Data On MDT Solution Architecture

Introduction

Lustre* file system read/write performance is currently optimized for large files (where *large* is more than a few megabytes in size). In addition to the initial file open RPC to the MDT, there are separate read/write RPCs to the OSTs to fetch the data, as well as disk IO on both MDT and OST for fetching and storing the objects and their attributes. This separation of functionality is acceptable (and even desirable) for large files since one MDT open RPC is typically a small fraction of the total number of read or write RPCs, but this hurts small file performance significantly when there is only a single read or write RPC for the file data. The Data On MDT (DOM) project aims to improve small file performance by allowing the data for small files to be placed only on the MDT, so that these additional RPCs and I/O overhead can be eliminated, and performance correspondingly improved. Since the MDT storage is typically configured as high-IOPS RAID-1+0 and optimized for small IO, Data-on-MDT will also be able to leverage this faster storage. Used in conjunction with the Distributed Namespace (DNE), this will improve efficiency without sacrificing horizontal scale.

In order to store file data on the MDT, users or system administrators will need to explicitly specify a layout at file creation time to store the data on the MDT, or set a default layout policy on any directory so that newly created files will store the data on the MDT. This is the same as the current Lustre mechanism to specify the stripe count or stripe size when creating new files with data on the OSTs.

The administrator will be able to specify a maximum file size for files that store their data on the MDT, to avoid users consuming too much space on the MDT and causing problems for the other users. If a file's layout specifies to store data on the MDT, but it grows beyond the specified maximum file size the data will be migrated to a layout with data on OST object(s). It will also be possible to use the file migration mechanism introduced in Lustre 2.4 as part of the HSM project to migrate existing small files that have data on OST to the MDT.

Use Cases

New files are created with an explicit layout to store the data on the MDT.

New files are created with an implicit layout to store the data on the MDT inherited from the default layout stored on the parent directory.

A small file is stored without the overhead of creating an OST object, and without OST RPCs.

A small file is retrieved without the overhead of accessing an OST object, and without OST RPCs.

Users, administrators, or policy engines can migrate existing small files stored on OSTs to an MDT.

A client accesses a small file and has the file attributes, lock, and data returned with a single RPC.

An administrator sets a global maximum size limit for small files stored on the MDT(s). Files larger than this value do not store their data on an MDT.

Solution Requirements

Design a new layout for files with data on MDT.

There needs to be some way for the client to know that the data is stored on the MDT. Currently there is no mechanism to record this information. The new DOM file layout is being designed in conjunction with the Layout Enhancement Solution Architecture. The DOM layout will only have a single implicit data stripe, which is the MDT inode itself.

Client IO stack must allow IO requests to an MDT device.

The current client IO stack is implemented to perform IO through the OSC layer to OSTs. There is no IO mechanism for the MDC layer. Storing data on the MDT requires that IO be read and written through the MDC layer.

Explicitly allocating files on an MDT by default directory striping.

Since Lustre allocates the file layout when the file is first opened, the DOM layout needs to be chosen before any file data is actually written. This

means it is not possible to use the file size or amount of data written by the client to decide after the file is opened whether the file data will reside on the MDT or OST. Applications would be able to explicitly specify a DOM layout for newly-created files using existing llapi interfaces. It would also be possible to inherit the default layout from the parent directory to allocate all new files in that directory on the MDT to avoid any changes to the application.

It would be possible to allow applications to `mknod()` a file and then `truncate()` it to the final size to allow the MDT to make the layout decision before the file is first opened. This was handled in most previous versions of Lustre, but was removed in recent versions since it was never properly documented and never used by applications to our knowledge. This mechanism would be useful for applications that know (or can reasonably estimate) the final file size in advance, such as `tar(1)` or `cp(1)` or applications writing a fixed amount of data to a file (e.g. known array size, or fixed data size per task).

A mechanism to perform migration from MDT to OST for newly created file that immediately exceed small file size limit.

If a client writes to a file with a DOM layout that immediately exceeds the small-file size limit, under current locking behavior clients would need to flush their dirty cache to the MDT and cancel their layout lock before changing the file layout. Clients should be able to avoid this overhead. Instead, it would be preferable to change the layout to contain an OST object and flush the data directly from the client cache to the OST, and not store anything to the MDT. One possibility is to create all small files with an OST object to handle the overflow case, but this would add overhead when it may not be needed. Another option is to allow the client PW layout lock for the object on the MDS and then the client can modify the layout directly on the MDT without having to drop the layout lock or flush its cache.

Functional Requirements

Administration

- system admin sets filesystem-wide filesize limit via procfs or lctl set_param for DOM feature
- system admin sets default layout directly on file or dir or fs using lctl setstripe

Small file size limit

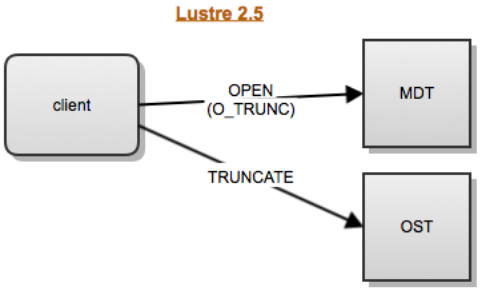
- The small file size limit is defined for small files and means that file growing beyound that limit should be migrated to OSTs.
- Migration process includes new layout creation for that file, object(s) allocation on OST(s) and data transfer from MDT to the OST object(s).

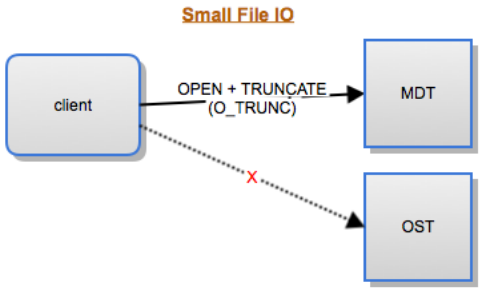
MDS_GETATTR request

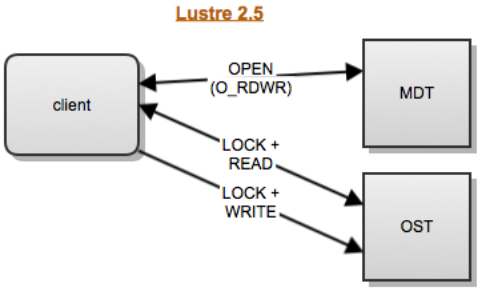
- Client to get LDLM locks and file size in addition to other metadata attributes from the MDT in the same RPC.

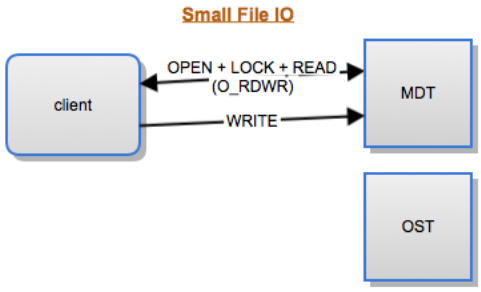
MDS OPEN optimizations

- Truncate to be done immediately during open with O_TRUNC
- files opened for write can also return data immediately if it is small to fit into the client reply buffer (also used for the return of large file layouts for widely striped files)







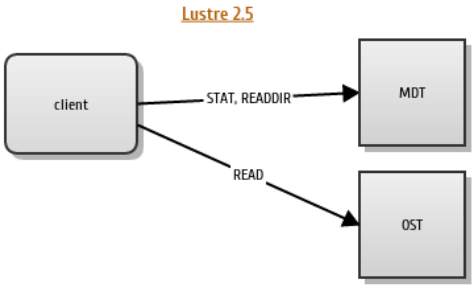


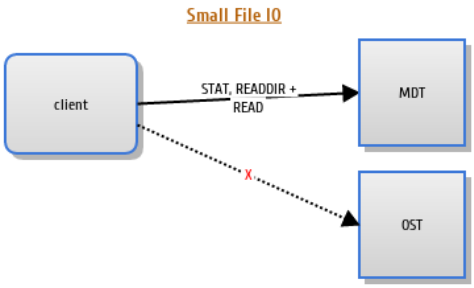
Layout bulk and DOM

- We can use the same bulk buffer used for large layouts for the file data transfer. A client cannot both be small and have data on the MDT at the same time it needs a large layout that needs a bulk reply buffer.

Read-ahead for small files

- With DOM there is possible optimization to pre-fetch files data during sequential readdir() or stat() operations. That allows efficient data access in common opeperations like `grep -r` or equivalent. This also avoids stalling the statahead pipeline because the client always needs to synchronously wait to read the data that is stored on the MDT.





Determine files that are not going to be small

- Upon first write to the file the client may understand that file is not going to be 'small' because write size exceeds DOM size limit.
- In that case normal layout is created with OST objects.

OST object creation policy

- The single stripe LOV layout may be created even for DOM files upon file create.
- Object could remain unused until size of file exceed DOM limit.
- When it happens that stripe will be used due to file exceeding size limit it will simplify migration.
- That makes migration easier but means more expensive DOM file operations like create, getattr, and destroy.
- Therefore administrator may choose what policy to use with DOM files - with or without LOV creation - depending on how Lustre is used and how often small files grows beyound limit.

Solution Proposal

The Data On MDT (DOM) feature can be implemented in two phases:

Phase I: server and client changes to support basic DOM mechanism without migration, DOM layout is set explicitly by lfs tool and cannot be changed.

Phase II: migration mechanism from MDT to OST for file becoming too large. Default directory striping for new files.

Phase III: performance tuning and optimization.

Phase I

New layout for DOM

DOM files need the layout indicating that data is stored on MDT. This is scope of Enhanced Layout feature and out of scope of this project. Meanwhile we indicate here that we need new layout to identify files with data on MDT. Client and server may identify that file has data on MDT and choose the way how to handle it. Details of the DOM locking are being resolved in the Enhanced Layout design.

Client MD stack to support IO

For files with DOM the client have to receive everything from MDT, that means the LMV/MDC devices should be able to pass/send combined requests to the MDT.

DOM locking

A new MDS IBITS lock to be used to protect file data that behaves similar to an extent lock to protect the data. Alternatively we may protect data with the LAYOUT lock bit that is used for the file layout. In order to correctly protect the file content from concurrent read and write operations, the client will need to be granted a write lock in the MDT inode. This is different than normal MDT inode locks, which are only ever granted to the client in read mode. The write lock will also allow the client to update attributes of the file (size, timestamps, etc) consistently between clients.

Since there is very little opportunity for parallel writes to such a small file, it is reasonable that only a single lock cover the entire data range on the MDT object. This avoids extra complexity in the MDT DLM locking protocol that would be needed for full extent range locking.

IO path for small files

CLIO to be able to use MDC export instead of OSC for small file IO requests. The underlying I/O implementation should share the same code as the OSC.

Enqueue for small files

Enqueue open/getattr request and parse their results properly for small files. We don't need to ask OST for anything and send any other request to MDT.

MDS stack to support IO requests

Unified request handler on the server makes it possible for the MDT device to accept IO requests by servers sharing the same request handlers and underlying OSD devices. This functionality is currently being implemented in the master code.

Phase II

With Phase II we plan to enable migration of small files on MDT to the OST by request or automatically when files grow beyond DOM limits. If a client starts writing to a file that was laid out on the MDT and it becomes too large, it will be migrated to an OST object. A procfs parameter (permanently tunable via `lctl set_param -P`) will be used for specifying size limit for files on MDT. There will also be an upper limit for the file size of the layout `stripe_size` for any object stored on the MDT.

With the automatic migration model the client takes layout lock, change the layout to the new one with OST stripes, reads data back from the MDT if it is not already cached on the client, and writes it to the OST.

The user or/and administrator can migrate files from MDT to OST or small files from OST to MDT using `lfs migrate` command introduced in Lustre 2.4. The migration will happen immediately when it is requested.

The default policy is remains to store all files on OST(s), unless explicitly requested to be stored on the MDT. Files may be created on the MDT by request only. Setting a default policy to create files on an MDT for a directory or file system will be possible for testing purposes only. It will not be a supported configuration until Phase III because this policy will trigger immediate migrations when file becomes too big for MDT upon first write request. See Big first write case for details. In Phase III we develop a client mechanism to efficiently migrate small files from the MDT to an OST as appropriate.

Phase III

This is 'performance improvements' phase. The possible areas of optimizations are the following:

- Read-ahead for small files. That returns data in open, getattr or readdir reply or bulk (if that mechanism will be used for layout transfer we can send data back instead of layout which is very small for DOM files).
- First write detection on client and avoiding DOM if size is bigger that limit changing of the layout from data on MDT to data on OST on the client without having to do an explicit data migration step. We expect this to take place at the llite point of the stack. This will intercept an excessively sized file before the data gets cached at the OSC layer.
- Pre-creation of OST object even for small files to use it later when migration occur. This is tunable option, administrator may choose to use it or not. It is not yet clear if this will be a performance improvement for most workloads, so implementation is not a priority.

Estimated performance benefits

Simple tests and log analysis were done to estimate possible Data-on-MDT benefits. All tests were performed on single-node test file system, so the network latency is zero. In a real-world deployment there would be an added round-trip network latency of **N** for each RPC sent. The file size tested was 2kB. Times reported are in microseconds per operation.

part of operation	server	time for operation, usec	Data-on-MDT goal
Write at the end of file			
open + lock	MDT	1296 + N	1296 + N
glimpse	OST	392 + N	0
IO lock	OST	1737 + N	0
IO	OST	1760 + N	1760 + N
		5185 + 4N	**3056 + 2N**
Read small file			
open + lock	MDT	844 + N	844 + IO read + N
glimpse	OST	333 + N	0
IO lock	OST	726 + N	0
IO read	OST	1392 + N	0
		3295 + 4N	**~ 1200 + N**
Stat of existent file with data			
getattr + lock	MDT	954 + N	954 + N
glimpse	OST	654 + N	0

		1608 + 2N	954 + N

These results indicate that small file performance will at least be double that of the OST-based files, not including the effect of the reduced network latency for DOM due to fewer RPCs being sent. There should also be a further improvement due to the MDT storage being RAID-1 and allowing small writes, compared to the OST storage having to do inefficient RAID-6 read-modify-write operations with 1MB or 4MB RAID stripe width.

Unit/Integration Test Plan

Functional tests

test	modification	pass condition
sanity	1. Phase I: tests to check small file has data on MDS and client is able to do all operations with it. 2. Phase II: test to check migration tool. It should move file data to OSTs.	Those function works as expected.
sanityn	Parallel access to the small file from different clients	
replay-single	Fail single MDTs with different operations with small file, like create, destroy, write, truncate	No application failure in those failover test.
replay-dual	The same as above but with two client working with the same small file	No application failure in those failover test.
sanity-quota	Do small file operation then check the inode quota and block quota	Quota reflects file data size as expected

Performance tests

- mdsrate tests should show performance benefits with operation like create, destroy, stat between files with 1 stripe on OST and small file with DOM
- obdecho test to work with small files having data on MDT
- Integral tests like Postfix or iozone with small file sizes may be used to see synergetic effect but that require Phase II with default policy for small files to have data on MDT

Interoperability

Old clients are not compatible with small files created by new client. They will return an error due to not understanding the new DOM file layout. There is also no way for these clients to access the file data on the MDT, so no compatibility mode is possible.

Accessing Lustre files systems from clients of different versions is a supported configuration. A large site may have a file system shared between different systems where the clients are upgraded independently. Since files are only created with DOM by request, it is possible for DOM-capable clients connected to a DOM-capable server to create DOM files while non-DOM-capable clients are still accessing the filesystem. To avoid an old client seeing errors when trying to access files created with DOM by a new client, either all of the clients should be upgraded before the servers, or an administrator may disable DOM explicitly on the MDT(s) until all clients are upgraded. The mechanism for how to disable Data on MDT will be discussed in the high level design.

Acceptance Criteria

All Unit/Integration Test Plan will pass

---

*Other names and brands may be the property of others.