# Lustre File System

## Aaron’s Manual Part-2

**Chapter 1. Understanding Lustre Architecture**

* 1. **What a Lustre File System Is (and What It Isn't)**

The Lustre architecture is a storage architecture for clusters. The Lustre storage architecture is used for many different kinds of clusters. It is best known for powering many of the largest high-performance computing (HPC) clusters worldwide, with tens of thousands of client systems, petabytes (PB) of storage and hundreds of gigabytes per second (GB/sec) of I/O throughput. Many HPC sites use a Lustre file system as a site-wide global file system, serving dozens of clusters.

The ability of a Lustre file system to scale capacity and performance for any need reduces the need to deploy many separate file systems, such as one for each compute cluster. Storage management is simplified by avoiding the need to copy data between compute clusters. In addition to aggregating storage capacity of many servers, the I/O throughput is also aggregated and scales with additional servers. Moreover, throughput and/or capacity can be easily increased by adding servers dynamically.

While a Lustre file system can function in many work environments, it is not necessarily the best choice for all applications. It is best suited for uses that exceed the capacity that a single server can provide, though in some use cases, a Lustre file system can perform better with a single server than other file systems due to its strong locking and data coherency.

A Lustre file system is currently not particularly well suited for "peer-to-peer" usage models where clients and servers are running on the same node, each sharing a small amount of storage, due to the lack of data replication at the Lustre software level. In such uses, if one client/server fails, then the data stored on that node will not be accessible until the node is restarted.

A Lustre installation can be scaled up or down with respect to the number of client nodes, disk storage and bandwidth. Scalability and performance are dependent on available disk and network bandwidth and the processing power of the servers in the system. A Lustre file system can be deployed in a wide variety of configurations that can be scaled well beyond the size and performance observed in production systems to date.

* + - * **High-performance heterogeneous networking:**The Lustre software supports a variety of high performance, low latency networks and permits Remote Direct Memory Access (RDMA) for InfiniBand

(utilizing OpenFabrics Enterprise Distribution (OFED\*), Intel OmniPath®, and other advanced

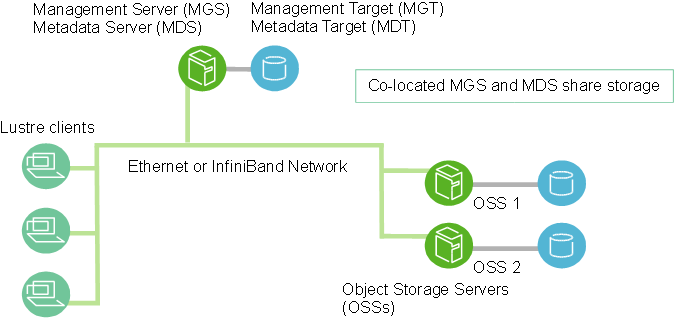
networks for fast and efficient network transport. Multiple RDMA networks can be bridged using Lustre routing for maximum performance. The Lustre software also includes integrated network diagnostics.

* + - * **High-availability:**The Lustre file system supports active/active failover using shared storage partitions for OSS targets (OSTs). Lustre software release 2.3 and earlier releases offer active/passive failover using a shared storage partition for the MDS target (MDT). The Lustre file system can work with a variety of high availability (HA) managers to allow automated failover and has no single point of failure (NSPF). This allows application transparent recovery. Multiple mount protection (MMP) provides integrated protection from errors in highly-available systems that would otherwise cause file system corruption.
      * **Security:**By default TCP connections are only allowed from privileged ports. UNIX group membership is verified on the MDS.
      * **Access control list (ACL), extended attributes:** The Lustre security model follows that of a UNIX file system, enhanced with POSIX ACLs. Noteworthy additional features include root squash.
      * **Quotas:** User and group quotas are available for a Lustre file system.
      * **Capacity growth:** The size of a Lustre file system and aggregate cluster bandwidth can be increased without interruption by adding new OSTs and MDTs to the cluster.
      * **Controlled file layout:** The layout of files across OSTs can be configured on a per file, per directory, or per file system basis. This allows file I/O to be tuned to specific application requirements within a single file system. The Lustre file system uses RAID-0 striping and balances space usage across OSTs.
      * **Network data integrity protection:** A checksum of all data sent from the client to the OSS protects against corruption during data transfer.
      * **MPI I/O:** The Lustre architecture has a dedicated MPI ADIO layer that optimizes parallel I/O to match the underlying file system architecture.
      * **NFS and CIFS export :**Lustre files can be re-exported using NFS (via Linux knfsd) or CIFS (via Samba) enabling them to be shared with non-Linux clients, such as Microsoft \*Windows \*and Apple \*Mac OS X \*.
      * **Disaster recovery tool:** The Lustre file system provides an online distributed file system check (LFSCK) that can restore consistency between storage components in case of a major file system error. A Lustre file system can operate even in the presence of file system inconsistencies, and LFSCK can run while the file system is in use, so LFSCK is not required to complete before returning the file system to production.
      * **Performance monitoring:** The Lustre file system offers a variety of mechanisms to examine performance and tuning.

## Lustre Components

An installation of the Lustre software includes a management server (MGS) and one or more Lustre file systems interconnected with Lustre networking (LNet).

A basic configuration of Lustre file system components is shown in [Figure 1.1, “Lustre file system](#_bookmark18) [components in a basic cluster”](#_bookmark18).



### Management Server (MGS)

The MGS stores configuration information for all the Lustre file systems in a cluster and provides this information to other Lustre components. Each Lustre target contacts the MGS to provide information, and Lustre clients contact the MGS to retrieve information.

It is preferable that the MGS have its own storage space so that it can be managed independently. However, the MGS can be co-located and share storage space with an MDS as shown in [Figure 1.1, “Lustre file](#_bookmark18) [system components in a basic cluster”](#_bookmark18).

### Lustre File System Components

Each Lustre file system consists of the following components:

* + - * **Metadata Servers (MDS)**- The MDS makes metadata stored in one or more MDTs available to Lustre clients. Each MDS manages the names and directories in the Lustre file system(s) and provides network request handling for one or more local MDTs.
      * **Metadata Targets (MDT**) - For Lustre software release 2.3 and earlier, each file system has one MDT. The MDT stores metadata (such as filenames, directories, permissions and file layout) on storage attached to an MDS. Each file system has one MDT. An MDT on a shared storage target can be available to multiple MDSs, although only one can access it at a time. If an active MDS fails, a standby MDS can serve the MDT and make it available to clients. This is referred to as MDS failover.
      * **Object Storage Servers (OSS)**: The OSS provides file I/O service and network request handling for one or more local OSTs. Typically, an OSS serves between two and eight OSTs, up to 16 TB each. A typical configuration is an MDT on a dedicated node, two or more OSTs on each OSS node, and a client on each of a large number of compute nodes.
      * **Object Storage Target (OST)**: User file data is stored in one or more objects, each object on a separate OST in a Lustre file system. The number of objects per file is configurable by the user and can be tuned to optimize performance for a given workload.
      * **Lustre clients**: Lustre clients are computational, visualization or desktop nodes that are running Lustre client software, allowing them to mount the Lustre file system.

The Lustre client software provides an interface between the Linux virtual file system and the Lustre servers. The client software includes a management client (MGC), a metadata client (MDC), and multiple object storage clients (OSCs), one corresponding to each OST in the file system.

A logical object volume (LOV) aggregates the OSCs to provide transparent access across all the OSTs. Thus, a client with the Lustre file system mounted sees a single, coherent, synchronized namespace. Several clients can write to different parts of the same file simultaneously, while, at the same time, other clients can read from the file.

A logical metadata volume (LMV) aggregates the MDCs to provide transparent access across all the MDTs in a similar manner as the LOV does for file access. This allows the client to see the directory tree on multiple MDTs as a single coherent namespace, and striped directories are merged on the clients to form a single visible directory to users and applications.

[Table 1.2, “ Storage and hardware requirements for Lustre file system components”](#_bookmark22) provides the requirements for attached storage for each Lustre file system component and describes desirable characteristics of the hardware used.

###### Table 1.2. Storage and hardware requirements for Lustre file system components

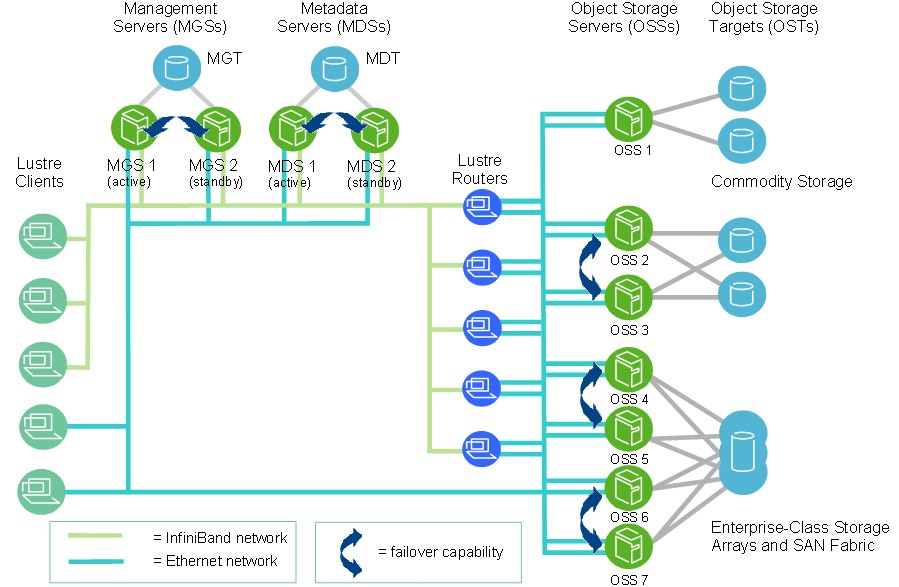
|  |  |  |
| --- | --- | --- |
|  | **Required attached storage** | **Desirable hardware characteristics** |
| **MDSs** | 1-2% of file system capacity | Adequate CPU power, plenty of memory, fast disk storage. |
| **OSSs** | 1-128 TB per OST, 1-8 OSTs per OSS | Good bus bandwidth. Recommended that storage be balanced evenly across OSSs and matched to network bandwidth. |
| **Clients** | No local storage needed | Low latency, high bandwidth network. |

For additional hardware requirements and considerations, see [Chapter 5, *Determining Hardware*](#_bookmark70)[*Configuration Requirements and Formatting Options*](#_bookmark70).

### Lustre Cluster

At scale, a Lustre file system cluster can include hundreds of OSSs and thousands of clients (see [Figure 1.2,](#_bookmark28) [“ Lustre cluster at scale”](#_bookmark28)). More than one type of network can be used in a Lustre cluster. Shared storage between OSSs enables failover capability. For more details about OSS failover, see [Chapter 3,](#_bookmark48) [*Understanding Failover in a Lustre File System*](#_bookmark48).

###### Figure 1.2. Lustre cluster at scale



* 1. **Lustre File System Storage and I/O**

In Lustre software release 2.0, Lustre file identifiers (FIDs) were introduced to replace UNIX inode numbers for identifying files or objects. A FID is a 128-bit identifier that contains a unique 64-bit sequence number, a 32-bit object ID (OID), and a 32-bit version number. The sequence number is unique across all Lustre targets in a file system (OSTs and MDTs). This change enabled future support for multiple MDTs (introduced in Lustre software release 2.4) and ZFS (introduced in Lustre software release 2.4).

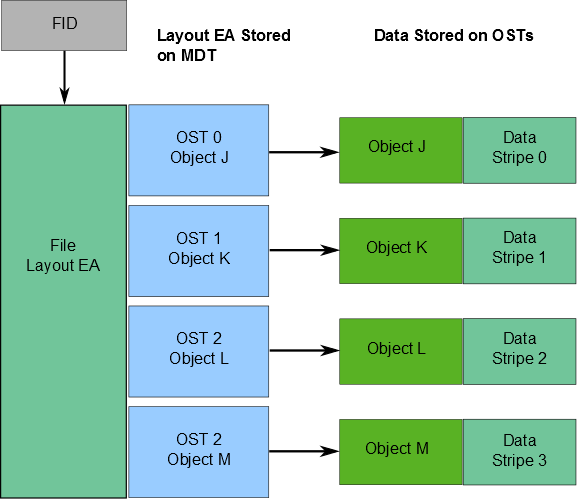
Also introduced in release 2.0 is an ldiskfs feature named *FID-in-dirent*(also known as *dirdata*) in which the FID is stored as part of the name of the file in the parent directory. This feature significantly improves performance for ls command executions by reducing disk I/O. The FID-in-dirent is generated at the time the file is created.

|  |  |  |
| --- | --- | --- |
|  |  |  |
| Introduced in Lustre 2.4 | |  |
|  | The LFSCK file system consistency checking tool released with Lustre software release 2.4 provides functionality that enables FID-in-dirent for existing files. It includes the following functionality: | |

* Generates IGIF mode FIDs for existing files from a 1.8 version file system files.
* Verifies the FID-in-dirent for each file and regenerates the FID-in-dirent if it is invalid or missing.
* Verifies the linkEA entry for each and regenerates the linkEA if it is invalid or missing. The *linkEA* consists of the file name and parent FID. It is stored as an extended attribute in the file itself. Thus, the linkEA can be used to reconstruct the full path name of a file.

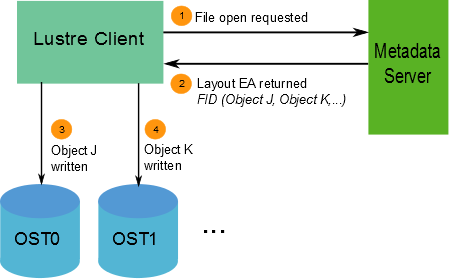
Information about where file data is located on the OST(s) is stored as an extended attribute called layout EA in an MDT object identified by the FID for the file (see [Figure 1.3, “Layout EA on MDT pointing to](#_bookmark33) [file data on OSTs”](#_bookmark33)). If the file is a regular file (not a directory or symbol link), the MDT object points to 1- to-N OST object(s) on the OST(s) that contain the file data. If the MDT file points to one object, all the file data is stored in that object. If the MDT file points to more than one object, the file data is *striped* across the objects using RAID 0, and each object is stored on a different OST. (For more information about how striping is implemented in a Lustre file system, see [Section 1.3.1, “ Lustre File System and Striping”](#_bookmark35).

###### Figure 1.3. Layout EA on MDT pointing to file data on OSTs



When a client wants to read from or write to a file, it first fetches the layout EA from the MDT object for the file. The client then uses this information to perform I/O on the file, directly interacting with the OSS nodes where the objects are stored. This process is illustrated in [Figure 1.4, “Lustre client requesting file data”](#_bookmark34) .

###### Figure 1.4. Lustre client requesting file data



The available bandwidth of a Lustre file system is determined as follows:

* The *network bandwidth* equals the aggregated bandwidth of the OSSs to the targets.
* The *disk bandwidth* equals the sum of the disk bandwidths of the storage targets (OSTs) up to the limit of the network bandwidth.
* The *aggregate bandwidth* equals the minimum of the disk bandwidth and the network bandwidth.
* The *available file system space* equals the sum of the available space of all the OSTs.

### Lustre File System and Striping

One of the main factors leading to the high performance of Lustre file systems is the ability to stripe data across multiple OSTs in a round-robin fashion. Users can optionally configure for each file the number of stripes, stripe size, and OSTs that are used.

Striping can be used to improve performance when the aggregate bandwidth to a single file exceeds the bandwidth of a single OST. The ability to stripe is also useful when a single OST does not have enough free space to hold an entire file. For more information about benefits and drawbacks of file striping, see [Section 19.2, “ Lustre File Layout (Striping) Considerations”](#_bookmark432).

Striping allows segments or 'chunks' of data in a file to be stored on different OSTs, as shown in [Figure 1.5,](#_bookmark38) [“File striping on a Lustre file system”](#_bookmark38). In the Lustre file system, a RAID 0 pattern is used in which data is "striped" across a certain number of objects. The number of objects in a single file is called the stripe\_count.

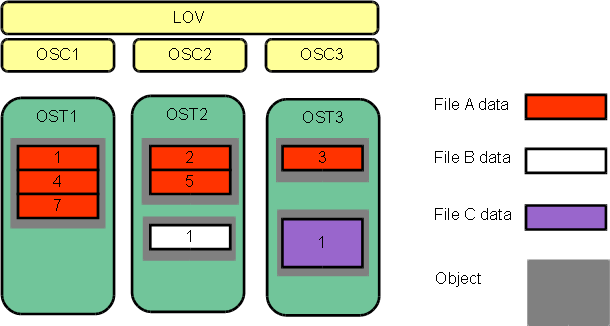
Each object contains a chunk of data from the file. When the chunk of data being written to a particular object exceeds the stripe\_size, the next chunk of data in the file is stored on the next object.

Default values for stripe\_count and stripe\_size are set for the file system. The default value for stripe\_count is 1 stripe for file and the default value for stripe\_size is 1MB. The user may change these values on a per directory or per file basis. For more details, see [Section 19.3, “Setting the](#_bookmark437) [File Layout/Striping Configuration (lfs setstripe)”](#_bookmark437).

[Figure 1.5, “File striping on a Lustre file system”](#_bookmark38), the stripe\_size for File C is larger than the stripe\_size for File A, allowing more data to be stored in a single stripe for File C. The stripe\_count for File A is 3, resulting in data striped across three objects, while the stripe\_count for File B and File C is 1.

No space is reserved on the OST for unwritten data. File A in [Figure 1.5, “File striping on a Lustre file](#_bookmark38) [system”](#_bookmark38).

###### Figure 1.5. File striping on a Lustre file system



The maximum file size is not limited by the size of a single target. In a Lustre file system, files can be striped across multiple objects (up to 2000), and each object can be up to 16 TB in size with ldiskfs, or up to 256PB with ZFS. This leads to a maximum file size of 31.25 PB for ldiskfs or 8EB with ZFS. Note that a Lustre file system can support files up to 2^63 bytes (8EB), limited only by the space available on the OSTs.

Although a single file can only be striped over 2000 objects, Lustre file systems can have thousands of OSTs. The I/O bandwidth to access a single file is the aggregated I/O bandwidth to the objects in a file, which can be as much as a bandwidth of up to 2000 servers. On systems with more than 2000 OSTs, clients can do I/O using multiple files to utilize the full file system bandwidth.

# Chapter 2. Understanding Lustre Networking (LNet)

## Introducing LNet

In a cluster using one or more Lustre file systems, the network communication infrastructure required by the Lustre file system is implemented using the Lustre networking (LNet) feature.

LNet supports many commonly-used network types, such as InfiniBand and IP networks, and allows simultaneous availability across multiple network types with routing between them. Remote direct memory access (RDMA) is permitted when supported by underlying networks using the appropriate Lustre network driver (LND). High availability and recovery features enable transparent recovery in conjunction with failover servers.

An LND is a pluggable driver that provides support for a particular network type, for example ksocklnd is the driver which implements the TCP Socket LND that supports TCP networks. LNDs are loaded into the driver stack, with one LND for each network type in use.

## Key Features of LNet

Key features of LNet include:

* + - RDMA, when supported by underlying networks
    - Support for many commonly-used network types
    - High availability and recovery
    - Support of multiple network types simultaneously
    - Routing among disparate networks

LNet permits end-to-end read/write throughput at or near peak bandwidth rates on a variety of network interconnects.

## Lustre Networks

A Lustre network is comprised of clients and servers running the Lustre software. It need not be confined to one LNet subnet but can span several networks provided routing is possible between the networks. In a similar manner, a single network can have multiple LNet subnets.

The Lustre networking stack is comprised of two layers, the LNet code module and the LND. The LNet layer operates above the LND layer in a manner similar to the way the network layer operates above the data link layer. LNet layer is connectionless, asynchronous and does not verify that data has been transmitted while the LND layer is connection oriented and typically does verify data transmission.

LNets are uniquely identified by a label comprised of a string corresponding to an LND and a number, such as tcp0, o2ib0, or o2ib1, that uniquely identifies each LNet. Each node on an LNet has at least one network identifier (NID). A NID is a combination of the address of the network interface and the LNet label in the form:*address*@*LNet\_label*.

Examples:

192.168.1.2@tcp0

10.13.24.90@o2ib1

In certain circumstances it might be desirable for Lustre file system traffic to pass between multiple LNets. This is possible using LNet routing. It is important to realize that LNet routing is not the same as network [routing. For more details about LNet routing, see Chapter 9, *Configuring Lustre Networking (LNet)*](#_bookmark149)

## Supported Network Types

The LNet code module includes LNDs to support many network types including:

* + - InfiniBand: OpenFabrics OFED (o2ib)
    - TCP (any network carrying TCP traffic, including GigE, 10GigE, and IPoIB)
    - RapidArray: ra
    - Quadrics: Elan

##### Note

Configuring LNet is optional.

LNet will use the first TCP/IP interface it discovers on a system (eth0) if it's loaded using the lctl network up. If this network configuration is sufficient, you do not need to configure LNet. LNet configuration is required if you are using Infiniband or multiple Ethernet interfaces.

# Chapter 3. Understanding Failover in a Lustre File System

## Failover:

In a high-availability (HA) system, unscheduled downtime is minimized by using redundant hardware and software components and software components that automate recovery when a failure occurs. If a failure condition occurs, such as the loss of a server or storage device or a network or software fault, the system's services continue with minimal interruption. Generally, availability is specified as the percentage of time the system is required to be available.

Availability is accomplished by replicating hardware and/or software so that when a primary server fails or is unavailable, a standby server can be switched into its place to run applications and associated resources. This process, called *failover*, is automatic in an HA system and, in most cases, completely application- transparent.

A failover hardware setup requires a pair of servers with a shared resource (typically a physical storage device, which may be based on SAN, NAS, hardware RAID, SCSI or Fibre Channel (FC) technology). The method of sharing storage should be essentially transparent at the device level; the same physical logical unit number (LUN) should be visible from both servers. To ensure high availability at the physical storage level, we encourage the use of RAID arrays to protect against drive-level failures.

##### Note

The Lustre software does not provide redundancy for data; it depends exclusively on redundancy of backing storage devices. The backing OST storage should be RAID 5 or, preferably, RAID 6 storage. MDT storage should be RAID 1 or RAID 10.

**Note**

The Lustre software provides failover mechanisms only at the layer of the Lustre file system. No failover functionality is provided for system-level components such as failing hardware or applications, or even for the entire failure of a node, as would typically be provided in a complete failover solution. Failover functionality such as node monitoring, failure detection, and resource fencing must be provided by external HA software, such as PowerMan or the open source Corosync and Pacemaker packages provided by Linux operating system vendors. Corosync provides support for detecting failures, and Pacemaker provides the actions to take once a failure has been detected.

### Failover Capabilities

To establish a highly-available Lustre file system, power management software or hardware and high availability (HA) software are used to provide the following failover capabilities:

* + - * **Resource fencing**- Protects physical storage from simultaneous access by two nodes.
      * **Resource management**- Starts and stops the Lustre resources as a part of failover, maintains the cluster state, and carries out other resource management tasks.
      * **Health monitoring**- Verifies the availability of hardware and network resources and responds to health indications provided by the Lustre software.

These capabilities can be provided by a variety of software and/or hardware solutions. HA software is responsible for detecting failure of the primary Lustre server node and controlling the failover. The Lustre software works with any HA software that includes resource (I/O) fencing. For proper resource fencing, the HA software must be able to completely power off the failed server or disconnect it from the shared storage device. If two active nodes have access to the same storage device, data may be severely corrupted.

### Types of Failover Configurations

Nodes in a cluster can be configured for failover in several ways. They are often configured in pairs (for example, two OSTs attached to a shared storage device), but other failover configurations are also possible. Failover configurations include:

* + - * **Active/passive** pair - In this configuration, the active node provides resources and serves data, while the passive node is usually standing by idle. If the active node fails, the passive node takes over and becomes active.
      * **Active/active** pair - In this configuration, both nodes are active, each providing a subset of resources. In case of a failure, the second node takes over resources from the failed node.

In Lustre software releases previous to Lustre software release 2.4, MDSs can be configured as an active/ passive pair, while OSSs can be deployed in an active/active configuration that provides redundancy without extra overhead. Often the standby MDS is the active MDS for another Lustre file system or the MGS, so no nodes are idle in the cluster.

## Failover Functionality in a Lustre File System

* + The failover functionality provided by the Lustre software can be used for the following failover scenario. When a client attempts to do I/O to a failed Lustre target, it continues to try until it receives an answer from any of the configured failover nodes for the Lustre target. A user-space application does not detect anything unusual, except that the I/O may take longer to complete.
  + Failover in a Lustre file system requires that two nodes be configured as a failover pair, which must share one or more storage devices. A Lustre file system can be configured to provide MDT or OST failover.
  + For MDT failover, two MDSs can be configured to serve the same MDT. Only one MDS node can serve an MDT at a time.
  + For OST failover, multiple OSS nodes can be configured to be able to serve the same OST. However, only one OSS node can serve the OST at a time. An OST can be moved between OSS nodes that have access to the same storage device using umount/mount commands.

The --servicenode option is used to set up nodes in a Lustre file system for failover at creation time (using mkfs.lustre) or later when the Lustre file system is active (using tunefs.lustre). For [explanations of these utilities, see Section 40.14, “ mkfs.lustre”](#_bookmark1111)and [Section 40.18, “ tunefs.lustre”](#_bookmark1137).

Failover capability in a Lustre file system can be used to upgrade the Lustre software between successive minor versions without cluster downtime.

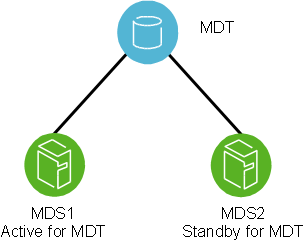
### 3.2.1. MDT Failover Configuration (Active/Passive)

Two MDSs are typically configured as an active/passive failover pair as shown in [Figure 3.1, “Lustre](#_bookmark59) [failover configuration for a active/passive MDT”](#_bookmark59). Note that both nodes must have access to shared storage for the MDT(s) and the MGS. The primary (active) MDS manages the Lustre system metadata resources. If the primary MDS fails, the secondary (passive) MDS takes over these resources and serves the MDTs and the MGS.

##### Note

In an environment with multiple file systems, the MDSs can be configured in a quasi active/active configuration, with each MDS managing metadata for a subset of the Lustre file system.

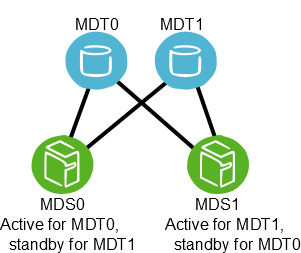
###### Figure 3.1. Lustre failover configuration for a active/passive MDT



**3.2.2.** **MDT Failover Configuration (Active/Active)**

Multiple MDTs became available with the advent of Lustre software release 2.4. MDTs can be setup as an active/active failover configuration. A failover cluster is built from two MDSs as shown in [Figure 3.2,](#_bookmark62) [“Lustre failover configuration for a active/active MDTs”](#_bookmark62).

**Figure 3.2. Lustre failover configuration for a active/active MDTs**



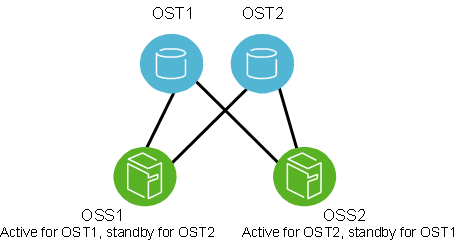
**3.2.3.** **OST Failover Configuration (Active/Active)**

OSTs are usually configured in a load-balanced, active/active failover configuration. A failover cluster is [built from two OSSs as shown in Figure 3.3, “Lustre failover configuration for an OSTs”](#_bookmark65).

##### Note

OSSs configured, as a failover pair must have shared disks/RAID.

###### Figure 3.3. Lustre failover configuration for an OSTs



In an active configuration, 50% of the available OSTs are assigned to one OSS and the remaining OSTs are assigned to the other OSS. Each OSS serves as the primary node for half the OSTs and as a failover node for the remaining OSTs.

In this mode, if one OSS fails, the other OSS takes over all of the failed OSTs. The clients attempt to connect to each OSS serving the OST, until one of them responds. Data on the OST is written synchronously, and the clients replay transactions that were in progress and uncommitted to disk before the OST failure.

# Chapter 5. Determining Hardware Configuration Requirements and Formatting Options

## Hardware Considerations

A Lustre file system can utilize any kind of block storage device such as single disks, software RAID, hardware RAID, or a logical volume manager. In contrast to some networked file systems, the block devices are only attached to the MDS and OSS nodes in a Lustre file system and are not accessed by the clients directly.

Since the block devices are accessed by only one or two server nodes, a storage area network (SAN) that is accessible from all the servers is not required. Expensive switches are not needed because point-to-point connections between the servers and the storage arrays normally provide the simplest and best attachments. (If failover capability is desired, the storage must be attached to multiple servers.)

For a production environment, it is preferable that the MGS have separate storage to allow future expansion to multiple file systems. However, it is possible to run the MDS and MGS on the same machine and have them share the same storage device.

For best performance in a production environment, dedicated clients are required. For a non-production Lustre environment or for testing, a Lustre client and server can run on the same machine. However, dedicated clients are the only supported configuration.

##### Warning

Performance and recovery issues can occur if you put a client on an MDS or OSS, so just don’t do this…

Only servers running on 64-bit CPUs are tested and supported. 64-bit CPU clients are typically used for testing to match expected customer usage and avoid limitations due to the 4 GB limit for RAM size, 1 GB low-memory limitation, and 16 TB file size limit of 32-bit CPUs. Also, due to kernel API limitations, performing backups of Lustre software release 2.x. file systems on 32-bit clients may cause backup tools to confuse files that have the same 32-bit inode number.

The storage attached to the servers typically uses RAID to provide fault tolerance and can optionally be organized with logical volume management (LVM), which is then formatted as a Lustre file system. Lustre OSS and MDS servers read, write and modify data in the format imposed by the file system.

The Lustre file system uses journaling file system technology on both the MDTs and OSTs. For a MDT, as much as a 20 percent performance gain can be obtained by placing the journal on a separate device.

The MDS can effectively utilize a lot of CPU cycles. A minimum of four processor cores are recommended. More are advisable for files systems with many clients.

##### Note

Lustre clients running on architectures with different endianness are supported. One limitation is that the PAGE\_SIZE kernel macro on the client must be as large as the PAGE\_SIZE of the server. In particular, ia64 or PPC clients with large pages (up to 64kB pages) can run with x86 servers (4kB pages). If you are running x86 clients with ia64 or PPC servers, you must compile the ia64 kernel with a 4kB PAGE\_SIZE (so the server page size is not larger than the client page size).

### MGT and MDT Storage Hardware Considerations

MGT storage requirements are small (less than 100 MB even in the largest Lustre file systems), and the data on an MGT is only accessed on a server/client mount, so disk performance is not a consideration. However, this data is vital for file system access, so the MGT should be reliable storage, preferably mirrored RAID1.

MDS storage is accessed in a database-like access pattern with many seeks and read-and-writes of small amounts of data. High throughput to MDS storage is not important. Storage types that provide much lower seek times, such as high-RPM SAS or SSD drives can be used for the MDT.

For maximum performance, the MDT should be configured as RAID1 with an internal journal and two disks from different controllers.

If you need a larger MDT, create multiple RAID1 devices from pairs of disks, and then make a RAID0 array of the RAID1 devices. This ensures maximum reliability because multiple disk failures only have a small chance of hitting both disks in the same RAID1 device.

Doing the opposite (RAID1 of a pair of RAID0 devices) has a 50% chance that even two disk failures can cause the loss of the whole MDT device. The first failure disables an entire half of the mirror and the second failure has a 50% chance of disabling the remaining mirror.

### OST Storage Hardware Considerations

The data access pattern for the OSS storage is a streaming I/O pattern that is dependent on the access patterns of applications being used. Each OSS can manage multiple object storage targets (OSTs), one for each volume with I/O traffic load-balanced between servers and targets. An OSS should be configured to have a balance between the network bandwidth and the attached storage bandwidth to prevent bottlenecks in the I/O path. Depending on the server hardware, an OSS typically serves between 2 and 8 targets, with each target between 24-48TB, but may be up to 256 terabytes (TBs) in size.

Lustre file system capacity is the sum of the capacities provided by the targets. For example, 64 OSSs, each with two 8 TB OSTs, provide a file system with a capacity of nearly 1 PB. If each OST uses ten 1 TB SATA disks (8 data disks plus 2 parity disks in a RAID-6 configuration), it may be possible to get 50 MB/sec from each drive, providing up to 400 MB/sec of disk bandwidth per OST. If this system is used as storage backend with a system network, such as the InfiniBand network, that provides a similar bandwidth, then each OSS could provide 800 MB/sec of end-to-end I/O throughput. (Although the architectural constraints described here are simple, in practice it takes careful hardware selection, benchmarking and integration to obtain such results.)

## Determining Space Requirements

The desired performance characteristics of the backing file systems on the MDT and OSTs are independent of one another. The size of the MDT backing file system depends on the number of inodes needed in the total Lustre file system, while the aggregate OST space depends on the total amount of data stored on the file system. If MGS data is to be stored on the MDT device (co-located MGT and MDT), add 100 MB to the required size estimate for the MDT.

Each time a file is created on a Lustre file system, it consumes one inode on the MDT and one OST object over which the file is striped. Normally, each file's stripe count is based on the system-wide default stripe count. However, this can be changed for individual files using the lfs setstripe option. For more [details, see Chapter 19, *Managing File Layout (Striping) and Free Space*](#_bookmark427).

In a Lustre ldiskfs file system, all the MDT inodes and OST objects are allocated when the file system is first formatted. When the file system is in use and a file is created, metadata associated with that file is stored in one of the pre-allocated inodes and does not consume any of the free space used to store file data. The total number of inodes on a formatted ldiskfs MDT or OST cannot be easily changed. Thus, the number of inodes created at format time should be generous enough to anticipate near term expected usage, with some room for growth without the effort of additional storage.

By default, the ldiskfs file system used by Lustre servers to store user-data objects and system data reserves 5% of space that cannot be used by the Lustre file system. Additionally, an ldiskfs Lustre file system reserves up to 400 MB on each OST, and up to 4GB on each MDT for journal use and a small amount of space outside the journal to store accounting data. This reserved space is unusable for general storage. Thus, at least this much space will be used per OST before any file object data is saved.

### Determining MGT Space Requirements

Less than 100 MB of space is typically required for the MGT. The size is determined by the total number of servers in the Lustre file system cluster(s) that are managed by the MGS.

### Determining MDT Space Requirements

When calculating the MDT size, the important factor to consider is the number of files to be stored in the file system, which depends on at least 4 KiB per inode of usable space on the MDT. Since MDTs typically use RAID-1+0 mirroring, the total storage needed will be double this.

Please note that the actual used space per MDT depends on the number of files per directory, the number of stripes per file, whether files have ACLs or user xattrs, and the number of hard links per file. The storage required for Lustre file system metadata is typically 1-2 percent of the total file system capacity depending upon file size.

For example, if the average file size is 5 MiB and you have 100 TiB of usable OST space, then you can calculate the minimum total number of inodes each for MDTs and OSTs as follows:

(500 TB \* 1000000 MB/TB) / 5 MB/inode = 100M inodes

For details about formatting options for ldiskfs MDT and OST file systems, see [Section 5.3.1, “Setting](#_bookmark93) [Formatting Options for an ldiskfs MDT”](#_bookmark93).

It is recommended that the MDT have at least twice the minimum number of inodes to allow for future expansion and allow for an average file size smaller than expected. Thus, the minimum space for an ldiskfs MDT should be approximately:

2 KiB/inode x 100 million inodes x 2 = 400 GiB ldiskfs MDT

##### Note

If the average file size is very small, 4 KB for example, the MDT will use as much space for each file as the space used on the OST. However, this is an uncommon usage for a Lustre filesystem.

##### Note

##### If the MDT has too few inodes, this can cause the space on the OSTs to be inaccessible since no new files can be created. Be sure to determine the appropriate size of the MDT needed to support the file system before formatting the file system. It is possible to increase the number of inodes after the file system is formatted, depending on the storage. For ldiskfs MDT filesystems the resize2fs tool can be used if the underlying block device is on a LVM logical volume and the underlying logical volume size can be increased. For ZFS new (mirrored) VDEVs can be added to the MDT pool to increase the total space available for inode storage. Inodes will be added approximately in proportion to space added.

### Determining OST Space Requirements

For the OST, the amount of space taken by each object depends on the usage pattern of the users/ applications running on the system. The Lustre software defaults to a conservative estimate for the average object size (between 64KB per object for 10GB OSTs, and 1MB per object for 16TB and larger OSTs). If you are confident that the average file size for your applications will be larger than this, you can specify a larger average file size (fewer total inodes for a given OST size) to reduce file system overhead and minimize file system check time. See [Section 5.3.2, “Setting Formatting Options for an ldiskfs OST”](#_bookmark95) for more details.

## Determining Memory Requirements

This section describes the memory requirements for each Lustre file system component.

### Client Memory Requirements

A minimum of 2 GB RAM is recommended for clients.

### MDS Memory Requirements

MDS memory requirements are determined by the following factors:

* + - * Number of clients
      * Size of the directories
      * Load placed on server

The amount of memory used by the MDS is a function of how many clients are on the system, and how many files they are using in their working set. This is driven, primarily, by the number of locks a client can hold at one time. The number of locks held by clients varies by load and memory availability on the server. Interactive clients can hold in excess of 10,000 locks at times. On the MDS, memory usage is approximately 2 KB per file, including the Lustre distributed lock manager (DLM) lock and kernel data structures for the files currently in use. Having file data in cache can improve metadata performance by a factor of 10x or more compared to reading it from disk.

MDS memory requirements include:

* + - * **File system metadata** : A reasonable amount of RAM needs to be available for file system metadata. While no hard limit can be placed on the amount of file system metadata, if more RAM is available, then the disk I/O is needed less often to retrieve the metadata.
      * **Network transport** : If you are using TCP or other network transport that uses system memory for send/receive buffers, this memory requirement must also be taken into consideration.
      * **Journal size** : By default, the journal size is 400 MB for each Lustre ldiskfs file system. This can pin up to an equal amount of RAM on the MDS node per file system.
      * **Failover configuration** : If the MDS node will be used for failover from another node, then the RAM for each journal should be doubled, so the backup server can handle the additional load if the primary server fails.

#### 5.3.2.1. Calculating MDS Memory Requirements

By default, 400 MB are used for the file system journal. Additional RAM is used for caching file data for the larger working set, which is not actively in use by clients but should be kept "hot" for improved access times. Approximately 1.5 KB per file is needed to keep a file in cache without a lock.

For example, for a single MDT on an MDS with 1,000 clients, 16 interactive nodes, and a 2 million file working set (of which 400,000 files are cached on the clients):

Operating system overhead = 512 MB File system journal = 400 MB

1000 \* 4-core clients \* 100 files/core \* 2kB = 800 MB 16 interactive clients \* 10,000 files \* 2kB = 320 MB 1,600,000 file extra working set \* 1.5kB/file = 2400 MB

Thus, the minimum requirement for a system with this configuration is at least 4 GB of RAM. However, additional memory may significantly improve performance.

For directories containing 1 million or more files, more memory may provide a significant benefit. For example, in an environment where clients randomly access one of 10 million files, having extra memory for the cache significantly improves performance.

### OSS Memory Requirements

When planning the hardware for an OSS node, consider the memory usage of several components in the Lustre file system (i.e., journal, service threads, file system metadata, etc.). Also, consider the effect of the OSS read cache feature, which consumes memory as it caches data on the OSS node.

In addition to the MDS memory requirements mentioned in [Section 5.2.2, “ Determining MDT Space](#_bookmark84) [Requirements”](#_bookmark84), the OSS requirements include:

* + - * **Service threads** : The service threads on the OSS node pre-allocate a 4 MB I/O buffer for each ost\_io service thread, so these buffers do not need to be allocated and freed for each I/O request.
      * **OSS read cache** : OSS read cache provides read-only caching of data on an OSS, using the regular Linux page cache to store the data. Just like caching from a regular file system in the Linux operating system, OSS read cache uses as much physical memory as is available.

The same calculation applies to files accessed from the OSS as for the MDS, but the load is distributed over many more OSSs nodes, so the amount of memory required for locks, inode cache, etc. listed under MDS is spread out over the OSS nodes.

Because of these memory requirements, the following calculations should be taken as determining the absolute minimum RAM required in an OSS node.

#### 5.3.3.1. Calculating OSS Memory Requirements

The minimum recommended RAM size for an OSS with two OSTs is computed below: Ethernet/TCP send/receive buffers (4 MB \* 512 threads) = 2048 MB

400 MB journal size \* 2 OST devices = 800 MB

1.5 MB read/write per OST IO thread \* 512 threads = 768 MB 600 MB file system read cache \* 2 OSTs = 1200 MB

1000 \* 4-core clients \* 100 files/core \* 2kB = 800MB 16 interactive clients \* 10,000 files \* 2kB = 320MB 1,600,000 file extra working set \* 1.5kB/file = 2400MB DLM locks + file system metadata TOTAL = 3520MB

Per OSS DLM locks + file system metadata = 3520MB/6 OSS = 600MB (approx.) Per OSS RAM minimum requirement = 4096MB (approx.)

This consumes about 1,400 MB just for the pre-allocated buffers, and an additional 2 GB for minimal file system and kernel usage. Therefore, for a non-failover configuration, the minimum RAM would be 4 GB for an OSS node with two OSTs. Adding additional memory on the OSS will improve the performance of reading smaller, frequently-accessed files.

For a failover configuration, the minimum RAM would be at least 6 GB. For 4 OSTs on each OSS in a failover configuration 10GB of RAM is reasonable. When the OSS is not handling any failed-over OSTs the extra RAM will be used as a read cache.

As a reasonable rule of thumb, about 2 GB of base memory plus 1 GB per OST can be used. In failover configurations, about 2 GB per OST is needed.

**Chapter 6. Configuring Storage on a Lustre File System**

* 1. **Selecting Storage for the MDT and OSTs**

The Lustre architecture allows the use of any kind of block device as backend storage. The characteristics of such devices, particularly in the case of failures, vary significantly and have an impact on configuration choices.

This section describes issues and recommendations regarding backend storage.

### Metadata Target (MDT)

I/O on the MDT is typically mostly reads and writes of small amounts of data. For this reason, we recommend that you use RAID 1 for MDT storage. If you require more capacity for an MDT than one disk provides, we recommend RAID 1 + 0 or RAID 10.

### Object Storage Server (OST)

A quick calculation makes it clear that without further redundancy, RAID 6 is required for large clusters and RAID 5 is not acceptable:

For a 2 PB file system (2,000 disks of 1 TB capacity) assume the mean time to failure (MTTF) of a disk is about 1,000 days. This means that the expected failure rate is 2000/1000 = 2 disks per day. Repair time at 10% of disk bandwidth is 1000 GB at 10MB/ sec = 100,000 sec, or about 1 day.

For a RAID 5 stripe that is 10 disks wide, during 1 day of rebuilding, the chance that a second disk in the same array will fail is about 9/1000 or about 1% per day. After 50 days, you have a 50% chance of a double failure in a RAID 5 array leading to data loss.

Therefore, RAID 6 or another double parity algorithm is needed to provide sufficient redundancy for OST storage.

For better performance, we recommend that you create RAID sets with 4 or 8 data disks plus one or two parity disks. Using larger RAID sets will negatively impact performance compared to having multiple independent RAID sets.

To maximize performance for small I/O request sizes, storage configured as RAID 1+0 can yield much better results but will increase cost or reduce capacity.

## Reliability Best Practices

RAID monitoring software is recommended to quickly detect faulty disks and allow them to be replaced to avoid double failures and data loss. Hot spare disks are recommended so that rebuilds happen without delays. Backups of the metadata file systems are recommended.

# Chapter 7. Setting Up Network Interface Bonding

* 1. **Network Interface Bonding Overview**

Bonding, also known as link aggregation, trunking and port trunking, is a method of aggregating multiple physical network links into a single logical link for increased bandwidth.

Several different types of bonding are available in the Linux distribution. All these types are referred to as 'modes', and use the bonding kernel module.

Modes 0 to 3 allow load balancing and fault tolerance by using multiple interfaces. Mode 4 aggregates a group of interfaces into a single virtual interface where all members of the group share the same speed and duplex settings. This mode is described under IEEE spec 802.3ad, and it is referred to as either 'mode 4' or '802.3ad.'

## Requirements

The most basic requirement for successful bonding is that both endpoints of the connection must be capable of bonding. In a normal case, the non-server endpoint is a switch. (Two systems connected via crossover cables can also use bonding.) Any switch used must explicitly handle 802.3ad Dynamic Link Aggregation.

The kernel must also be configured with bonding. All supported Lustre kernels have bonding functionality. The network driver for the interfaces to be bonded must have the ethtool functionality to determine slave speed and duplex settings. All recent network drivers implement it.

# Chapter 12. Monitoring a Lustre File System

## Lustre Changelogs

The changelogs feature records events that change the file system namespace or file metadata. Changes such as file creation, deletion, renaming, attribute changes, etc. are recorded with the target and parent file identifiers (FIDs), the name of the target, and a timestamp. These records can be used for a variety of purposes:

* + - Capture recent changes to feed into an archiving system.
    - Use changelog entries to exactly replicate changes in a file system mirror.
    - Set up "watch scripts" that take action on certain events or directories.
    - Maintain a rough audit trail (file/directory changes with timestamps, but no userb information). Changelogs record types are:

|  |  |
| --- | --- |
| **Value** | **Description** |
| MARK | Internal record keeping |
| CREATE | Regular file creation |
| MKDIR | Directory creation |
| HLINK | Hard link |
| SLINK | Soft link |
| MKNOD | Other file removal |
| UNLINK | Regular file removal |
| RMDIR | Directory removal |
| RNMFM | Rename, original |
| RNMTO | Rename, final |
| IOCTL | Ioctl on file or directory |
| TRUNC | Regular file truncated |
| SATTR | Attribute Change |
| XATTR | Extended attribute change |
| UNKNW | Unknown operation |

FID-to-full-pathname and pathname-to-FID functions are also included to map target and parent FIDs into the file system namespace.

### Working with Changelogs

Several commands are available to work with changelogs.

#### lctl changelog\_register

Because changelog records take up space on the MDT, the system administration must register changelog users. The registrants specify which records they are "done with", and the system purges up to the greatest common record.

To register a new changelog user, run:

lctl --device *fsname*-*MDTnumber* changelog\_register

Changelog entries are not purged beyond a registered user's set point (see lfs changelog\_clear).

#### lfs changelog

To display the metadata changes on an MDT (the changelog records), run: lfs changelog *fsname*-*MDTnumber* [startrec [endrec]] It is optional whether to specify the start and end records.

These are sample changelog records:

2 02MKDIR 4298396676 0x0 t=[0x200000405:0x15f9:0x0] p=[0x13:0x15e5a7a3:0x0]\

pics

3 01CREAT 4298402264 0x0 t=[0x200000405:0x15fa:0x0] p=[0x200000405:0x15f9:0\

x0] chloe.jpg

4 06UNLNK 4298404466 0x0 t=[0x200000405:0x15fa:0x0] p=[0x200000405:0x15f9:0\

x0] chloe.jpg

5 07RMDIR 4298405394 0x0 t=[0x200000405:0x15f9:0x0] p=[0x13:0x15e5a7a3:0x0]\

pics

##### lfs changelog\_clear

To clear old changelog records for a specific user (records that the user no longer needs), run:

lfs changelog\_clear *mdt\_name userid endrec*

The changelog\_clear command indicates that changelog records previous to *endrec* are no longer of interest to a particular user *userid*, potentially allowing the MDT to free up disk space. An *endrec* value of 0 indicates the current last record. To run changelog\_clear, the changelog user must be registered on the MDT node using lctl. When all changelog users are done with records < X, the records are deleted.

##### lctl changelog\_deregister

To deregister (unregister) a changelog user, run:

lctl --device *mdt\_device* changelog\_deregister *userid*

changelog\_deregister cl1 effectively does a changelog\_clear cl1 0 as it deregisters.

### Changelog Examples

This section provides examples of different changelog commands.

#### Registering a Changelog User

To register a new changelog user for a device (lustre-MDT0000):

# lctl --device lustre-MDT0000 changelog\_register lustre-MDT0000: Registered changelog userid 'cl1'

##### Displaying Changelog Records

To display changelog records on an MDT (lustre-MDT0000):

$ lfs changelog lustre-MDT0000

1 00MARK 19:08:20.890432813 2010.03.24 0x0 t=[0x10001:0x0:0x0] p=[0:0x0:0x\

0] mdd\_obd-lustre-MDT0000-0

##### Clearing Changelog Records

To notify a device that a specific user (cl1) no longer needs records (up to and including 3):

$ lfs changelog\_clear lustre-MDT0000 cl1 3

To confirm that the changelog\_clear operation was successful, run lfs changelog; only records after id-3 are listed:

$ lfs changelog lustre-MDT0000

4 01CREAT 19:10:37.113847713 2010.03.24 0x0 t=[0x200000420:0x4:0x0] p=[0x20\

0000420:0x3:0x0] hosts

##### Deregistering a Changelog User

To deregister a changelog user (cl1) for a specific device (lustre-MDT0000):

# lctl --device lustre-MDT0000 changelog\_deregister cl1 lustre-MDT0000: Deregistered changelog user 'cl1'

The deregistration operation clears all changelog records for the specified user (cl1).

$ lfs changelog lustre-MDT0000

5 00MARK 19:13:40.858292517 2010.03.24 0x0 t=[0x40001:0x0:0x0] p=[0:0x0:0x\

0] mdd\_obd-lustre-MDT0000-0

##### Note

MARK records typically indicate changelog recording status changes.

##### Displaying the Changelog Index and Registered Users

To display the current, maximum changelog index and registered changelog users for a specific device (lustre-MDT0000):

# lctl get\_param mdd.lustre-MDT0000.changelog\_users mdd.lustre-MDT0000.changelog\_users=current index: 8 ID index

cl2 8

#### Displaying the Changelog Mask

To show the current changelog mask on a specific device (lustre-MDT0000):

# lctl get\_param mdd.lustre-MDT0000.changelog\_mask mdd.lustre-MDT0000.changelog\_mask=

MARK CREAT MKDIR HLINK SLINK MKNOD UNLNK RMDIR RNMFM RNMTO OPEN CLOSE IOCTL\ TRUNC SATTR XATTR HSM

##### Setting the Changelog Mask

To set the current changelog mask on a specific device (lustre-MDT0000):

# lctl set\_param mdd.lustre-MDT0000.changelog\_mask=HLINK mdd.lustre-MDT0000.changelog\_mask=HLINK

$ lfs changelog\_clear lustre-MDT0000 cl1 0

$ mkdir /mnt/lustre/mydir/foo

$ cp /etc/hosts /mnt/lustre/mydir/foo/file

$ ln /mnt/lustre/mydir/foo/file /mnt/lustre/mydir/myhardlink

Only item types that are in the mask show up in the changelog.

$ lfs changelog lustre-MDT0000

9 03HLINK 19:19:35.171867477 2010.03.24 0x0 t=[0x200000420:0x6:0x0] p=[0x20\

0000420:0x3:0x0] myhardlink

## Lustre Jobstats

The Lustre jobstats feature is available starting in Lustre software release 2.3. It collects file system operation statistics for user processes running on Lustre clients, and exposes them via procfs on the server using the unique Job Identifier (JobID) provided by the job scheduler for each job. Job schedulers known to be able to work with jobstats include: SLURM, SGE, LSF, Loadleveler, PBS and Maui/MOAB.

Since jobstats is implemented in a scheduler-agnostic manner, it is likely that it will be able to work with other schedulers also.

### How Jobstats Works

The Lustre jobstats code on the client extracts the unique JobID from an environment variable within the user process, and sends this JobID to the server with the I/O operation. The server tracks statistics for operations whose JobID is given, indexed by that ID.

A Lustre setting on the client, jobid\_var, specifies which variable to use. Any environment variable can be specified. For example, SLURM sets the SLURM\_JOB\_ID environment variable with the unique job ID on each client when the job is first launched on a node, and the SLURM\_JOB\_ID will be inherited by all child processes started below that process.

Lustre can also be configured to generate a synthetic JobID from the user's process name and User ID, by setting jobid\_var to a special value, procname\_uid.

The setting of jobid\_var need not be the same on all clients. For example, one could use SLURM\_JOB\_ID on all clients managed by SLURM, and use procname\_uid on clients not managed by SLURM, such as interactive login nodes.

It is not possible to have different jobid\_var settings on a single node, since it is unlikely that multiple job schedulers are active on one client. However, the actual JobID value is local to each process environment and it is possible for multiple jobs with different JobIDs to be active on a single client at one time.

### Enable/Disable Jobstats

Jobstats are disabled by default. The current state of jobstats can be verified by checking lctl get\_param jobid\_var on a client:

$ lctl get\_param jobid\_var jobid\_var=disable

To enable jobstats on the testfs file system with SLURM:

# lctl conf\_param testfs.sys.jobid\_var=SLURM\_JOB\_ID

The lctl conf\_param command to enable or disable jobstats should be run on the MGS as root. The change is persistent, and will be propagated to the MDS, OSS, and client nodes automatically when it is set on the MGS and for each new client mount.

To temporarily enable jobstats on a client, or to use a different jobid\_var on a subset of nodes, such as nodes in a remote cluster that use a different job scheduler, or interactive login nodes that do not use a job scheduler at all, run the lctl set\_param command directly on the client node(s) after the filesystem is mounted. For example, to enable the procname\_uid synthetic JobID on a login node run:

# lctl set\_param jobid\_var=procname\_uid

The lctl set\_param setting is not persistent, and will be reset if the global jobid\_var is set on the MGS or if the filesystem is unmounted.

The following table shows the environment variables which are set by various job schedulers. Set

jobid\_var to the value for your job scheduler to collect statistics on a per job basis.

|  |  |
| --- | --- |
| **Job Scheduler** | **Environment Variable** |
| Simple Linux Utility for Resource Management (SLURM) | SLURM\_JOB\_ID |
| Sun Grid Engine (SGE) | JOB\_ID |
| Load Sharing Facility (LSF) | LSB\_JOBID |
| Loadleveler | LOADL\_STEP\_ID |
| Portable Batch Scheduler (PBS)/MAUI | PBS\_JOBID |
| Cray Application Level Placement Scheduler (ALPS) | ALPS\_APP\_ID |

There are two special values for jobid\_var: disable and procname\_uid. To disable jobstats, specify jobid\_var as disable:

# lctl conf\_param testfs.sys.jobid\_var=disable

To track job stats per process name and user ID (for debugging, or if no job scheduler is in use on some nodes such as login nodes), specify jobid\_var as procname\_uid:

# lctl conf\_param testfs.sys.jobid\_var=procname\_uid

### Check Job Stats

Metadata operation statistics are collected on MDTs. These statistics can be accessed for all file systems and all jobs on the MDT via the lctl get\_param mdt.\*.job\_stats. For example, clients running with jobid\_var=procname\_uid:

# lctl get\_param mdt.\*.job\_stats

Data operation statistics are collected on OSTs. Data operations statistics can be accessed via lctl get\_param obdfilter.\*.job\_stats, for example:

$ lctl get\_param obdfilter.\*.job\_stats

### Clear Job Stats

Accumulated job statistics can be reset by writing proc file job\_stats. Clear statistics for all jobs on the local node:

# lctl set\_param obdfilter.\*.job\_stats=clear

Clear statistics only for job 'bash.0' on lustre-MDT0000:

# lctl set\_param mdt.lustre-MDT0000.job\_stats=bash.0

### Configure Auto-cleanup Interval

By default, if a job is inactive for 600 seconds (10 minutes) statistics for this job will be dropped. This expiration value can be changed temporarily via:

# lctl set\_param \*.\*.job\_cleanup\_interval={max\_age}

It can also be changed permanently, for example to 700 seconds via:

# lctl conf\_param testfs.mdt.job\_cleanup\_interval=700

The job\_cleanup\_interval can be set as 0 to disable the auto-cleanup. Note that if auto-cleanup of Jobstats is disabled, then all statistics will be kept in memory forever, which may eventually consume all memory on the servers. In this case, any monitoring tool should explicitly clear individual job statistics as they are processed, as shown above.

## Lustre Monitoring Tool (LMT)

The Lustre Monitoring Tool (LMT) is a Python-based, distributed system that provides a top-like display of activity on server-side nodes (MDS, OSS and portals routers) on one or more Lustre file systems. It does not provide support for monitoring clients. For more information on LMT, including the setup procedure, see:

[https://github.com/chaos/lmt/wiki](http://code.google.com/p/lmt/) [<http://code.google.com/p/lmt/>] LMT questions can be directed to:

[lmt-discuss@googlegroups.com](mailto:lmt-discuss@googlegroups.com) [<mailto:lmt-discuss@googlegroups.com>]

## CollectL

CollectL is another tool that can be used to monitor a Lustre file system. You can run CollectL on a Lustre system that has any combination of MDSs, OSTs and clients. The collected data can be written to a file for continuous logging and played back at a later time. It can also be converted to a format suitable for plotting.

For more information about CollectL, see: [http://collectl.sourceforge.net](http://collectl.sourceforge.net/)

Lustre-specific documentation is also available. See:<http://collectl.sourceforge.net/Tutorial-Lustre.html>

## Other Monitoring Options

A variety of standard tools are available publicly including the following:

* + - lltop [- Lustre load monitor with batch scheduler integration. https://github.com/jhammond/lltop](https://github.com/jhammond/lltop)
    - tacc\_stats - A job-oriented system monitor, analyzation, and visualization tool that probes Lustre [interfaces and collects statistics. https://github.com/jhammond/tacc\_stats](https://github.com/jhammond/tacc_stats)
    - xltop - A continuous Lustre monitor with batch scheduler integration. [https://github.com/jhammond/](https://github.com/jhammond/xltop) [xltop](https://github.com/jhammond/xltop)

Another option is to script a simple monitoring solution that looks at various reports from ipconfig, as well as the procfs files generated by the Lustre software.

# Chapter 13. Lustre Operations

Once you have the Lustre file system up and running, you can use the procedures in this section to perform these basic Lustre administration tasks.

## Mounting by Label

The file system name is limited to 8 characters. We have encoded the file system and target information in the disk label, so you can mount by label. This allows system administrators to move disks around without worrying about issues such as SCSI disk reordering or getting the /dev/device wrong for a shared target. Soon, file system naming will be made as fail-safe as possible. Currently, Linux disk labels are limited to 16 characters. To identify the target within the file system, 8 characters are reserved, leaving 8 characters for the file system name:

## Starting Lustre

On the first start of a Lustre file system, the components must be started in the following order:

* + 1. Mount the MGT.

##### Note

If a combined MGT/MDT is present, Lustre will correctly mount the MGT and M

* + 1. Mount the MDT.
    2. Mount the OST(s).
    3. Mount the client(s).

## Mounting a Server

Starting a Lustre server is straightforward and only involves the mount command. Lustre servers can be added to /etc/fstab:

mount -t lustre

The mount command generates output similar to this:

/dev/sda1 on /mnt/test/mdt type lustre (rw)

/dev/sda2 on /mnt/test/ost0 type lustre (rw) 192.168.0.21@tcp:/testfs on /mnt/testfs type lustre (rw)

In this example, the MDT, an OST (ost0) and file system (testfs) are mounted.

LABEL=testfs-MDT0000 /mnt/test/mdt lustre defaults,\_netdev,noauto 0 0

LABEL=testfs-OST0000 /mnt/test/ost0 lustre defaults,\_netdev,noauto 0 0

In general, it is wise to specify no auto and let your high-availability (HA) package manage when to mount the device. If you are not using failover, make sure that networking has been started before mounting a Lustre server. If you are running Red Hat Enterprise Linux, SUSE Linux Enterprise Server, Debian operating system (and perhaps others), use the \_netdev flag to ensure that these disks are mounted after the network is up.

We are mounting by disk label here. The label of a device can be read with e2label. The label of a newly- formatted Lustre server may end in FFFF if the --index option is not specified to mkfs.lustre, meaning that it has yet to be assigned. The assignment takes place when the server is first started, and the disk label is updated. It is recommended that the --index option always be used, which will also ensure that the label is set at format time automatically.

## Stopping the Filesystem

A complete Lustre filesystem shutdown occurs by unmounting all clients and servers in the order shown below. Please note that unmounting a block device causes the Lustre software to be shut down on that node.

##### Note

Please note that the -a -t lustre in the commands below is not the name of a filesystem, but rather is specifying to unmount all entries in /etc/mtab that are of type lustre

* + 1. Unmount the clients

On each client node, unmount the filesystem on that client using the umount command:

umount -a -t lustre

The example below shows the unmount of the testfs filesystem on a client node:

[root@client1 ~]# mount |grep testfs

XXX.XXX.0.11@tcp:/testfs on /mnt/testfs type lustre (rw,lazystatfs)

[root@client1 ~]# umount -a -t lustre [154523.177714] Lustre: Unmounted testfs-client

* + 1. Unmount the MDT and MGT

On the MGS and MDS node(s), use the umount command:

umount -a -t lustre

The example below shows the unmount of the MDT and MGT for the testfs filesystem on a combined MGS/MDS:

[root@mds1 ~]# mount |grep lustre

/dev/sda on /mnt/mgt type lustre (ro)

/dev/sdb on /mnt/mdt type lustre (ro)

[root@mds1 ~]# umount -a -t lustre

[155263.566230] Lustre: Failing over testfs-MDT0000 [155263.775355] Lustre: server umount testfs-MDT0000 complete [155269.843862] Lustre: server umount MGS complete

For a seperate MGS and MDS, the same command is used, first on the MDS and then followed by the MGS.

* + 1. Unmount all the OSTs

On each OSS node, use the umount command:

umount -a -t lustre

The example below shows the unmount of all OSTs for the testfs filesystem on server OSS1: [root@oss1 ~]# mount |grep lustre

/dev/sda on /mnt/ost0 type lustre (ro)

/dev/sdb on /mnt/ost1 type lustre (ro)

/dev/sdc on /mnt/ost2 type lustre (ro)

[root@oss1 ~]# umount -a -t lustre

[155336.491445] Lustre: Failing over testfs-OST0002 [155336.556752] Lustre: server umount testfs-OST0002 complete

## Unmounting a Specific Target on a Server

To stop a Lustre OST, MDT, or MGT , use the umount */mount\_point* command.

The example below stops an OST, ost0, on mount point /mnt/ost0 for the testfs filesystem:

[root@oss1 ~]# umount /mnt/ost0

[ 385.142264] Lustre: Failing over testfs-OST0000

[ 385.210810] Lustre: server umount testfs-OST0000 complete

Gracefully stopping a server with the umount command preserves the state of the connected clients. The next time the server is started, it waits for clients to reconnect, and then goes through the recovery procedure.

If the force ( -f) flag is used, then the server evicts all clients and stops WITHOUT recovery. Upon restart, the server does not wait for recovery. Any currently connected clients receive I/O errors until they reconnect.

##### Note

If you are using loopback devices, use the -d flag. This flag cleans up loop devices and can always be safely specified.

## Specifying Failout/Failover Mode for OSTs

In a Lustre file system, an OST that has become unreachable because it fails, is taken off the network, or is unmounted can be handled in one of two ways:

* In failout mode, Lustre clients immediately receive errors (EIOs) after a timeout, instead of waiting for the OST to recover.
* In failover mode, Lustre clients wait for the OST to recover.

By default, the Lustre file system uses failover mode for OSTs. To specify failout mode instead, use the --param="failover.mode=failout" option as shown below (entered on one line):

oss# mkfs.lustre --fsname= *fsname* --mgsnode=

*mgs\_NID* --param=failover.mode=failout--ost --index=

*ost\_index*

*/dev/ost\_block\_device*

In the example below, failout mode is specified for the OSTs on the MGS mds0 in the file system

##### Caution

Before running this command, unmount all OSTs that will be affected by a change in failover/ failout mode.

##### Note

After initial file system configuration, use the tunefs.lustre utility to change the mode. For example, to set the failout mode, run:

$ tunefs.lustre --param failover.mode=failout

*/dev/ost\_device*

## Handling Degraded OST RAID Arrays

Lustre includes functionality that notifies Lustre if an external RAID array has degraded performance (resulting in reduced overall file system performance), either because a disk has failed and not been replaced, or because a disk was replaced and is undergoing a rebuild. To avoid a global performance slowdown due to a degraded OST, the MDS can avoid the OST for new object allocation if it is notified of the degraded state.

A parameter for each OST, called degraded, specifies whether the OST is running in degraded mode or not.

To mark the OST as degraded, use:

lctl set\_param obdfilter.{OST\_name}.degraded=1

To mark that the OST is back in normal operation, use:

lctl set\_param obdfilter.{OST\_name}.degraded=0

To determine if OSTs are currently in degraded mode, use:

lctl get\_param obdfilter.\*.degraded

If the OST is remounted due to a reboot or other condition, the flag resets to 0.

It is recommended that this be implemented by an automated script that monitors the status of individual RAID devices.

**13.9.** **Creating a sub-directory on a given MDT**

Lustre 2.4 enables individual sub-directories to be serviced by unique MDTs. An administrator can allocate a sub-directory to a given MDT using the command:

client# lfs mkdir –i *mdt\_index* */mount\_point/remote\_dir*

This command will allocate the sub-directory remote\_dir onto the MDT of index mdt\_index. For more information on adding additional MDTs and mdt\_index [see 2](#_bookmark205).

**Warning**

An administrator can allocate remote sub-directories to separate MDTs. Creating remote sub- directories in parent directories not hosted on MDT0 is not recommended. This is because the failure of the parent MDT will leave the namespace below it inaccessible. For this reason, by default it is only possible to create remote sub-directories off MDT0. To relax this restriction and enable remote sub-directories off any MDT, an administrator must issue the following command on the MGS:

mgs# lctl conf\_param *fsname*.mdt.enable\_remote\_dir=1

For Lustre filesystem 'scratch', the command executed is:

mgs# lctl conf\_param scratch.mdt.enable\_remote\_dir=1 To verify the configuration setting execute the following command on any MDS: mds# lctl get\_param mdt.\*.enable\_remote\_dir

**13.10.** **Creating a directory striped across multiple MDTs**

The Lustre 2.8 DNE feature enables individual files in a given directory to store their metadata on separate MDTs (a *striped directory*) once additional MDTs have been added to the filesystem, see [Section 14.6,](#_bookmark314) [“Adding a New MDT to a Lustre File System”](#_bookmark314). The result of this is that metadata requests for files in a striped directory are serviced by multiple MDTs and metadata service load is distributed over all the MDTs that service a given directory. By distributing metadata service load over multiple MDTs, performance can be improved beyond the limit of single MDT performance. Prior to the development of this feature all files in a directory must record their metadata on a single MDT.

This command to stripe a directory over *mdt\_count* MDTs is:

client# lfs mkdir –c *mdt\_count /mount\_point/new\_directory*

The striped directory feature is most useful for distributing single large directories (50k entries or more) across multiple MDTs, since it incurs more overhead than non-striped directories.

## Setting and Retrieving Lustre Parameters

Several options are available for setting parameters in Lustre:

* + - When creating a file system, use mkfs.lustre. See [Section 13.11.1, “Setting Tunable Parameters with](#_bookmark286)

[mkfs.lustre”](#_bookmark286)below.

* + - When a server is stopped, use tunefs.lustre. See [Section 13.11.2, “Setting Parameters with](#_bookmark287)

[tunefs.lustre”](#_bookmark287)below.

* + - When the file system is running, use lctl to set or retrieve Lustre parameters. See [Section 13.11.3,](#_bookmark288) [“Setting Parameters with lctl”](#_bookmark288)and [Section 13.11.3.5, “Reporting Current Parameter Values”](#_bookmark291)below.

### Setting Tunable Parameters with mkfs.lustre

When the file system is first formatted, parameters can simply be added as a --param option to the

mkfs.lustre command. For example:

mds# mkfs.lustre --mdt --param="sys.timeout=50" /dev/sda

For more details about creating a file system,see [Chapter 10, *Configuring a Lustre File System*](#_bookmark202). For more details about mkfs.lustre, see [Chapter 40, *System Configuration Utilities*](#_bookmark1033).

### Setting Parameters with tunefs.lustre

If a server (OSS or MDS) is stopped, parameters can be added to an existing file system using the -- param option to the tunefs.lustre command. For example:

oss# tunefs.lustre --param=failover.node=192.168.0.13@tcp0 /dev/sda

With tunefs.lustre, parameters are *additive*-- new parameters are specified in addition to old parameters, they do not replace them. To erase all old tunefs.lustre parameters and just use newly- specified parameters, run:

mds# tunefs.lustre --erase-params --param=

*new\_parameters*

The tunefs.lustre command can be used to set any parameter settable in a /proc/fs/lustre file and that has its own OBD device, so it can be specified as *obdname|fsname*. *obdtype*. *proc\_file\_name*= *value*. For example:

mds# tunefs.lustre --param mdt.identity\_upcall=NONE /dev/sda1

For more details about tunefs.lustre[, see Chapter 40,](#_bookmark1033) *[System Configuration Utilities](#_bookmark1033)*.

### Setting Parameters with lctl

When the file system is running, the lctl command can be used to set parameters (temporary or permanent) and report current parameter values. Temporary parameters are active as long as the server or client is not shut down. Permanent parameters live through server and client reboots.

##### Note

The lctl list\_param command enables users to list all parameters that can be set. See [Section 13.11.3.4, “Listing Parameters”](#_bookmark290).

For more details about the lctl command, see the examples in the sections below and [Chapter 40, *System*](#_bookmark1033)[*Configuration Utilities*](#_bookmark1033).

* + - 1. **Setting Temporary Parameters**

Use lctl set\_param to set temporary parameters on the node where it is run. These parameters map to items in /proc/{fs,sys}/{lnet,lustre}. The lctl set\_param command uses this syntax:

lctl set\_param [-n] *obdtype*. *obdname*. *proc\_file\_name*= *value*

For example:

# lctl set\_param osc.\*.max\_dirty\_mb=1024 osc.myth-OST0000-osc.max\_dirty\_mb=32 osc.myth-OST0001-osc.max\_dirty\_mb=32 osc.myth-OST0002-osc.max\_dirty\_mb=32 osc.myth-OST0003-osc.max\_dirty\_mb=32 osc.myth-OST0004-osc.max\_dirty\_mb=32

##### Setting Permanent Parameters

Use the lctl conf\_param command to set permanent parameters. In general, the lctl conf\_param command can be used to specify any parameter settable in a /proc/fs/lustre file, with its own OBD device. The lctl conf\_param command uses this syntax (same as the mkfs.lustre and tunefs.lustre commands):

Here are a few examples of lctl conf\_param commands:

mgs# lctl conf\_param testfs-MDT0000.sys.timeout=40

$ lctl conf\_param testfs-MDT0000.mdt.identity\_upcall=NONE

$ lctl conf\_param testfs.llite.max\_read\_ahead\_mb=16

$ lctl conf\_param testfs-MDT0000.lov.stripesize=2M

$ lctl conf\_param testfs-OST0000.osc.max\_dirty\_mb=29.15

$ lctl conf\_param testfs-OST0000.ost.client\_cache\_seconds=15

$ lctl conf\_param testfs.sys.timeout=40

##### Caution

Parameters specified with the lctl conf\_param command are set permanently in the file system's configuration file on the MGS.

**13.11.3.3. Setting Permanent Parameters with lctl set\_param -P**

Use the lctl set\_param -P to set parameters permanently. This command must be issued on the MGS. The given parameter is set on every host using lctl upcall. Parameters map to items in /proc/

{fs,sys}/{lnet,lustre}. The lctl set\_param command uses this syntax:

lctl set\_param –P *obdtype*. *obdname*. *proc\_file\_name*= *value*

For example:

# lctl set\_param -P osc.\*.max\_dirty\_mb=1024 osc.myth-OST0000-osc.max\_dirty\_mb=32 osc.myth-OST0001-osc.max\_dirty\_mb=32 osc.myth-OST0002-osc.max\_dirty\_mb=32 osc.myth-OST0003-osc.max\_dirty\_mb=32 osc.myth-OST0004-osc.max\_dirty\_mb=32

Use -d(only with -P) option to delete permanent parameter. Syntax:

lctl set\_param -P –d *obdtype*. *obdname*. *proc\_file\_name*

For example:

# lctl set\_param -P -d osc.\*.max\_dirty\_mb

##### Listing Parameters

To list Lustre or LNet parameters that are available to set, use the lctl list\_param command. For example:

lctl list\_param [-FR]*obdtype*. *obdname*

The following arguments are available for the lctl list\_param command.

-F Add ' /', ' @' or ' =' for directories, symlinks and writeable files, respectively

-R Recursively lists all parameters under the specified path For example:

oss# lctl list\_param obdfilter.lustre-OST0000

#### Reporting Current Parameter Values

To report current Lustre parameter values, use the lctl get\_param command with this syntax:

lctl get\_param [-n] *obdtype*. *obdname*. *proc\_file\_name*

This example reports data on RPC service times.

oss# lctl get\_param -n ost.\*.ost\_io.timeouts

service : cur 1 worst 30 (at 1257150393, 85d23h58m54s ago) 1 1 1 1

This example reports the amount of space this client has reserved for writeback cache with each OST:

client# lctl get\_param osc.\*.cur\_grant\_bytes

osc.myth-OST0000-osc-ffff8800376bdc00.cur\_grant\_bytes=2097152 osc.myth-OST0001-osc-ffff8800376bdc00.cur\_grant\_bytes=33890304 osc.myth-OST0002-osc-ffff8800376bdc00.cur\_grant\_bytes=35418112 osc.myth-OST0003-osc-ffff8800376bdc00.cur\_grant\_bytes=2097152 osc.myth-OST0004-osc-ffff8800376bdc00.cur\_grant\_bytes=33808384

## Specifying NIDs and Failover

If a node has multiple network interfaces, it may have multiple NIDs, which must all be identified so other nodes can choose the NID that is appropriate for their network interfaces. Typically, NIDs are specified in a list delimited by commas ( ,). However, when failover nodes are specified, the NIDs are delimited by a colon ( :) or by repeating a keyword such as --mgsnode= or --servicenode=).

To display the NIDs of all servers in networks configured to work with the Lustre file system, run (while LNet is running):

lctl list\_nids

In the example below, mds0 and mds1 are configured as a combined MGS/MDT failover pair and oss0 and oss1 are configured as an OST failover pair. The Ethernet address for mds0 is 192.168.10.1, and for mds1 is 192.168.10.2. The Ethernet addresses for oss0 and oss1 are 192.168.10.20 and 192.168.10.21 respectively.

mds0# mkfs.lustre --fsname=testfs --mdt --mgs \

--servicenode=192.168.10.2@tcp0 \

-–servicenode=192.168.10.1@tcp0 /dev/sda1

mds0# mount -t lustre /dev/sda1 /mnt/test/mdt

oss0# mkfs.lustre --fsname=testfs --servicenode=192.168.10.20@tcp0 \

--servicenode=192.168.10.21 --ost --index=0 \--mgsnode=192.168.10.1@tcp0 --mgsnode=192.168.10.2@tcp0 \

/dev/sdb

oss0# mount -t lustre /dev/sdb /mnt/test/ost0

client# mount -t lustre 192.168.10.1@tcp0:192.168.10.2@tcp0:/testfs \

/mnt/testfs mds0# umount /mnt/mdt

mds1# mount -t lustre /dev/sda1 /mnt/test/mdt

mds1# lctl get\_param mdt.testfs-MDT0000.recovery\_status

Where multiple NIDs are specified separated by commas (for example, 10.67.73.200@tcp,192.168.10.1@tcp), the two NIDs refer to the same host, and the Lustre software chooses the *best* one for communication. When a pair of NIDs is separated by a colon (for example, 10.67.73.200@tcp:10.67.73.201@tcp), the two NIDs refer to two different hosts and are treated as a failover pair (the Lustre software tries the first one, and if that fails, it tries the second one.)

Two options to mkfs.lustre can be used to specify failover nodes. Introduced in Lustre software release 2.0, the --servicenode option is used to specify all service NIDs, including those for primary nodes and failover nodes. When the --servicenode option is used, the first service node to load the target device becomes the primary service node, while nodes corresponding to the other specified NIDs become failover locations for the target device. An older option, --failnode, specifies just the NIDS of failover nodes. For more information about the --servicenode and --failnode options, see [Chapter 11, *Configuring Failover in a Lustre File System*](#_bookmark221).

## Erasing a File System

If you want to erase a file system and permanently delete all the data in the file system, run this command on your targets:

$ "mkfs.lustre --reformat"

If you are using a separate MGS and want to keep other file systems defined on that MGS, then set the writeconf flag on the MDT for that file system. The writeconf flag causes the configuration logs to be erased; they are regenerated the next time the servers start.

To set the writeconf flag on the MDT:

* + 1. Unmount all clients/servers using this file system, run:

$ umount /mnt/lustre

* + 1. Permanently erase the file system and, presumably, replace it with another file system, run:

$ mkfs.lustre --reformat --fsname spfs --mgs --mdt --index=0 /dev/

*{mdsdev}*

* + 1. If you have a separate MGS (that you do not want to reformat), then add the --writeconf flag to

mkfs.lustre on the MDT, run:

$ mkfs.lustre --reformat --writeconf --fsname spfs --mgsnode=

*mgs\_nid* --mdt --index=0 */dev/mds\_device*

##### Note

If you have a combined MGS/MDT, reformatting the MDT reformats the MGS as well, causing all configuration information to be lost; you can start building your new file system. Nothing needs to be done with old disks that will not be part of the new file system, just do not mount them.

## Reclaiming Reserved Disk Space

All current Lustre installations run the ldiskfs file system internally on service nodes. By default, ldiskfs reserves 5% of the disk space to avoid file system fragmentation. In order to reclaim this space, run the following command on your OSS for each OST in the file system:

tune2fs [-m reserved\_blocks\_percent] /dev/

*{ostdev}*

You do not need to shut down Lustre before running this command or restart it afterwards.

##### Warning

Reducing the space reservation can cause severe performance degradation as the OST file system becomes more than 95% full, due to difficulty in locating large areas of contiguous free space. This performance degradation may persist even if the space usage drops below 95% again. It is recommended NOT to reduce the reserved disk space below 5%.

## Replacing an Existing OST or MDT

To copy the contents of an existing OST to a new OST (or an old MDT to a new MDT), follow the process for either OST/MDT backups in [Section 18.2, “ Backing Up and Restoring an MDT or OST (ldiskfs Device](#_bookmark406) [Level)”](#_bookmark406)or [Section 18.3, “ Backing Up an OST or MDT (ldiskfs File System Level)”](#_bookmark408). For more information on removing a MDT, see [Section 14.8.1, “Removing a MDT from the File System”](#_bookmark321).

## Identifying To Which Lustre File an OST Object Belongs

Use this procedure to identify the file containing a given object on a given OST.

* + 1. On the OST (as root), run debugfs to display the file identifier ( FID) of the file associated with the object.

For example, if the object is 34976 on /dev/lustre/ost\_test2, the debug command is:

# debugfs -c -R "stat /O/0/d$((34976 % 32))/34976" /dev/lustre/ost\_test2

The command output is:

debugfs 1.42.3.wc3 (15-Aug-2012)

/dev/lustre/ost\_test2: catastrophic mode - not reading inode or group bitmaps Inode: 352365 Type: regular Mode: 0666 Flags: 0x80000 Generation: 2393149953 Version: 0x0000002a:00005f81

User: 1000 Group: 1000 Size: 260096

File ACL: 0 Directory ACL: 0

Links: 1 Blockcount: 512

Fragment: Address: 0 Number: 0 Size: 0

ctime: 0x4a216b48:00000000 -- Sat May 30 13:22:16 2009

atime: 0x4a216b48:00000000 -- Sat May 30 13:22:16 2009

mtime: 0x4a216b48:00000000 -- Sat May 30 13:22:16 2009

crtime: 0x4a216b3c:975870dc -- Sat May 30 13:22:04 2009 Size of extra inode fields: 24

Extended attributes stored in inode body:

fid = "b9 da 24 00 00 00 00 00 6a fa 0d 3f 01 00 00 00 eb 5b 0b 00 00 00 0000

00 00 00 00 00 00 00 00 " (32)

fid: objid=34976 seq=0 parent=[0x24dab9:0x3f0dfa6a:0x0] stripe=1 EXTENTS:

(0-64):4620544-4620607

* + 1. For Lustre software release 2.x file systems, the parent FID will be of the form [0x200000400:0x122:0x0] and can be resolved directly using the lfs fid2path [0x200000404:0x122:0x0] /mnt/lustre command on any Lustre client, and the process is complete.
    2. In this example the parent inode FID is an upgraded 1.x inode (due to the first part of the FID being below 0x200000400), the MDT inode number is 0x24dab9 and generation 0x3f0dfa6a and the pathname needs to be resolved using debugfs.
    3. On the MDS (as root), use debugfs to find the file associated with the inode:

# debugfs -c -R "ncheck 0x24dab9" /dev/lustre/mdt\_test

Here is the command output:

debugfs 1.42.3.wc2 (15-Aug-2012)

/dev/lustre/mdt\_test: catastrophic mode - not reading inode or group bitmap\ s

Inode Pathname

2415289 /ROOT/brian-laptop-guest/clients/client11/~dmtmp/PWRPNT/ZD16.BMP

The command lists the inode and pathname associated with the object.

##### Note

Debugfs' ''ncheck'' is a brute-force search that may take a long time to complete.

##### Note

To find the Lustre file from a disk LBA, follow the steps listed in the document at this URL: <http://smartmontools.sourceforge.net/badblockhowto.html> [[http://](http://smartmontools.sourceforge.net/badblockhowto.html) [smartmontools.sourceforge.net/badblockhowto.html](http://smartmontools.sourceforge.net/badblockhowto.html)]. Then, follow the steps above to resolve the Lustre filename.

# Chapter 14. Lustre Maintenance

Once you have the Lustre file system up and running, you can use the procedures in this section to perform these basic Lustre maintenance tasks.

## Working with Inactive OSTs

To mount a client or an MDT with one or more inactive OSTs, run commands similar to this:

client# mount -o exclude=testfs-OST0000 -t lustre \ uml1:/testfs /mnt/testfs

client# lctl get\_param lov.testfs-clilov-\*.target\_obd

To activate an inactive OST on a live client or MDT, use the lctl activate command on the OSC device. For example:

lctl --device 7 activate

##### Note

A colon-separated list can also be specified. For example, exclude=testfs- OST0000:testfs-OST0001.

## Finding Nodes in the Lustre File System

There may be situations in which you need to find all nodes in your Lustre file system or get the names of all OSTs.

To get a list of all Lustre nodes, run this command on the MGS:

# lctl get\_param mgs.MGS.live.\*

##### Note

This command must be run on the MGS.

To get the names of all OSTs, run this command on the MDS:

mds:/root# lctl get\_param lov.\*-mdtlov.target\_obd

##### Note

This command must be run on the MDS.

In this example, there are two OSTs, testfs-OST0000 and testfs-OST0001, which are both active.

mgs:/root# lctl get\_param lov.testfs-mdtlov.target\_obd 0: testfs-OST0000\_UUID ACTIVE

1: testfs-OST0001\_UUID ACTIVE

## Mounting a Server Without Lustre Service

If you are using a combined MGS/MDT, but you only want to start the MGS and not the MDT, run this command:

mount -t lustre */dev/mdt\_partition* -o nosvc */mount\_point*

The *mdt\_partition* variable is the combined MGS/MDT block device.

In this example, the combined MGS/MDT is testfs-MDT0000 and the mount point is /mnt/test/ mdt.

$ mount -t lustre -L testfs-MDT0000 -o nosvc /mnt/test/mdt

## Regenerating Lustre Configuration Logs

If the Lustre file system configuration logs are in a state where the file system cannot be started, use the writeconf command to erase them. After the writeconf command is run and the servers restart, the configuration logs are re-generated and stored on the MGS (as in a new file system).

You should only use the writeconf command if:

* + - The configuration logs are in a state where the file system cannot start
    - A server NID is being changed

The writeconf command is destructive to some configuration items (i.e., OST pools information and items set via conf\_param), and should be used with caution. To avoid problems:

* + - Shut down the file system before running the writeconf command
    - Run the writeconf command on all servers (MDT first, then OSTs)
    - Start the file system in this order:
      * MGS (or the combined MGS/MDT)
      * MDT
      * OSTs
      * Lustre clients

##### Caution

The OST pools feature enables a group of OSTs to be named for file striping purposes. If you use OST pools, be aware that running the writeconf command erases **all** pools information (as well as any other parameters set via lctl conf\_param). We recommend that the pools definitions (and conf\_param settings) be executed via a script, so they can be reproduced easily after a writeconf is performed.

To regenerate Lustre file system configuration logs:

1. Shut down the file system in this order.
   1. Unmount the clients.
   2. Unmount the MDT.
   3. Unmount all OSTs.
2. Make sure the the MDT and OST devices are available.

Run the writeconf command on all servers.

Run writeconf on the MDT first, and then the OSTs.

* 1. On the MDT, run:

mdt# tunefs.lustre --writeconf */dev/mdt\_device*

* 1. On each OST, run:

ost# tunefs.lustre --writeconf */dev/ost\_device*

1. Restart the file system in this order.
   1. Mount the MGS (or the combined MGS/MDT).
   2. Mount the MDT.
   3. Mount the OSTs.
   4. Mount the clients.

After the writeconf command is run, the configuration logs are re-generated as servers restart.

## Changing a Server NID

In Lustre software release 2.3 or earlier, the tunefs.lustre --writeconf command is used to rewrite all of the configuration files.

Change a server NID in these situations:

* + - New server hardware is added to the file system, and the MDS or an OSS is being moved to the new machine.
    - New network card is installed in the server.
    - You want to reassign IP addresses. To change a server NID:

1. Update the LNet configuration in the /etc/modprobe.conf file so the list of server NIDs is correct. Use lctl list\_nids to view the list of server NIDS.

The lctl list\_nids command indicates which network(s) are configured to work with the Lustre file system.

1. Shut down the file system in this order:

Unmount the clients.

* 1. Unmount the MDT.
  2. Unmount all OSTs.

1. If the MGS and MDS share a partition, start the MGS only:

mount -t lustre *MDT partition* -o nosvc *mount\_point*

1. Run the replace\_nids command on the MGS:

lctl replace\_nids *devicename nid1*[,nid2,nid3 ...]

where *devicename* is the Lustre target name, e.g. testfs-OST0013

1. If the MGS and MDS share a partition, stop the MGS:

umount *mount\_point*

##### Note

The replace\_nids command also cleans all old, invalidated records out of the configuration log, while preserving all other current settings.

##### Note

The previous configuration log is backed up on the MGS disk with the suffix '.bak'

**14.6. Adding a New MDT to a Lustre File System**

Additional MDTs can be added using the DNE feature to serve one or more remote sub-directories within a filesystem, in order to increase the total number of files that can be created in the filesystem, to increase aggregate metadata performance, or to isolate user or application workloads from other users of the filesystem. It is possible to have multiple remote sub-directories reference the same MDT. However, the root directory will always be located on MDT0. To add a new MDT into the file system:

1. Discover the maximum MDT index. Each MDT must have unique index.

client$ lctl dl | grep mdc

36 UP mdc testfs-MDT0000-mdc-ffff88004edf3c00 4c8be054-144f-9359-b063 37 UP mdc testfs-MDT0001-mdc-ffff88004edf3c00 4c8be054-144f-9359-b063 38 UP mdc testfs-MDT0002-mdc-ffff88004edf3c00 4c8be054-144f-9359-b063 39 UP mdc testfs-MDT0003-mdc-ffff88004edf3c00 4c8be054-144f-9359-b063

1. Add the new block device as a new MDT at the next available index. In this example, the next available index is 4.

mds# mkfs.lustre --reformat --fsname=*testfs* --mdt --mgsnode=*mgsnode* –

1. Mount the MDTs.

mds# mount –t lustre */dev/mdt4\_blockdevice* /mnt/mdt4

1. 4. In order to start creating new files and directories on the new MDT(s) they need to be attached into the namespace at one or more subdirectories using the lfs mkdir command. All files and directories below those created with lfs mkdir will also be created on the same MDT unless otherwise specified.

client# lfs mkdir -i 3 /mnt/testfs/new\_dir\_on\_mdt3 client# lfs mkdir –I /mnt/testfs/new\_dir\_on\_mdt4

client# lfs mkdir -c 4 /mnt/testfs/new\_directory\_striped\_across\_4\_mdts

## Adding a New OST to a Lustre File System

To add an OST to existing Lustre file system:

* + 1. Add a new OST by passing on the following commands, run:

oss# mkfs.lustre --fsname=spfs --mgsnode=mds16@tcp0 --ost --index=12 /dev/sda oss# mkdir -p /mnt/test/ost12

oss# mount -t lustre /dev/sda /mnt/test/ost12

* + 1. Migrate the data (possibly).

The file system is quite unbalanced when new empty OSTs are added. New file creations are automatically balanced. If this is a scratch file system or files are pruned at a regular interval, then no further work may be needed.

New files being created will preferentially be placed on the empty OST. As old files are deleted, they will release space on the old OST.

Files existing prior to the expansion can optionally be rebalanced with an in-place copy, which can be done with a simple script. The basic method is to copy existing files to a temporary file, then move the temp file over the old one. This should not be attempted with files which are currently being written to by users or applications. This operation redistributes the stripes over the entire set of OSTs.

For example, to rebalance all files within /mnt/lustre/dir, enter:

client# lfs\_migrate /mnt/lustre/file

To migrate files within the /test file system on OST0004 that are larger than 4GB in size, enter: client# lfs find /test -obd test-OST0004 -size +4G | lfs\_migrate -y [See Section 36.2, “ lfs\_migrate ”](#_bookmark957) for more details.

## Removing and Restoring OSTs

OSTs can be removed from and restored to a Lustre file system. Removing a OST means the OST is *deactivated* in the file system, not permanently removed.

##### Note

A removed OST still appears in the file system; do not create a new OST with the same name.

You may want to remove (deactivate) an OST and prevent new files from being written to it in several situations:

* Hard drive has failed and a RAID resync/rebuild is underway
* OST is nearing its space capacity
* OST storage has failed permanently

**14.8.1.** **Removing a MDT from the File System**

If the MDT is permanently inaccessible, lfs rm\_entry {directory} can be used to delete the directory entry on the unavailable MDT. A normal rmdir will report an IO error due to the remote MDT being inactive. Please note that if the MDT is available, a standard rm -r should be used to delete the remote directory. After the remote directory has been removed, the administrator should mark the MDT as permanently inactive with:

lctl conf\_param {MDT name}.mdc.active=0

A user can identify which MDT holds a remote sub-directory using the lfs utility. For example:

client$ lfs getstripe -M /mnt/lustre/remote\_dir1 1

client$ mkdir /mnt/lustre/local\_dir0

client$ lfs getstripe -M /mnt/lustre/local\_dir0 0

The getstripe [--mdt-index|-M] parameters return the index of the MDT that is serving the given directory.

**14.8.2.** **Working with Inactive MDTs**

Files located on or below an inactive MDT are inaccessible until the MDT is activated again. Clients accessing an inactive MDT will receive an EIO error.

### Removing an OST from the File System

When removing an OST, remember that the MDT does not communicate directly with OSTs. Rather, each OST has a corresponding OSC which communicates with the MDT. It is necessary to determine the device number of the OSC that corresponds to the OST. Then, you use this device number to deactivate the OSC on the MDT.

To remove an OST from the file system:

1. For the OST to be removed, determine the device number of the corresponding OSC on the MDT.
   1. List all OSCs on the node, along with their device numbers. Run:

lctl dl | grep osc

For example: lctl dl | grep

11 UP osc testfs-OST-0000-osc-cac94211 4ea5b30f-6a8e-55a0-7519-2f20318ebdb4 5

12 UP osc testfs-OST-0001-osc-cac94211 4ea5b30f-6a8e-55a0-7519-2f20318ebdb4 5

1. IN osc testfs-OST-0000-osc testfs-MDT0000-mdtlov\_UUID 5
2. UP osc testfs-OST-0001-osc testfs-MDT0000-mdtlov\_UUID 5
   1. Determine the device number of the OSC that corresponds to the OST to be removed.
3. Temporarily deactivate the OSC on the MDT. On the MDT, run:

mds# lctl --device *lustre\_devno* deactivate

For example, based on the command output in Step 1, to deactivate device 13 (the MDT’s OSC for

OST-0000), the command would be:

mds# lctl --device 13 deactivate

This marks the OST as inactive on the MDS, so no new objects are assigned to the OST. This does not prevent use of existing objects for reads or writes.

##### Note

Do not deactivate the OST on the clients. Do so causes errors (EIOs), and the copy out to fail.

##### Caution

Do not use lctl conf\_param to deactivate the OST. It permanently sets a parameter in the file system configuration.

1. Discover all files that have objects residing on the deactivated OST.

Depending on whether the deactivated OST is available or not, the data from that OST may be migrated to other OSTs, or may need to be restored from backup.

* 1. If the OST is still online and available, find all files with objects on the deactivated OST, and copy them to other OSTs in the file system to:

client# lfs find --obd *ost\_name /mount/point* | lfs\_migrate -y

* 1. If the OST is no longer available, delete the files on that OST and restore them from backup:

client# lfsfind --obd *ost\_uuid* -print0 */mount/point* | \ tee /tmp/files\_to\_restore | xargs -0 -n 1 unlink

The list of files that need to be restored from backup is stored in /tmp/files\_to\_restore. Restoring these files is beyond the scope of this document.

1. Deactivate the OST.
   1. If there is expected to be a replacement OST in some short time (a few days), the OST can temporarily be deactivated on the clients using:

client# lctl set\_param osc.*fsname*-OST*number*-\*.active=0

##### Note

This setting is only temporary and will be reset if the clients are remounted or rebooted. It needs to be run on all clients.

If there is not expected to be a replacement for this OST in the near future, permanently deactivate it on all clients and the MDS by running the following command on the MGS:

mgs# lctl conf\_param *ost\_name*.osc.active=0

##### Note

A deactivated OST still appears in the file system configuration, though a new OST with the same name can be created using the --replace option for mkfs.lustre.

### Backing Up OST Configuration Files

If the OST device is still accessible, then the Lustre configuration files on the OST should be backed up and saved for future use in order to avoid difficulties when a replacement OST is returned to service. These files rarely change, so they can and should be backed up while the OST is functional and accessible. If the deactivated OST is still available to mount (i.e. has not permanently failed or is unmountable due to severe corruption), an effort should be made to preserve these files.

* + - 1. Mount the OST file system.

oss# mkdir -p /mnt/ost

[oss]# mount -t ldiskfs */dev/ost\_device* /mnt/ost

* + - 1. Back up the OST configuration files.

oss# tar cvf *ost\_name*.tar -C /mnt/ost last\_rcvd \ CONFIGS/ O/0/LAST\_ID

* + - 1. Unmount the OST file system.

oss# umount /mnt/ost

### Restoring OST Configuration Files

If the original OST is still available, it is best to follow the OST backup and restore procedure given in either [Section 18.2, “ Backing Up and Restoring an MDT or OST (ldiskfs Device Level)”](#_bookmark406), or [Section 18.3,](#_bookmark408) [“ Backing Up an OST or MDT (ldiskfs File System Level)”](#_bookmark408) and [Section 18.4, “ Restoring a File-Level](#_bookmark411) [Backup”](#_bookmark411).

To replace an OST that was removed from service due to corruption or hardware failure, the file system needs to be formatted using mkfs.lustre, and the Lustre file system configuration should be restored, if available.

If the OST configuration files were not backed up, due to the OST file system being completely inaccessible, it is still possible to replace the failed OST with a new one at the same OST index.

* + - 1. Format the OST file system.

oss# mkfs.lustre --ost --index=*old\_ost\_index other\_options* \ */dev/new\_ost\_dev*

* + - 1. Mount the OST file system.

oss# mkdir /mnt/ost

oss# mount -t ldiskfs */dev/new\_ost\_dev /mnt/ost*

* + - 1. Restore the OST configuration files, if available.

oss# tar xvf *ost\_name*.tar -C /mnt/ost

* + - 1. Recreate the OST configuration files, if unavailable.

Follow the procedure in [Section 31.3.4, “Fixing a Bad LAST\_ID on an OST”](#_bookmark780) to recreate the LAST\_ID file for this OST index. The last\_rcvd file will be recreated when the OST is first mounted using the default parameters, which are normally correct for all file systems. The CONFIGS/mountdata file is created by mkfs.lustre at format time, but has flags set that request it to register itself with the MGS. It is possible to copy these flags from another working OST (which should be the same):

oss1# debugfs -c -R "dump CONFIGS/mountdata /tmp/ldd" */dev/other\_osdev*

oss1# scp /tmp/ldd oss0:/tmp/ldd

oss0# dd if=/tmp/ldd of=/mnt/ost/CONFIGS/mountdata bs=4 count=1 seek=5 skip=5

* + - 1. Unmount the OST file system.

oss# umount /mnt/ost

### Returning a Deactivated OST to Service

If the OST was permanently deactivated, it needs to be reactivated in the MGS configuration.

mgs# lctl conf\_param *ost\_name*.osc.active=1

If the OST was temporarily deactivated, it needs to be reactivated on the MDS and clients.

mds# lctl --device *lustre\_devno* activate

client# lctl set\_param osc.*fsname*-OST*number*-\*.active=1

## Aborting Recovery

You can abort recovery with either the lctl utility or by mounting the target with the abort\_recov

option (mount -o abort\_recov). When starting a target, run:

mds# mount -t lustre -L *mdt\_name* -o abort\_recov */mount\_point*

##### Note

The recovery process is blocked until all OSTs are available.

Determining Which Machine is Serving an OST In the course of administering a Lustre file system, you may need to determine which machine is serving a specific OST. It is not as simple as identifying the machine’s IP address, as IP is only one of several networking protocols that the Lustre software uses and, as such, LNet does not use IP addresses as node identifiers, but NIDs instead. To identify the NID that is serving a specific OST, run one of the following commands on a client (you do not need to be a root user):

client$ lctl get\_param osc.*fsname*-OST*number*\*.ost\_conn\_uuid

## Changing the Address of a Failover Node

To change the address of a failover node (e.g, to use node X instead of node Y), run this command on the OSS/OST partition (depending on which option was used to originally identify the NID):

oss# tunefs.lustre --erase-params --servicenode=*NID /dev/ost\_device*

or

oss# tunefs.lustre --erase-params --failnode=*NID /dev/ost\_device*

For more information about the --servicenode and --failnode options, see [Chapter 11,](#_bookmark221)

[*Configuring Failover in a Lustre File System*](#_bookmark221).

## Separate a combined MGS/MDT

These instructions assume the MGS node will be the same as the MDS node. For instructions on how to [move MGS to a different node, see Section 14.5, “ Changing a Server NID”](#_bookmark312).

These instructions are for doing the split without shutting down other servers and clients.

* + 1. Stop the MDS.

Unmount the MDT

umount -f */dev/mdt\_device*

* + 1. Create the MGS.

mds# mkfs.lustre --mgs --device-size=*size /dev/mgs\_device*

* + 1. Copy the configuration data from MDT disk to the new MGS disk.

mds# mount -t ldiskfs -o ro */dev/mdt\_device /mdt\_mount\_point*

mds# mount -t ldiskfs -o rw */dev/mgs\_device /mgs\_mount\_point*

mds# cp -r */mdt\_mount\_point*/CONFIGS/*filesystem\_name*-\* */mgs\_mount\_point*/CONFIGS/.

mds# umount */mgs\_mount\_point*

mds# umount */mdt\_mount\_point*

[See Section 14.4, “ Regenerating Lustre Configuration Logs”](#_bookmark310) for alternative method.

* + 1. Start the MGS.

mgs# mount -t lustre */dev/mgs\_device /mgs\_mount\_point*

Check to make sure it knows about all your file system

mgs:/root# lctl get\_param mgs.MGS.filesystems

* + 1. Remove the MGS option from the MDT, and set the new MGS nid.

mds# tunefs.lustre --nomgs --mgsnode=*new\_mgs\_nid /dev/mdt-device*

* + 1. Start the MDT.

mds# mount -t lustre */dev/mdt\_device /mdt\_mount\_point*

Check to make sure the MGS configuration looks right:

mgs# lctl get\_param mgs.MGS.live.*filesystem\_name*

# Chapter 15. Managing Lustre Networking (LNet)

This chapter describes some tools for managing Lustre networking (LNet).

## Updating the Health Status of a Peer or Router

There are two mechanisms to update the health status of a peer or a router:

* + - LNet can actively check health status of all routers and mark them as dead or alive automatically. By default, this is off. To enable it set auto\_down and if desired check\_routers\_before\_use. This initial check may cause a pause equal to router\_ping\_timeout at system startup, if there are dead routers in the system.
    - When there is a communication error, all LNDs notify LNet that the peer (not necessarily a router) is down. This mechanism is always on, and there is no parameter to turn it off. However, if you set the LNet module parameter auto\_down to 0, LNet ignores all such peer-down notifications.

Several key differences in both mechanisms:

* + - The router pinger only checks routers for their health, while LNDs notices all dead peers, regardless of whether they are a router or not.
    - The router pinger actively checks the router health by sending pings, but LNDs only notice a dead peer when there is network traffic going on.
    - The router pinger can bring a router from alive to dead or vice versa, but LNDs can only bring a peer down.

## Starting and Stopping LNet

The Lustre software automatically starts and stops LNet, but it can also be manually started in a standalone manner. This is particularly useful to verify that your networking setup is working correctly before you attempt to start the Lustre file system.

### Starting LNet

To start LNet, run:

$ modprobe lnet

$ lctl network up

To see the list of local NIDs, run:

$ lctl list\_nids

This command tells you the network(s) configured to work with the Lustre file system.

If the networks are not correctly setup, see the modules.conf "networks=" line and make sure the network layer modules are correctly installed and configured.

To get the best remote NID, run:

$ lctl which\_nid *NIDs*

where *NIDs* is the list of available NIDs.

This command takes the "best" NID from a list of the NIDs of a remote host. The "best" NID is the one that the local node uses when trying to communicate with the remote node.

##### Starting Clients

To start a TCP client, run:

mount -t lustre mdsnode:/mdsA/client /mnt/lustre/

To start an Elan client, run:

mount -t lustre 2@elan0:/mdsA/client /mnt/lustre

### Stopping LNet

Before the LNet modules can be removed, LNet references must be removed. In general, these references are removed automatically when the Lustre file system is shut down, but for standalone routers, an explicit step is needed to stop LNet. Run:

lctl network unconfigure

##### Note

Attempting to remove Lustre modules prior to stopping the network may result in a crash or an LNet hang. If this occurs, the node must be rebooted (in most cases). Make sure that the Lustre network and Lustre file system are stopped prior to unloading the modules. Be extremely careful using rmmod -f.

To unconfigure the LNet network, run:

modprobe -r *lnd\_and\_lnet\_modules*

##### Note

To remove all Lustre modules, run:

$ lustre\_rmmod

# Chapter 17. Upgrading a Lustre File System

This chapter describes interoperability between Lustre software releases. It also provides procedures for upgrading from Lustre software release 1.8 to Lustre software release 2.x , from a Lustre software release 2.x to a more recent Lustre software release 2.x (major release upgrade).

## Release Interoperability and Upgrade Requirements

Lustre software release 2.x (major) upgrade:

* + - All servers must be upgraded at the same time, while some or all clients may be upgraded independently of the servers.
    - All servers must be be upgraded to a Linux kernel supported by the Lustre software. See the Lustre Release Notes for your Lustre version for a list of tested Linux distributions.
    - Clients to be upgraded must be running a compatible Linux distribution as described in the Release Notes.

Lustre software release 2.x.y release (minor) upgrade:

* + - All servers must be upgraded at the same time, while some or all clients may be upgraded.
    - Rolling upgrades are supported for minor releases allowing individual servers and clients to be upgraded without stopping the Lustre file system.

## Upgrading to Lustre Software Release

The procedure for upgrading from a Lustre software release 2.x to a more recent 2.x release of the Lustre software is described in this section.

##### Note

This procedure can also be used to upgrade Lustre software release 1.8.6-wc1 or later to any Lustre software release 2.x. To upgrade other versions of Lustre software release 1.8.x, contact your support provider.

##### Note

In Lustre software release 2.2, a feature has been added for ldiskfs-based MDTs that allows striping a single file across up to 2000 OSTs. By default, this "wide striping" feature is disabled. It is activated by setting the ea\_inode option on the MDT using either mkfs.lustre or tune2fs. For example after upgrading an existing file system to Lustre software release 2.2 or later, wide striping can be enabled by running the following command on the MDT device before mounting it:

tune2fs -O large\_xattr

Once the wide striping feature is enabled and in use on the MDT, it is not possible to directly downgrade the MDT file system to an earlier version of the Lustre software that does not support wide striping. To disable wide striping:

* + 1. Delete all wide-striped files, *OR* use lfs\_migrate -c 160 (or fewer stripes) to migrate the files to use fewer OSTs. This does not affect the total number of OSTs that the whole filesystem can access.
    2. Unmount the MDT.
    3. Run the following command to turn off the large\_xattr option:

tune2fs -O ^large\_xattr

Using either mkfs.lustre or tune2fs with large\_xattr or ea\_inode option reseults in ea\_inode in the file system feature list.

To upgrade a Lustre software release 2.x to a more recent major release, complete these steps:

1. Create a complete, restorable file system backup.

##### Caution

Before installing the Lustre software, back up ALL data. The Lustre software contains kernel modifications that interact with storage devices and may introduce security issues and data loss if not installed, configured, or administered properly. If a full backup of the file system is not practical, a device-level backup of the MDT file system is recommended. See [Chapter 18,](#_bookmark399) [*Backing Up and Restoring a File System*](#_bookmark399)for a procedure.

1. Shut down the entire file system by following Section 13.4 “Stopping the file system”
2. Upgrade the Linux operating system on all servers to a compatible (tested) Linux distribution and reboot.
3. Upgrade the Linux operating system on all clients to Red Hat Enterprise Linux 6 or other compatible (tested) distribution and reboot.
4. Download the Lustre server RPMs for your platform from the [Lustre Releases](https://wiki.hpdd.intel.com/display/PUB/Lustre%2BReleases) [[https://](https://wiki.hpdd.intel.com/display/PUB/Lustre%2BReleases) [wiki.hpdd.intel.com/display/PUB/Lustre+Releases](https://wiki.hpdd.intel.com/display/PUB/Lustre%2BReleases)]repository. See [Table 8.1, “Packages Installed on](#_bookmark144) [Lustre Servers”](#_bookmark144)for a list of required packages.
5. Install the Lustre server packages on all Lustre servers (MGS, MDSs, and OSSs).
   1. Log onto a Lustre server as the root user
   2. Use the yum command to install the packages:

# yum --nogpgcheck install pkg1.rpm pkg2.rpm ...

* 1. Verify the packages are installed correctly:

rpm -qa|egrep "lustre|wc"

* 1. Repeat these steps on each Lustre server.

1. Download the Lustre client RPMs for your platform from the [Lustre Releases](https://wiki.hpdd.intel.com/display/PUB/Lustre%2BReleases) [[https://](https://wiki.hpdd.intel.com/display/PUB/Lustre%2BReleases) [wiki.hpdd.intel.com/display/PUB/Lustre+Releases](https://wiki.hpdd.intel.com/display/PUB/Lustre%2BReleases)]repository. See [Table 8.2, “Packages Installed on](#_bookmark145) [Lustre Clients”](#_bookmark145)for a list of required packages.

##### Note

The version of the kernel running on a Lustre client must be the same as the version of the lustre-client-modules- *ver*package being installed. If not, a compatible kernel must be installed on the client before the Lustre client packages are installed.

1. Install the Lustre client packages on each of the Lustre clients to be upgraded.
   1. Log onto a Lustre client as the root user.
   2. Use the yum command to install the packages:

# yum --nogpgcheck install pkg1.rpm pkg2.rpm ...

* 1. Verify the packages were installed correctly:

# rpm -qa|egrep "lustre|kernel"

* 1. Repeat these steps on each Lustre client.

1. (Optional) For upgrades to Lustre software release 2.2 or higher, to enable wide striping on an existing MDT, run the following command on the MDT:

tune2fs -O ea\_inode /dev/*mdtdev*

[For more information about wide striping, see Section 19.7, “Lustre Striping Internals”](#_bookmark474).

1. (Optional) For upgrades to Lustre software release 2.4 or higher, to format an additional MDT, complete these steps:
   1. Determine the index used for the first MDT (each MDT must have unique index). Enter:

client$ lctl dl | grep mdc

36 UP mdc lustre-MDT0000-mdc-ffff88004edf3c00 4c8be054-144f-9359-b063-8477566eb84e 5

In this example, the next available index is 1.

* 1. Add the new block device as a new MDT at the next available index by entering (on one line):

mds# mkfs.lustre --reformat --fsname=*filesystem\_name* --mdt \

--mgsnode=*mgsnode* --index *1*

*/dev/mdt1\_device*

10. (Optional) If you are upgrading to Lustre software release 2.3 or higher from Lustre software release

2.2 or earlier and want to enable the quota feature, complete these steps:

1. Before setting up the file system, enter on both the MDS and OSTs:

tunefs.lustre --quota

1. (Optional) If you are upgrading before Lustre software release 2.10, to enable the project quota feature enter the following on every ldiskfs backend target:

tune2fs –O project /dev/*dev*

##### Note

Enabling the project feature will prevent the filesystem from being used by older versions of ldiskfs, so it should only be enabled if the project quota feature is required and/ or after it is known that the upgraded release does not need to be downgraded.

1. When setting up the file system, enter:

conf\_param $FSNAME.quota.mdt=$QUOTA\_TYPE conf\_param $FSNAME.quota.ost=$QUOTA\_TYPE

11. (Optional) If you are upgrading from Lustre software release 1.8, you must manually enable the FID- in-dirent feature. On the MDS, enter:

tune2fs –O dirdata /dev/*mdtdev*

##### Warning

This step is not reversible. Do not complete this step until you are sure you will not be downgrading the Lustre software.

12. Start the Lustre file system by starting the components in the order shown in the following steps:

* 1. Mount the MGT. On the MGS, run

mgs# mount -a -t lustre

* 1. Mount the MDT(s). On each MDT, run:

mds# mount -a -t lustre

* 1. Mount all the OSTs. On each OSS node, run:

oss# mount -a -t lustre

##### Note

This command assumes that all the OSTs are listed in the /etc/fstab file. OSTs that are not listed in the /etc/fstab file, must be mounted individually by running the mount command:

mount -t lustre */dev/block\_device/mount\_point*

* 1. Mount the file system on the clients. On each client node, run:

client# mount -a -t lustre

##### Note

The mounting order described in the steps above must be followed for the initial mount and registration of a Lustre file system after an upgrade. For a normal start of a Lustre file system, the mounting order is MGT, OSTs, MDT(s), clients.

[If you have a problem upgrading a Lustre file system, see Section 31.2, “ Reporting a Lustre File System](#_bookmark772) [Bug”](#_bookmark772) for some ways to get help.

# Chapter 18. Backing Up and Restoring a File System

This chapter describes how to backup and restore at the file system-level, device-level and file-level in a Lustre file system.

It is *strongly* [recommended that sites perform periodic device-level backup of the MDT(s) (Section 18.2,](#_bookmark406) [“ Backing Up and Restoring an MDT or OST (ldiskfs Device Level)”](#_bookmark406)), for example twice a week with alternate backups going to a separate device, even if there is not enough capacity to do a full backup of all of the filesystem data. Even if there are separate file-level backups of some or all files in the filesystem, having a device-level backup of the MDT can be very useful in case of MDT failure or corruption. Being able to restore a device-level MDT backup can avoid the significantly longer process of restoring the entire filesystem from backup. Since the MDT is required for access to all files, its loss would otherwise force full restore of the filesystem (if that is even possible) even if the OSTs are still OK.

Performing a periodic device-level MDT backup can be done relatively inexpensively because the storage need only be connected to the primary MDS (it can be manually connected to the backup MDS in the rare case it is needed), and only needs good linear read/write performance. While the device-level MDT backup is not useful for restoring individual files, it is most efficient to handle the case of MDT failure or corruption.

## Backing up a File System

Backing up a complete file system gives you full control over the files to back up, and allows restoration of individual files as needed. File system-level backups are also the easiest to integrate into existing backup solutions.

File system backups are performed from a Lustre client (or many clients working parallel in different directories) rather than on individual server nodes; this is no different than backing up any other file system.

However, due to the large size of most Lustre file systems, it is not always possible to get a complete backup. We recommend that you back up subsets of a file system. This includes subdirectories of the entire file system, filesets for a single user, files incremented by date, and so on, so that restores can be done more efficiently.

##### Note

Lustre internally uses a 128-bit file identifier (FID) for all files. To interface with user applications, the 64-bit inode numbers are returned by the stat(), fstat(), and readdir() system calls on 64-bit applications, and 32-bit inode numbers to 32-bit applications.

Some 32-bit applications accessing Lustre file systems (on both 32-bit and 64-bit CPUs) may experience problems with the stat(), fstat() or readdir() system calls under certain circumstances, though the Lustre client should return 32-bit inode numbers to these applications.

In particular, if the Lustre file system is exported from a 64-bit client via NFS to a 32-bit client, the Linux NFS server will export 64-bit inode numbers to applications running on the NFS client. If the 32-bit applications are not compiled with Large File Support (LFS), then they return EOVERFLOW errors when accessing the Lustre files. To avoid this problem, Linux NFS clients can use the kernel command-line option "nfs.enable\_ino64=0" in order to force the NFS client to export 32-bit inode numbers to the client.

**Workaround**: We very strongly recommend that backups using tar(1) and other utilities that depend on the inode number to uniquely identify an inode to be run on 64-bit clients. The 128-bit Lustre file identifiers cannot be uniquely mapped to a 32-bit inode number, and as a result these utilities may operate incorrectly on 32-bit clients. While there is still a small chance of inode number collisions with 64-bit inodes, the FID allocation pattern is designed to avoid collisions for long periods of usage.

### Lustre\_rsync

The lustre\_rsync feature keeps the entire file system in sync on a backup by replicating the file system's changes to a second file system (the second file system need not be a Lustre file system, but it must be sufficiently large). lustre\_rsync uses Lustre changelogs to efficiently synchronize the file systems without having to scan (directory walk) the Lustre file system. This efficiency is critically important for large file systems, and distinguishes the Lustre lustre\_rsync feature from other replication/backup solutions. For more information on how to use Lustre rsync, please consult the Lustre manual.

## Backing Up and Restoring an MDT or OST (ldiskfs Device Level)

In some cases, it is useful to do a full device-level backup of an individual device (MDT or OST), before replacing hardware, performing maintenance, etc. Doing full device-level backups ensures that all of the data and configuration files is preserved in the original state and is the easiest method of doing a backup. For the MDT file system, it may also be the fastest way to perform the backup and restore, since it can do large streaming read and write operations at the maximum bandwidth of the underlying devices.

##### Note

Keeping an updated full backup of the MDT is especially important because permanent failure or corruption of the MDT file system renders the much larger amount of data in all the OSTs largely inaccessible and unusable. The storage needed for one or two full MDT device backups is much smaller than doing a full filesystem backup, and can use less expensive storage than the actual MDT device(s) since it only needs to have good streaming read/write speed instead of high random IOPS.

[[http://jira.hpdd.intel.com/browse/LU-957](http://jira.hpdd.intel.com/browse/LU-957)]), and file-level backup is supported (see [Section 18.3,](#_bookmark408) [“ Backing Up an OST or MDT (ldiskfs File System Level)”](#_bookmark408)).

If hardware replacement is the reason for the backup or if a spare storage device is available, it is possible to do a raw copy of the MDT or OST from one block device to the other, as long as the new device is at least as large as the original device. To do this, run:

dd if=/dev/{original} of=/dev/{newdev} bs=4M

If hardware errors cause read problems on the original device, use the command below to allow as much data as possible to be read from the original device while skipping sections of the disk with errors:

dd if=/dev/{original} of=/dev/{newdev} bs=4k conv=sync,noerror / count={original size in 4kB blocks}

Even in the face of hardware errors, the ldiskfs file system is very robust and it may be possible to recover the file system data after running e2fsck -fy /dev/{newdev} on the new device, along with ll\_recover\_lost\_found\_objs for OST devices.

In order to ensure that the backup is fully consistent, the MDT or OST must be unmounted, so that there are no changes being made to the device while the data is being transferred. If the reason for the backup is preventative (i.e. MDT backup on a running MDS in case of future failures) then it is possible to perform a consistent backup from an LVM snapshot. If an LVM snapshot is not available, and taking the MDS offline for a backup is unacceptable, it is also possible to perform a backup from the raw MDT block device. While the backup from the raw device will not be fully consistent due to ongoing changes, the vast majority of ldiskfs metadata is statically allocated, and inconsistencies in the backup can be fixed by running e2fsck on the backup device, and is still much better than not having any backup at all.

## Restoring a File-Level Backup

To restore data from a file-level backup, you need to format the device, restore the file data and then restore the EA data.

* + 1. Format the new device.

[oss]# mkfs.lustre --ost --index {*OST index*} {*other options*} /dev/*{newdev}*

* + 1. Set the file system label.

[oss]# e2label {fsname}-OST{index in hex} /mnt/ost

* + 1. Mount the file system.

[oss]# mount -t ldiskfs /dev/*{newdev}* /mnt/ost

* + 1. Change to the new file system mount point.

[oss]# cd /mnt/ost

* + 1. Restore the file system backup.

[oss]# tar xzvpf *{backup file}* [--xattrs] --sparse

* + 1. Restore the file system extended attributes.

[oss]# setfattr --restore=ea-${date}.bak

##### Note

If --xattrs option is supported by tar and specified in the step above, this step is redundant.

* + 1. Verify that the extended attributes were restored.

[oss]# getfattr -d -m ".\*" -e hex O/0/d0/100992 trusted.fid= \ 0x0d822200000000004a8a73e500000000808a0100000000000000000000000000

* + 1. Remove old OI files.

[oss]# rm -f oi.16\*

* + 1. Remove old CATALOGS.

[oss]# rm -f CATALOGS

##### Note

This is optional for the MDT side only. The CATALOGS record the llog file handlers that are used for recovering cross-server updates. Before OI scrub rebuilds the OI mappings for the llog files, the related recovery will get a failure if it runs faster than the background OI scrub. This will result in a failure of the whole mount process. OI scrub is an online tool, therefore, a mount failure means that the OI scrub will be stopped. Removing the old CATALOGS will avoid this potential trouble. The side-effect of removing old CATALOGS is that the recovery for related cross-server updates will be aborted. However, this can be handled by LFSCK after the system mount is up.

* + 1. Change directory out of the file system.

[oss]# cd -

* + 1. Unmount the new file system.

[oss]# umount /mnt/ost

## Using LVM Snapshots with the Lustre File System

If you want to perform disk-based backups (because, for example, access to the backup system needs to be as fast as to the primary Lustre file system), you can use the Linux LVM snapshot tool to maintain multiple, incremental file system backups.

Because LVM snapshots cost CPU cycles as new files are written, taking snapshots of the main Lustre file system will probably result in unacceptable performance losses. You should create a new, backup Lustre file system and periodically (e.g., nightly) back up new/changed files to it. Periodic snapshots can be taken of this backup file system to create a series of "full" backups.

##### Note

Creating an LVM snapshot is not as reliable as making a separate backup, because the LVM snapshot shares the same disks as the primary MDT device, and depends on the primary MDT device for much of its data. If the primary MDT device becomes corrupted, this may result in the snapshot being corrupted.

### 18.4.1. Creating an LVM-based Backup File System

Use this procedure to create a backup Lustre file system for use with the LVM snapshot mechanism.

1. Create LVM volumes for the MDT and OSTs.

Create LVM devices for your MDT and OST targets. Make sure not to use the entire disk for the targets; save some room for the snapshots. The snapshots start out as 0 size, but grow as you make changes to the current file system. If you expect to change 20% of the file system between backups, the most recent snapshot will be 20% of the target size, the next older one will be 40%, etc.

1. Format the LVM volumes as Lustre targets.

### Backing up New/Changed Files to the Backup File System

At periodic intervals e.g., nightly, back up new and changed files to the LVM-based backup file system.

cfs21:~# cp /etc/passwd /mnt/main cfs21:~# cp /etc/fstab /mnt/main cfs21:~# ls /mnt/main

fstab passwd

### 

### 18.4.3. Creating Snapshot Volumes

Whenever you want to make a "checkpoint" of the main Lustre file system, create LVM snapshots of all target MDT and OSTs in the LVM-based backup file system. You must decide the maximum size of a snapshot ahead of time, although you can dynamically change this later. The size of a daily snapshot is dependent on the amount of data changed daily in the main Lustre file system. It is likely that a two-day old snapshot will be twice as big as a one-day old snapshot.

You can create as many snapshots as you have room for in the volume group. If necessary, you can dynamically add disks to the volume group.

The snapshots of the target MDT and OSTs should be taken at the same point in time. Make sure that the cronjob updating the backup file system is not running, since that is the only thing writing to the disks.

### 18.4.4. Restoring the File System From a Snapshot

Use this procedure to restore the file system from an LVM snapshot.

1. Rename the LVM snapshot.

Rename the file system snapshot from "main" to "back" so you can mount it without unmounting "main". This is recommended, but not required. Use the --reformat flag to tunefs.lustre to force the name change.

When renaming a file system, we must also erase the last\_rcvd file from the snapshots

1. Mount the file system from the LVM snapshot.
2. Note the old directory contents, as of the snapshot time.

### Deleting Old Snapshots

To reclaim disk space, you can erase old snapshots as your backup policy dictates. Run:

lvremove /dev/vgmain/MDT0.b1

### Changing Snapshot Volume Size

You can also extend or shrink snapshot volumes if you find your daily deltas are smaller or larger than expected. Run:

lvextend -L10G /dev/vgmain/MDT0.b1

# Chapter 19. Managing File Layout (Striping) and Free Space

This chapter describes file layout (striping) and I/O options.

## How Lustre File System Striping Works

In a Lustre file system, the MDS allocates objects to OSTs using either a round-robin algorithm or a weighted algorithm. When the amount of free space is well balanced (i.e., by default, when the free space across OSTs differs by less than 17%), the round-robin algorithm is used to select the next OST to which a stripe is to be written. Periodically, the MDS adjusts the striping layout to eliminate some degenerated cases in which applications that create very regular file layouts (striping patterns) preferentially use a particular OST in the sequence.

Normally the usage of OSTs is well balanced. However, if users create a small number of exceptionally large files or incorrectly specify striping parameters, imbalanced OST usage may result. When the free space across OSTs differs by more than a specific amount (17% by default), the MDS then uses weighted random allocations with a preference for allocating objects on OSTs with more free space. (This can reduce I/O performance until space usage is rebalanced again.) For a more detailed description of how striping [is allocated, see Section 19.6, “Managing Free Space”](#_bookmark467).

Files can only be striped over a finite number of OSTs, based on the maximum size of the attributes that can be stored on the MDT. If the MDT is ldiskfs-based without the ea\_inode feature, a file can be striped across at most 160 OSTs. With a ZFS-based MDT, or if the ea\_inode feature is enabled for an ldiskfs-based MDT, a file can be striped across up to 2000 OSTs. For more information, see [Section 19.7,](#_bookmark474) [“Lustre Striping Internals”](#_bookmark474).

## Lustre File Layout (Striping) Considerations

Whether you should set up file striping and what parameter values you select depends on your needs. A good rule of thumb is to stripe over as few objects as will meet those needs and no more.

Some reasons for using striping include:

* + - **Providing high-bandwidth access.** Many applications require high-bandwidth access to a single file, which may be more bandwidth than can be provided by a single OSS. Examples are a scientific application that writes to a single file from hundreds of nodes, or a binary executable that is loaded by many nodes when an application starts.

In cases like these, a file can be striped over as many OSSs as it takes to achieve the required peak aggregate bandwidth for that file. Striping across a larger number of OSSs should only be used when the file size is very large and/or is accessed by many nodes at a time. Currently, Lustre files can be striped across up to 2000 OSTs, the maximum stripe count for an ldiskfs file system.

* + - **Improving performance when OSS bandwidth is exceeded.** Striping across many OSSs can improve performance if the aggregate client bandwidth exceeds the server bandwidth and the application reads and writes data fast enough to take advantage of the additional OSS bandwidth. The largest useful stripe count is bounded by the I/O rate of the clients/jobs divided by the performance per OSS.
    - **Providing space for very large files.** Striping is useful when a single OST does not have enough free space to hold the entire file.

Some reasons to minimize or avoid striping:

* + - **Increased overhead.** Striping results in more locks and extra network operations during common operations such as stat and unlink. Even when these operations are performed in parallel, one network operation takes less time than 100 operations.

Increased overhead also results from server contention. Consider a cluster with 100 clients and 100 OSSs, each with one OST. If each file has exactly one object and the load is distributed evenly, there is no contention and the disks on each server can manage sequential I/O. If each file has 100 objects, then the clients all compete with one another for the attention of the servers, and the disks on each node seek in 100 different directions resulting in needless contention.

* + - **Increased risk.** When files are striped across all servers and one of the servers breaks down, a small part of each striped file is lost. By comparison, if each file has exactly one stripe, fewer files are lost, but they are lost in their entirety. Many users would prefer to lose some of their files entirely than all of their files partially.

### Choosing a Stripe Size

Choosing a stripe size is a balancing act, but reasonable defaults are described below. The stripe size has no effect on a single-stripe file.

* + - * **The stripe size must be a multiple of the page size.** Lustre software tools enforce a multiple of 64 KB (the maximum page size on ia64 and PPC64 nodes) so that users on platforms with smaller pages do not accidentally create files that might cause problems for ia64 clients.
      * **The smallest recommended stripe size is 512 KB.** Although you can create files with a stripe size of 64 KB, the smallest practical stripe size is 512 KB because the Lustre file system sends 1MB chunks over the network. Choosing a smaller stripe size may result in inefficient I/O to the disks and reduced performance.
      * **A good stripe size for sequential I/O using high-speed networks is between 1 MB and 4 MB.** In most situations, stripe sizes larger than 4 MB may result in longer lock hold times and contention during shared file access.
      * **The maximum stripe size is 4 GB.** Using a large stripe size can improve performance when accessing very large files. It allows each client to have exclusive access to its own part of a file. However, a large stripe size can be counterproductive in cases where it does not match your I/O pattern.

**Choose a stripe pattern that takes into account the write patterns of your application.** Writes that cross an object boundary are slightly less efficient than writes that go entirely to one server. If the file is written in a consistent and aligned way, make the stripe size a multiple of the write() size.

## Setting the File Layout/Striping Configuration (lfs setstripe)

Use the lfs setstripe command to create new files with a specific file layout (stripe pattern) configuration.

lfs setstripe [--size|-s stripe\_size] [--count|-c stripe\_count] \ [--index|-i start\_ost] [--pool|-p pool\_name] *filename|dirname*

stripe\_size

The stripe\_size indicates how much data to write to one OST before moving to the next OST. The default stripe\_size is 1 MB. Passing a stripe\_size of 0 causes the default stripe size to be used. Otherwise, the stripe\_size value must be a multiple of 64 KB.

stripe\_count

The stripe\_count indicates how many OSTs to use. The default stripe\_count value is 1. Setting stripe\_count to 0 causes the default stripe count to be used. Setting stripe\_count to -1 means stripe over all available OSTs (full OSTs are skipped).

start\_ost

The start OST is the first OST to which files are written. The default value for start\_ost is -1, which allows the MDS to choose the starting index. This setting is strongly recommended, as it allows space and load balancing to be done by the MDS as needed. If the value of start\_ost is set to a value other than

-1, the file starts on the specified OST index. OST index numbering starts at 0.

**Note**

If the specified OST is inactive or in a degraded mode, the MDS will silently choose another target.

##### Note

If you pass a start\_ost value of 0 and a stripe\_count value of *1*, all files are written to OST 0, until space is exhausted. *This is probably not what you meant to do.* If you only want to adjust the stripe count and keep the other parameters at their default settings, do not specify any of the other parameters:

client# lfs setstripe -c *stripe\_count filename*

pool\_name

The pool\_name specifies the OST pool to which the file will be written. This allows limiting the OSTs used to a subset of all OSTs in the file system.

### Specifying a File Layout (Striping Pattern) for a Single File

It is possible to specify the file layout when a new file is created using the command lfs setstripe. This allows users to override the file system default parameters to tune the file layout more optimally for their application. Execution of an lfs setstripe command fails if the file already exists.

##### Setting the Stripe Size

The command to create a new file with a specified stripe size is similar to:

[client]# lfs setstripe -s 4M /mnt/lustre/new\_file

This example command creates the new file /mnt/lustre/new\_file with a stripe size of 4 MB. Now, when the file is created, the new stripe setting creates the file on a single OST with a stripe size of 4M:

[client]# lfs getstripe /mnt/lustre/new\_file

/mnt/lustre/4mb\_file

|  |  |  |
| --- | --- | --- |
| lmm\_stripe\_count: | 1 |  |
| lmm\_stripe\_size: | 4194304 |
| lmm\_pattern: | 1 |
| lmm\_layout\_gen: | 0 |
| lmm\_stripe\_offset: | 1 |
| obdidx objid | objid | group |
| 1 690550 | 0xa8976 | 0 |

In this example, the stripe size is 4 MB.

##### Setting the Stripe Count

The command below creates a new file with a stripe count of -1 to specify striping over all available OSTs:

[client]# lfs setstripe -c -1 /mnt/lustre/full\_stripe

The example below indicates that the file full\_stripe is striped over all six active OSTs in the configuration:

[client]# lfs getstripe /mnt/lustre/full\_stripe

/mnt/lustre/full\_stripe

|  |  |  |  |
| --- | --- | --- | --- |
| obdidx | objid | objid | group |
| 0 | 8 | 0x8 | 0 |
| 1 | 4 | 0x4 | 0 |
| 2 | 5 | 0x5 | 0 |
| 3 | 5 | 0x5 | 0 |
| 4 | 4 | 0x4 | 0 |
| 5 | 2 | 0x2 | 0 |

This is in contrast to the output in [Section 19.3.1.1, “Setting the Stripe Size”](#_bookmark440), which shows only a single object for the file.

### Setting the Striping Layout for a Directory

In a directory, the lfs setstripe command sets a default striping configuration for files created in the directory. The usage is the same as lfs setstripe for a regular file, except that the directory must exist prior to setting the default striping configuration. If a file is created in a directory with a default stripe configuration (without otherwise specifying striping), the Lustre file system uses those striping parameters instead of the file system default for the new file.

To change the striping pattern for a sub-directory, create a directory with desired file layout as described above. Sub-directories inherit the file layout of the root/parent directory.

### Setting the Striping Layout for a File System

Setting the striping specification on the root directory determines the striping for all new files created in the file system unless an overriding striping specification takes precedence (such as a striping layout specified by the application, or set using lfs setstripe, or specified for the parent directory).

##### Note

The striping settings for a root directory are, by default, applied to any new child directories created in the root directory, unless striping settings have been specified for the child directory.

### Creating a File on a Specific OST

You can use lfs setstripe to create a file on a specific OST. In the following example, the file

file1 is created on the first OST (OST index is 0).

$ lfs setstripe --count 1 --index 0 file1

$ dd if=/dev/zero of=file1 count=1 bs=100M 1+0 records in

1+0 records out

$ lfs getstripe file1

/mnt/testfs/file1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| lmm\_stripe\_count: | | 1 | | |
| lmm\_stripe\_size: | | 1048576 | | |
| lmm\_pattern: | | 1 | | |
| lmm\_layout\_gen: | | 0 | | |
| lmm\_stripe\_offset: | | 0 | | |
| obdidx | objid | | objid | group |
| 0 | 37364 | | 0x91f4 | 0 |

## Retrieving File Layout/Striping Information (getstripe)

The lfs getstripe command is used to display information that shows over which OSTs a file is distributed. For each OST, the index and UUID is displayed, along with the OST index and object ID for each stripe in the file. For directories, the default settings for files created in that directory are displayed.

### Displaying the Current Stripe Size

To see the current stripe size for a Lustre file or directory, use the lfs getstripe command. For example, to view information for a directory, enter a command similar to:

[client]# lfs getstripe /mnt/lustre

This command produces output similar to:

/mnt/lustre

(Default) stripe\_count: 1 stripe\_size: 1M stripe\_offset: -1

In this example, the default stripe count is 1 (data blocks are striped over a single OST), the default stripe size is 1 MB, and the objects are created over all available OSTs.

To view information for a file, enter a command similar to:

$ lfs getstripe /mnt/lustre/foo

/mnt/lustre/foo

|  |  |  |
| --- | --- | --- |
| lmm\_stripe\_count: | 1 |  |
| lmm\_stripe\_size: | 1048576 |
| lmm\_pattern: | 1 |
| lmm\_layout\_gen: | 0 |
| lmm\_stripe\_offset: | 0 |
| obdidx objid | objid | group |
| 2 835487 | m0xcbf9f | 0 |

In this example, the file is located on obdidx 2, which corresponds to the OST lustre-OST0002. To see which node is serving that OST, run:

$ lctl get\_param osc.lustre-OST0002-osc.ost\_conn\_uuid osc.lustre-OST0002-osc.ost\_conn\_uuid=192.168.20.1@tcp

### Inspecting the File Tree

To inspect an entire tree of files, use the lfs find command:

lfs find [--recursive | -r] *file|directory* ...

**19.5.** **Progressive File Layout(PFL)**

The Lustre Progressive File Layout (PFL) feature simplifies the use of Lustre so that users can expect reasonable performance for a variety of normal file IO patterns without the need to explicitly understand their IO model or Lustre usage details in advance. In particular, users do not necessarily need to know the size or concurrency of output files in advance of their creation and explicitly specify an optimal layout for each file in order to achieve good performance for both highly concurrent shared-single-large-file IO or parallel IO to many smaller per-process files.

The layout of a PFL file is stored on disk as composite layout. A PFL file is essentially an array of sub-layout components, with each sub-layout component being a plain layout covering different and non-overlapped extents of the file. For PFL files, the file layout is composed of a series of components, therefore it's possible that there are some file extents are not described by any components. Please reference the Lustre manual in order to know how to create a PFL file.

## Managing Free Space

To optimize file system performance, the MDT assigns file stripes to OSTs based on two allocation algorithms. The *round-robin* allocator gives preference to location (spreading out stripes across OSSs to increase network bandwidth utilization) and the weighted allocator gives preference to available space (balancing loads across OSTs). Threshold and weighting factors for these two algorithms can be adjusted by the user. The MDT reserves 0.1 percent of total OST space and 32 inodes for each OST. The MDT stops object allocation for the OST if available space is less than reserved or the OST has fewer than 32 free inodes. The MDT starts object allocation when available space is twice as big as the reserved space and the OST has more than 64 free inodes. Note, clients could append existing files no matter what object allocation state is.

This section describes how to check available free space on disks and how free space is allocated. It then describes how to set the threshold and weighting factors for the allocation algorithms.

### Checking File System Free Space

Free space is an important consideration in assigning file stripes. The lfs df command can be used to show available disk space on the mounted Lustre file system and space consumption per OST. If multiple Lustre file systems are mounted, a path may be specified, but is not required. Options to the lfs df command are shown below.

|  |  |
| --- | --- |
| **Option** | **Description** |
| -h | Displays sizes in human readable format (for example: 1K, 234M, 5G). |
| -i, --inodes | Lists inodes instead of block usage. |

##### Note

The df -i and lfs df -i commands show the *minimum* number of inodes that can be created in the file system at the current time. If the total number of objects available across all of the OSTs is smaller than those available on the MDT(s), taking into account the default file striping, then df -i will also report a smaller number of inodes than could be created. Running lfs df

-i will report the actual number of inodes that are free on each target.

For ZFS file systems, the number of inodes that can be created is dynamic and depends on the free space in the file system. The Free and Total inode counts reported for a ZFS file system are only an estimate based on the current usage for each target. The Used inode count is the actual number of inodes used by the file system.

### Stripe Allocation Methods

Two stripe allocation methods are provided:

* **Round-robin allocator** - When the OSTs have approximately the same amount of free space, the round- robin allocator alternates stripes between OSTs on different OSSs, so the OST used for stripe 0 of each file is evenly distributed among OSTs, regardless of the stripe count. In a simple example with eight OSTs numbered 0-7, objects would be allocated like this:

File 1: OST1, OST2, OST3, OST4 File 2: OST5, OST6, OST7

File 3: OST0, OST1, OST2, OST3, OST4, OST5 File 4: OST6, OST7, OST0

Here are several more sample round-robin stripe orders (each letter represents a different OST on a single OSS):

3: AAA

3x3: ABABAB

3x4: BBABABA

3x5: BBABBABA

3x3x3: ABCABCABC

One 3-OST OSS

Two 3-OST OSSs

One 3-OST OSS (A) and one 4-OST OSS (B) One 3-OST OSS (A) and one 5-OST OSS (B)

Three 3-OST OSSs

* **Weighted allocator** - When the free space difference between the OSTs becomes significant, the weighting algorithm is used to influence OST ordering based on size (amount of free space available on each OST) and location (stripes evenly distributed across OSTs). The weighted allocator fills the emptier OSTs faster, but uses a weighted random algorithm, so the OST with the most free space is not necessarily chosen each time.

The allocation method is determined by the amount of free-space imbalance on the OSTs. When free space is relatively balanced across OSTs, the faster round-robin allocator is used, which maximizes network balancing. The weighted allocator is used when any two OSTs are out of balance by more than the specified threshold (17% by default). The threshold between the two allocation methods is defined in the file / proc/fs/*fsname*/lov/*fsname*-mdtlov/qos\_threshold\_rr.

To set the qos\_threshold\_r to 25, enter this command on the MGS:

lctl set\_param lov.*fsname*-mdtlov.qos\_threshold\_rr=25

### 19.6.3. Adjusting the Weighting Between Free Space and Location

The weighting priority used by the weighted allocator is set in the file /proc/fs/*fsname*/lov/ *fsname*-mdtlov/qos\_prio\_free. Increasing the value of qos\_prio\_free puts more weighting on the amount of free space available on each OST and less on how stripes are distributed across OSTs. The default value is 91 (percent). When the free space priority is set to 100 (percent), weighting is based entirely on free space and location is no longer used by the striping algorithm.

To change the allocator weighting to 100, enter this command on the MGS:

lctl conf\_param *fsname*-MDT0000.lov.qos\_prio\_free=100

##### Note

When qos\_prio\_free is set to 100, a weighted random algorithm is still used to assign stripes, so, for example, if OST2 has twice as much free space as OST1, OST2 is twice as likely to be used, but it is not guaranteed to be used.

## 19.7. Lustre Striping Internals

Individual files can only be striped over a finite number of OSTs, based on the maximum size of the attributes that can be stored on the MDT. If the MDT is ldiskfs-based without the ea\_inode feature, a file can be striped across at most 160 OSTs. With ZFS-based MDTs, or if the ea\_inode feature is enabled for an ldiskfs-based MDT, a file can be striped across up to 2000 OSTs.

Lustre inodes use an extended attribute to record on which OST each object is located, and the identifier each object on that OST. The size of the extended attribute is a function of the number of stripes.

If using an ldiskfs-based MDT, the maximum number of OSTs over which files can be striped can been raised to 2000 by enabling the ea\_inode feature on the MDT:

tune2fs -O ea\_inode /dev/*mdtdev*

##### Note

The maximum stripe count for a single file does not limit the maximum number of OSTs that are in the filesystem as a whole, only the maximum possible size and maximum aggregate bandwidth for the file.

# Chapter 20. Managing the File System and I/O

## Handling Full OSTs

Sometimes a Lustre file system becomes unbalanced, often due to incorrectly-specified stripe settings, or when very large files are created that are not striped over all of the OSTs. If an OST is full and an attempt is made to write more information to the file system, an error occurs. The procedures below describe how to handle a full OST.

The MDS will normally handle space balancing automatically at file creation time, and this procedure is normally not needed, but may be desirable in certain circumstances (e.g. when creating very large files that would consume more than the total free space of the full OSTs).

### Checking OST Space Usage

The example below shows an unbalanced file system:

client# lfs df -h

UUID bytes Used Available \ Use% Mounted on

testfs-MDT0000\_UUID 4.4G 214.5M 3.9G \

4% /mnt/testfs[MDT:0]

testfs-OST0000\_UUID 2.0G 751.3M 1.1G \

37% /mnt/testfs[OST:0]

testfs-OST0001\_UUID 2.0G 755.3M 1.1G \

37% /mnt/testfs[OST:1]

testfs-OST0002\_UUID 2.0G 1.7G 155.1M \

86% /mnt/testfs[OST:2] \*\*\*\*

testfs-OST0003\_UUID 2.0G 751.3M 1.1G \

37% /mnt/testfs[OST:3]

testfs-OST0004\_UUID 2.0G 747.3M 1.1G \

37% /mnt/testfs[OST:4]

testfs-OST0005\_UUID 2.0G 743.3M 1.1G \

36% /mnt/testfs[OST:5]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| filesystem | summary: 11.8G | 5.4G | 5.8G | \ |
| 45% | /mnt/testfs |  |  |  |

In this case, OST0002 is almost full and when an attempt is made to write additional information to the file system (even with uniform striping over all the OSTs).

### Taking a Full OST Offline

To avoid running out of space in the file system, if the OST usage is imbalanced and one or more OSTs are close to being full while there are others that have a lot of space, the full OSTs may optionally be deactivated at the MDS to prevent the MDS from allocating new objects there.

1. Log into the MDS server:

[client# ssh root@192.168.0.10](mailto:root@192.168.0.10) root@192.168.0.10's password:

Last login: Wed Nov 26 13:35:12 2008 from 192.168.0.6

1. Use the lctl dl command to show the status of all file system components:
2. Use lctl deactivate to take the full OST offline:

mds# lctl --device 7 deactivate

1. Display the status of the file system components:

mds# lctl dl

0 UP mgs MGS MGS 9

1 UP mgc MGC192.168.0.10@tcp e384bb0e-680b-ce25-7bc9-81655dd1e813 5

1. UP mdt MDS MDS\_uuid 3
2. UP lov testfs-mdtlov testfs-mdtlov\_UUID 4
3. UP mds testfs-MDT0000 testfs-MDT0000\_UUID 5
4. UP osc testfs-OST0000-osc testfs-mdtlov\_UUID 5
5. UP osc testfs-OST0001-osc testfs-mdtlov\_UUID 5
6. IN osc testfs-OST0002-osc testfs-mdtlov\_UUID 5
7. UP osc testfs-OST0003-osc testfs-mdtlov\_UUID 5
8. UP osc testfs-OST0004-osc testfs-mdtlov\_UUID 5
9. UP osc testfs-OST0005-osc testfs-mdtlov\_UUID 5

The device list shows that OST0002 is now inactive. When new files are created in the file system, they will only use the remaining active OSTs. Either manual space rebalancing can be done by migrating data to other OSTs, as shown in the next section, or normal file deletion and creation can be allowed to passively rebalance the space usage.

### Migrating Data within a File System

If there is a need to migrate the file data from the current OST(s) to new OSTs, the data must be migrated (copied) to the new location. The simplest way to do this is to use the lfs\_migrate command (see [Section 36.2, “ lfs\_migrate ”](#_bookmark957)). However, the steps for migrating a file by hand are also shown here for reference.

* + - 1. Identify the file(s) to be moved.

In the example below, the object information portion of the output from the lfs getstripe

command below shows that the test\_2file is located entirely on OST0002:

client# lfs getstripe /mnt/testfs/test\_2

/mnt/testfs/test\_2

|  |  |  |
| --- | --- | --- |
| obdidx | objid objid | group |
| 2 | 8 0x8 | 0 |

* + - 1. To move the data, create a copy and remove the original:

client# cp -a /mnt/testfs/test\_2 /mnt/testfs/test\_2.tmp

client# mv /mnt/testfs/test\_2.tmp /mnt/testfs/test\_2

* + - 1. If the space usage of OSTs is severely imbalanced, it is possible to find and migrate large files from their current location onto OSTs that have more space, one could run:

client# lfs find --ost

*ost\_name* -size +1G | lfs\_migrate -y

* + - 1. Check the file system balance.

The lfs df output in the example below shows a more balanced system compared to the lfs df [output in the example in Section 20.1, “ Handling Full OSTs”](#_bookmark477).

### Returning an Inactive OST Back Online

Once the deactivated OST(s) no longer are severely imbalanced, due to either active or passive data redistribution, they should be reactivated so they will again have new files allocated on them.

[mds]# lctl --device 7 activate [mds]# lctl dl

0 UP mgs MGS MGS 9

1 UP mgc MGC192.168.0.10@tcp e384bb0e-680b-ce25-7bc9-816dd1e813 5

1. UP mdt MDS MDS\_uuid 3
2. UP lov testfs-mdtlov testfs-mdtlov\_UUID 4
3. UP mds testfs-MDT0000 testfs-MDT0000\_UUID 5
4. UP osc testfs-OST0000-osc testfs-mdtlov\_UUID 5
5. UP osc testfs-OST0001-osc testfs-mdtlov\_UUID 5
6. UP osc testfs-OST0002-osc testfs-mdtlov\_UUID 5
7. UP osc testfs-OST0003-osc testfs-mdtlov\_UUID 5
8. UP osc testfs-OST0004-osc testfs-mdtlov\_UUID 5
9. UP osc testfs-OST0005-osc testfs-mdtlov\_UUID

## Creating and Managing OST Pools

The OST pools feature enables users to group OSTs together to make object placement more flexible. A 'pool' is the name associated with an arbitrary subset of OSTs in a Lustre cluster.

OST pools follow these rules:

* An OST can be a member of multiple pools.
* No ordering of OSTs in a pool is defined or implied.
* Stripe allocation within a pool follows the same rules as the normal stripe allocator.
* OST membership in a pool is flexible, and can change over time.

When an OST pool is defined, it can be used to allocate files. When file or directory striping is set to a pool, only OSTs in the pool are candidates for striping. If a stripe\_index is specified which refers to an OST that is not a member of the pool, an error is returned.

OST pools are used only at file creation. If the definition of a pool changes (an OST is added or removed or the pool is destroyed), already-created files are not affected.

##### Note

An error ( EINVAL) results if you create a file using an empty pool.

##### Note

If a directory has pool striping set and the pool is subsequently removed, the new files created in this directory have the (non-pool) default striping pattern for that directory applied and no error is returned.

### Working with OST Pools

OST pools are defined in the configuration log on the MGS. Use the lctl command to:

* Create/destroy a pool
* Add/remove OSTs in a pool
* List pools and OSTs in a specific pool

The lctl command MUST be run on the MGS. Another requirement for managing OST pools is to either have the MDT and MGS on the same node or have a Lustre client mounted on the MGS node, if it is separate from the MDS. This is needed to validate the pool commands being run are correct.

##### Caution

Running the writeconf command on the MDS erases all pools information (as well as any other parameters set using lctl conf\_param). We recommend that the pools definitions (and conf\_param settings) be executed using a script, so they can be reproduced easily after a writeconf is performed.

To create a new pool, run:

mgs# lctl pool\_new *fsname*. *poolname*

##### Note

The pool name is an ASCII string up to 15 characters.

To add the named OST to a pool, run:

mgs# lctl pool\_add *fsname*. *poolname ost\_list*

Where:

* *ost\_list*is *fsname*-OST *index\_range*
* *index\_range*is *ost\_index\_start*- *ost\_index\_end[,index\_range]* or

*ost\_index\_start*- *ost\_index\_end/step*

If the leading *fsname* and/or ending \_UUID are missing, they are automatically added.

For example, to add even-numbered OSTs to pool1 on file system testfs, run a single command (pool\_add) to add many OSTs to the pool at one time:

lctl pool\_add testfs.pool1 OST[0-10/2]

##### Note

Each time an OST is added to a pool, a new llog configuration record is created. For convenience, you can run a single command.

To remove a named OST from a pool, run:

mgs# lctl pool\_remove *fsname*. *poolname ost\_list*

To destroy a pool, run:

mgs# lctl pool\_destroy *fsname*. *poolname*

##### Note

All OSTs must be removed from a pool before it can be destroyed.

To list pools in the named file system, run:

mgs# lctl pool\_list *fsname|pathname*

To list OSTs in a named pool, run:

lctl pool\_list fsname.poolname

**20.2.1.1. Using the lfs Command with OST Pools**

Several lfs commands can be run with OST pools. Use the lfs setstripe command to associate a directory with an OST pool. This causes all new regular files and directories in the directory to be created in the pool. The lfs command can be used to list pools in a file system and OSTs in a named pool.

To associate a directory with a pool, so all new files and directories will be created in the pool, run:

client# lfs setstripe --pool|-p pool\_name

*filename|dirname*

To set striping patterns, run:

client# lfssetstripe [--size|-s stripe\_size] [--offset|-o start\_ost] [--count|-c stripe\_count] [--pool|-p pool\_name]

*dir|filename*

##### Note

If you specify striping with an invalid pool name, because the pool does not exist or the pool name was mistyped, lfs setstripe returns an error. Run lfs pool\_list to make sure the pool exists and the pool name is entered correctly.

##### Note

The --pool option for lfs setstripe is compatible with other modifiers. For example, you can set striping on a directory to use an explicit starting index.

### Tips for Using OST Pools

Here are several suggestions for using OST pools.

* + - * A directory or file can be given an extended attribute (EA), that restricts striping to a pool.
      * Pools can be used to group OSTs with the same technology or performance (slower or faster), or that are preferred for certain jobs. Examples are SATA OSTs versus SAS OSTs or remote OSTs versus local OSTs.
      * A file created in an OST pool tracks the pool by keeping the pool name in the file LOV EA.

## Adding an OST to a Lustre File System

To add an OST to existing Lustre file system:

* + 1. Add a new OST by passing on the following commands, run:

oss# mkfs.lustre --fsname=testfs --mgsnode=mds16@tcp0 --ost --index=12 /dev/sda oss# mkdir -p /mnt/testfs/ost12

oss# mount -t lustre /dev/sda /mnt/testfs/ost12

* + 1. Migrate the data (possibly).

The file system is quite unbalanced when new empty OSTs are added. New file creations are automatically balanced. If this is a scratch file system or files are pruned at a regular interval, then no further work may be needed. Files existing prior to the expansion can be rebalanced with an in-place copy, which can be done with a simple script.

The basic method is to copy existing files to a temporary file, then move the temp file over the old one. This should not be attempted with files which are currently being written to by users or applications. This operation redistributes the stripes over the entire set of OSTs.

A very clever migration script would do the following:

* + - * Examine the current distribution of data.
      * Calculate how much data should move from each full OST to the empty ones.
      * Search for files on a given full OST (using lfs getstripe).
      * Force the new destination OST (using lfs setstripe).
      * Copy only enough files to address the imbalance.

If a Lustre file system administrator wants to explore this approach further, per-OST disk-usage statistics can be found under /proc/fs/lustre/osc/\*/rpc\_stats

## Performing Direct I/O

The Lustre software supports the O\_DIRECT flag to open.

Applications using the read() and write() calls must supply buffers aligned on a page boundary (usually 4 K). If the alignment is not correct, the call returns -EINVAL. Direct I/O may help performance in cases where the client is doing a large amount of I/O and is CPU-bound (CPU utilization 100%).

### 20.4.1. Making File System Objects Immutable

An immutable file or directory is one that cannot be modified, renamed or removed. To do this:

chattr +i

*file*

To remove this flag, use chattr -i

## Other I/O Options

This section describes other I/O options, including checksums, and the ptlrpcd thread pool.

### Lustre Checksums

To guard against network data corruption, a Lustre client can perform two types of data checksums: in- memory (for data in client memory) and wire (for data sent over the network). For each checksum type, a 32-bit checksum of the data read or written on both the client and server is computed, to ensure that the data has not been corrupted in transit over the network. The ldiskfs backing file system does NOT do any persistent checksumming, so it does not detect corruption of data in the OST file system.

The checksumming feature is enabled, by default, on individual client nodes. If the client or OST detects a checksum mismatch, then an error is logged in the syslog of the form:

LustreError: BAD WRITE CHECKSUM: changed in transit before arrival at OST: \ from 192.168.1.1@tcp inum 8991479/2386814769 object 1127239/0 extent [10240\ 0-106495]

If this happens, the client will re-read or re-write the affected data up to five times to get a good copy of the data over the network. If it is still not possible, then an I/O error is returned to the application.

To enable both types of checksums (in-memory and wire), run:

lctl set\_param llite.\*.checksum\_pages=1

To disable both types of checksums (in-memory and wire), run:

lctl set\_param llite.\*.checksum\_pages=0

To check the status of a wire checksum, run:

lctl get\_param osc.\*.checksums

#### Changing Checksum Algorithms

By default, the Lustre software uses the adler32 checksum algorithm, because it is robust and has a lower impact on performance than crc32. The Lustre file system administrator can change the checksum algorithm via lctl get\_param, depending on what is supported in the kernel.

To check which checksum algorithm is being used by the Lustre software, run:

$ lctl get\_param osc.\*.checksum\_type

To change the wire checksum algorithm, run:

$ lctl set\_param osc.\*.checksum\_type=

*algorithm*

##### Note

The in-memory checksum always uses the adler32 algorithm, if available, and only falls back to crc32 if adler32 cannot be used.

In the following example, the lctl get\_param command is used to determine that the Lustre software is using the adler32 checksum algorithm. Then the lctl set\_param command is used to change the checksum algorithm to crc32. A second lctl get\_param command confirms that the crc32 checksum algorithm is now in use.

$ lctl get\_param osc.\*.checksum\_type

osc.testfs-OST0000-osc-ffff81012b2c48e0.checksum\_type=crc32 [adler]

$ lctl set\_param osc.\*.checksum\_type=crc32

osc.testfs-OST0000-osc-ffff81012b2c48e0.checksum\_type=crc32

$ lctl get\_param osc.\*.checksum\_type

osc.testfs-OST0000-osc-ffff81012b2c48e0.checksum\_type=[crc32] adler

### Ptlrpc Thread Pool

Releases prior to Lustre software release 2.2 used two portal RPC daemons for each client/server pair. One daemon handled all synchronous IO requests, and the second daemon handled all asynchronous (non-IO) RPCs. The increasing use of large SMP nodes for Lustre servers exposed some scaling issues. The lack of threads for large SMP nodes resulted in cases where a single CPU would be 100% utilized and other CPUs would be relativity idle. This is especially noticeable when a single client traverses a large directory.

Lustre software release 2.2.x implements a ptlrpc thread pool, so that multiple threads can be created to serve asynchronous RPC requests. The number of threads spawned is controlled at module load time using module options. By default one thread is spawned per CPU, with a minimum of 2 threads spawned irrespective of module options.

One of the issues with thread operations is the cost of moving a thread context from one CPU to another with the resulting loss of CPU cache warmth. To reduce this cost, ptlrpc threads can be bound to a CPU. However, if the CPUs are busy, a bound thread may not be able to respond quickly, as the bound CPU may be busy with other tasks and the thread must wait to schedule.

Because of these considerations, the pool of ptlrpc threads can be a mixture of bound and unbound threads. The system operator can balance the thread mixture based on system size and workload.

##### ptlrpcd parameters

These parameters should be set in /etc/modprobe.conf or in the etc/modprobe.d directory, as options for the ptlrpc module.

options ptlrpcd max\_ptlrpcds=XXX

Sets the number of ptlrpcd threads created at module load time. The default if not specified is one thread per CPU, including hyper-threaded CPUs. The lower bound is 2 (old prlrpcd behaviour)

options ptlrpcd ptlrpcd\_bind\_policy=[1-4]

Controls the binding of threads to CPUs. There are four policy options.

* + - * + PDB\_POLICY\_NONE(ptlrpcd\_bind\_policy=1) All threads are unbound.
        + PDB\_POLICY\_FULL(ptlrpcd\_bind\_policy=2) All threads attempt to bind to a CPU.
        + PDB\_POLICY\_PAIR(ptlrpcd\_bind\_policy=3) This is the default policy. Threads are allocated as a bound/unbound pair. Each thread (bound or free) has a partner thread. The partnering is used by the ptlrpcd load policy, which determines how threads are allocated to CPUs.
        + PDB\_POLICY\_NEIGHBOR(ptlrpcd\_bind\_policy=4) Threads are allocated as a bound/unbound pair. Each thread (bound or free) has two partner threads.

# Chapter 22. Configuring and Managing Quotas

## Working with Quotas

Quotas allow a system administrator to limit the amount of disk space a user, group, or project can use. Quotas are set by root, and can be specified for individual users, groups, and/or projects. Before a file is written to a partition where quotas are set, the quota of the creator's group is checked. If a quota exists, then the file size counts towards the group's quota. If no quota exists, then the owner's user quota is checked before the file is written. Similarly, inode usage for specific functions can be controlled if a user over- uses the allocated space.

Lustre quota enforcement differs from standard Linux quota enforcement in several ways:

* + - Quotas are administered via the lfs and lctl commands (post-mount).
    - The quota feature in Lustre software is distributed throughout the system (as the Lustre file system is a distributed file system). Because of this, quota setup and behavior on Lustre is different from local disk quotas in the following ways:
      * No single point of administration: some commands must be executed on the MGS, other commands on the MDSs and OSSs, and still other commands on the client.
      * Granularity: a local quota is typically specified for kilobyte resolution, Lustre uses one megabyte as the smallest quota resolution.
      * Accuracy: quota information is distributed throughout the file system and can only be accurately calculated with a completely quite file system.
    - Quotas are allocated and consumed in a quantized fashion.
    - Client does not set the usrquota or grpquota options to mount. As of Lustre software release 2.4, space accounting is always enabled by default and quota enforcement can be enabled/disabled on a per-file system basis with lctl conf\_param. It is worth noting that both lfs quotaon and quota\_type are deprecated as of Lustre software release 2.4.0.

##### Caution

Although a quota feature is available in the Lustre software, root quotas are NOT enforced.

lfs setquota -u root (limits are not enforced)

lfs quota -u root (usage includes internal Lustre data that is dynamic in size and does not accurately reflect mount point visible block and inode usage).

## Enabling Disk Quotas

The design of quotas on Lustre has management and enforcement separated from resource usage and accounting. Lustre software is responsible for management and enforcement. The back-end file system is responsible for resource usage and accounting. Because of this, it is necessary to begin enabling quotas by enabling quotas on the back-end disk system. Because quota setup is dependent on the Lustre software version in use, you may first need to run lctl get\_param version to identify [which version?](#_bookmark7) you are currently using.

### Enabling Disk Quotas

Quota setup is orchestrated by the MGS and *all setup commands in this section must be run on the MGS and project quotas need lustre Relase 2.10 and later*. Once setup, verification of the quota state must be performed on the MDT. Although quota enforcement is managed by the Lustre software, each OSD implementation relies on the back-end file system to maintain per-user/group/project block and inode usage. Hence, differences exist when setting up quotas with ldiskfs or ZFS back-ends:

* + - For ldiskfs backends, mkfs.lustre now creates empty quota files and enables the QUOTA feature flag in the superblock which turns quota accounting on at mount time automatically. e2fsck was also modified to fix the quota files when the QUOTA feature flag is present. The project quota feature is disabled by default, and tune2fs needs to be run to enable every target manually.
    - For ZFS backend, *the project quota feature is not supported yet.* Accounting ZAPs are created and maintained by the ZFS file system itself. While ZFS tracks per-user and group block usage, it does not handle inode accounting for ZFS versions prior to zfs-0.7.0. The ZFS OSD implements its own support for inode tracking. Two options are available:
      1. The ZFS OSD can estimate the number of inodes in-use based on the number of blocks used by a given user or group. This mode can be enabled by running the following command on the server running the target: lctl set\_param osd-zfs.${FSNAME}-

${TARGETNAME}.quota\_iused\_estimate=1.

* + - 1. Similarly to block accounting, dedicated ZAPs are also created the ZFS OSD to maintain per-user and group inode usage. This is the default mode which corresponds to quota\_iused\_estimate set to 0.

##### Note

Lustre file systems formatted with a Lustre release prior to 2.4.0 can be still safely upgraded to release 2.4.0, but will not have functional space usage report until tunefs.lustre --quota is run against all targets. This command sets the QUOTA feature flag in the superblock and runs e2fsck (as a result, the target must be offline) to build the per-UID/GID disk usage database.

##### Caution

Lustre software release 2.4 and later requires a version of e2fsprogs that supports quota (i.e. newer or equal to 1.42.13.wc5, 1.42.13.wc6 or newer is needed for project quota support) to be installed on the server nodes using ldiskfs backend (e2fsprogs is not needed with ZFS backend). In general, we recommend to use the latest e2fsprogs version available on [http://downloads.hpdd.intel.com/](http://downloads.hpdd.intel.com/e2fsprogs/) [public/e2fsprogs/](http://downloads.hpdd.intel.com/e2fsprogs/) [<http://downloads.hpdd.intel.com/e2fsprogs/>].

The ldiskfs OSD relies on the standard Linux quota to maintain accounting information on disk. As a consequence, the Linux kernel running on the Lustre servers using ldiskfs backend must have CONFIG\_QUOTA, CONFIG\_QUOTACTL and CONFIG\_QFMT\_V2 enabled.

As of Lustre software release 2.4.0, quota enforcement is thus turned on/off independently of space accounting which is always enabled. lfs quota *on|off* as well as the per-target quota\_type parameter are deprecated in favor of a single per-file system quota parameter controlling inode/block quota enforcement. Once the quota parameters have been configured, all targets which are part of the file system will be automatically notified of the new quota settings and enable/disable quota enforcement as needed.

## Quota Administration

Once the file system is up and running, quota limits on blocks and inodes can be set for user, group, and project. This is *controlled entirely from a client* via three quota parameters:

**Grace period**-- The period of time (in seconds) within which users are allowed to exceed their soft limit. There are six types of grace periods:

* + - user block soft limit
    - user inode soft limit
    - group block soft limit
    - group inode soft limit
    - project block soft limit
    - project inode soft limit

The grace period applies to all users. The user block soft limit is for all users who are using a blocks quota.

**Soft limit** -- The grace timer is started once the soft limit is exceeded. At this point, the user/group/project can still allocate block/inode. When the grace time expires and if the user is still above the soft limit, the soft limit becomes a hard limit and the user/group/project can't allocate any new block/inode any more. The user/group/project should then delete files to be under the soft limit. The soft limit MUST be smaller than the hard limit. If the soft limit is not needed, it should be set to zero (0).

**Hard limit** -- Block or inode allocation will fail with EDQUOT(i.e. quota exceeded) when the hard limit is reached. The hard limit is the absolute limit. When a grace period is set, one can exceed the soft limit within the grace period if under the hard limit.

Due to the distributed nature of a Lustre file system and the need to maintain performance under load, those quota parameters may not be 100% accurate. The quota settings can be manipulated via the lfs command, executed on a client, and includes several options to work with quotas:

* + - quota -- displays general quota information (disk usage and limits)
    - setquota -- specifies quota limits and tunes the grace period. By default, the grace period is one week. Usage:

lfs quota [-q] [-v] [-h] [-o obd\_uuid] [-u|-g|-p *uname|uid|gname|gid|projid*] */moun*

lfs quota -t {-u|-g|-p} */mount\_point*

lfs setquota {-u|--user|-g|--group|-p|--project} *username|groupname* [-b *block-soft*

[-B *block\_hardlimit*] [-i *inode\_softlimit*] \ [-I *inode\_hardlimit*] */mount\_point*

To display general quota information (disk usage and limits) for the user running the command and his primary group, run:

$ lfs quota /mnt/testfs

To display general quota information for a specific user (" bob" in this example), run:

$ lfs quota -u bob /mnt/testfs

To display general quota information for a specific user (" bob" in this example) and detailed quota statistics for each MDT and OST, run:

$ lfs quota -u bob -v /mnt/testfs

To display general quota information for a specific project (" 1" in this example), run:

$ lfs quota -p 1 /mnt/testfs

To display general quota information for a specific group (" eng" in this example), run:

$ lfs quota -g eng /mnt/testfs

To limit quota usage for a specific project ID on a specific directory ("/mnt/testfs/dir" in this example), run:

$ chattr +P /mnt/testfs/dir

$ chattr -p 1 /mnt/testfs/dir

$ lfs setquota -p 1 -b 307200 -B 309200 -i 10000 -I 11000 /mnt/testfs

Please note that if it is desired to have lfs quota -p show the space/inode usage under the directory properly (much faster than du), then the user/admin needs to use different project IDs for different directories.

To display block and inode grace times for user quotas, run:

$ lfs quota -t -u /mnt/testfs

To set user or group quotas for a specific ID ("bob" in this example), run:

$ lfs setquota -u bob -b 307200 -B 309200 -i 10000 -I 11000 /mnt/testfs

In this example, the quota for user "bob" is set to 300 MB (309200\*1024) and the hard limit is 11,000 files. Therefore, the inode hard limit should be 11000.

The quota command displays the quota allocated and consumed by each Lustre target. Using the previous

setquota example, running this lfs quota command:

$ lfs quota -u bob -v /mnt/testfs

displays this command output:

Disk quotas for user bob (uid 6000):

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Filesystem | kbytes | quota | limit | grace | files | quota | limit | grace |
| /mnt/testfs | 0 | 30720 | 30920 | - | 0 | 10000 | 11000 | - |
| testfs-MDT0000\_UUID | 0 | - | 8192 | - | 0 | - | 2560 | - |
| testfs-OST0000\_UUID | 0 | - | 8192 | - | 0 | - | 0 | - |
| testfs-OST0001\_UUID | 0 | - | 8192 | - | 0 | - | 0 | - |

Total allocated inode limit: 2560, total allocated block limit: 24576

Global quota limits are stored in dedicated index files (there is one such index per quota type) on the quota master target (aka QMT). The QMT runs on MDT0000 and exports the global indexes via /proc. The global indexes can thus be dumped via the following command:

# lctl get\_param qmt.testfs-QMT0000.\*.glb-\*

The format of global indexes depends on the OSD type. The ldiskfs OSD uses an IAM files while the ZFS OSD creates dedicated ZAPs.

Each slave also stores a copy of this global index locally. When the global index is modified on the master, a glimpse callback is issued on the global quota lock to notify all slaves that the global index has been modified. This glimpse callback includes information about the identifier subject to the change. If the global index on the QMT is modified while a slave is disconnected, the index version is used to determine whether the slave copy of the global index isn't up to date any more. If so, the slave fetches the whole index again and updates the local copy. The slave copy of the global index is also exported via /proc and can be accessed via the following command:

lctl get\_param osd-\*.\*.quota\_slave.limit\*

##### Note

Prior to 2.4, global quota limits used to be stored in administrative quota files using the on-disk format of the linux quota file. When upgrading MDT0000 to 2.4, those administrative quota files are converted into IAM indexes automatically, conserving existing quota limits previously set by the administrator.

## Quota Allocation

In a Lustre file system, quota must be properly allocated or users may experience unnecessary failures. The file system block quota is divided up among the OSTs within the file system. Each OST requests an allocation which is increased up to the quota limit. The quota allocation is then *quantized* to reduce the number of quota-related request traffic.

The Lustre quota system distributes quotas from the Quota Master Target (aka QMT). Only one QMT instance is supported for now and only runs on the same node as MDT0000. All OSTs and MDTs set up a Quota Slave Device (aka QSD) which connects to the QMT to allocate/release quota space. The QSD is setup directly from the OSD layer.

To reduce quota requests, quota space is initially allocated to QSDs in very large chunks. How much unused quota space can be hold by a target is controlled by the qunit size. When quota space for a given ID is close to exhaustion on the QMT, the qunit size is reduced and QSDs are notified of the new qunit size value via a glimpse callback. Slaves are then responsible for releasing quota space above the new qunit value. The qunit size isn't shrunk indefinitely and there is a minimal value of 1MB for blocks and 1,024 for inodes. This means that the quota space rebalancing process will stop when this minimum value is reached. As a result, quota exceeded can be returned while many slaves still have 1MB or 1,024 inodes of spare quota space.

If we look at the setquota example again, running this lfs quota command:

# lfs quota -u bob -v /mnt/testfs

displays this command output:

Disk quotas for user bob (uid 500):

Filesystem kbytes quota limit grace files quota limit gra

/mnt/testfs 30720\* 30720 30920 6d23h56m44s 10101\* 10000 11000

6d23h59m50s

testfs-MDT0000\_UUID 0 - 0 - 10101 - 10240 testfs-OST0000\_UUID 0 - 1024 - - - - testfs-OST0001\_UUID 30720\* - 29896 - - - -

Total allocated inode limit: 10240, total allocated block limit: 30920

The total quota limit of 30,920 is allocated to user bob, which is further distributed to two OSTs.

Values appended with ' \*' show that the quota limit has been exceeded, causing the following error when trying to write or create a file:

$ cp: writing `/mnt/testfs/foo`: Disk quota exceeded.

##### Note

It is very important to note that the block quota is consumed per OST and the inode quota per MDS. Therefore, when the quota is consumed on one OST (resp. MDT), the client may not be able to create files regardless of the quota available on other OSTs (resp. MDTs).

Setting the quota limit below the minimal qunit size may prevent the user/group from all file creation. It is thus recommended to use soft/hard limits which are a multiple of the number of OSTs \* the minimal qunit size.

To determine the total number of inodes, use lfs df -i(and also lctl get\_param

\*.\*.filestotal). For more information on using the lfs df -i command and the command [output, see Section 19.6.1, “Checking File System Free Space”](#_bookmark469).

Unfortunately, the statfs interface does not report the free inode count directly, but instead reports the total inode and used inode counts. The free inode count is calculated for df from (total inodes

- used inodes). It is not critical to know the total inode count for a file system. Instead, you should know (accurately), the free inode count and the used inode count for a file system. The Lustre software manipulates the total inode count in order to accurately report the other two values.

## Quotas and Version Interoperability

The new quota protocol introduced in Lustre software release 2.4.0 **is not compatible** with previous versions. As a consequence, **all Lustre servers must be upgraded to release 2.4.0 for quota to be functional**. Quota limits set on the Lustre file system prior to the upgrade will be automatically migrated to the new quota index format. As for accounting information with ldiskfs backend, they will be regenerated by running tunefs.lustre --quota against all targets. It is worth noting that running tunefs.lustre --quota is **mandatory** for all targets formatted with a Lustre software release older than release 2.4.0, otherwise quota enforcement as well as accounting won't be functional.

Besides, the quota protocol in release 2.4 takes for granted that the Lustre client supports the OBD\_CONNECT\_EINPROGRESS connect flag. Clients supporting this flag will retry indefinitely when the server returns EINPROGRESS in a reply. Here is the list of Lustre client version which are compatible with release 2.4:

* + - Release 2.3-based clients and later
    - Release 1.8 clients newer or equal to release 1.8.9-wc1
    - Release 2.1 clients newer or equal to release 2.1.4

## Granted Cache and Quota Limits

In a Lustre file system, granted cache does not respect quota limits. In this situation, OSTs grant cache to a Lustre client to accelerate I/O. Granting cache causes writes to be successful in OSTs, even if they exceed the quota limits, and will overwrite them.

The sequence is:

1. A user writes files to the Lustre file system.
2. If the Lustre client has enough granted cache, then it returns 'success' to users and arranges the writes to the OSTs.
3. Because Lustre clients have delivered success to users, the OSTs cannot fail these writes.

Because of granted cache, writes always overwrite quota limitations. For example, if you set a 400 GB quota on user A and use IOR to write for user A from a bundle of clients, you will write much more data than 400 GB, and cause an out-of-quota error ( EDQUOT).

##### Note

The effect of granted cache on quota limits can be mitigated, but not eradicated. Reduce the maximum amount of dirty data on the clients (minimal value is 1MB):

* + lctl set\_param osc.\*.max\_dirty\_mb=8

## Lustre Quota Statistics

The Lustre software includes statistics that monitor quota activity, such as the kinds of quota RPCs sent during a specific period, the average time to complete the RPCs, etc. These statistics are useful to measure performance of a Lustre file system.

Each quota statistic consists of a quota event and min\_time, max\_time and sum\_time values for the event.

|  |  |
| --- | --- |
| **Quota Event** | **Description** |
| **sync\_acq\_req** | Quota slaves send a acquiring\_quota request and wait for its return. |
| **sync\_rel\_req** | Quota slaves send a releasing\_quota request and wait for its return. |
| **async\_acq\_req** | Quota slaves send an acquiring\_quota request and do not wait for its return. |
| **async\_rel\_req** | Quota slaves send a releasing\_quota request and do not wait for its return. |
| **Quota Event** | **Description** |
| **wait\_for\_ino\_quota (lquota\_chkquota)** | Before files are created on the MDS, the MDS checks if the remaining inode quota is sufficient. This is done in the lquota\_chkquota function. |
| **wait\_for\_blk\_quota (lquota\_pending\_commit)** | After blocks are written to OSTs, relative quota information is updated. This is done in the lquota\_pending\_commit function. |
| **wait\_for\_ino\_quota (lquota\_pending\_commit)** | After files are created, relative quota information is updated. This is done in the lquota\_pending\_commit function. |
| **wait\_for\_pending\_blk\_quota\_req (qctxt\_wait\_pending\_dqacq)** | On the MDS or OSTs, there is one thread sending a quota request for a specific UID/GID for block quota at any time. At that time, if other threads need to do this too, they should wait. This is done in the qctxt\_wait\_pending\_dqacq function. |
| **wait\_for\_pending\_ino\_quota\_req (qctxt\_wait\_pending\_dqacq)** | On the MDS, there is one thread sending a quota request for a specific UID/GID for inode quota at any time. If other threads need to do this too, they should wait. This is done in the qctxt\_wait\_pending\_dqacq function. |
| **nowait\_for\_pending\_blk\_quota\_req (qctxt\_wait\_pending\_dqacq)** | On the MDS or OSTs, there is one thread sending a quota request for a specific UID/GID for block quota at any time. When threads enter qctxt\_wait\_pending\_dqacq, they do not need to wait. This is done in the qctxt\_wait\_pending\_dqacq function. |
| **nowait\_for\_pending\_ino\_quota\_req (qctxt\_wait\_pending\_dqacq)** | On the MDS, there is one thread sending a quota request for a specific UID/GID for inode quota at any time. When threads enter qctxt\_wait\_pending\_dqacq, they do not need to wait. This is done in the qctxt\_wait\_pending\_dqacq function. |
| **quota\_ctl** | The quota\_ctl statistic is generated when lfs  setquota, lfs quota and so on, are issued. |
| **adjust\_qunit** | Each time qunit is adjusted, it is counted. |
| **wait\_for\_blk\_quota (lquota\_chkquota)** | Before data is written to OSTs, the OSTs check if the remaining block quota is sufficient. This is done in the lquota\_chkquota function. |

### Interpreting Quota Statistics

Quota statistics are an important measure of the performance of a Lustre file system. Interpreting these statistics correctly can help you diagnose problems with quotas, and may indicate adjustments to improve system performance.

For example, if you run this command on the OSTs:

lctl get\_param lquota.testfs-OST0000.stats

You will get a result similar to this:

snapshot\_time 1219908615.506895 secs.usecs

|  |  |  |
| --- | --- | --- |
| async\_acq\_req | 1 samples [us] | 32 32 32 |
| async\_rel\_req | 1 samples [us] | 5 5 5 |

nowait\_for\_pending\_blk\_quota\_req(qctxt\_wait\_pending\_dqacq) 1 samples [us] 2\ 2 2

|  |  |  |
| --- | --- | --- |
| quota\_ctl | 4 samples [us] 80 3470 | 4293 |
| adjust\_qunit | 1 samples [us] | 70 70 70 |
| .... |  |  |

In the first line, snapshot\_time indicates when the statistics were taken. The remaining lines list the quota events and their associated data.

In the second line, the async\_acq\_req event occurs one time. The min\_time, max\_time and

sum\_time statistics for this event are 32, 32 and 32, respectively. The unit is microseconds (µs).

In the fifth line, the quota\_ctl event occurs four times. The min\_time, max\_time and sum\_time

statistics for this event are 80, 3470 and 4293, respectively. The unit is microseconds (µs).

**Chapter 24. Mapping UIDs and GIDs with Nodemap**

This chapter describes how to map UID and GIDs across a Lustre file system using the nodemap feature, and includes the following sections:

* [Section 24.1, “Setting a Mapping”](#_bookmark576)
* [Section 24.2, “Altering Properties”](#_bookmark580)
* [Section 24.3, “Enabling the Feature”](#_bookmark583)
* [Section 24.4, “Verifying Settings”](#_bookmark584)
* [Section 24.5, “Ensuring Consistency”](#_bookmark585)

## Setting a Mapping

The nodemap feature supported in Lustre 2.9 was first introduced in Lustre 2.7 as a technology preview. It allows UIDs and GIDs from remote systems to be mapped to local sets of UIDs and GIDs while retaining POSIX ownership, permissions and quota information. As a result, multiple sites with conflicting user and group identifiers can operate on a single Lustre file system without creating collisions in UID or GID space.

### Defining Terms

When the nodemap feature is enabled, client file system access to a Lustre system is filtered through the nodemap identity mapping policy engine. Lustre connectivity is governed by network identifiers, or *NIDs*, such as 192.168.7.121@tcp. When an operation is made from a NID, Lustre decides if that NID is part of a *nodemap*, a policy group consisting of one or more NID ranges. If no policy group exists for that NID, access is squashed to user nobody by default. Each policy group also has several *properties*, such as trusted and admin, which determine access conditions. A collection of identity maps or *idmaps* are kept for each policy group. These idmaps determine how UIDs and GIDs on the client are translated into the canonical user space of the local Lustre file system.

In order for nodemap to function properly, the MGS, MDS, and OSS systems must all have a version of Lustre which supports nodemap. Clients operate transparently and do not require special configuration or knowledge of the nodemap setu

# Chapter 29. Benchmarking Lustre File System Performance (Lustre I/O Kit)

This chapter describes the Lustre I/O kit, a collection of I/O benchmarking tools for a Lustre cluster. It includes:

## Using Lustre I/O Kit Tools

The tools in the Lustre I/O Kit are used to benchmark Lustre file system hardware and validate that it is working as expected before you install the Lustre software. It can also be used to to validate the performance of the various hardware and software layers in the cluster and also to find and troubleshoot I/O issues.

Typically, performance is measured starting with single raw devices and then proceeding to groups of devices. Once raw performance has been established, other software layers are then added incrementally and tested.

### Contents of the Lustre I/O Kit

The I/O kit contains three tests, each of which tests a progressively higher layer in the Lustre software stack:

* sgpdd-survey - Measure basic 'bare metal' performance of devices while bypassing the kernel block device layers, buffer cache, and file system.
* obdfilter-survey - Measure the performance of one or more OSTs directly on the OSS node or alternately over the network from a Lustre client.
* ost-survey - Performs I/O against OSTs individually to allow performance comparisons to detect if an OST is performing sub-optimally due to hardware issues.

Typically with these tests, a Lustre file system should deliver 85-90% of the raw device performance.

A utility stats-collect is also provided to collect application profiling information from Lustre clients and servers. See [Section 29.6, “Collecting Application Profiling Information (stats-collect)”](#_bookmark701) for more information.

### Preparing to Use the Lustre I/O Kit

The following prerequisites must be met to use the tests in the Lustre I/O kit:

* Password-free remote access to nodes in the system (provided by ssh or rsh).
* LNet self-test completed to test that Lustre networking has been properly installed and configured. See [Chapter 28, *Testing Lustre Network Performance (LNet Self-Test)*](#_bookmark662).
* Lustre file system software installed.
* sg3\_utils package providing the sgp\_dd tool (sg3\_utils is a separate RPM package available online using YUM).

Download the Lustre I/O kit (lustre-iokit)from:<http://downloads.hpdd.intel.com/>

## Testing I/O Performance of Raw Hardware (sgpdd-survey)

The sgpdd-survey tool is used to test bare metal I/O performance of the raw hardware, while bypassing as much of the kernel as possible. This survey may be used to characterize the performance of a SCSI device by simulating an OST serving multiple stripe files. The data gathered by this survey can help set expectations for the performance of a Lustre OST using this device.

The script uses sgp\_dd to carry out raw sequential disk I/O. It runs with variable numbers of sgp\_dd

threads to show how performance varies with different request queue depths.

The script spawns variable numbers of sgp\_dd instances, each reading or writing a separate area of the disk to demonstrate performance variance within a number of concurrent stripe files.

Several tips and insights for disk performance measurement are described below. Some of this information is specific to RAID arrays and/or the Linux RAID implementation.

* + - *Performance is limited by the slowest disk.*

Before creating a RAID array, benchmark all disks individually. We have frequently encountered situations where drive performance was not consistent for all devices in the array. Replace any disks that are significantly slower than the rest.

* + - *Disks and arrays are very sensitive to request size.*

To identify the optimal request size for a given disk, benchmark the disk with different record sizes ranging from 4 KB to 1 to 2 MB.

##### Caution

The sgpdd-survey script overwrites the device being tested, which results in the ***LOSS OF ALL DATA*** on that device. Exercise caution when selecting the device to be tested.

##### Note

Array performance with all LUNs loaded does not always match the performance of a single LUN when tested in isolation.

Prerequisites:

* + - sgp\_dd tool in the sg3\_utils package
    - Lustre software is *NOT* required

The device(s) being tested must meet one of these two requirements:

* + - If the device is a SCSI device, it must appear in the output of sg\_map (make sure the kernel module

sg is loaded).

* + - If the device is a raw device, it must appear in the output of raw -qa. Raw and SCSI devices cannot be mixed in the test specification.

##### Note

If you need to create raw devices to use the sgpdd-survey tool, note that raw device 0 cannot be used due to a bug in certain versions of the "raw" utility (including the version shipped with Red Hat Enterprise Linux 4U4.)

### Tuning Linux Storage Devices

To get large I/O transfers (1 MB) to disk, it may be necessary to tune several kernel parameters as specified:

/sys/block/sdN/queue/max\_sectors\_kb = 4096

/sys/block/sdN/queue/max\_phys\_segments = 256

/proc/scsi/sg/allow\_dio = 1

/sys/module/ib\_srp/parameters/srp\_sg\_tablesize = 255

/sys/block/sdN/queue/scheduler

##### Note

Recommended schedulers are **deadline** and **noop**. The scheduler is set by default to **deadline**, unless it has already been set to **noop**.

### Running sgpdd-survey

The sgpdd-survey script must be customized for the particular device being tested and for the location where the script saves its working and result files (by specifying the ${rslt} variable). Customization variables are described at the beginning of the script.

When the sgpdd-survey script runs, it creates a number of working files and a pair of result files. The names of all the files created start with the prefix defined in the variable ${rslt}. (The default value is /tmp.) The files include:

* + - File containing standard output data (same as stdout)

*rslt\_date\_time*.summary

* + - Temporary (tmp) files

*rslt\_date\_time*\_\*

* + - Collected tmp files for post-mortem

*rslt\_date\_time*.detail

The stdout and the .summary file will contain lines like this:

total\_size 8388608K rsz 1024 thr 1 crg 1 180.45 MB/s 1 x 180.50 \

= 180.50 MB/s

Each line corresponds to a run of the test. Each test run will have a different number of threads, record size, or number of regions.

* + - total\_size - Size of file being tested in KBs (8 GB in above example).
    - rsz - Record size in KBs (1 MB in above example).
    - thr - Number of threads generating I/O (1 thread in above example).
    - crg - Current regions, the number of disjoint areas on the disk to which I/O is being sent (1 region in above example, indicating that no seeking is done).
    - MB/s - Aggregate bandwidth measured by dividing the total amount of data by the elapsed time (180.45 MB/s in the above example).
    - MB/s - The remaining numbers show the number of regions X performance of the slowest disk as a sanity check on the aggregate bandwidth.

If there are so many threads that the sgp\_dd script is unlikely to be able to allocate I/O buffers, then

ENOMEM is printed in place of the aggregate bandwidth result.

If one or more sgp\_dd instances do not successfully report a bandwidth number, then FAILED is printed in place of the aggregate bandwidth result.

## Testing OST Performance (obdfilter- survey)

The obdfilter-survey script generates sequential I/O from varying numbers of threads and objects (files) to simulate the I/O patterns of a Lustre client.

The obdfilter-survey script can be run directly on the OSS node to measure the OST storage performance without any intervening network, or it can be run remotely on a Lustre client to measure the OST performance including network overhead.

The obdfilter-survey is used to characterize the performance of the following:

* + - **Local file system** - In this mode, the obdfilter-survey script exercises one or more instances of the obdfilter directly. The script may run on one or more OSS nodes, for example, when the OSSs are all attached to the same multi-ported disk subsystem.

Run the script using the case=disk parameter to run the test against all the local OSTs. The script automatically detects all local OSTs and includes them in the survey.

To run the test against only specific OSTs, run the script using the targets=parameter to list the OSTs to be tested explicitly. If some OSTs are on remote nodes, specify their hostnames in addition to the OST name (for example, oss2:lustre-OST0004).

All obdfilter instances are driven directly. The script automatically loads the obdecho module (if required) and creates one instance of echo\_client for each obdfilter instance in order to generate I/O requests directly to the OST.

[For more details, see Section 29.3.1, “Testing Local Disk Performance”](#_bookmark688).

* + - **Network** - In this mode, the Lustre client generates I/O requests over the network but these requests are not sent to the OST file system. The OSS node runs the obdecho server to receive the requests but discards them before they are sent to the disk.

Pass the parameters case=network and targets=*hostname|IP\_of\_server* to the script. For each network case, the script does the required setup.

[For more details, see Section 29.3.2, “Testing Network Performance”](#_bookmark690)

* + - **Remote file system over the network** - In this mode the obdfilter-survey script generates I/O from a Lustre client to a remote OSS to write the data to the file system.

To run the test against all the local OSCs, pass the parameter case=netdisk to the script. Alternately you can pass the target= parameter with one or more OSC devices (e.g., lustre-OST0000-osc- ffff88007754bc00) against which the tests are to be run.

[For more details, see Section 29.3.3, “Testing Remote Disk Performance”](#_bookmark692).

##### Caution

The obdfilter-survey script is potentially destructive and there is a small risk data may be lost. To reduce this risk, obdfilter-survey should not be run on devices that contain data that needs to be preserved. Thus, the best time to run obdfilter-survey is before the Lustre file system is put into production. The reason obdfilter-survey may be safe to run on a production file system is because it creates objects with object sequence 2. Normal file system objects are typically created with object sequence 0.

##### Note

If the obdfilter-survey test is terminated before it completes, some small amount of space is leaked. you can either ignore it or reformat the file system.

##### Note

The obdfilter-survey script is *NOT* scalable beyond tens of OSTs since it is only intended to measure the I/O performance of individual storage subsystems, not the scalability of the entire system.

##### Note

The obdfilter-survey script must be customized, depending on the components under test and where the script's working files should be kept. Customization variables are described at the beginning of the obdfilter-survey script. In particular, pay attention to the listed maximum values listed for each parameter in the script.

### Testing Local Disk Performance

The obdfilter-survey script can be run automatically or manually against a local disk. This script profiles the overall throughput of storage hardware, including the file system and RAID layers managing the storage, by send ing workloads to the OSTs that vary in thread count, object count, and I/O size.

When the obdfilter-survey script is run, it provides information about the performance abilities of the storage hardware and shows the saturation points.

The plot-obdfilter script generates from the output of the obdfilter-survey a CSV file and parameters for importing into a spreadsheet or gnuplot to visualize the data.

To run the obdfilter-survey script, create a standard Lustre file system configuration; no special setup is needed.

To perform an automatic run:

1. Start the Lustre OSTs.

The Lustre OSTs should be mounted on the OSS node(s) to be tested. The Lustre client is not required to be mounted at this time.

1. Verify that the obdecho module is loaded. Run:

modprobe obdecho

1. Run the obdfilter-survey script with the parameter case=disk.

For example, to run a local test with up to two objects (nobjhi), up to two threads (thrhi), and 1024 MB transfer size (size):

$ nobjhi=2 thrhi=2 size=1024 case=disk sh obdfilter-survey

1. Performance measurements for write, rewrite, read etc are provided below:

# example output

Fri Sep 25 11:14:03 EDT 2015 Obdfilter-survey for case=disk from hds1fnb6123

ost 10 sz 167772160K rsz 1024K obj 10 thr 10 write 10982.73 [ 601.97,2912.91

...

The file ./lustre-iokit/obdfilter-survey/README.obdfilter-survey provides an explaination for the output as follows:

ost 10 is the total number of OSTs under test.

sz 167772160K is the total amount of data read or written (in bytes).

rsz 1024K is the record size (size of each echo\_client I/O, in bytes). obj 10 is the total number of objects over all OSTs

thr 10 is the total number of threads over all OSTs and objects write is the test name. If more tests have been specified they

all appear on the same line.

10982.73 is the aggregate bandwidth over all OSTs measured by dividing the total number of MB by the elapsed time.

[601.97,2912.91] are the minimum and maximum instantaneous bandwidths seen on any individual OST.

Note that although the numbers of threads and objects are specifed per-OST in the customization section of the script, results are reported aggregated over all OSTs.

*To perform a manual run:*

1. Start the Lustre OSTs.

The Lustre OSTs should be mounted on the OSS node(s) to be tested. The Lustre client is not required to be mounted at this time.

Verify that the obdecho module is loaded. Run:

modprobe obdecho

1. Determine the OST names.

On the OSS nodes to be tested, run the lctl dl command. The OST device names are listed in the fourth column of the output. For example:

$ lctl dl |grep obdfilter

0 UP obdfilter lustre-OST0001 lustre-OST0001\_UUID 1159

1. UP obdfilter lustre-OST0002 lustre-OST0002\_UUID 1159

...

1. List all OSTs you want to test.

Use the targets=parameter to list the OSTs separated by spaces. List the individual OSTs by name using the format *fsname*-*OSTnumber* (for example, lustre-OST0001). You do not have to specify an MDS or LOV.

1. Run the obdfilter-survey script with the targets=parameter.

For example, to run a local test with up to two objects (nobjhi), up to two threads (thrhi), and 1024 Mb (size) transfer size:

$ nobjhi=2 thrhi=2 size=1024 targets="lustre-OST0001 \ lustre-OST0002" sh obdfilter-survey

### Testing Network Performance

The obdfilter-survey script can only be run automatically against a network; no manual test is provided.

To run the network test, a specific Lustre file system setup is needed. Make sure that these configuration requirements have been met.

To perform an automatic run:

1. Start the Lustre OSTs.

The Lustre OSTs should be mounted on the OSS node(s) to be tested. The Lustre client is not required to be mounted at this time.

1. Verify that the obdecho module is loaded. Run:

modprobe obdecho

1. Start lctl and check the device list, which must be empty. Run:

lctl dl

1. Run the obdfilter-survey script with the parameters case=network and

targets=*hostname|ip\_of\_server*. For example:

$ nobjhi=2 thrhi=2 size=1024 targets="oss0 oss1" \ case=network sh obdfilter survey

On the server side, view the statistics at:

/proc/fs/lustre/obdecho/*echo\_srv*/stats

where *echo\_srv* is the obdecho server created by the script.

### Testing Remote Disk Performance

The obdfilter-survey script can be run automatically or manually against a network disk. To run the network disk test, start with a standard Lustre configuration. No special setup is needed.

To perform an automatic run:

1. Start the Lustre OSTs.

The Lustre OSTs should be mounted on the OSS node(s) to be tested. The Lustre client is not required to be mounted at this time.

1. Verify that the obdecho module is loaded. Run:

modprobe obdecho

1. Run the obdfilter-survey script with the parameter case=netdisk. For example:

$ nobjhi=2 thrhi=2 size=1024 case=netdisk sh obdfilter-survey

To perform a manual run:

1. Start the Lustre OSTs.

The Lustre OSTs should be mounted on the OSS node(s) to be tested. The Lustre client is not required to be mounted at this time.

1. Verify that the obdecho module is loaded. Run: modprobe obdecho
2. Determine the OSC names.

On the OSS nodes to be tested, run the lctl dl command. The OSC device names are listed in the fourth column of the output. For example:

$ lctl dl |grep obdfilter

1. UP osc lustre-OST0000-osc-ffff88007754bc00 \ 54b91eab-0ea9-1516-b571-5e6df349592e 5
2. UP osc lustre-OST0001-osc-ffff88007754bc00 \ 54b91eab-0ea9-1516-b571-5e6df349592e 5

...

1. List all OSCs you want to test.

Use the targets=parameter to list the OSCs separated by spaces. List the individual OSCs by name separated by spaces using the format *fsname*-*OST\_name*-osc-*instance* (for example, lustre-OST0000-osc-ffff88007754bc00). You *do not have to specify an MDS or LOV.*

1. Run the obdfilter-survey script with the targets=*osc* and case=netdisk.

An example of a local test run with up to two objects (nobjhi), up to two threads (thrhi), and 1024 Mb (size) transfer size is shown below:

$ nobjhi=2 thrhi=2 size=1024 \

targets="lustre-OST0000-osc-ffff88007754bc00 \

lustre-OST0001-osc-ffff88007754bc00" sh obdfilter-survey

## Testing OST I/O Performance (ost- survey)

The ost-survey tool is a shell script that uses lfs setstripe to perform I/O against a single OST. The script writes a file (currently using dd) to each OST in the Lustre file system, and compares read and write speeds. The ost-survey tool is used to detect anomalies between otherwise identical disk subsystems.

##### Note

We have frequently discovered wide performance variations across all LUNs in a cluster. This may be caused by faulty disks, RAID parity reconstruction during the test, or faulty network hardware.

To run the ost-survey script, supply a file size (in KB) and the Lustre file system mount point. For example, run:

$ ./ost-survey.sh -s 10 /mnt/lustre

Typical output is:

Number of Active OST devices : 4

Worst Read OST indx: 2 speed: 2835.272725 Best Read OST indx: 3 speed: 2872.889668 Read Average: 2852.508999 +/- 16.444792 MB/s Worst Write OST indx: 3 speed: 17.705545 Best Write OST indx: 2 speed: 128.172576 Write Average: 95.437735 +/- 45.518117 MB/s

Ost# Read(MB/s) Write(MB/s) Read-time Write-time

---------------------------------------------------- 0 2837.440 126.918 0.035 0.788

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2864.433 | 108.954 | 0.035 | 0.918 |
| 2 | 2835.273 | 128.173 | 0.035 | 0.780 |
| 3 | 2872.890 | 17.706 | 0.035 | 5.648 |

## Testing MDS Performance (mds-survey)

mds-survey is available in Lustre software release 2.2 and beyond. The mds-survey script tests the local metadata performance using the echo\_client to drive different layers of the MDS stack: mdd, mdt, osd (the Lustre software only supports mdd stack). It can be used with the following classes of operations:

* + - Open-create/mkdir/create
    - Lookup/getattr/setxattr
    - Delete/destroy
    - Unlink/rmdir

These operations will be run by a variable number of concurrent threads and will test with the number of directories specified by the user. The run can be executed such that all threads operate in a single directory (dir\_count=1) or in private/unique directory (dir\_count=x thrlo=x thrhi=x).

The mdd instance is driven directly. The script automatically loads the obdecho module if required and creates instance of echo\_client.

This script can also create OST objects by providing stripe\_count greater than zero.

To perform a run:

1. Start the Lustre MDT.

The Lustre MDT should be mounted on the MDS node to be tested.

1. Start the Lustre OSTs (optional, only required when test with OST objects) The Lustre OSTs should be mounted on the OSS node(s).
2. Run the mds-survey script as explain below

The script must be customized according to the components under test and where it should keep its working files. Customization variables are described as followed:

* + thrlo - threads to start testing. skipped if less than dir\_count
  + thrhi - maximum number of threads to test
  + targets - MDT instance
  + file\_count - number of files per thread to test
  + dir\_count - total number of directories to test. Must be less than or equal to thrhi
  + stripe\_count - number stripe on OST objects
  + tests\_str - test operations. Must have at least "create" and "destroy"
  + start\_number - base number for each thread to prevent name collisions
  + layer - MDS stack's layer to be tested Run without OST objects creation:

Setup the Lustre MDS without OST mounted. Then invoke the mds-survey script

$ thrhi=64 file\_count=200000 sh mds-survey

Run with OST objects creation:

Setup the Lustre MDS with at least one OST mounted. Then invoke the mds-survey script with

stripe\_count parameter

$ thrhi=64 file\_count=200000 stripe\_count=2 sh mds-survey

Note: a specific MDT instance can be specified using targets variable.

$ targets=lustre-MDT0000 thrhi=64 file\_count=200000 stripe\_count=2 sh mds-survey

## Collecting Application Profiling Information (stats-collect)

The stats-collect utility contains the following scripts used to collect application profiling information from Lustre clients and servers:

* + - lstat.sh - Script for a single node that is run on each profile node.
    - gather\_stats\_everywhere.sh - Script that collect statistics.
    - config.sh - Script that contains customized configuration descriptions. The stats-collect utility requires:
    - Lustre software to be installed and set up on your cluster
    - SSH and SCP access to these nodes without requiring a password

### 29.6.1. Using stats-collect

The stats-collect utility is configured by including profiling configuration variables in the config.sh script. Each configuration variable takes the following form, where 0 indicates statistics are to be collected only when the script starts and stops and *n* indicates the interval in seconds at which statistics are to be collected:

*statistic*\_INTERVAL=*0|n*

Statistics that can be collected include:

* + - VMSTAT - Memory and CPU usage and aggregate read/write operations
    - SERVICE - Lustre OST and MDT RPC service statistics
    - BRW - OST bulk read/write statistics (brw\_stats)
    - SDIO - SCSI disk IO statistics (sd\_iostats)
    - MBALLOC - ldiskfs block allocation statistics
    - IO - Lustre target operations statistics
    - JBD - ldiskfs journal statistics
    - CLIENT - Lustre OSC request statistics To collect profile information:

Begin collecting statistics on each node specified in the config.sh script.

1. Starting the collect profile daemon on each node by entering:

sh gather\_stats\_everywhere.sh config.sh start

1. Run the test.
2. Stop collecting statistics on each node, clean up the temporary file, and create a profiling tarball. Enter:

sh gather\_stats\_everywhere.sh config.sh stop *log\_name*.tgz

When *log\_name*.tgz is specified, a profile tarball */tmp/log\_name*.tgz is created.

1. Analyze the collected statistics and create a csv tarball for the specified profiling data.

sh gather\_stats\_everywhere.sh config.sh analyse log\_tarball.tgz csv

# Chapter 30. Tuning a Lustre File System

This chapter contains information about tuning a Lustre file system for better performance.

##### Note

Many options in the Lustre software are set by means of kernel module parameters. These parameters are contained in the /etc/modprobe.d/lustre.conf file.

* 1. **Optimizing the Number of Service Threads**

An OSS can have a minimum of two service threads and a maximum of 512 service threads. The number of service threads is a function of how much RAM and how many CPUs are on each OSS node (1 thread / 128MB \* num\_cpus). If the load on the OSS node is high, new service threads will be started in order to process more requests concurrently, up to 4x the initial number of threads (subject to the maximum of 512). For a 2GB 2-CPU system, the default thread count is 32 and the maximum thread count is 128.

Increasing the size of the thread pool may help when:

* + - Several OSTs are exported from a single OSS
    - Back-end storage is running synchronously
    - I/O completions take excessive time due to slow storage Decreasing the size of the thread pool may help if:
    - Clients are overwhelming the storage capacity
    - There are lots of "slow I/O" or similar messages

Increasing the number of I/O threads allows the kernel and storage to aggregate many writes together for more efficient disk I/O. The OSS thread pool is shared--each thread allocates approximately 1.5 MB (maximum RPC size + 0.5 MB) for internal I/O buffers.

It is very important to consider memory consumption when increasing the thread pool size. Drives are only able to sustain a certain amount of parallel I/O activity before performance is degraded, due to the high number of seeks and the OST threads just waiting for I/O. In this situation, it may be advisable to decrease the load by decreasing the number of OST threads.

Determining the optimum number of OSS threads is a process of trial and error, and varies for each particular configuration. Variables include the number of OSTs on each OSS, number and speed of disks, RAID configuration, and available RAM. You may want to start with a number of OST threads equal to the number of actual disk spindles on the node. If you use RAID, subtract any dead spindles not used for actual data (e.g., 1 of N of spindles for RAID5, 2 of N spindles for RAID6), and monitor the performance of clients during usual workloads. If performance is degraded, increase the thread count and see how that works until performance is degraded again or you reach satisfactory performance.

##### Note

If there are too many threads, the latency for individual I/O requests can become very high and should be avoided. Set the desired maximum thread count permanently using the method described above.

### Specifying the OSS Service Thread Count

The oss\_num\_threads parameter enables the number of OST service threads to be specified at module load time on the OSS nodes:

options ost oss\_num\_threads={N}

After startup, the minimum and maximum number of OSS thread counts can be set via the

{service}.thread\_{min,max,started} tunable. To change the tunable at runtime, run:

lctl {get,set}\_param {service}.thread\_{min,max,started}

This works in a similar fashion to binding of threads on MDS. MDS thread tuning is covered in [Section 30.2, “ Binding MDS Service Thread to CPU Partitions”](#_bookmark712).

* + - * oss\_cpts=[EXPRESSION] binds the default OSS service on CPTs defined by [EXPRESSION].
      * oss\_io\_cpts=[EXPRESSION] binds the IO OSS service on CPTs defined by [EXPRESSION]. [For further details, see Section 35.9, “Setting MDS and OSS Thread Counts”](#_bookmark943).

### Specifying the MDS Service Thread Count

The mds\_num\_threads parameter enables the number of MDS service threads to be specified at module load time on the MDS node:

options mds mds\_num\_threads={N}

After startup, the minimum and maximum number of MDS thread counts can be set via the

{service}.thread\_{min,max,started} tunable. To change the tunable at runtime, run:

lctl {get,set}\_param {service}.thread\_{min,max,started}

[For details, see Section 35.9, “Setting MDS and OSS Thread Counts”](#_bookmark943).

The number of MDS service threads started depends on system size and the load on the server, and has a default maximum of 64. The maximum potential number of threads (MDS\_MAX\_THREADS) is 1024.

##### Note

The OSS and MDS start two threads per service per CPT at mount time, and dynamically increase the number of running service threads in response to server load. Setting the \*\_num\_threads module parameter starts the specified number of threads for that service immediately and disables automatic thread creation behavior.

* + - * mds\_rdpg\_num\_threads controls the number of threads in providing the read page service. The read page service handles file close and readdir operations.
      * m ds\_attr\_num\_threads controls the number of threads in providing the setattr service to clients running Lustre software release 1.8.
  1. **Improving Lustre I/O Performance for Small Files**

An environment where an application writes small file chunks from many clients to a single file can result in poor I/O performance. To improve the performance of the Lustre file system with small files:

Have the application aggregate writes some amount before submitting them to the Lustre file system. By default, the Lustre software enforces POSIX coherency semantics, so it results in lock ping-pong between client nodes if they are all writing to the same file at one time.

Using MPI-IO Collective Write functionality in the Lustre ADIO driver is one way to achieve this in a straight forward manner if the application is already using MPI-IO.

Have the application do 4kB O\_DIRECT sized I/O to the file and disable locking on the output file. This avoids partial-page IO submissions and, by disabling locking, you avoid contention between clients.

Have the application write contiguous data.

Add more disks or use SSD disks for the OSTs. This dramatically improves the IOPS rate. Consider creating larger OSTs rather than many smaller OSTs due to less overhead (journal, connections, etc).

Use RAID-1+0 OSTs instead of RAID-5/6. There is RAID parity overhead for writing small chunks of data to disk.

## Understanding Why Write Performance is Better Than Read Performance

Typically, the performance of write operations on a Lustre cluster is better than read operations. When doing writes, all clients are sending write RPCs asynchronously. The RPCs are allocated, and written to disk in the order they arrive. In many cases, this allows the back-end storage to aggregate writes efficiently.

In the case of read operations, the reads from clients may come in a different order and need a lot of seeking to get read from the disk. This noticeably hampers the read throughput. Currently, there is no readahead on the OSTs themselves, though the clients do readahead. If there are lots of clients doing reads it would not be possible to do any readahead in any case because of memory consumption (consider that even a single RPC (1 MB) readahead for 1000 clients would consume 1 GB of RAM).

For file systems that use socklnd (TCP, Ethernet) as interconnect, there is also additional CPU overhead because the client cannot receive data without copying it from the network buffers. In the write case, the client CAN send data without the additional data copy. This means that the client is more likely to become CPU-bound during reads than writes.

# Chapter 35. Lustre Parameters

The /proc and /sys file systems acts as an interface to internal data structures in the kernel. This chapter describes parameters and tunables that are useful for optimizing and monitoring aspects of a Lustre file system.

## Introduction to Lustre Parameters

Lustre parameters and statistics files provide an interface to internal data structures in the kernel that enables monitoring and tuning of many aspects of Lustre file system and application performance. These data structures include settings and metrics for components such as memory, networking, file systems, and kernel housekeeping routines, which are available throughout the hierarchical file layout.

Typically, metrics are accessed via lctl get\_param files and settings are changed by via lctl set\_param. Some data is server-only, some data is client-only, and some data is exported from the client to the server and is thus duplicated in both locations.

Some examples are shown below:

* + - To obtain data from a Lustre client:

# lctl list\_param osc.\*

osc.testfs-OST0000-osc-ffff881071d5cc00 osc.testfs-OST0001-osc-ffff881071d5cc00 osc.testfs-OST0002-osc-ffff881071d5cc00 osc.testfs-OST0003-osc-ffff881071d5cc00 osc.testfs-OST0004-osc-ffff881071d5cc00 osc.testfs-OST0005-osc-ffff881071d5cc00 osc.testfs-OST0006-osc-ffff881071d5cc00 osc.testfs-OST0007-osc-ffff881071d5cc00 osc.testfs-OST0008-osc-ffff881071d5cc00

In this example, information about OST connections available on a client is displayed (indicated by "osc").

* + - To see multiple levels of parameters, use multiple wildcards:

# lctl list\_param osc.\*.\*

osc.testfs-OST0000-osc-ffff881071d5cc00.active osc.testfs-OST0000-osc-ffff881071d5cc00.blocksize osc.testfs-OST0000-osc-ffff881071d5cc00.checksum\_type osc.testfs-OST0000-osc-ffff881071d5cc00.checksums osc.testfs-OST0000-osc-ffff881071d5cc00.connect\_flags

osc.testfs-OST0000-osc-ffff881071d5cc00.contention\_seconds

osc.testfs-OST0000-osc-ffff881071d5cc00.cur\_dirty\_bytes

...

osc.testfs-OST0000-osc-ffff881071d5cc00.rpc\_stats

* + - To view a specific file, use lctl get\_param:

# lctl get\_param osc.lustre-OST0000\*.rpc\_stats

For more information about using lctl[, see Section 13.11.3, “Setting Parameters with lctl”](#_bookmark288).

Data can also be viewed using the cat command with the full path to the file. The form of the cat command is similar to that of the lctl get\_param command with some differences. Unfortunately, as the Linux kernel has changed over the years, the location of statistics and parameter files has also changed, which means that the Lustre parameter files may be located in either the /proc directory, in the /sys directory, and/or in the /sys/kernel/debug directory, depending on the kernel version and the Lustre version being used. The lctl command insulates scripts from these changes and is preferred over direct file access, unless as part of a high-performance monitoring system. In the cat command:

* + - Replace the dots in the path with slashes.
    - Prepend the path with the following as appropriate:

/{proc,sys}/{fs,sys}/{lustre,lnet}

For example, an lctl get\_param command may look like this:

# lctl get\_param osc.\*.uuid

osc.testfs-OST0000-osc-ffff881071d5cc00.uuid=594db456-0685-bd16-f59b-e72ee90e9819 osc.testfs-OST0001-osc-ffff881071d5cc00.uuid=594db456-0685-bd16-f59b-e72ee90e9819

...

The equivalent cat command may look like this:

# cat /proc/fs/lustre/osc/\*/uuid 594db456-0685-bd16-f59b-e72ee90e9819

594db456-0685-bd16-f59b-e72ee90e9819

...

or like this:

# cat /sys/fs/lustre/osc/\*/uuid 594db456-0685-bd16-f59b-e72ee90e9819

594db456-0685-bd16-f59b-e72ee90e9819

...

The llstat utility can be used to monitor some Lustre file system I/O activity over a specified time [period. For more details, see Section 40.8, “ llstat”](#_bookmark1076)

Some data is imported from attached clients and is available in a directory called exports located in the corresponding per-service directory on a Lustre server. For example:

oss:/root# lctl list\_param obdfilter.testfs-OST0000.exports.\*

# hash ldlm\_stats stats uuid

### 35.1.1. Identifying Lustre File Systems and Servers

Several /proc files on the MGS list existing Lustre file systems and file system servers. The examples below are for a Lustre file system called testfs with one MDT and three OSTs.

* + - To view all known Lustre file systems, enter:

mgs# lctl get\_param mgs.\*.filesystems testfs

* + - To view the names of the servers in a file system in which least one server is running, enter:

lctl get\_param mgs.\*.live.*<filesystem name>*

* + - To view the names of all live servers in the file system as listed in /proc/fs/lustre/devices, enter:

# lctl device\_list

The information provided on each line includes:

* + - * Device number
      * Device status (UP, INactive, or STopping)
      * Device name
      * Device UUID
      * Reference count (how many users this device has)
    - To display the name of any server, view the device label:

root$ e2label /dev/sda

## Monitoring Lustre File System I/O

A number of system utilities are provided to enable collection of data related to I/O activity in a Lustre file system. In general, the data collected describes:

* + - Data transfer rates and throughput of inputs and outputs external to the Lustre file system, such as network requests or disk I/O operations performed
    - Data about the throughput or transfer rates of internal Lustre file system data, such as locks or allocations.

##### Note

It is highly recommended that you complete baseline testing for your Lustre file system to determine normal I/O activity for your hardware, network, and system workloads. Baseline data will allow you to easily determine when performance becomes degraded in your system. Two particularly useful baseline statistics are:

* + - * brw\_stats – Histogram data characterizing I/O requests to the OSTs. For more details, see [Section 35.3.5, “Monitoring the OST Block I/O Stream”](#_bookmark919).
      * rpc\_stats – Histogram data showing information about RPCs made by clients. For more [details, see Section 35.3.1, “Monitoring the Client RPC Stream”](#_bookmark911).

### Monitoring the Client RPC Stream

The rpc\_stats file contains histogram data showing information about remote procedure calls (RPCs) that have been made since this file was last cleared. The histogram data can be cleared by writing any value into the rpc\_stats file.

The header information includes:

* + - snapshot\_time - UNIX epoch instant the file was read.
    - read RPCs in flight - Number of read RPCs issued by the OSC, but not complete at the time of the snapshot. This value should always be less than or equal to max\_rpcs\_in\_flight.
    - write RPCs in flight - Number of write RPCs issued by the OSC, but not complete at the time of the snapshot. This value should always be less than or equal to max\_rpcs\_in\_flight.
    - dio read RPCs in flight - Direct I/O (as opposed to block I/O) read RPCs issued but not completed at the time of the snapshot.
    - dio write RPCs in flight - Direct I/O (as opposed to block I/O) write RPCs issued but not completed at the time of the snapshot.
    - pending write pages - Number of pending write pages that have been queued for I/O in the OSC.
    - pending read pages - Number of pending read pages that have been queued for I/O in the OSC.

The tabular data is described in the table below. Each row in the table shows the number of reads or writes (ios) occurring for the statistic, the relative percentage (%) of total reads or writes, and the cumulative percentage (cum %) to that point in the table for the statistic.

|  |  |
| --- | --- |
| **Field** | **Description** |
| pages per RPC | Shows cumulative RPC reads and writes organized according to the number of pages in the RPC. A single page RPC increments the 0: row. |
| RPCs in flight | Shows the number of RPCs that are pending when an RPC is sent. When the first RPC is sent, the 0: row is incremented. If the first RPC is sent while another RPC is pending, the 1: row is incremented and so on. |
| offset | The page index of the first page read from or written to the object by the RPC. |

Analysis:

This table provides a way to visualize the concurrency of the RPC stream. Ideally, you will see a large clump around the max\_rpcs\_in\_flight value, which shows that the network is being kept busy.

### Monitoring Client Activity

The stats file maintains statistics accumulate during typical operation of a client across the VFS interface of the Lustre file system. Only non-zero parameters are displayed in the file.

Client statistics are enabled by default.

##### Note

Statistics for all mounted file systems can be discovered by entering:

lctl get\_param llite.\*.stats

The statistics can be cleared by echoing an empty string into the stats file or by using the command:

lctl set\_param llite.\*.stats=0

The statistics displayed are described in the table below.

|  |  |  |
| --- | --- | --- |
| **Entry** | **Description** | |
| snapshot\_time | UNIX epoch instant the stats file was read. | |
| dirty\_page\_hits | The number of write operations that have been satisfied by the dirty page cache. See [Section 35.4.1, “Tuning the Client I/O RPC Stream”](#_bookmark922) for more information about dirty cache behavior in a Lustre file system. | |
| dirty\_page\_misses | The number of write operations that were not satisfied by the dirty page cache. | |
| read\_bytes | The number of read operations that have occurred. Three additional parameters are displayed:  min The minimum number of bytes read in a single request since the counter was reset.  max The maximum number of bytes read in a single request since the counter was reset.  sum The accumulated sum of bytes of all read requests since the counter was reset. | |
| write\_bytes | The number of write operations that have occurred. Three additional parameters are displayed:  min The minimum number of bytes written in a single request since the counter was reset. | |
|  | max The maximum number of bytes written in a single request since  the counter was reset.  sum The accumulated sum of bytes of all write requests since the counter was reset. | |
| brw\_read | The number of pages that have been read. Three additional parameters are displayed:  min The minimum number of bytes read in a single block read/write (brw) read request since the counter was reset.  max The maximum number of bytes read in a single brw read requests since the counter was reset.  sum The accumulated sum of bytes of all brw read requests since the counter was reset. | |
| ioctl | The number of combined file and directory ioctl operations. | |
| open | The number of open operations that have succeeded. | |
| close | The number of close operations that have succeeded. | |
| seek | The number of times seek has been called. | |
| fsync | The number of times fsync has been called. | |
| truncate | The total number of calls to both locked and lockless truncate. | |
| setxattr | The number of times extended attributes have been set. | |
| getxattr | | The number of times value(s) of extended attributes have been fetched. | |

Information is provided about the amount and type of I/O activity is taking place on the client.

### Monitoring Client Read-Write Offset Statistics

When the offset\_stats parameter is set, statistics are maintained for occurrences of a series of read or write calls from a process that did not access the next sequential location. The OFFSET field is reset to 0 (zero) whenever a different file is read or written.

##### Note

By default, statistics are not collected in the offset\_stats, extents\_stats, and extents\_stats\_per\_process files to reduce monitoring overhead when this information is not needed. The collection of statistics in all three of these files is activated by writing anything, except for 0 (zero) and "disable", into any one of the files.

Example:

# lctl get\_param llite.testfs-f57dee0.offset\_stats snapshot\_time: 1155748884.591028 (secs.usecs)

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | RANGE | | RANGE | | SMALLEST | | LARGEST | |  | |
| R/W | | PID | START | | END | | EXTENT | | EXTENT | | OFFSET | |
| R | | 8385 | 0 | | 128 | | 128 | | 128 | | 0 | |
| R | | 8385 | 0 | | 224 | | 224 | | 224 | | -128 | |
| W | 8385 | | 0 | 250 | | 50 | | 100 | | 0 | |
| W | 8385 | | 100 | 1110 | | 10 | | 500 | | -150 | |
| W | 8384 | | 0 | 5233 | | 5233 | | 5233 | | 0 | |
| R | 8385 | | 500 | 600 | | 100 | | 100 | | -610 | |

In this example, snapshot\_time is the UNIX epoch instant the file was read. The tabular data is described in the table below.

The offset\_stats file can be cleared by entering:

lctl set\_param llite.\*.offset\_stats=0

|  |  |
| --- | --- |
| **Field** | **Description** |
| R/W | Indicates if the non-sequential call was a read or write |
| PID | Process ID of the process that made the read/write call. |
| RANGE START/RANGE END | Range in which the read/write calls were sequential. |
| SMALLEST EXTENT | Smallest single read/write in the corresponding range (in bytes). |
| LARGEST EXTENT | Largest single read/write in the corresponding range (in bytes). |
| OFFSET | Difference between the previous range end and the current range start. |

Analysis:

This data provides an indication of how contiguous or fragmented the data is. For example, the fourth entry in the example above shows the writes for this RPC were sequential in the range 100 to 1110 with the minimum write 10 bytes and the maximum write 500 bytes. The range started with an offset of -150 from the RANGE END of the previous entry in the example.

### Monitoring Client Read-Write Extent Statistics

For in-depth troubleshooting, client read-write extent statistics can be accessed to obtain more detail about read/write I/O extents for the file system or for a particular process.

##### Note

By default, statistics are not collected in the offset\_stats, extents\_stats, and extents\_stats\_per\_process files to reduce monitoring overhead when this information is not needed. The collection of statistics in all three of these files is activated by writing anything, except for 0 (zero) and "disable", into any one of the files.

## Tuning Lustre File System I/O

Each OSC has its own tree of tunables. For example:

$ ls -d /proc/fs/testfs/osc/OSC\_client\_ost1\_MNT\_client\_2 /localhost

/proc/fs/testfs/osc/OSC\_uml0\_ost1\_MNT\_localhost

/proc/fs/testfs/osc/OSC\_uml0\_ost2\_MNT\_localhost

/proc/fs/testfs/osc/OSC\_uml0\_ost3\_MNT\_localhost

$ ls /proc/fs/testfs/osc/OSC\_uml0\_ost1\_MNT\_localhost blocksizefilesfree max\_dirty\_mb ost\_server\_uuid stats

...

The following sections describe some of the parameters that can be tuned in a Lustre file system.

### Tuning the Client I/O RPC Stream

Ideally, an optimal amount of data is packed into each I/O RPC and a consistent number of issued RPCs are in progress at any time. To help optimize the client I/O RPC stream, several tuning variables are provided to adjust behavior according to network conditions and cluster size. For information about monitoring the [client I/O RPC stream, see Section 35.3.1, “Monitoring the Client RPC Stream”](#_bookmark911).

RPC stream tunables include:

* osc.*osc\_instance*.max\_dirty\_mb - Controls how many MBs of dirty data can be written and queued up in the OSC. POSIX file writes that are cached contribute to this count. When the limit is reached, additional writes stall until previously-cached writes are written to the server. This may be changed by writing a single ASCII integer to the file. Only values between 0 and 2048 or 1/4 of RAM are allowable. If 0 is specified, no writes are cached. Performance suffers noticeably unless you use large writes (1 MB or more).

To maximize performance, the value for max\_dirty\_mb is recommended to be 4 \*

max\_pages\_per\_rpc \* max\_rpcs\_in\_flight.

* osc.*osc\_instance*.cur\_dirty\_bytes - A read-only value that returns the current number of bytes written and cached on this OSC.

osc.*osc\_instance*.max\_pages\_per\_rpc - The maximum number of pages that will undergo I/O in a single RPC to the OST. The minimum setting is a single page and the maximum setting is 1024 (for systems with a PAGE\_SIZE of 4 KB), with the default maximum of 1 MB in the RPC. It is also possible to specify a units suffix (e.g. 4M), so that the RPC size can be specified independently of the client PAGE\_SIZE.

* osc.*osc\_instance*.max\_rpcs\_in\_flight - The maximum number of concurrent RPCs in flight from an OSC to its OST. If the OSC tries to initiate an RPC but finds that it already has the same number of RPCs outstanding, it will wait to issue further RPCs until some complete. The minimum setting is 1 and maximum setting is 256.

To improve small file I/O performance, increase the max\_rpcs\_in\_flight value.

* llite.*fsname-instance*/max\_cache\_mb - Maximum amount of inactive data cached by the client (default is 3/4 of RAM). For example:

# lctl get\_param llite.testfs-ce63ca00.max\_cached\_mb 128

##### Note

The value for *osc\_instance* is typically *fsname*-OST*ost\_index*-osc- *mountpoint\_instance*, where the value for *mountpoint\_instance* is unique to each mount point to allow associating osc, mdc, lov, lmv, and llite parameters with the same mount point.

### Tuning OSS Read Cache

The OSS read cache feature provides read-only caching of data on an OSS. This functionality uses the Linux page cache to store the data and uses as much physical memory as is allocated.

OSS read cache improves Lustre file system performance in these situations:

* + - * + Many clients are accessing the same data set (as in HPC applications or when diskless clients boot from the Lustre file system).
        + One client is storing data while another client is reading it (i.e., clients are exchanging data via the OST).
        + A client has very limited caching of its own. OSS read cache offers these benefits:
        + Allows OSTs to cache read data more frequently.
        + Improves repeated reads to match network speeds instead of disk speeds.
        + Provides the building blocks for OST write cache (small-write aggregation).

##### Using OSS Read Cache

OSS read cache is implemented on the OSS, and does not require any special support on the client side. Since OSS read cache uses the memory available in the Linux page cache, the appropriate amount of memory for the cache should be determined based on I/O patterns; if the data is mostly reads, then more cache is required than would be needed for mostly writes.

OSS read cache is managed using the following tunables:

* + - * + read\_cache\_enable - Controls whether data read from disk during a read request is kept in memory and available for later read requests for the same data, without having to re-read it from disk. By default, read cache is enabled (read\_cache\_enable=1).

When the OSS receives a read request from a client, it reads data from disk into its memory and sends the data as a reply to the request. If read cache is enabled, this data stays in memory after the request from the client has been fulfilled. When subsequent read requests for the same data are received, the OSS skips reading data from disk and the request is fulfilled from the cached data. The read cache is managed by the Linux kernel globally across all OSTs on that OSS so that the least recently used cache pages are dropped from memory when the amount of free memory is running low.

If read cache is disabled (read\_cache\_enable=0), the OSS discards the data after a read request from the client is serviced and, for subsequent read requests, the OSS again reads the data from disk.

To disable read cache on all the OSTs of an OSS, run:

root@oss1# lctl set\_param obdfilter.\*.read\_cache\_enable=0

To re-enable read cache on one OST, run:

root@oss1# lctl set\_param obdfilter.{OST\_name}.read\_cache\_enable=1

To check if read cache is enabled on all OSTs on an OSS, run:

root@oss1# lctl get\_param obdfilter.\*.read\_cache\_enable

* + - * + writethrough\_cache\_enable - Controls whether data sent to the OSS as a write request is kept in the read cache and available for later reads, or if it is discarded from cache when the write is completed. By default, the writethrough cache is enabled (writethrough\_cache\_enable=1).

When the OSS receives write requests from a client, it receives data from the client into its memory and writes the data to disk. If the writethrough cache is enabled, this data stays in memory after the write request is completed, allowing the OSS to skip reading this data from disk if a later read request, or partial-page write request, for the same data is received.

If the writethrough cache is disabled (writethrough\_cache\_enabled=0), the OSS discards the data after the write request from the client is completed. For subsequent read requests, or partial-page write requests, the OSS must re-read the data from disk.

Enabling writethrough cache is advisable if clients are doing small or unaligned writes that would cause partial-page updates, or if the files written by one node are immediately being accessed by other nodes. Some examples where enabling writethrough cache might be useful include producer-consumer I/O models or shared-file writes with a different node doing I/O not aligned on 4096-byte boundaries.

Disabling the writethrough cache is advisable when files are mostly written to the file system but are not re-read within a short time period, or files are only written and re-read by the same node, regardless of whether the I/O is aligned or not.

To disable the writethrough cache on all OSTs of an OSS, run:

root@oss1# lctl set\_param obdfilter.\*.writethrough\_cache\_enable=0

To re-enable the writethrough cache on one OST, run:

root@oss1# lctl set\_param obdfilter.{OST\_name}.writethrough\_cache\_enable=1

To check if the writethrough cache is enabled, run:

root@oss1# lctl get\_param obdfilter.\*.writethrough\_cache\_enable

* + - * + readcache\_max\_filesize - Controls the maximum size of a file that both the read cache and writethrough cache will try to keep in memory. Files larger than readcache\_max\_filesize will not be kept in cache for either reads or writes.

Setting this tunable can be useful for workloads where relatively small files are repeatedly accessed by many clients, such as job startup files, executables, log files, etc., but large files are read or written only once. By not putting the larger files into the cache, it is much more likely that more of the smaller files will remain in cache for a longer time.

When setting readcache\_max\_filesize, the input value can be specified in bytes, or can have a suffix to indicate other binary units such as K (kilobytes), M (megabytes), G (gigabytes), T (terabytes), or P (petabytes).

To limit the maximum cached file size to 32 MB on all OSTs of an OSS, run:

root@oss1# lctl set\_param obdfilter.\*.readcache\_max\_filesize=32M

To disable the maximum cached file size on an OST, run:

root@oss1# lctl set\_param obdfilter.{OST\_name}.readcache\_max\_filesize=-1

To check the current maximum cached file size on all OSTs of an OSS, run:

root@oss1# lctl get\_param obdfilter.\*.readcache\_max\_filesize

# Chapter 36. User Utilities

This chapter describes user utilities.

## lfs

The lfs utility can be used for user configuration routines and monitoring.

### Description

The lfs utility is used to create a new file with a specific striping pattern, determine the default striping pattern, gather the extended attributes (object numbers and location) for a specific file, find files with specific attributes, list OST information or set quota limits. It can be invoked interactively without any arguments or in a non-interactive mode with one of the supported arguments.

## lfs\_migrate

The lfs\_migrate utility is a simple tool to migrate files between Lustre OSTs.

### Description

The lfs\_migrate utility is a simple tool to assist migration of files between Lustre OSTs. The utility copies each specified file to a new file, verifies the file contents have not changed, and then renames the new file to the original filename. This allows balanced space usage between OSTs, moving files off OSTs that are starting to show hardware problems (though are still functional) or OSTs that will be discontinued.

##### Warning

For versions of Lustre before 2.5, lfs\_migrate is not closely integrated with the MDS, it cannot determine whether a file is currently open and/or in-use by other applications or nodes. This makes it UNSAFE for use on files that might be modified by other applications, since the migrated file is only a copy of the current file. This results in the old file becoming an open- unlinked file and any modifications to that file are lost.

Files to be migrated can be specified as command-line arguments. If a directory is specified on the command-line then all files within the directory are migrated. If no files are specified on the command- line, then a list of files is read from the standard input, making lfs\_migrate suitable for use with lfs The current file allocation policies on the MDS dictate where the new files are placed, taking into account whether specific OSTs have been disabled on the MDS via lctl(preventing new files from being allocated there), whether some OSTs are overly full (reducing the number of files placed on those OSTs), or if there is a specific default file striping for the target directory (potentially changing the stripe count, stripe size, OST pool, or OST index of a new file).

##### Note

The lfs\_migrate utility can also be used in some cases to reduce file fragmentation. File fragmentation will typically reduce Lustre file system performance. File fragmentation may be observed on an aged file system and will commonly occur if the file was written by many threads. Provided there is sufficient free space (or if it was written when the file system was nearly full) that is less fragmented than the file being copied, re-writing a file with lfs\_migrate will result in a migrated file with reduced fragmentation. The tool filefrag can be used to report file [fragmentation. See Section 36.3, “ filefrag ”](#_bookmark965)

##### Note

As long as a file has extent lengths of tens of megabytes ( *read\_bandwidth \* seek\_time*) or more, the read performance for the file will not be significantly impacted by fragmentation, since the read pipeline can be filled by large reads from disk even with an occasional disk seek.

## filefrag

The e2fsprogs package contains the filefrag tool which reports the extent of file fragmentation.

### Description

The filefrag utility reports the extent of fragmentation in a given file. The filefrag utility obtains the extent information from Lustre files using the FIEMAP ioctl, which is efficient and fast, even for very large files.

In default mode 1, filefrag prints the number of physically discontiguous extents in the file. In extent or verbose mode, each extent is printed with details such as the blocks allocated on each OST. For a Lustre file system, the extents are printed in device offset order (i.e. all of the extents for one OST first, then the next OST, etc.), not file logical offset order. If the file logical offset order was used, the Lustre striping would make the output very verbose and difficult to see if there was file fragmentation or not.

##### Note

Note that as long as a file has extent lengths of tens of megabytes or more (i.e. *read\_bandwidth \* seek\_time > extent\_length*), the read performance for the file will not be significantly impacted by fragmentation, since file readahead can fully utilize the disk disk bandwidth even with occasional seeks.

In default mode 2, filefrag returns the number of physically discontiguous extents in the file. In extent or verbose mode, each extent is printed with details. For a Lustre file system, the extents are printed in device offset order, not logical offset order.

# Chapter 40. System Configuration Utilities

This chapter includes system configuration utilities and includes the following sections:

## e2scan

The e2scan utility is an ext2 file system-modified inode scan program. The e2scan program uses libext2fs to find inodes with ctime or mtime newer than a given time and prints out their pathname. Use e2scan to efficiently generate lists of files that have been modified. The e2scan tool is included in the e2fsprogs package, located at:

<http://downloads.hpdd.intel.com/public/e2fsprogs/latest/>

### Description

When invoked, the e2scan utility iterates all inodes on the block device, finds modified inodes, and prints their inode numbers. A similar iterator, using libext2fs(5), builds a table (called parent database) which lists the parent node for each inode. With a lookup function, you can reconstruct modified pathnames from root.

* 1. **l\_getidentity**

The l\_getidentity utility handles Lustre user / group cache upcall.

### Description

The group upcall file contains the path to an executable file that is invoked to resolve a numeric UID to a group membership list. This utility opens /proc/fs/lustre/mdt/${FSNAME}- MDT{xxxx}/identity\_info and writes the related identity\_downcall\_data structure (see [Section 37.1.4, “Data Structures”](#_bookmark980).) The data is persisted with lctl set\_param mdt.${FSNAME}- MDT{xxxx}.identity\_info.

The l\_getidentity utility is the reference implementation of the user or group cache upcall.

## lctl

The lctl utility is used for root control and configuration. With lctl you can directly control Lustre via an ioctl interface, allowing various configuration, maintenance and debugging features to be accessed.

### Description

The lctl utility can be invoked in interactive mode by issuing the lctl command. After that, commands are issued as shown below. The most common lctl commands are:

dl

dk

device

network up | down

list\_nids

ping nidhelp

quit

For a complete list of available commands, type help at the lctl prompt. To get basic help on command meaning and syntax, type help *command*. Command completion is activated with the TAB key (depending on compile options), and command history is available via the up- and down-arrow keys.

For non-interactive use, use the second invocation, which runs the command after connecting to the device.

### Setting Parameters with lctl

Lustre parameters are not always accessible using the procfs interface, as it is platform-specific. As a solution, lctl {get,set}\_param has been introduced as a platform-independent interface to the Lustre tunables. Avoid direct references to /proc/{fs,sys}/{lustre,lnet}. For future portability, use lctl

{get,set}\_param .

When the file system is running, use the lctl set\_param command on the affected node(s) to *temporarily* set parameters (mapping to items in /proc/{fs,sys}/{lnet,lustre}). The lctl set\_param command uses this syntax:

## ll\_decode\_filter\_fid

The ll\_decode\_filter\_fid utility displays the Lustre object ID and MDT parent FID.

### Description

The ll\_decode\_filter\_fid utility decodes and prints the Lustre OST object ID, MDT FID, stripe index for the specified OST object(s), which is stored in the "trusted.fid" attribute on each OST object. This is accessible to ll\_decode\_filter\_fid when the OST file system is mounted locally as type ldiskfs The "trusted.fid" extended attribute is stored on each OST object when it is first modified (data written or attributes set), and is not accessed or modified by Lustre after that time.

The OST object ID (objid) is useful in case of OST directory corruption, though normally the ll\_recover\_lost\_found\_objs(8) utility is able to reconstruct the entire OST object directory hierarchy. The MDS FID can be useful to determine which MDS inode an OST object is (or was) used by. The stripe index can be used in conjunction with other OST objects to reconstruct the layout of a file even if the MDT inode was lost.

## ll\_recover\_lost\_found\_objs

The ll\_recover\_lost\_found\_objs utility helps recover Lustre OST objects (file data) from a lost and found directory and return them to their correct locations.

### Description

The first time Lustre modifies an object, it saves the MDS inode number and the objid as an extended attribute on the object, so in case of directory corruption of the OST, it is possible to recover the objects. Running e2fsck fixes the corrupted OST directory, but it puts all of the objects into a lost and found directory, where they are inaccessible to Lustre. Use the ll\_recover\_lost\_found\_objs utility to recover all (or at least most) objects from a lost and found directory and return them to the O/0/d\* directories.

To use ll\_recover\_lost\_found\_objs, mount the file system locally (using the -t ldiskfs, or -t zfs command), run the utility and then unmount it again. The OST must not be mounted by Lustre when ll\_recover\_lost\_found\_objs is run.

## llobdstat

The llobdstat utility displays OST statistics.

### Description

The llobdstat utility displays a line of OST statistics for the given ost\_name every interval seconds. It should be run directly on an OSS node. Type CTRL-C to stop statistics printing.

## llog\_reader

The llog\_reader utility translates a Lustre configuration log into human-readable form.

### Description

The llog\_reader utility parses the binary format of Lustre's on-disk configuration logs. Llog\_reader can only read logs; use tunefs.lustre to write to them.

To examine a log file on a stopped Lustre server, mount its backing file system as ldiskfs or zfs, then use llog\_reader to dump the log file's contents, for example:

mount -t ldiskfs /dev/sda /mnt/mgs llog\_reader /mnt/mgs/CONFIGS/tfs-client

To examine the same log file on a running Lustre server, use the ldiskfs-enabled debugfs utility (called debug.ldiskfs on some distributions) to extract the file, for example:

debugfs -c -R 'dump CONFIGS/tfs-client /tmp/tfs-client' /dev/sda llog\_reader /tmp/tfs-client

##### Caution

Although they are stored in the CONFIGS directory, mountdata files do not use the configuration log format and will confuse the llog\_reader utility.

## llstat

The llstat utility displays Lustre statistics.

### Description

The llstat utility displays statistics from any of the Lustre statistics files that share a common format and are updated at interval seconds. To stop statistics printing, use ctrl-c.

## llverdev

The llverdev verifies a block device is functioning properly over its full size.

### Description

Sometimes kernel drivers or hardware devices have bugs that prevent them from accessing the full device size correctly, or possibly have bad sectors on disk or other problems which prevent proper data storage. There are often defects associated with major system boundaries such as 2^32 bytes, 2^31 sectors, 2^31 blocks, 2^32 blocks, etc.

The llverdev utility writes and verifies a unique test pattern across the entire device to ensure that data is accessible after it was written, and that data written to one part of the disk is not overwriting data on another part of the disk.

It is expected that llverdev will be run on large size devices (TB). It is always better to run llverdev in verbose mode, so that device testing can be easily restarted from the point where it was stopped.

Running a full verification can be time-consuming for very large devices. We recommend starting with a partial verification to ensure that the device is minimally sane before investing in a full verification.

## lshowmount

The lshowmount utility shows Lustre exports.

### Description

The lshowmount utility shows the hosts that have Lustre mounted to a server. This utility looks for exports from the MGS, MDS, and obdfilter.

## lst

The lst utility starts LNet self-test.

### Description

LNet self-test helps site administrators confirm that Lustre Networking (LNet) has been properly installed and configured. The self-test also confirms that LNet and the network software and hardware underlying it are performing as expected.

Each LNet self-test runs in the context of a session. A node can be associated with only one session at a time, to ensure that the session has exclusive use of the nodes on which it is running. A session is create, controlled and monitored from a single node; this is referred to as the self-test console.

Any node may act as the self-test console. Nodes are named and allocated to a self-test session in groups. This allows all nodes in a group to be referenced by a single name.

Test configurations are built by describing and running test batches. A test batch is a named collection of tests, with each test composed of a number of individual point-to-point tests running in parallel. These individual point-to-point tests are instantiated according to the test type, source group, target group and distribution specified when the test is added to the test batch.

### Modules

To run LNet self-test, load these modules: libcfs, lnet, lnet\_selftest and any one of the klnds (ksocklnd, ko2iblnd...). To load all necessary modules, run modprobe lnet\_selftest, which recursively loads the modules on which lnet\_selftest depends.

There are two types of nodes for LNet self-test: the console node and test nodes. Both node types require all previously-specified modules to be loaded. (The userspace test node does not require these modules).

Test nodes can be in either kernel or in userspace. A console user can invite a kernel test node to join the test session by running lst add\_group NID, but the user cannot actively add a userspace test node to the test session. However, the console user can passively accept a test node to the test session while the test node runs lst client to connect to the console.

### Utilities

LNet self-test includes two user utilities, lst and lstclient.

lst is the user interface for the self-test console (run on the console node). It provides a list of commands to control the entire test system, such as create session, create test groups, etc.

lstclient is the userspace self-test program which is linked with userspace LNDs and LNet. A user can invoke lstclient to join a self-test session:

lstclient -sesid CONSOLE\_NID group NAME

## lustre\_rmmod.sh

The lustre\_rmmod.sh utility removes all Lustre and LNet modules (assuming no Lustre services are running). It is located in /usr/bin.

##### Note

The lustre\_rmmod.sh utility does not work if Lustre modules are being used or if you have manually run the lctl network up command.

## lustre\_rsync

The lustre\_rsync utility synchronizes (replicates) a Lustre file system to a target file system.

### Description

The lustre\_rsync utility is designed to synchronize (replicate) a Lustre file system (source) to another file system (target). The target can be a Lustre file system or any other type, and is a normal, usable file system. The synchronization operation is efficient and does not require directory walking, as lustre\_rsync uses Lustre MDT changelogs to identify changes in the Lustre file system.

Before using lustre\_rsync:

* + - * A changelog user must be registered (see lctl (8) changelog\_register)

- AND -

* + - * Verify that the Lustre file system (source) and the replica file system (target) are identical before the changelog user is registered. If the file systems are discrepant, use a utility, e.g. regular rsync (not lustre\_rsync) to make them identical.

## mkfs.lustre

The mkfs.lustre utility formats a disk for a Lustre service.

### Description

mkfs.lustre is used to format a disk device for use as part of a Lustre file system. After formatting, a disk can be mounted to start the Lustre service defined by this command.

When the file system is created, parameters can simply be added as a --param option to the

mkfs.lustre [command. See Section 13.11.1, “Setting Tunable Parameters with mkfs.lustre”](#_bookmark286).

## mount.lustre

The mount.lustre utility starts a Lustre client or target service.

### Description

The mount.lustre utility starts a Lustre client or target service. This program should not be called directly; rather, it is a helper program invoked through mount(8), as shown above. Use the umount command to stop Lustre clients and targets.

* + - * [Section 36.1, “ lfs ”](#_bookmark950)

## plot-llstat

The plot-llstat utility plots Lustre statistics.

### Description

The plot-llstat utility generates a CSV file and instruction files for gnuplot from the output of llstat. Since llstat is generic in nature, plot-llstat is also a generic script. The value of parameter\_index can be 1 for count per interval, 2 for count per second (default setting) or 3 for total count.

The plot-llstat utility creates a .dat (CSV) file using the number of operations specified by the user. The number of operations equals the number of columns in the CSV file. The values in those columns are equal to the corresponding value of parameter\_index in the output file.

The plot-llstat utility also creates a .scr file that contains instructions for gnuplot to plot the graph. After generating the .dat and .scr files, the plot-llstat tool invokes gnuplot to display the graph.

## routerstat

The routerstat utility prints Lustre router statistics.

### Description

The routerstat utility displays LNet router statistics. If no *interval* is specified, then statistics are sampled and printed only one time. Otherwise, statistics are sampled and printed at the specified *interval* (in seconds).

## tunefs.lustre

The tunefs.lustre utility modifies configuration information on a Lustre target disk.

### Description

tunefs.lustre is used to modify configuration information on a Lustre target disk. This does not reformat the disk or erase the target information, but modifying the configuration information can result in an unusable file system.

##### Caution

Changes made here affect a file system only when the target is mounted the next time.

## Additional System Configuration Utilities

This section describes additional system configuration utilities for Lustre.

The stats-collect utility contains scripts used to collect application profiling information from Lustre clients and servers.

/proc/fs/lustre/mdt/\*/md\_stats

### Testing / Debugging Utilities

Lustre offers the following test and debugging utilities.

#### lr\_reader

The lr\_reader utility translates the content of the last\_rcvd and reply\_data files into human- readable form.

The following utilities are part of the Lustre I/O kit. For more information, see [Chapter 29, *Benchmarking*](#_bookmark676)[*Lustre File System Performance (Lustre I/O Kit)*](#_bookmark676).

#### sgpdd-survey

The sgpdd-survey utility tests 'bare metal' performance, bypassing as much of the kernel as possible. The sgpdd-survey tool does not require Lustre, but it does require the sgp\_dd package.

##### Caution

The sgpdd-survey utility erases all data on the device.

##### obdfilter-survey

The obdfilter-survey utility is a shell script that tests performance of isolated OSTS, the network via echo clients, and an end-to-end test.

##### ior-survey

The ior-survey utility is a script used to run the IOR benchmark. Lustre includes IOR version 2.8.6.

#### *ost-survey*

The ost-survey utility is an OST performance survey that tests client-to-disk performance of the individual OSTs in a Lustre file system.

##### stats-collect

The stats-collect utility contains scripts used to collect application profiling information from Lustre clients and servers.