Sun microsystems

Lustre™ Scalability

An Oak Ridge National Laboratory/
Lustre Center of Excellence White Paper
March 2009

lustre®

Table of Contents

Executive Summary

The computational performance of High Performance Computing (HPC) systems is rapidly increasing. Each year, the performance of supercomputing sites in the Top500 list doubles. At this rate, we can expect the deployment of production-level exaflop systems within the next 10 years. This paper discusses this trend and the challenges it presents to delivering high-performance I/O for HPC systems. In particular, this paper identifies scalability issues in the Lustre file system that must be addressed to successfully support exascale systems, and it presents several approaches to address these issues.

HPC Trends

Steadily increasing demands for computing capability in the HPC community have been met by creating systems with ever larger numbers of processing elements. Today's high-end computer systems consist of clusters with thousands or tens of thousands of nodes. These clusters have successfully met the demand for computing capacity that has outpaced the growth of single processor performance.

While the growth in processor performance has lagged the growth in compute demand, the growth in I/O performance of disk drives has lagged far behind improvements in processor performance. To provide the I/O bandwidth required for large HPC centers, storage systems with very large numbers of disk drives are being deployed. As processor performance improvements continue to exceed disk drive bandwidth improvements, the ratio of the number of disk drives to compute nodes will continue to grow. With this growth, the cost of drives and other storage system components will become an increasing percentage of system cost.

Increasingly, HPC customers want to implement solutions that provide shared access to a common pool of storage for all systems in a data center, rather than directly attach storage to individual systems. This design enables better utilization of the storage system and makes it easier to move users and workload from one system to another. Centralizing a storage solution in this way greatly increases the importance of a high-performance, reliable storage system. When the shared storage system is degraded or down, the entire site is affected.

Very large component counts[1] in current and planned HPC systems also create issues for the storage system. A recent study[2] showed average annual replacement rates of 3% for disk drives in large HPC sites and a minimum processor failure rate of .1 failures per processor per year. Applying these results to a site with 25,000 drives and 50,000 processor sockets, one could expect 2 drive failures and 13 processor failures per day.

Large HPC sites must continue to operate in the presence of component failures. Applications running on these systems write checkpoint files at frequent intervals (every 1 to 2 hours), so they can restart from a known, recent state if they are interrupted. These checkpoint files have become a major portion of the I/O workload.

As these trends continue, the following issues have become more significant:

- I/O performance is becoming the key constraint on overall system performance. The storage system has to effectively aggregate I/O to and from tens or hundreds of thousands of compute nodes and disk drives, as well as enable administrators to manage them effectively.
- If I/O bandwidth is the limiting resource in system performance, then mechanisms must be provided to allocate this resource among users and jobs (much as CPU cycles are allocated today).
- The storage system must function in the presence of constant component failures and nearly continuous drive reconstructions.
- The storage system must enable the effective administration of configurations containing trillions of files and hundreds of thousands of drives.

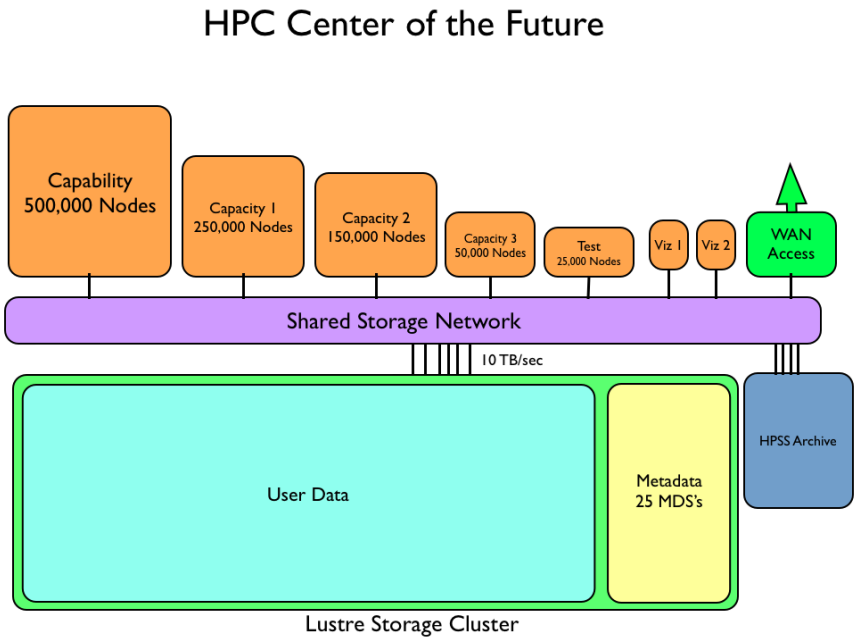
Lustre File System

Today, Lustre is the leading clustered file system in the HPC market. Lustre effectively scales to support systems with tens of thousands of compute nodes. As HPC systems increase node counts to increase overall performance, Lustre is challenged to scale even further. To prepare for this challenge, this paper identifies the obstacles to scale Lustre to support 1,000,000 application nodes and high-level approaches to overcoming these obstacles. It also discusses key issues in the areas of Lustre architecture, operation, performance improvements and RAS.

1 A “very large component count” is tens of thousands of compute nodes and disk drives, and associated components, such as routers and cables.
2 *Understanding Failures in Petascale Computers*, Bianca Schroeder, Garth A. Gibson, SciDAC 2007 Journal of Physics: Conference Series 78 (2007) 012022.

HPC Center of the Future

Scaling a storage system to support 1,000,000 or more application nodes raises issues beyond simply enabling this number of nodes. To place this goal in context, this section describes the high-level configuration of a hypothetical HPC center of the future.



This hypothetical HPC site will have several large computing clusters that share access to a single Lustre storage cluster. In this example, the systems are:

- **Capability** - A 500,000 node compute cluster used for Grand Challenge-type problems.
- **Capacity** - Three systems used for throughput workloads:
  - Capacity 1 - 250,000 nodes
  - Capacity 2 - 150,000 nodes
  - Capacity 3 - 50,000 nodes
- **Test** - A 25,000 node system used as a development and test platform.
- **Viz 1** and **Viz 2** - Two visualization clusters used to drive very large display devices.

All of these systems connect to a large shared Lustre Storage Cluster through a shared high-performance network. The network is attached to a tape archive used to back up the Lustre Storage System. It is also connected to external networks (WAN, Internet, etc.).

The Shared Storage System is a Lustre Storage Cluster consisting of tens of thousands of disk drives with a storage capacity of hundreds of petabytes. The drives are attached to hundreds of Object Storage Servers (OSSs). The metadata is housed on a cluster of Metadata Servers (MDSs). The storage system provides aggregate bandwidth of 10 TB/sec (sustained over a 10-minute interval).

Connecting these different systems to shared storage enables data sharing between the systems, makes it easier for users to move their applications from system to system, and simplifies the addition or removal of large systems to the HPC center because data management is decoupled from the compute systems.

Architectural Improvements to Lustre Scalability

This section identifies the architectural and functional enhancements needed to improve Lustre scalability to the point where it can support 1,000,000 clients. Currently, there are two main architectural limits to Lustre's scalability.

The first limitation is that only one MDS can be active in a Lustre file system at a given time. This means that metadata operations can be processed only as quickly as a single server can manage. To date, this has not been a serious limitation, and it has been addressed by selecting an MDS that is capable of handling the required load. However, as a Lustre system scales, a single MDS becomes a bottleneck. This issue is being addressed by the addition of Clustered Metadata Server (CMD) functionality to Lustre.

The second limitation is the flat nature of the current Lustre communications model. In a Lustre cluster, all Lustre clients must communicate with all Lustre servers. This many-to-many communication pattern reduces scalability. Adding a hierarchy to Lustre communications addresses this concern. There are two independent approaches to creating such a hierarchy, proxy servers and I/O forwarding.

CMD Functionality

With CMD functionality, a collection of metadata servers manages the file system's metadata operations. Each MDS manages a set of directories in the file system. A hash function, based on a directory's name, determines which MDS will manage the directory and its contents. The MDSs are connected to one another in failover pairs to deal with MDS failures.

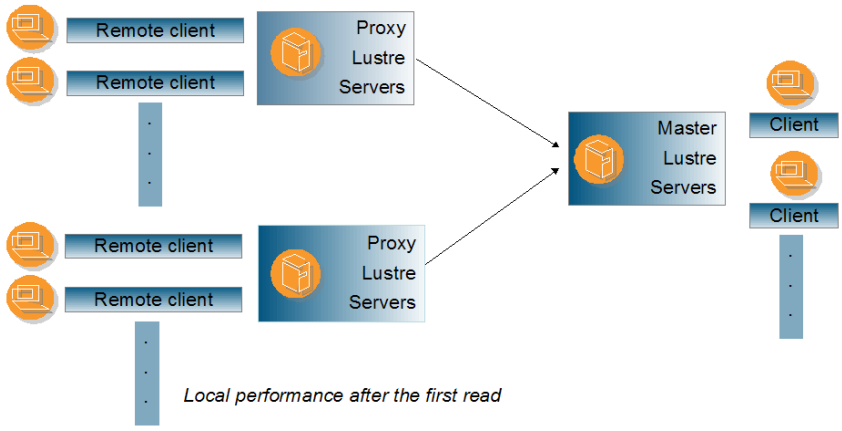
While there is some overhead in CMD compared to a single, active metadata server, the majority of metadata operations are still performed on a single MDS, so performance is the same as a system with a single MDS. Operations involving more than one server can occur in parallel.

Communications Hierarchy

Lustre's many-to-many communication pattern reduces scalability. Adding a hierarchy, such as proxy servers or I/O forwarding, to Lustre communications addresses this concern. Both of these mechanisms effectively reduce the number of clients visible to the Lustre file system, and improve the opportunity to consolidate requests.

Proxy Servers

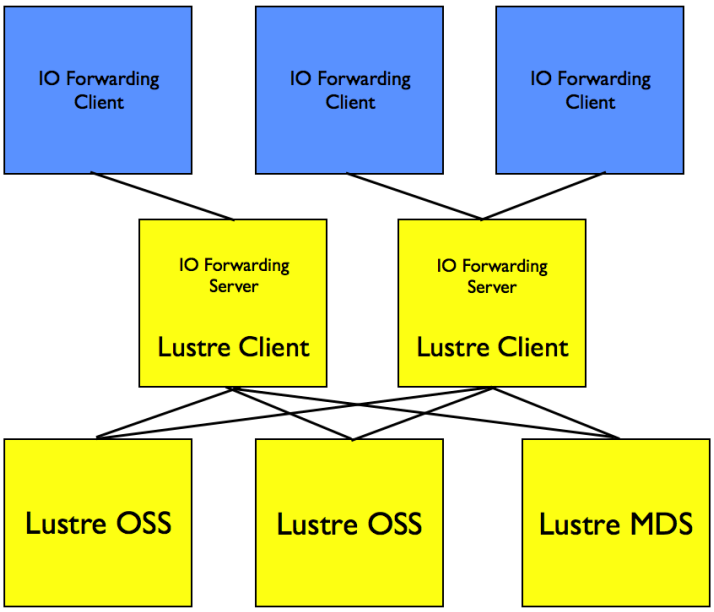
Proxy servers are a layer of servers between the OSSs and the Lustre clients. The proxy servers appear as a Lustre file system to the clients, and appear as Lustre clients to the parent servers. The proxy servers cache data and aggregate I/O and lock requests. This mechanism can reduce packet handling on the network by replacing hundreds of individual requests with a single aggregated request.



I/O Forwarding

I/O forwarding mechanisms are used in some clusters to forward I/O requests from application nodes to dedicated I/O nodes. IBM's Blue Gene system uses I/O forwarding, and it is also available on some Cray systems. In a Lustre system using I/O forwarding, the Lustre client is only installed on the dedicated I/O nodes. This significantly reduces the number of Lustre clients to be serviced. The I/O forwarders enable the Lustre clients to which they connect to aggregate I/O from multiple application nodes. The I/O forwarders also reduce the number of Lustre clients in a system and the amount of state information that the Lustre servers need to track. (For example, IBM uses a ratio of 64 application nodes to 1 I/O node; the reduction in the number of Lustre clients is substantial.)

These I/O forwarding mechanisms will need to be modified to better integrate with future Lustre features. For example, when ZFS support is added to Lustre, ZFS checksumming capability will need to be added to ensure end-to-end data integrity. The Lustre team may partner with other vendors to introduce this functionality to the I/O forwarding software or, if the technique is deemed valuable enough, we may develop I/O forwarding capability specific to Lustre. Additionally, liblustre could be adapted to include this functionality.



Proxy servers and I/O forwarding functionality can be used independently or in combination. For example, the 1,000,000 nodes in the HPC center of the future could communicate with 20,000 Lustre proxy servers through I/O forwarders.

This would enable the parent servers in the Lustre file system to see and service requests from 20,000 clients.

Resource Management and Performance Enhancements

In addition to the architectural features discussed above, Lustre scalability will be enhanced by performance improvements to existing areas of the system, and by adding functionality to manage and tune system performance.

SMP Scalability

System-wide Lustre scalability can be enhanced by improving the efficiency and scalability of individual Lustre servers. Scalability improvements are being achieved with the introduction of improved functionality. Two examples are:

- **Finer granularity locks**. Currently in Lustre, LNET has one global lock. LNET operations that require locking are, therefore, essentially single threaded. The SMP performance of LNET will be improved by replacing a single global lock with multiple locks, to enable greater parallelism and scalability of LNET operations.
- **CPU affinity**. At the LND level, keeping the processing of given peers on particular CPU cores has resulted in substantial performance improvements. This improvement could be extended into a generic facility, so all levels of the software do this identically.

Metadata Writeback Cache

Metadata writeback cache is a mechanism to cache metadata operations in a manner similar to existing caches for data operations, such as client data cache. It enables a client to cache metadata operations to a server (instead of sending them immediately), simulate their local effects, and later send a batched description of the cached operations to the server for execution. Writeback cache offers these advantages:

- Improved network efficiency because of the batched transfer of metadata operations.
- Decreased workload and more efficient operations execution on the server.
- Greater concurrency on the client.
- Client operation without communication to/from the MDS. (Note: This requires a suitable locking mechanism, such as sub-tree locking.)

Network Request Scheduler

The Network Request Scheduler (NRS) manages incoming RPC requests on a Lustre server, providing improved and more consistent performance. The NRS re-orders request execution to avoid client starvation and to present a workload to the backend file system that can be optimized more easily. This is analogous to the disk elevator feature in the Linux kernel which allows I/O requests to sit in a queue, so the I/O scheduler can coalesce requests and improve throughput. Unlike the disk elevator, NRS does not require bulk data buffers to be allocated in memory to schedule the I/O, so it has lower memory requirements.

NRS implements an allocation policy to give clients a fair share of the servers' resources. Extending NRS to enforce a global allocation policy is one way to implement Quality of Service (QoS). Potentially, NRS could be extended to throttle the request load from clients, thereby avoiding server overload and degradations in responsiveness and performance.

Channel Bonding

Lustre channel bonding combines multiple, physical Network Interfaces (NIs) into a single, logical NI. The bonded NIs do not need to be the same type; for example, Ethernet and InfiniBand networks can be bonded together. Channel bonding can be used in two modes. In standby mode, one NI is active; communication fails over to a spare NI if the active NI fails. In load-balancing mode, communication is distributed across all of the bonded NIs, according to the load-balancing policy.

RAS

In a file system of the size being discussed, with hundreds of thousands of disk drives and thousands of servers, some components will be in a degraded or failed state at any given time. For example, a system with 200,000 drives could expect replacement of 2,000 or more drives per year. Studies have shown that at large HPC sites, annual replacement rates of drives increase over time and can approach 5% a year after several years. Meeting this challenge with an effective Reliability, Availability and Serviceability (RAS) strategy is a vital part of system scalability.

Virtual Health Network

Today, Lustre server failures are detected by RPC timeouts. The RPC timeout period tends to be long, to avoid confusing network congestion with server failure. Long timeout periods can result in large delays in detecting and recovering from failures.

To address this issue, investigation is underway to add a virtual health network to LNET that provides a specific mechanism to monitor the storage cluster's health. Traffic on the health network would have a higher priority than other LNET traffic, and would not be delayed by congested “normal” communication. This network would enable the prompt detection of server or client failures and the rapid removal of communication associated with a failed client or server from the network, as well as prompt notification to clients about recovered servers. This mechanism should improve failover/recovery times from tens of minutes (or longer) to seconds. One enhancement under consideration is adding network priority levels to improve lock conflict resolution.

Rebuild Performance

When disk failures occur, it is important that recovery (the rebuild of the RAID group) occur as quickly as possible. This reduces the risk of a second drive failure occurring in the same group of disks while the rebuild is in progress. The other important aspect of disk rebuilding is that the process minimally impact the normal operation of the disk group. Both of these concerns are being addressed by the addition of declustered parity and distributed sparing functionality to Lustre. This functionality will spread the rebuild load across multiple disks.

ZFS

Lustre's current node file system, ext3, has several limitations. It is limited to an 8 TB maximum file system size and it offers no guarantee of data integrity.

To improve the reliability and resilience of the underlying file system on the OSS and MDS components, Lustre will add ZFS support. Lustre supporting ZFS will offer a number of advantages, such as improved data integrity with transaction-based, copy-on-write operations and end-to-end checksumming on every block. Copy-on-write means that ZFS never overwrites existing data. Changed information is written to a new block, and the block pointer to in-use data is only moved after the write transaction is completed. This mechanism is used all the way up to the file system block structure at the top block. To avoid data corruption, ZFS performs end-to-end checksumming. The checksum is not stored with the data block, but rather in the pointer to the block.

All checksums are done in server memory, so errors not caught by other file systems are detected in ZFS, such as:

- Phantom writes, where the write is dropped on the floor.
- Misdirected reads or writes, where the disk accesses the wrong block.
- DMA parity errors between the array and server memory (or from the driver), since the checksum validates data inside the array.
- Driver errors, where data winds up in the wrong buffer inside the kernel.
- Accidental overwrites, such as swapping to a live file system, are all detected by ZFS.

In Lustre, data checksumming will be done by the Lustre client on the application node. This will detect any data corruption introduced into the network between the application node and the disk drive in the Lustre storage system. In testing, Lustre data checksumming has detected previously unknown problems in network cards. These cards silently introduced data corruption that went undetected without Lustre data checksumming.

An implementation note: ZFS support is being developed and tested with a user space implementation of the ZFS DMU. In the future, the DMU will run in kernel space. Also, the Lustre DMU code is almost entirely common with the Solaris version of ZFS, so Lustre support for ZFS will closely parallel the Solaris release of ZFS.

Lustre support of ZFS will offer several specific advantages:

- **ZFS is self-healing** - In a mirrored or RAID configuration, ZFS not only detects data corruption, but it automatically corrects the bad data.
- **Improved administration** - Because ZFS detects and reports data corruption on all read and write errors at the block level, it is easier for system administrators to quickly identify which hardware components are corrupting data.
- **SSD support** - ZFS supports the addition of high-speed I/O devices, such as SSDs, to the storage pool. The Read Cache Pool or L2ARC acts as a cache layer between memory and the disk. This support can substantially improve the performance of random read operations.
- **Scalability** - ZFS is a 128-bit file system. This means that current restrictions on maximum size file systems for a single MDS or OST, maximum stripe size, maximum number of files, fixed number of inodes in the MDS file system, and maximum size of a single file will be removed. ZFS support will also remove the current 8 TB limitation on LUNs.

RAS Database

Lustre's RAS strategy needs more than a reliable, redundant, scalable architecture; it requires a software component to provide status, diagnostics and an overall view of the system. The software component under consideration is a RAS Database (RASd).

Primarily, this database would contain real-time information regarding the cluster's status, node status (up/down, available to be scheduled, etc.), physical locations of nodes (rack number, coordinate location, chassis number, slot number, etc.), inventory of support hardware (InfiniBand switches, PDUs, etc.), etc. Ideally, the RAS database would contain as much information about software status (e.g., process states, job states, operating system health, etc.) as about hardware status (e.g., memory errors, fan speed, CPU temperatures, etc.). This database would use as much automation as possible to populate its initial information. While the system is in use, RASd information will be updated by a mechanism that impacts the system as little as possible.

Much like a hardware RAS solution is incomplete without a software counterpart, the RAS database is incomplete without tools to use the information it contains. These tools can monitor information in the database, look for events, or even implement, over time, ways to data mine information and help predict errors before they occur. The database could also be used to manipulate the node's states; for example, send reboot or power cycle signals to the hardware in question. Additionally, the database could be used as a way to provide node availability or suggestions to the job scheduler used on the system, to better increase overall system use.

Arguably, the most important piece of the RAS software solution is its redundancy and availability. Access to the RASd information and its tools will need to be granted securely to any service node in the system, not just for ease of use, but also to provide redundant access in case of problems. In addition, the database itself will need to be replicated and mirrored, with updates made simultaneously to all clones (so the state is consistent between them). This provides confidence that at any time the system could switch between these clones and continue to function. Lastly, a good backup and restore method needs to be available to allow for a quicker recovery if an unforeseen, catastrophic event occurs.

The proposed RASd system will be based on a MySQL database. The RASd database will capture the following types of information:

- **Machine topology** – Basic node information including node type, hostname, IP address and physical location.
- **System inventory** – Information on nodes, switches, PDUs and any other support hardware, including associated IP addresses.
- **System state** – Availability of nodes in the system (up/down, running, etc.).

The RASd database is not limited to these types of data; it can be populated with even more useful information.

Operational Issues

This section identifies issues with the administration and operation of an extremely large-scale Lustre system.

Lustre HSM

Lustre HSM, a collaborative project between CEA and the Sun Lustre team, will provide an interface between Lustre and the hierarchical storage system. HPSS will be the initial system supported by Lustre HSM, with others, such as SAM/QFS to follow.

Tiered Storage

The combination of Lustre HSM's ability to connect a Lustre file system to an archive system such as HPSS, and ZFS's ability to manage SSD storage as cache, provides a basic capability for system administrators to map Lustre to tiered storage. Effective management of a tiered storage system with Lustre will require the implementation of a policy manager that enables the system administrator to specify policies, such as which directories or systems to map to which class of storage device. A Policy Manager will be implemented as part of the HSM project, and it is likely that this will form the basis of future Lustre policy-based file system management.