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 2012. Based on these findings, the TWG recommends development of features to address requirements of immediate importance to the OpenSFS membership.

Table of Contents:

Recommendations to the OpenSFS Board

During the course of our investigation this year, the TWG found significant agreement among members on four broad areas--availability, storage management, performance, and Lustre networking--that OpenSFS should address in its forthcoming development RFPs.

Availability encompasses requirements to make Lustre more robust and better able to tolerate errors. The highest priority by consensus is to address Lustre's dependence on timeouts and requires development to avoid timeouts as is best possible. This category also requires improved fault management.

Storage Management is a new area for Lustre that builds on the foundational work started by CEA for their HSM project. In this category, we seek to extend CEA's work to make Lustre more relevant in modern data centers and more competitive with other file systems by enabling enterprise class features such as object migration, file mirroring, and replication. We know that these changes will require a common infrastructure, on which creating these closely related features and enhancements would be easily possible.

Performance continues to be an area of concern. We are hopeful that the new metadata features currently under development (SMP affinity, distributed namespace) will greatly improve performance of the metadata server. Nonetheless, there are workloads where Lustre performance continues to need improvement and we want to address architectural bottlenecks for both bulk I/O and metadata performance of a single Lustre client.

Lustre networking (LNET) is the transport for remote procedure calls (RPCs) to the Lustre servers. We identified a number of requirements for improving Lustre scalability, configurability and reliability of Lustre by enhancing core networking functionality.

We believe the requirements and features described in the availability and storage management categories are foundational in nature and, as a result, OpenSFS needs to start working on them now to ensure feature landings in years to come. The table below describes the four categories and identifies requirements for each that could be addressed in upcoming RFPs. A complete listing of all requirements gathered over the last several months follows this section.

Category (prioritized)	**Requirements** (not prioritized)
File system availability and robustness	Avoid RPC timeouts Scalable fault management
Storage management	HSM and storage management infrastructure OST migration/rebalancing
Performance	Single client IO performance File create performance Directory traversal and attribute retrieval
Lustre networking (LNET)	LNET channel bonding Improved LNET robustness Dynamic LNET configuration

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 features delivered in the 2014 time frame.

In addition, we note that there has been considerable interest in restructuring the validation tests included with each Lustre build. Although this level of interest suggests that community investment may be warranted in the future to help create a consistent solution, we believe the requirement should be addressed by the Community Development Working Group (CDWG) rather than be defined as a roadmap item by the TWG.

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 in the table above. These requirements have been discussed by the members of the TWG and the category ordering has been reached by consensus (http://goo.gl/Lqg7s).

Gathered Requirements

The OpenSFS TWG published its first list of requirements and recommendations to the OpenSFS Board in March 2011 (http://goo.gl/cZSWG). As a result, the OpenSFS has funded development of features that address requirements to improve metadata server performance, metadata server scalability, online file system consistency checks, and user identity mapping.

In preparation for the 2012 Lustre Users Group meeting, the OpenSFS TWG held weekly conference calls since February 2012 to consider new requirements to add to the remaining requirements from its original list. Minutes from these meetings have been posted to the OpenSFS Discuss reflector (http://lists.opensfs.org/pipermail/discuss-opensfs.org/).

The working group discussed a number of new requirements during this year's investigation that are listed in the appendix. These topics have been integrated into the requirements sections that follow. Requirements from 2011 that were moved or merged have been saved in the appendix.

Please note that the order of requirements and features below does NOT imply any ranking or preference.

Availability Requirements

The following requirements address improvements to Lustre availability, fault tolerance and recovery at future system scales. Investment in one of these technologies now, will provide the foundation that Lustre needs to achieve the next levels of system scale.

Avoid RPC timeouts

Users sometimes perceive Lustre as unstable because of periodic pauses in execution as Lustre waits for an overloaded server, or a timeout to expire. We have discussed health networks as a means of providing lower latency fault detection and improved error handling. This will be a scalability feature that by providing an active, deterministic mechanism for communicating system status will avoid the sequence of cascading timeouts that limits Lustre at scale. For example, the current use of pings is difficult to tune at scale, and also imposes a significant overhead on the network, impact to IO performance, and impact to OS noise/jitter. The health network should be a high priority, efficient, and reliable communications channel to avoid the need for timeouts and make client/server interactions more deterministic. As a result, we expect work fulfilling this requirement to improve recovery times, facilitate error detection within Lustre and improve file system responsiveness.

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.

Storage Management Requirements

For purposes of this discussion, *migration* is the ability to move objects within the same file system. *Mirroring* is the ability to create replicas of objects referenced by the same metadata within the same file system. This is different from *remote replication*, which is the process of copying data from one file system to another separate file system (namespace). Remote replication is possible today with Lustre rsync. More intelligent tools based on change logs, rather than walking the directory tree, are anticipated. Dynamic layouts, object migration, and mirroring will benefit from a common infrastructure introduced with the current Hierarchical Storage Management (HSM) project.

HSM and storage management infrastructure

There is an ongoing project to implement Hierarchical Storage Management (HSM) for Lustre that will provide the foundational infrastructure needed for several of the features listed here. In particular, migration relies on the layout lock feature to ensure that object migration maintains file coherency. The current layout lock implementation will require work to manage files that are not at rest. Furthermore, the Lustre client will need to be made aware of these layout changes. The Lustre changelog feature may need improved scalability so that policy engines that utilize these features do not negatively impact performance.

OST migration/rebalancing

Move objects between OSTs to more evenly distribute free space among the OSTs or to distribute objects to new OSTs added to expand the file system. This same facility can be used to manage space usage between tiers (OST pools) of storage to allow configurations with burst buffers, and archival disks. Similarly, it is possible to migrate all objects off of an OST before it is replaced or removed from the file system.

Asynchronous file replication (mirroring)

Create multiple copies of a file within the same file system namespace (ala HDFS) after the file is initially written. This is useful for availability of important system files, without the need to add copies to all files in the filesystem. Real-time mirroring as the file is written would require additional coordination and recovery, or synchronous IO operations, and should be considered as a separate feature.

Complex file layouts

As application middleware, such as HDF5, become more common, Lustre should allow different file layouts tuned for the different file types. This would also allow the layout of a single file, or different parts of the file, to change as it grows or gains concurrent writers, to reduce overhead for small files, and increase bandwidth/concurrency for large files.

Dynamic layout for subset of a file

Data managed through an HSM, needs to move from slow data (tape) to fast (disk) during job execution. However, to speed access to critical data, it may be necessary to only restore part of a file. Any layout definition needs to support files split between different media types.

Storage pool quotas

Labeling heterogeneous OST classes is done via OST pools today. There needs to be a mechanism to control access and resource usage of OST pools. OST pool quotas would allow more flexible resource allocation than binary (allow/disallow) permissions like ACLs. A related issue is labeling subsets of OSTs for management and user convenience.

Unified storage target

It may be useful to remove the distinction between MDTs and OSTs and instead consider storage in terms of performance for different workloads. We then use the storage pool most appropriate for the workload. Small I/O's, or the beginning of a file, for example, could use storage tuned for this access pattern.

Performance Requirements

The requirements in this section highlight areas where Lustre performance could be improved. Deliverables that meet these requirements should provide immediate benefits.

Single client I/O performance

Single client performance can be CPU bound,. Although new multi-threaded RPC code has improved performance of a single-threaded reader/writer, there are still bottlenecks, such as the number of simultaneous RPCs in flight, internal lock contention, and SMP-unfriendly code, that prevent a single client from maximizing the performance available from the Lustre file system and its interconnect network. The client is currently also limited to a single metadata-modifying RPC in flight, which will also impact DNE MDT performance.

File create performance

Previous work to improve metadata performance on a single MDS have gained significant improvements to directory and device node creation, but the benefit to creating files with objects on the OSTs has been less clear. Thus, file creation performance (in particular, OST object precreation) remains an area where Lustre requires significant effort to meet user requirements (for example, see the Appendix). As with the MDS performance tuning, there are likely also SMP scaling bottlenecks in the OST code that can be addressed as part of a larger performance review.

Directory traversal and attribute retrieval

Directory listings performance has increased with recent metadata projects, but “ls -l” speeds for a single client should still be improved. Some possible areas for exploration are client-OST interactions or some version of a size-on-mds mechanism (e.g. synchronous recording of open-for-write, using the HSM “dirty” flag, or simplifications for single-client access).

Single shared file performance

Users desire single, shared files for certain applications, 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. WAN users should be able to avoid frequent health communication.

Small and medium I/O performance

Small and medium I/O requests (4KB – 256KB) are common in many HPC workloads and can be substantially lower on Lustre than on local file systems, even when the local file system is supported by disk arrays over a SAN environment as disk seek latencies dominate any network overheads. Lustre should be optimized for a variety of I/O request sizes including small and medium request sizes particularly in file per process workloads where distributed lock management is not a bottleneck.

LNET Requirements

As more compute systems use shared Lustre file systems, the robustness and configuration of the LNET layer will become more critical to successful file system deployments.

LNET channel bonding

LNET routers and servers are currently limited to a single channel provided by a single instance of the LND. This restriction limits bandwidth and reliability of an LNET connection to a single interconnect. LNET should allow multiple LNDs to be bonded as a group in order to enable load balancing and failover between LNET endpoints.

Improved LNET robustness

The original LNET design used multiple routers to guarantee connectivity, but performance suffers when there are large numbers of routers. This effect can be exasperated when there are multiple network levels as router transmit credits become depleted within a network. Furthermore, performance of a group of routers can suffer by one poorly behaving router. This investigation should consider mechanisms for improving LNET robustness and router performance.

Dynamic LNET configuration

LNET configuration currently uses static routes and requires LNET to restart to capture configuration changes. LNET needs to adopt a more IP-like configuration so that network changes can be more easily programmed and qualified.

IPv6

Support for IPv6 requires a change in the NID format to accommodate a 128-bit IPv6 address in the address-within-network field. Adding support for IPv6 will change the network protocol so the impact is more involved than just changing internal data structures. This affects all protocol levels: LNDs, LNET and Lustre. Lustre should support a phased approach to make the feature available before the site upgrades to IPv6.

Manageability and Administrative Requirements

Requirements in this section highlight areas of improvement for Lustre to reduce administrative burden or address shortcomings in its integration to the compute 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. Also, updated distros require new kernels to be supported. Lustre must reduce or eliminate its need to patch the kernel on the server, and should support recent kernel.org kernels.

Improved configuration robustness

Need to make file system configuration robust in the face failures, such as during OST additions to an existing file system. We should anticipate similar errors with DNE when we expand the number of MDTs in the namespace. Dynamic configuration using registration data from the MGS is one possible replacement for the current static configuration.
Other possible improvements are unifying the syntax between set_param and conf_param, and possibly replacing the old llog-based config file with a simple editable text file.

Administrative shutdown

Lustre needs a safe server shutdown to ensure that clients flush state to disk. This capability will be necessary for transparent server version upgrades, and to avoid potential loss of unwritten data in the case of known server shutdown.

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.

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.

Arbitrary OST assignment

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

Application Interface Requirements

The requirements in this section highlight areas of improvement for Lustre to reduce the overhead 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 by 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 usability as a library. 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.

POSIX extensions for file sets

Migration and replication could be facilitated if Lustre allowed operations on sets of files. GPFS has this today. If done right, the set will appear in the directory and quotas can be allocated on the set, not just the component files.

POSIX extensions for scalable opens

The Linux kernel recently added support for “open by handle” system call. Wiring this feature into Lustre would improve shared file (N:1) performance on large systems.

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 currently allows clients and servers to use different page sizes. It currently supports same-sized or larger 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 different 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 different byte orders. Incremental development and testing effort is needed to maintain this cross-endian functionality. Long-term, Lustre should support servers on either byte ordering.

Improved security infrastructure

Lustre will need to support Public Key Infrastructure (PKI) to provide secure authenticated access over WAN. Data Grids, such as Open Science Grid, TeraGrid (XSEDE) and others, natively use PKI/X.509 rather than Kerberos for security and user authentication. They have an established infrastructure and provide tools to manage certificates to user communities. Smaller communities conveniently can use these tools or self-signed certificates. PKI/X.509 support can be added following within the framework of the GSSAPI project at Indiana University.

Improved Lustre Tests

The Lustre tests included with each release need to be cleaned up. The following are areas that should be addressed to create a more robust, usable set of tests: refactor unused tests, ensure interoperability of tests between different Lustre versions, document requirements and coverage of existing tests, client failure shouldn't stop the tests, and increased code coverage.

Improved Lustre internals documentation

Lustre internals and architecture documentation is important for the dissemination of Lustre knowledge and encouraging more developers to contribute to the Lustre community. ORNL published the first internals documentation in 2009 for Lustre 1.6 (http://wiki.lustre.org/images/d/da/Understanding_Lustre_Filesystem_Internals.pdf). In addition, Sun provided Lustre Internals Documentaion (LID) through the lustre.org wiki (http://wiki.lustre.org/lid/index.html). Both of these documents are now out of date. We need

updated, well-documented internals for all Lustre components (MDS, MGS, OSS, OST, OSD API, etc.).

Appendices

2011 Required Rates and Capacities

The following section was prepared in 2011 and is included here for reference only. The table describes specific requirements for file system performance and scalability that the community thinks Lustre will need to accommodate HPC systems in the near term (2012) and beyond (2014). The table was ***not*** updated during the TWG's 2012 requirements review.

Metric	**Lustre 2.1**[1]	**Lustre 2.2**[2]	**Q2 2012**	**Q1 2014**
maximum number of files in file system	4 billion	4 billion	100 billion	1 trillion
maximum number of files in directory	10 million	10 million[3]	50 million	10 billion
maximum number of subdirectories	10 million	10 million	1 million	10 million
maximum number of clients	131072	128 thousand	64 thousand	128 thousand
maximum number of OSS nodes	-	-	1 thousand	4 thousand
maximum number of OSTs	8150	8150	2 thousand	8 thousand
maximum OST size	16 TB	128 TB	32 TB	128 TB
maximum file system size	64 PB	1 EB	100 PB	256 PB

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

[2] projected performance in 2011. This table has not been updated since 2.2 was released.

[3]The Lustre 2.2 ldiskfs code supports directories with over 10M entries, but as yet there is no support for e2fsck of such directories, so this feature is currently disabled by default.

Metric	Lustre 2.1[1]	Lustre 2.2[2]	Q2 2012	Q1 2014
maximum file size	320 TB	64 PB	1 PB	-
maximum object size	2 TB	16 TB	16 TB	64 TB
peak aggregate file creates/s	20 thousand	40 thousand	200 thousand	400 thousand
peak directory listings/s (ls -l, 4-stripe)	5 thousand	30 thousand	-	100 thousand
maximum single client open files	~3 thousand[4]		100 thousand	-
peak single client file creates/s	3 thousand	3 thousand	30 thousand	-

[4] "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."
[4]

Requirements funded in 2011

The following requirements appeared in the 2011 document and have been removed from the 2012 version because community funded development contracts during 2011/2012 have resulted in features that address the requirements. These new features began landing in Lustre 2.2 and will continue to appear through 2012 and 2013 with release of Lustre 2.3, 2.4 and 2.5. See the community roadmap page for more information.

Performance Requirements

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 multi-core 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.

Foundational Requirements

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.

Manageability and Administrative Requirements

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.

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.

2011 Deprecated Requirements

The following requirements appeared in the TWG's 2011 Requirements document, but have been removed from the 2012 document because they were either merged with other requirements or had been deemed unnecessary.

Merged features

The following two items were merged into new requirements for the storage management section of the current document.

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.

Deleted features

The following were deleted from the current list of requirements. Whamcloud and ORNL reported on using btrfs as a Lustre backend at LUG'11. Their conclusion was that the btrfs file system is not appropriate at this time for Lustre storage. In addition, Lustre clients can already restrict access to specific OSTs. The real issue is flushing client cache before server shutdown, which is addressed by a new requirement this year.

Backend storage investigation

To accommodate the capacities above, the backend store must expand beyond current ldiskfs limits (eg 128TB 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.

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.

New Requirements for 2012

The following is a brief list of new requirements gathered during our meetings in 2012.

Requirement Category	Feature
Availability	● Scalable fault management ● Avoid RPC timeouts
Storage Pool Management	● HSM and storage management infrastructure ● OST migration/rebalancing ● Asynchronous file mirroring ● Complex file layouts ● Dynamic layout for subset of a file ● Storage pool quotas ● Unified storage target
Performance	● Single client I/O performance ● File create performance ● Directory traversal and attribute retrieval ● Single file performance ● Small & medium I/O performance
LNET	● LNET Channel Bonding ● Improved LNET robustness ● Dynamic LNET configuration ● IPv6
Manageability and Administrative	● Administrative shutdown ● Improved configuration robustness
Application Interface	● POSIX extensions for file sets ● POSIX extensions for scalable opens
Other	● Improved security infrastructure ● Improved Lustre Tests ● Improved Lustre internals documentation

Change Log

date	initials	notes
4/11/2012	jc	initial draft integrating 2012 notes into 2011 template
4/11/2012	aed	add limits for 2.2 filesystems
4/19/2012	twg	group edit of document during TWG concall
5/3/2012	twg	add 2 new metadata performance requirements
5/30/2012	jc	add recommendation text
5/31/2012	twg	final draft following group review
6/7/2012	jc	board feedback: match order of gathered requirements to table