Hewlett Packard
Enterprise

Experience Building DMF7 on Lustre

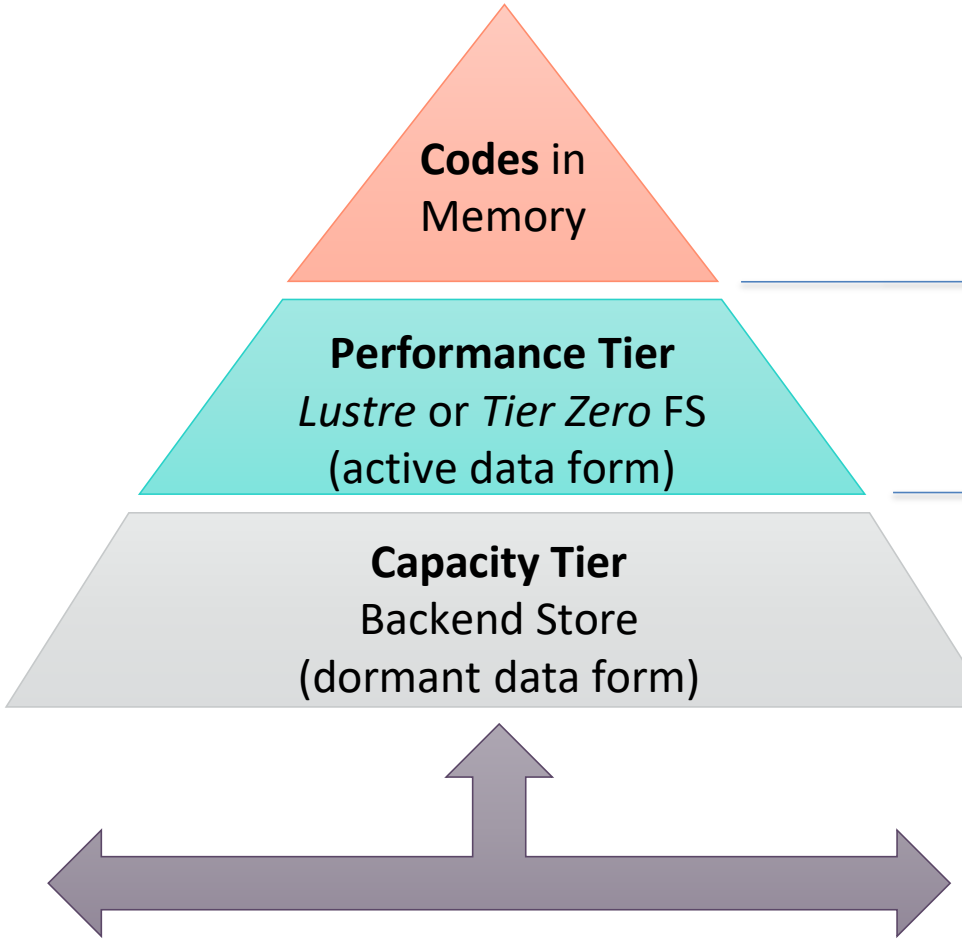
Olaf Weber
Master Technologist
HPC Data Management & Storage

Disclaimer

- The information contained in this presentation is proprietary to Hewlett Packard Enterprise (HPE) and may contain forward-looking information regarding products or services that are not yet available.
- Do not remove this slide from the presentation
- HPE does not warrant or represent that it will introduce any product to which the information relates
- The information contained herein is subject to change without notice
- HPE makes no warranties regarding the accuracy of this information
- The only warranties for HPE products and services are set forth in the express warranty statements accompanying such products and services
- Nothing herein should be construed as constituting an additional warranty
- HPE shall not be liable for technical or editorial errors or omissions contained herein

Data Management Framework 7

Data Tiers



HPC codes require POSIX
Buffered or Direct I/O
MPI-IO libraries

Highest bandwidth and IOPS
Limited namespace (up to 100M)
Flexible or no data redundancy

Good streaming performance
Very large namespace (over 10B)
Strong data protection

Multi-site support
Geo replication & Backup
Cloud integration

Data Management Framework 7

Hierarchical Storage Management → Tiered Data Management

Data Migration Facility (DMF6)

- Filesystem *is* the metadata database
- Entire namespace is in filesystem
  - Database does not have directory info
- File data is migrated transparently
  - Policy engine drives put/punch/get
  - Access drives get
- Migration leaves inodes in place
- Migration leaves extended attributes in place

Data Management Framework (DMF7)

- Separate Metadata Database for a filesystem
- Entire namespace is in Metadata Database
  - Metadata Database does have directory info
- Object Database tracks *all* known objects
- File data is migrated transparently
  - Policy engine drives put/punch/get
  - Access drives get
  - But only for *staged* files
  - Policy engine drives *destage*/*stage*
  - Other processes can also drive *destage*/*stage*
- *Destaging* removes inodes from the filesystem
- *Destaging* removes extended attributes

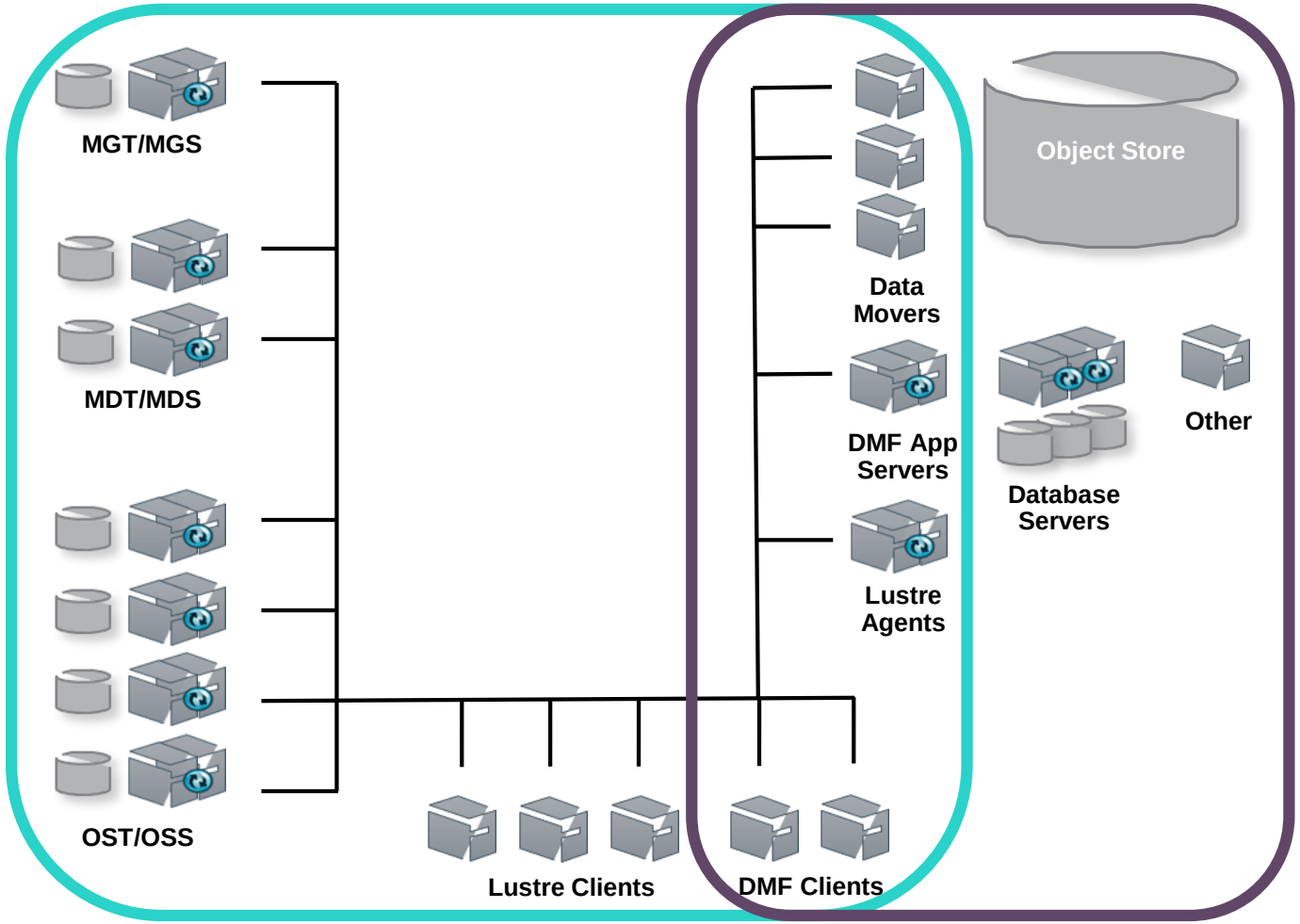
Data Management Framework 7

Features

- Redesigned from the ground up
  - Incorporate lessons from DMF6
- Designed for Tiered Data Management
- Designed for horizontal scaling
  - Scale by adding more servers
  - Distributed NoSQL database
- Many single-purpose components working together
- Most components are filesystem-agnostic
  - Lustre is the second filesystem to be supported, after HPE EXFS

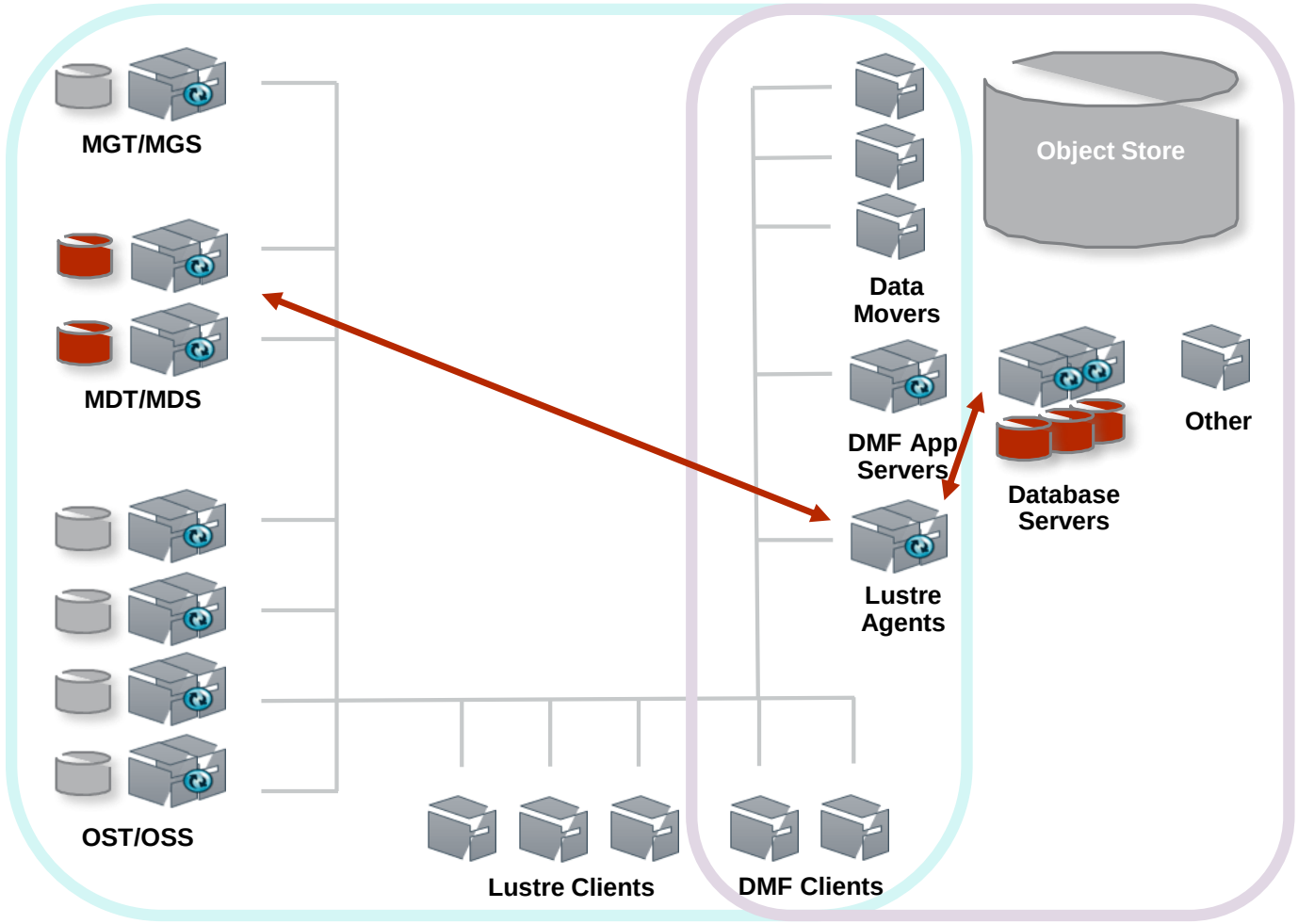
Data Management Framework 7 on Lustre

Roles of DMF7 nodes



- DMF Application Servers
  - Manage the other nodes
  - Provide the registry
  - Manage namespaces / filesystems
- Database Servers
  - Manage the DMF database
  - Policy Agent
- Data Movers
  - Move data between filesystem and backend
- Lustre Agents
  - Changelog processor
  - Filesystem scanner
  - Database scrubber
- DMF Clients
  - DMF CLI available

Filesystem Reflection



- Synchronized copy of filesystem metadata
  - Inode metadata
  - Directory tree
  - Extended attributes
  - HSM state
- Maps Lustre FIDs to Object Store
- NoSQL database
- Maintained by the Lustre Agents
  - Filesystem scanner
  - Changelog processor
  - Database scrubber
- Used by policy engine
  - Parallel data mover framework
  - Copytool interfaces with Lustre HSM

Interfacing with Lustre

Components that interface with Lustre

- Filesystem scanner
  - Walks the filesystem to fill in the filesystem reflection
  - Has to work in the presence of “overlong” pathnames
- Database scrubber
  - Checks the filesystem reflection for possibly-stale entries
  - Verifies on the filesystem whether the file is gone
- Changelog processor
  - Consumes Lustre changelog
  - Updates the filesystem reflection
  - Need filesystem access to obtain all necessary data
  - Most issues we encountered relate to changelog processing
- Copytool
  - Translates Lustre HSM interface to and from DMF7 internals
- Dispatcher
  - Forwards HSM requests from CLI
  - Validates file state on completion

Access a file by its FID

```
int llapi_open_by_fid(
        const char *lustre_dir,
        const struct lu_fid *fid,
        int flags)
{
        char mntdir[PATH_MAX];
        char path[PATH_MAX];
        int rc;

        rc = llapi_search_mounts(lustre_dir, 0, mntdir, NULL);
        if (rc != 0)
                return -1;

        snprintf(path, sizeof(path),
                 "%s/.lustre/fid/"DFID, mntdir, PFID(fid));
        return open(path, flags);
}
```

Most components require this ability:

– **`llapi_open_by_fid()`**

But it pays to examine the internals:

– **`llapi_search_mounts()`** is expensive

Use **`/<mount>/.lustre/fid/<fid>`**

Access a file by its FID

`/<mount>/.lustre/fid/<fid>`

– Provides a fixed name for every file and directory in the filesystem

– It is a fixed-length name, independent of the location of the file

Limitations

– Need to be root.

– **`open()`** yields **`ENXIO`** for device files

  – Do not put device files on shared filesystem

– **`open()`** yields **`ELOOP`** for symbolic links

  – This interface cannot dereference symbolic links

  – But we get **`ELOOP`** even with **`O_NOFOLLOW`**

– The FID encodes the MDT, so migration of an inode to another MDT changes the FID

Direct I/O

- On HPE EXFS we use direct I/O extensively for performance
- On Lustre Direct I/O performance is “disappointing”
  - Especially write performance
- Direct I/O is synchronous
  - The **`write()`** call returns after the OSS has responded
- Direct I/O can still be useful to avoid spoiling caches
  - But the application doing it should be multi-threaded or use aio

Lustre Changelog

```
# mkdir tmp

2112648 02MKDIR 09:30:25.501859712 2018.08.24 0x0
t=[0x200019271:0x2:0x0] ef=0xf u=0:0 nid=192.168.131.17@tcp1
p=[0x200000007:0x1:0x0] tmp

# chmod a+rwxt tmp

2112649 14SATTR 09:30:28.739566509 2018.08.24 0x14
t=[0x200019271:0x2:0x0] ef=0xf u=0:0 nid=192.168.131.17@tcp1

xfs_mkfile 1m file.1m

2112650 01CREAT 09:31:11.661327380 2018.08.24 0x0
t=[0x200019271:0x3:0x0] ef=0xf u=0:0 nid=192.168.131.17@tcp1
p=[0x200019271:0x2:0x0] file.1m

2112651 13TRUNC 09:31:11.741270796 2018.08.24 0xe
t=[0x200019271:0x3:0x0] ef=0xf u=0:0 nid=192.168.131.17@tcp1

2112652 11CLOSE 09:31:11.747861801 2018.08.24 0x243
t=[0x200019271:0x3:0x0] ef=0xf u=0:0 nid=192.168.131.17@tcp1

# touch file.1m

2112653 11CLOSE 09:36:18.556856115 2018.08.24 0x42
t=[0x200019271:0x3:0x0] ef=0xf u=0:0 nid=192.168.131.17@tcp1
```

- Tracks Metadata changes
  - Updated by MDS
  - Stored on MDT
  - Part of filesystem transactions
- Can only be read on Lustre client nodes
  - Must be root or equivalent
- Three types of metadata changes
  - Namespace
  - Side effects
  - Audit trail
- Controlled by per-MDT event mask
- Not a full log of the filesystem actions
  - Tracks that something changed…
  - …but not necessarily what changed

Lustre Changelog Records

```
2112648 02MKDIR 09:30:25.501859712 2018.08.24 0x0 t=[0x200019271:0x2:0x0] ef=0xf
u=0:0 nid=192.168.131.17@tcp1 p=[0x200000007:0x1:0x0] tmp
```

– A Lustre Changelog Record provides the following information:
  – Record index (a sequence number)
  – Record timestamp in nanoseconds
  – Record type (kind of metadata change)
  – The FID of the object (file or directory) affected
  – If appropriate, the FID of the parent directory and the FIDs of the other objects involved in an operation
  – If appropriate, the file name or names involved in an operation
  – If appropriate, the name of the extended attribute affected
  – Stuff we don't care about
    – User and Group ID of process making the change
    – Project ID of process making the change
    – NID (Lustre's way of identifying a cluster node) of node making the change
    – If appropriate, flags given to `open()`

Lustre Changelog Issues

The Changelog Reader ID

– A *Changelog Reader ID* is used to track when entries can be removed from the MDT

– The Reader ID is registered on the MDS for the MDT

```
# lctl changelog_register
# lctl changelog_deregister
```

– The Reader ID is *used* on the Lustre client that reads the changelog

  – The *only* use is clearing entries

```
# lfs changelog_clear
llapi_changelog_clear()
```

  – A Lustre client cannot tell whether a reader ID is valid, except by trying to clear entries

– If you forget that a reader was registered…

  – You may not notice until space on the MDT runs out

  – We made this mistake during testing, and only noticed after a few weeks

  – There were several *billion* changelog entries on disk at that point

  – Accidentally a useful stress test

Lustre Changelog Issues

The Start-Stop Interface

Client-Side

– Start reading

`llapi_changelog_start()`

– Read records

`llapi_changelog_recv()`

– Stop reading

`llapi_changelog_fini()`

– Repeat

  – **`CHANGELOG_FLAG_FOLLOW`** is defined…

  – …but not implemented

  – Pause between repeats if there are no new records

Under the Hood

– The client asks the MDS for the records for the MDT

– The MDS starts a kernel thread to handle the work

  – This tread reads the records

  – Then pushes them to the client

  – On reading the last available record the thread exits

– Implementing follow semantics looks to be difficult

Lustre Changelog Issues

Out-of-order entries

– We have seen cases where the index of subsequent entries goes like this:
  – **2112648**
  – **2112649**
  – **2112651**
  – **2112650**
  – **2112652**
  – **2112653**
– A changelog processor needs to handle this with some care
  – Clearing to **2112651** before **2112650** has been read causes **2112650** to be lost
– Not seen with 2.10 or later MDS (yet?)
  – No LU filed for now

Lustre Changelog Issues

Oddball Errors

– Before Lustre 2.10 the changelog reading loop gets **`EPROTO`** errors in some cases
  – These are returned by **`llapi_changelog_start()`**
  – Just call **`llapi_changelog_fini()`** and retry
  – Fixed in 2.10 when the internals of the interface were rewritten
– In Lustre 2.10 and later you get records with type **`-1`**
  – Treat as **`SETXATTR`** records
  – **`LU-10579`**
– In Lustre 2.10 and later you can get spurious HSM records
  – “File successfully archived”
  – **`LU-11258`**
– There is an issue with directories that are striped across MDTs
  – The “parent directory” fid for **`CREATE`** records refers to a stripe, not the directory as a whole
  – **`LU-10283`**

Lustre Changelog Wishlist

- Implement **`CHANGELOG_FLAG_FOLLOW`**
  - As noted earlier, this may be a non-trivial amount of work
- Per-reader changelog masks
  - Lets each changelog reader choose which changelog events to receive
- Have changelog timestamp match operation timestamps
  - In particular, ctime/mtime/atime timestamps of inodes
  - Avoids the need for **`stat()`** calls after namespace changes (create/rename/link/unlink)
- Reading changelog on MDS
  - Would eliminate a network hop
  - Preferably combined with fast access to inode state
  - In a multi-MDT setup, we may need inode state from a different MDS – for namespace changes

Copytool Registration

– A copytool registers with a filesystem using **`llapi_hsm_copytool_register()`**

– For HA purposes we want to be able to re-register on a different node

– The interface does not really provide for this

– Solution: start copytools on all the applicable nodes and sort out the complications from there

  – The DMF7 copytool only manages the translation to and from the Lustre HSM interface

Preliminary Performance Numbers

Engineering cluster

Cluster

- DMF
  - 5 x DMF application server / database server / Lustre agent
  - 3 x DMF mover
  - SATA SSD for DMF database
- Lustre
  - Cray appliance
  - 3 x SSU
  - 800TB
  - EDR Infiniband
- CEPH
  - 3 RGW nodes
  - EDR Infiniband

Performance

- Filesystem scan
  - 5 million entries
  - 25 minutes
- Data movement
  - Put: 8.4 GB/s
  - Get: 3.3 GB/s
- Changelog
  - 9000 to 10000 records/s

Hewlett Packard
Enterprise

Thank you

Olaf Weber
olaf.weber@hpe.com