# OpenSFS TWG Lustre Requirements

This document summarizes requirements for new Lustre features, which the OpenSFS Technical Working Group (TWG) gathered from the community in early 2011. Based on these findings, the TWG recommends development of features to address requirements of immediate importance to the OpenSFS membership.

## Table of Contents:

IPv6

## Recommendations to the OpenSFS Board

A complete listing of requirements gathered over the last several months follows this section. During the course of our investigation, we identified near-term requirements for improving Lustre performance, scalability, manageability, and administration. From this list, the TWG has identified requirements for Lustre metadata performance and scalability as being of the most immediate benefit to its members' needs.

In addition, the TWG members expect OpenSFS to take a broad view and consider work that needs to be done in Lustre to pave the way for future functionality. Investment in these foundational requirements will resolve some of the remaining technical debt in the Lustre code and set the stage for feature deliveries in the 2013-2014 time frame.

### Near-term requirements

1. improve metadata server performance
2. improve metadata server scalability

### Foundational Requirements

1. support for alternate backend file systems
2. scalable fault management
3. backend storage investigation

At this time, the TWG recommends that OpenSFS pursue RFPs for both performance and foundational requirements, focusing on at least one requirement from the prioritized categories above. These requirements have been discussed by the members of the TWG and the ordering has been reached by consensus.

## Gathered Requirements

### Required Rates and Capacities

This section describes specific requirements for file system performance and scalabilty that the community thinks Lustre will need to accommodate HPC systems in the near term (2012) and beyond (2014).

| **Metric** | **Lustre 2.0/2.1**[1] | **Q2 2012** | **Q1 2014** |
|---|---|---|---|
| maximum number of files in file system | 4 billion | 100 billion | 1 trillion |

| | | | |
|---|---|---|---|
| maximum number of files in directory | 10 million | 50 million | 10 billion |
| maximum number of subdirectories | 10 million | 1 million | 10 million |
| maximum number of clients | 131072 | 64 thousand | 128 thousand |
| maximum number of OSS nodes | - | 1 thousand | 4 thousand |
| maximum number of OSTs | 8150 | 2 thousand | 8 thousand |
| maximum OST size | 16 TB | 32 TB | 128 TB |
| maximum file system size | 64 PB | 100 PB | 256 PB |
| maximum file size | 320 TB | 1 PB | - |
| maximum object size | 2 TB | 16 TB | 64 TB |
| peak aggregate file creates/s | - | 200 thousand | 400 thousand |
| peak directory listings/s (ls -l) | - | - | 100 thousand |
| maximum single client open files | ~3 thousand[2] | 100 thousand | - |
| peak single client file creates/s | - | 30 thousand | - |

[1] source : Lustre 2.0 Manual, Table 5-1

[2] ”Lustre does not impose a maximum for the number of open files, but the practical limit depends on the amount of RAM on the MDS. No "tables" for open files exist on the MDS, as they are only linked in a list to a given client's export. Each client process probably has a limit of several thousands of open files which depends on the ulimit.”

## Performance Requirements

The requirements in this section address limitations to Lustre which the TWG believes can be addressed through immediate development. Deliverables that meet these requirements should provide immediate benefits.

### Metadata server performance

Interactive workloads (ls -l, du) do not perform as well with Lustre as they do on local file systems. The MDS software architecture has not kept pace with the capabilities of current multicore hardware architectures. It is a requirement that the MDS be capable of using efficiently all of the compute resources available on commodity server platforms.

### Metadata server scalability

The single metadata server is Lustre's greatest architectural liability. The file system provides horizontal scalability of the data store across multiple object storage servers, but the metadata services are still limited to a single metadata server.    Lustre performance and capacity can be improved by enabling horizontal scale-out of the MDS, allowing the file system namespace to be distributed.  Maximal performance will result when directories themselves can be distributed across multiple MDS hosts.

### Single file performance

Users desire single, shared files for more sane file management, though this does not always provide the best performance due to locking and other implementation issues. Lustre must allow single files to take full advantage of capabilities of the storage system, and must have interfaces that allow users to exploit their knowledge of their IO patterns.

### Quality of service

Existing deployments currently have no mechanism to balance the performance needs of interactive users against the needs of large-scale compute jobs.  For example, it is possible and likely that directory listings will encounter absurdly long execution times when competing against a 200,000 core checkpoint operation. To maintain usability in such scenarios, Lustre must be able to allocate a {job,cluster,user} a share of IOPS and bandwidth consummate with the priority levels assigned by an administrator.

### Locality and scalability

Large systems will need to use client locality to determine the OSSs for storage in order to avoid contention across the interconnect. Locality should allow clients to reduce the time they spend checking the status of all servers in the file system.

### LNET channel bonding

LNET routers are currently limited to a single channel provided by a single instance of the LND. This restriction limits bandwidth and reliability of an LNET router to a single interconnect.  LNET should allow multiple LNDs to be bonded as a group in order to provide increased bandwidth and increased availability.

## Foundational Requirements

The following describes requirements of Lustre to meet future needs. Investment in one of these technologies now, will provide the foundation for features needed for Lustre to achieve the next levels of performance.

### Support for alternate backend file systems

Ldiskfs is at the limits of its useful life.  Selection of an alternate backend store is impossible unless Lustre supports interchangeable backend file systems. Lustre engineers had started a project to create a backend abstraction for arbitrary Object Storage Devices (OSDs). This effort was not completed.  There remains considerable code reorganization to facilitate interoperability with different backend devices whether through OSD or some other interface.

### Backend storage investigation

To accommodate the capacities above, the backend store must expand beyond current ldiskfs limits (eg 32TB LUN sizes). We seek backend solutions that improve Lustre reliability and resiliency. LLNL is pursuing ZFS as an ldiskfs replacement. Btrfs has many features in common with ZFS. It is of interest because it is licensed under the GPL and included with Linux.

Assuming that Lustre can be restructured to accommodate alternate backend stores, we need to investigate alternative file systems to understand their architecture, layering, stability, and performance. These baseline investigations should be completed _before_ there any attempts to implement an OSD interface for the file system.

### Scalable fault management

While it is already the case that today's supercomputers have a marked dependence on their file systems for productive use, this dependency will continue to rise as we see more and more center-wide file systems. To minimize the downtime for the entire center, reliability must increase and recovery from faults must be bounded in time. Lustre must be able to recover in O(log n) time or better as a mid-term goal to meet this requirement.

Lustre must expose errors it detects to standard administrative infrastructures. We cannot continue with error logs as being used today. Instead, Lustre must detect, collect, and parse faults then distribute the errors in a scalable manner to the administrative interface for notification.

## Manageability and Administrative Requirements

The requirements in this section highlight areas of improvement for Lustre in order to reduce the administrative burden or address shortcomings in its integration to the computing environment.

### Better support for newer kernels

Security updates for Lustre kernels remain a sore point for system administrators. Using patches to the kernel on the server side introduces a potential delay to rolling out updates. In addition, newer releases require new kernel revisions to be supported. Lustre must reduce or eliminate its need to patch the kernel on the server, and must support the newer kernels in RHEL6 as a minimum, and should support recent kernel.org kernels.

### Improved configuration of Lustre

The current mechanism of using module parameters is relatively inflexible and can constrain large, complex Lustre configurations. As a short- to mid-term requirement, Lustre must have the ability to allow dynamic configuration of the LNET interfaces and routes. This ability should include the ability to bind targets (OST/MDT/etc) to specific LNET interfaces, and to hot-add or hot-remove LNET interfaces.

### Allowing for controlled partial-system maintenance

Currently, to upgrade a Lustre installation or perform maintenance on a subset of the comprising hardware, one must unmount the filesystem from all clients or risk hanging processes until the hardware is back online (maintenance) or other odd, undefined client behavior once the upgrade completes. To allow more flexible administration, the file system must be able to handle these situations gracefully, and allow the clients to avoid attempting to use hardware known to be down.

### Balancing storage use

Currently, ensuring a balanced use of the storage space available to Lustre relies on a haphazard set of setting default striping, storage pools, and manual rebalancing of overfull OSTs. As a mid-term goal, Lustre must be able to allow automatic emptying of an OST, migrating the data to other devices in the filesystem. Similarly, Lustre must be able to rebalance the storage load over new OSTs as they are added. Additionally, Lustre must be able to require authorization for use of specific storage pools.

### Adaptive storage layout

Users are often confused by the relationship between maximum file size and object count. In addition, they often make poor striping choices, causing massive imbalances in OST use. Lustre should have the ability to adapt the storage layout of the file as it grows and/or ages, such as adding more objects as needed. This adaptive layout should be able to be set as the default striping pattern by administrators, but must not preclude knowledgeable users from continuing to set a specific layout. Additionally, users must be able to specify the exact layout of the file if so desired, to include specific OSTs and their order in the striping.

### Arbitrary OST assignment

Lustre should allow specific stripes to be assigned to specific OSTs rather than just heuristically. This has a potential impact on WAN operations.

### Better userspace tools

The lctl command is confusing to use, with mixes of fixed and non-fixed positional parameters, and poor documentation.  Further, the output of many of the sub commands are not particularly well designed, making them both difficult to read for a human, and difficult to parse by command-line scripting.  We desire clean, well-designed command line interfaces.

### File system consistency checks

Compute centers can ill-afford the downtime required to ensure the consistency of Lustre or its backing file systems.  There must be online integrity check and repair processes that can continually run in the background to verify the consistency of both file systems. These processes must identify and repair various inconsistencies including, but not limited to, orphaned and missing data objects. The processes should have low impact on normal operations of the file system.

### Snapshots

While backing up a large scale Lustre file system to offline storage may not be a practical endeavor, allowing the possibility is a desirable goal. To support this activity, Lustre must be able to efficiently quiesce the the file system and make a stable snapshot. Snapshots made in this way must be easily accessible to users -- subject to normal access control measures -- to allow easy recovery from simple mistakes without requiring administrator assistance.

### User Identity Mapping

As Lustre use expands over the WAN into environments that have differing models of user management, there is a growing need to map identities from one management domain to another, on a per-NID basis. This mapping must be performed in such a manner that continued operation of Lustre's quota system is achieved.

## Application Interface Requirements

The requirements in this section highlight areas of improvement for Lustre in order to reduce the overheads experienced by applications, both in development and in operation.

### Improved storage semantics/interfaces

Lustre should explore alternatives to POSIX access methods that can be used to support exascale file sytem requirements.   At the scales of today's large systems -- and as those scales are expected to grow in the future -- the familiar semantics of POSIX incur challenges to developers seeking to extract maximum  performance from the hardware. In the future, Lustre must allow developers to avoid the performance pitfalls -- both by improving the performance when operating in POSIX mode, or by allowing one to tell the system "I know what I am doing" and step outside of those semantics. Applications should be able to inform Lustre that they do not need the locking implied to reach POSIX semantics and/or give hints to the file system as to what their usage pattern is expected  to look like. Applications should be able to submit requests that avoid copies without blocking on the completion of those requests.

### Better user tool API

llapi as it exists now is really largely the internals of the lfs command.  As a result, many of the functions print directly to stdout, which does not lend itself to reusablity.  We desire clean APIs which can be used by many programs to interact with lustre to gather and present data in a format of the program's own choosing.

## Other Requirements

These requirements do not properly fit into other categories, but remain important as Lustre and the hardware it runs on continue to evolve.

### Varying page-sizes

Lustre must allow clients and servers to use different page sizes. We expect smaller pages on the client as compared to the server, but future hardware may challenge this expectation. Lustre should strive for flexibility in this area, and allow for heterogeneous page sizes among concurrently connected clients and servers.

### Mixed endian support

The two platforms with the largest share of the HPC market have differing byte orders. In order to extend the benefits of sharing storage between multiple systems to shops supporting both platforms, Lustre must be able to interoperate seamlessly between clients and servers of differing byte orders. Long-term, Lustre should support servers on either byte ordering.

### IPv6

Support for IPv6 requires a change in the NID format to accommodate a 128bit IPv6 address in the address-within-network field.  This affects all protocol levels: LNDs, LNET and Lustre.