OpenSFS TWG Lustre Requirements

DRAFT: 3/17/2011

This document summarizes requirements for new Lustre features gathered from the community and presents recommendations for features which the OpenSFS Technical Working Group considers to be of immediate importance.

Recommendations to the OpenSFS Board

The TWG believes that the OpenSFS should pursue RFPs for both performance and foundational requirements. The TWG expects OpenSFS to pursue at least one requirement from each category. The order of requirements in each list represents the consensus of the TWG.

Near-term requirements

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

Foundational Requirements

1. support for alternate backend stores
2. scalable fault management
3. Backend storage investigation

Gathered Requirements

Performance and Capacity Requirements

This section describes specific requirements that the community thinks Lustre will need to accommodate HPC systems in the near term (2012) and longer term beyond the near term release (2014).

Metric	**Q2 2012**	**Q1 2014**
# files in file system	100 billion	1 trillion
# files in directory	50 million	10 billion
max subdirectories		? ?
# clients	64 thousand	128 thousand
# OSS nodes	1 thousand	10 thousand
# OSTs	1 thousand	10 thousand
OST size	32 TB	64 TB
file system capacity	100 PB	256 PB
file size	1 PB	?
aggregate file creates/s	200 thousand	400 thousand
stats per second/s	?	?
directory listings/s (ls -l)	?	100 thousand

open files per process		100 thousand
single file creates/s	?	30 thousand

Performance Requirements

The requirements in this section address limitations to Lustre which the TWG believe 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 files for their sanity, 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.

Foundational Requirements

The following describe 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 stores

Ldiskfs is at the limits of its useful life. Selection of an alternate backend store is impossible unless Lustre supports interchangeable backends. Sun 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.

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.

These are some of the reasons that CFS/Sun and LLNL have pursued ZFS as a backing store. Despite efforts from the Lustre community to integrate ZFS with Lustre, ZFS is not a commercially viable solution because the CDDL license is incompatible with the GPL in regards to distribution. Btrfs has many features in common with ZFS, but is available within Linux under GPL.

Assuming that Lustre can be restructured to accommodate alternate backend file systems, 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.

Varying page-sizes

Lustre must allow clients and servers to use different page sizes. (Expect smaller on client, larger on server.)

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.

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.

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.

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.

Assured disk i/o

Lustre should experience zero down-time while checking the file system. Failover is a non-starter if the the file system is offline performing an fsck. Both Lustre and its backing store should have online checking and scrubbing.

Dynamic OST addition/deletion

Lustre should allow dynamic OST addition and deletion. This is possible in experimental file systems like Ceph, which uses the CRUSH algorithm to determine striping. For Lustre, we need to consider the locality of the new OSTs while enabling dynamic striping.

Replicas & snapshots

Replicas and snapshots will be needed to do Lustre backups. The requirement is for consistent use of file system at a point in time.

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

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.

Lustre should explore alternatives to POSIX access methods that can be used to support exascale file sytem requirements.

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.

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.

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.

Incomplete Requirements

The following were in Dave's original notes but are lacking supporting text for inclusion above. REMOVE THESE FROM FINAL DRAFT.

[[adaptive object layout, especially as file size grows, mutable layouts]]

Fujitsu

[[support for 1024 OSSes?]]
[[max subdirectories requirement?]]
[[continuted support for big-endian clients, perhaps servers?]]
[[support larger page sizes/mismatched page sizes between clients and servers]]
[[advisory lock support? MPI-IO requirement?]]
[[full OST specification for shared file?]]
[[maximum file size? 1 PB]]
[[Snapshotting the filesystems]]

[[There is a question as to what's driving 10 billion files in a directory? How does one expect ls to work in that directory? Is that expected to be POSIX compliant?]]
[[open files per client instead of open files per process? or is that total open files per system or job? We've already passed that it seems...]]

[[Eric notes that find has a similar set of requirements as ls]]
[[Improved small file IO performance -- need to quantify]]

[[Replication to acheive fault tolerance was mentioned --
requirement vs implementation detail?]]

[[Covered Linux AIO support (needed for more useful O_DIRECT, in turn needed for lockless IO) and ability to use fadvice() to express access patterns, though these are implementations.

Need to figure out requirements underlying collective open and ADIO and MPI-IO improvements.

Would like to motivate readdir+/statlite/collective open from POSIX HECEWG, but need requirements for those; perhaps they come in through the metadata performance section.

[[full OST specification for shared file?]]
]]

[[
how to handle integrity check requirements, split online/offline?
-- leave to HW vendors for low level
-- handle Lustre issues here, and add lowlevel pieces as well

Improved analysis and visualization tools?
]]

[[
Improved security
-- user name mapping
-- propogating authorization from MDT to OST through
trusted channels rather than untrusted clients
]]