications expecting ‘POSIX-like’ file trees

What is MarFS?

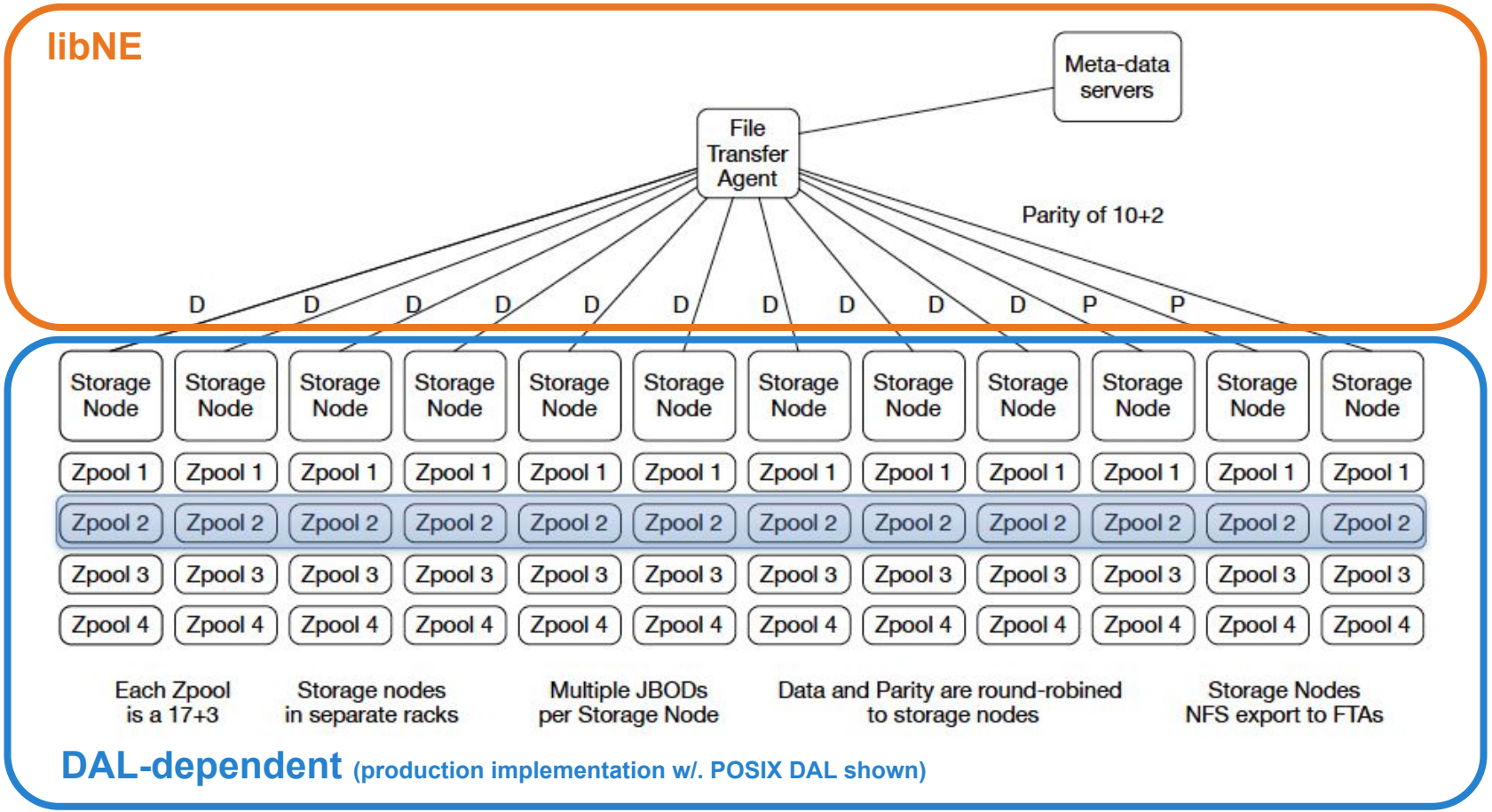
- A near-POSIX interface layered over distinct metadata and data implementations
  - Data stored as erasure coded objects
  - Metadata mirrored within a parallel file system
  - Object IDs written as extended attributes of metadata files
- Familiar semantics, fast metadata, stable objects
  - Metadata in a real PLFS gives us POSIX-style directory trees and permissions almost for free
  - Data as objects simplifies implementation and data protection
- Production system: 60PB capacity, 25 GB/sec
- Currently working on a significant refactor



Multi-Component System: libNE

- Cross-server erasure coding atop ZFS pools
  - Allows failure tolerance at both the disk and server level
    - More reliability allows the use of cheaper disk
    - Erasure coding performed through Intel's Storage Acceleration Library (isa-l)
  - Performance through parallelism
    - Threaded I/O to multiple servers
- Load-Balancing
  - Objects hashed and balanced across available servers
- Highly Resilient
  - Objects are accessible even after losing entire servers
- Transparent Architecture
  - Object stripes stored as files in ZFS plainly visible to administrators

Multi-Component System Structure



libNE DAL Implementations

- POSIX DAL
  - Utilizes standard I/O syscalls
- AWS S3 DAL
  - Calls functions from the libs3 library
- Recursive DAL
  - Built on another libNE instance (can handle multiple levels of recursion)
- Fuzzing DAL
  - Passes calls through to an underlying DAL while injecting errors
- NO-OP DAL
  - Temporarily caches data without any I/O access
- Timer DAL
  - Collects timing data from an underlying DAL

MarFS Interface Methods

- MarFS API
  - Utilized by other user-level interfaces
  - Similar to POSIX I/O syscalls (with several additions)
- FUSE (Filesystem in Userspace): Interactive Interface
  - Intended for metadata/read operations
  - Library providing a standard interface which allows filesystems to be implemented as an application in userspace (no kernel manipulation)
- pftool (Parallel File Tool): Batch Interface
  - Main method of data movement
  - Utility leveraging MPI to transfer/compare large groups of files, including across filesystems
  - Different classes specify how to handle different filesystem types

Questions?

Please direct further discussion to danmperry@lanl.gov

Special thanks to my mentors, managers, and all those who have provided assistance over the course of the summer.

Some details sourced from LA-UR-18-24846 and LA-UR-17-26375