Clustered Metadata

Andreas Dilger
Sun Microsystems
Agenda

- What is CMD?
- How does it work?
- What are FIDs?
- CMD features
- CMD tricks
- Upcoming development
Lustre Scalability with CMD

Capacity will be 100's of billion of files
Throughput will grow to a million operations per second
What is CMD?

- Clustered MetaData allows storing the filesystem namespace spread over multiple MDTs
- First version was developed about 4 years ago as a part of Hendrix project
- Working on 3rd version of CMD and it is currently being tested internally
CMD Benefits

- Better metadata performance due to parallel access from different clients
- Scale memory, network, and disk IO cost-effectively
- Increase total metadata capacity
- Parallel object creation on OSTs from different MDSes
- Parallelize big directories by splitting and storing them on multiple MDTs
How does it work?

- Cluster has a number of MDS nodes which communicate with each other.
- Each MDS has an independent MDT file system for storage.
- All clients connect to all MDSes and request root data and volume stats.
- All clients do operations (getattr, setattr, unlink) directly with the MDS that holds needed part of namespace.
What are FIDs?

- FID (File identifier) is cluster-wide 128-bit unique identifier
- FID contains 64-bit sequence number (SEQ), 32-bit object id (OBJ), and 32-bit version number (VER)
- FID itself does not contain backing-store related information like inode number/generation, or MDS number
- Uses sequence number as index in FID Location Database to find MDT
- FID also stored in OI (object index) for FID->inode mapping on each MDT
FID Location Database

- FLD (FID Location Database) records which MDT holds each FID sequence
- All FIDs in one sequence live in the same MDT
- CMD uses FLD to find out which MDT should be contacted to perform an operation on an object
FID Location DB/Object Index

SEQ=51  OBJ=93  VER=0

FUTURE USE BY SNAPSHOTS

MDT0
SEQ 0..47
SEQ 48..63
SEQ 64..72

MDT1
SEQ 1B..

FID Location DB (FLD)

Object Index (OI)

OBJ 47  INO 99
OBJ 88  INO 157
OBJ 93  INO 332
OBJ 104  INO 20

INO 332

INO 33  INO 39
INO 133  INO 3132

INO 99  INO 157  INO 332  INO 20

Regensburg HPC Workshop 2009
Create a File (local inode)

- Client generates FID for new file and sends create RPC to the MDS which holds the parent directory.
- This MDS inserts \{filename, FID\} into the directory and allocates a local inode.
- This happens for all non-directory inodes, so is the common case for file creates.
- FID is chosen by client to put inode on the same MDS as the parent directory.
Create a directory (remote inode)

- Client generates FID for new object and sends create RPC to the MDS which holds the parent directory (call this MDT1)
- MDT1 inserts {filename, FID} in dir
- MDS1 finds (through FID Location Database) which MDS should hold new file inode (call this MDS2)
- Create RPC sent from MDS1 to MDS2 to create the new directory with FID
- This happens for new subdirectories to balance load across MDSes, and also in the case of hard-links across MDSes
Creates on Local and Remote MDS

Create dir2/f2: 'dir2' lives at MDS2

Mkdir dir1/d1: 'dir1' lives at MDS1

Create inode for f1
Split Directories

- One user-visible directory stored in multiple MDT directory objects
- Hash of filename + directory layout map filename to MDT + directory object
- If we create file in split directory, directory 'striping' attribute allows client to decide which MDT holds a given filename for RPCs
- This allows parallel access and more scalability for single directories
- Currently specified at mkdir time
- Very similar to striped files
2-MDS Split Directory

Create bigdir/f1 hashing 'f1' -> MDS2

Create bigdir/f2 hashing 'f2' -> MDS4
Operations and Updates

- An *Operation* is a complete change in the filesystem namespace
  - Corresponds to a VFS-level syscall
    - mkdir, rmdir, create, link, unlink, symlink, rename
  - May involve one or multiple MDTs
  - Need to ensure Operation consistency

- An *Update* is a component of an Operation
  - An operation may have 1 or more updates
  - Cannot possibly do updates atomically
Recovery with CMD

- Multi-node operations need to be atomic to ensure namespace coherency
- This is hard
- Keep operations local if possible
  - Create files on same MDT as parent
  - Most getattr, unlink to one RPC
  - Faster, can use local filesystem recovery
- Non-local operations order updates for safety
  - Create remote inode first, commit
  - Insert local directory entry second, async
  - Any crash leaves consistent namespace
  - At worst unused inode is leaked, clean later
Minor Issues

- Split operations depend on updates on 2-4 MDTs
  - Latency is twice as long, and recovery crosses multiple nodes
  - Multiple MDTs can make up for reduced performance for the less-common ops
- Check for rename of directory into subdirectory is more complex as it needs to check more than one MDS
  - Cross-MDS renames not very common
  - Only needed for multi-parent renames
- Replication for FLD, root inode
Current State of CMD

- Largely finished development, and it passed many tests
- Users can currently investigate this feature, without any warranty
- Recovery not yet complete
- 2.0 release (end 2009) has much of the CMD functionality, unsupported
- Likely released as production in 2.2 release (late 2010)
Thank you

Andreas Dilger
<adilger@sun.com>