Cluster File Systems, Inc.
530 Showers Drive # 7 – 147
Mountain View, CA 94040

Phone	**650 799 8578**
Fax	**403 678 6922**
Email	**braam@clusterfilesystem.com**
WWW	**http://www.clusterfilesystem.com**

Lustre Technical Project Summary
(Attachment A to RFP B514193 Response)

Authors: Peter J. Braam, Cluster File Systems
and
Rumi Zahir, Intel Labs

Date: Version 2, July 29, 2001

Table of Contents

1 Background

The Tri-Labs/NSA (Los Alamos, Livermore, Sandia, DOE and DOD) are requesting a proposal for a scalable global secure file system (SGS file system) in the context of an Advanced Strategic Computing Initiative (ASCI) "Pathforward" grant [1]. Cluster File Systems, Inc is responding to this RFP [2]. Focus of this project is the development of a highly scalable file system, named Lustre, which will be evolved from the current Lustre open-source object-based file system [3]. Scalability spans many dimensions, including data and metadata performance, large numbers of clients and inter-site file access, management and security. All these dimensions are of importance to the Tri-Labs/NSA sites and global enterprises alike.

Object-based storage continues a long history of increasing the level of abstraction of storage devices as seen by operating systems. Whereas early system software needed to be aware of arm positions and other disk internals SCSI and ATA disk access protocols abstracted disk drive geometry away, and exposed disks as an abstraction of a linear sequence of fixed size. Object storage concepts date as far back as the 1980 "Universal File Server" paper [36] by Birrel and Needham. Kaashoek started an interesting project at MIT confirming some of the benefits [2], and the first large scale implementation was pioneered at Carnegie Mellon University and executed as part of the NSIC/NASD project [22], [46]. NASD focused on abstracting block allocation and providing system software with an object-based storage abstraction. The NASD architecture enabled scalable I/O bandwidth through third party transfer, and primarily focused on secure access to storage devices. NASD explored object-based file systems (ERDFS) but it "interact[ed] minimally with its host operating system" [22], and its API did not provide explicit support for file system recovery or clustering. One of the most important outcomes of the NASD work was a SCSI based OSD object command set proposal that is currently under consideration by the ANSI T10 standards committee [47].

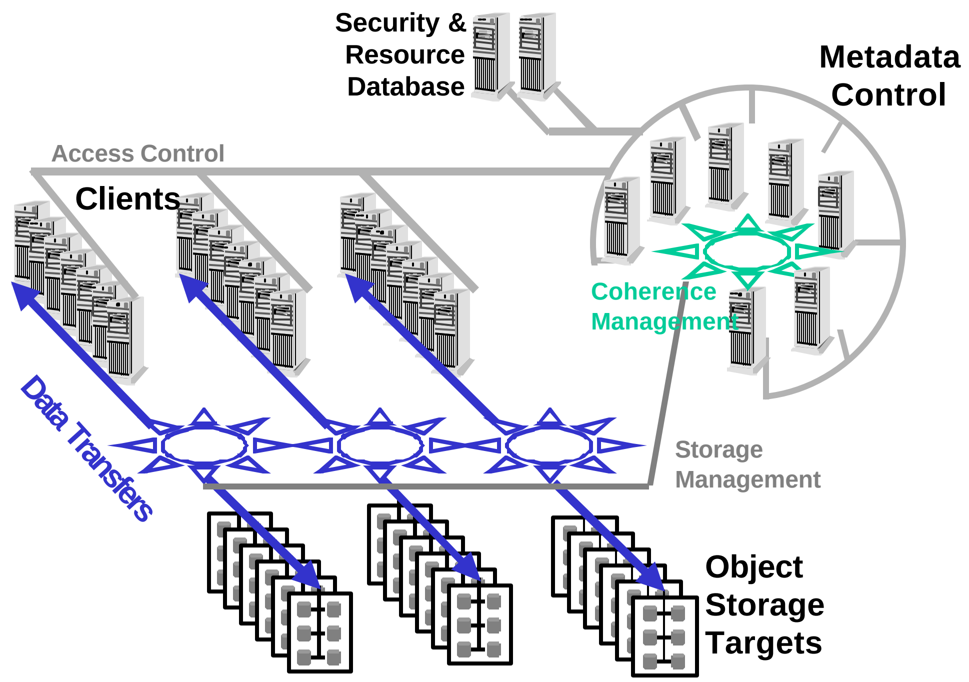
The original Lustre project [5,6] in 1999 also originated from CMU. It initially sought to build an object-based file system, with cluster-wide Unix semantics. This file system design has evolved to become the core of this proposal. Lustre object storage categorized the device drivers that can build up an object storage stack. Direct drivers, clients / target pairs and logical drivers underlie storage object applications such as file systems or object databases. Lustre did not tie its command set to SCSI and added features to support advanced file systems such as parallel I/O abstractions, object pre-allocation, locks and hooks for journaling to provide faster file system recovery. Lustre allows protocol modules to be loaded into and executed by the storage device. A prototype open-source Linux implementation of Lustre is available at [3], and currently runs under Linux 2.4.

Since January 2000, the Lustre development efforts have been heavily influenced by scalable cluster file system requirements outlined by the National Labs in the ASCI I/O SGPFS [23] and the more recent Tri-Labs/NSA SGPFS requirements document [1]. In response to the Tri-Labs/NSA RFI [1], Braam submitted a design document [4] that outlines how the Lustre architecture can be evolved to meet the Tri-Labs/NSA's needs.

2 Lustre Structural Overview and Rationale

This section describes the overall Lustre architecture [4,6], and enumerates various capabilities and interfaces of each of the components and gives a brief rationale for our approach.

2.1 Structural Overview



As shown in the picture above a Lustre-based cluster consists of three types of systems:

1. *Clients*: Applications running on clients see a shared file system with standard POSIX file system semantics. The file system is built up of filesets and provides a global namespace. Specialized applications can bypass the file system and may directly access objects stored in the cluster.
2. *Metadata Control Systems*: manage name space and file system meta-data coherence, security, cluster recovery, and coordinate storage management functions. *Metadata Control* systems require direct access to storage for meta-data, i.e. file system and object attributes as well as directory contents. *Metadata Control* systems do not handle file data, file allocation data and file locking semantics. Instead, they direct clients to do file I/O directly and securely with storage targets. The metadata cluster is free of single points of failure.

3. *Storage Targets*: provide persistent storage for objects. Objects can represent files, stripes or extents of files or serve other purposes. Files are represented by container objects in the metadata cluster and by constituent objects on storage targets. Rich interfaces to perform I/O are provided including block allocation, locking, parallel I/O, storage networking optimizations and storage management as well as active disk interactions with loadable logical storage modules. Storage targets can be made free of single points of failure.
4. *Resource and security databases:* provide configuration management information to systems, provide security services, user and group databases and file set location databases. Inter-site referrals provide key mechanisms for global infrastructure. The redundancy of these systems is provided by shared storage fail-over solutions.

The protocols among the systems can be summarized as follows:

1. **Clients – Storage Targets:** Clients interact on a client/server basis with storage targets directly for file I/O. Clients can exploit specialized parallel I/O interfaces or benefit from storage management modules running in the storage target (e.g. data migration, network adaptation, data mining). Locking of file data is managed with storage target based lock service supplemented by revocation services on the client. Storage targets accept security capabilities from clients. Client and target failures invoke recovery protocols among these systems, which include I/O fencing, journal recovery and lock revocation/re-establishment.
2. **Clients – Metadata Control**: Changes to the namespaces are requested by clients and directed to metadata control. The file system protocol is supplemented with resource location services, lock services for metadata (including revocation services offered by the clients). The client/metadata file protocol dynamically adapts to cover low contention and high contention cases. Aggressive write-back caching is used in case of low contention, while a scalable client/server model is used when contention is high. In case of high contention, resource management distributes the load across the metadata cluster. Implicit in this protocol is the allocation of storage target resources to objects, which is communicated to clients. When clients die, a simple recovery protocol is followed similar to that between clients and storage targets. Changes in the membership of the metadata control cluster first provoke recovery of that system and then recover clients and storage targets.
3. **Storage Targets – Metadata Control:** The protocol is a client/server protocol enabling storage targets (clients in this protocol) to update metadata control (servers) with information regarding constituent attributes and summary information on target load and capacity resources. There is a recovery protocol to re-establish distributed consistency among containers and constituent objects.
4. **Metadata Control – Metadata Control:** The communication in this system is much like that of a VAX Cluster, including a tightly coupled metadata file system, cluster transition and distributed lock management functionality.
5. **Client – Client:** There are no direct interactions except for client and storage target clusters performing hierarchical flood-fill notifications from metadata

control, resource databases (or storage targets) to all systems. Flood/fill notification services can be made redundant by virtual service techniques because they are stateless and memory based.

6. **Storage Target – Storage Target:** Storage targets communicate with each other as part of storage management for object migrations directly between storage targets. For file system operation these nodes communicate among each other like client – client interaction.
7. **Resource and security services:** after new (racks of) systems have been added to the configuration databases they are self-configuring, using DHCP, LDAP and related systems. Security, user/group and file set databases are queried accessed by clients, metadata control and storage target nodes for a variety of services. In some cases information is cached.

The Lustre object-based protocol permits more than simple stacking of object-protocol modules. It also allows various cluster file system functions can be partitioned across multiple systems in different ways. The following client, metadata control and target configurations are possible:

- *Single system*: {Client = Metadata Control = Target} – All functions execute on a single system. In this case, Lustre behaves like a local file system, but can add features such as snapshots, DMAPI or encryption through loadable modules.
- *Shared object storage file system*: {Clients incorporate Metadata Control, Targets} – A symmetrical object-based cluster file system that performs metadata control functions between clients and shares storage on targets.
- *File manager object file system*: {Clients, Single Metadata Control, Targets} – Multiple clients that manage coherence through a *single* metadata control systems. This is an object-based cluster file system with a file manager.
- *Client-server distributed file system*: {Clients, Metadata Control with direct-attached Targets} – This is a client server network file system configuration.
  Lustre: {Clients, Metadata Control Systems, Targets} – Multiple clients that manage coherence through multiple metadata control systems, and that manage access to multiple targets

2.2 Rationale and Alternatives

A traditional cluster file system has aimed to provide high performance Unix file sharing semantics in a tightly connected cluster. Distributed file systems (such as NFS, SMB and AFS) have provided file service to larger groups of clients. Newer file systems have addressed object storage, and others (InterMezzo [20]) have introduced extremely aggressive write back caching techniques suitable for wide area operations. Our solution will draw on innovations from many such systems.

The requirements posed by the SGS File System emphasize all dimension of scalability. The brief rationale here provides motivation for our solutions and mentions some alternatives.

First, much of the current design was arrived at in joint work with Sandia spanning several years, based on the I/O subsystem used in CPLANT. Secondly, we have tried to draw on successes in the industry: VAX Clusters, AFS scalability, Kerberos security, and journal file systems like ReiserFS and XFS. Finally we have wanted to build a system that is closely aligned with core developments in the storage industry such as NFS v4, DAFS-style storage networking and commodity hardware infrastructure.

Object-based I/O is a natural way to offload a sub-protocol of file service to storage targets. The NASD project has demonstrated the opportunities for scalable I/O and security. Our work is an enhancement of this providing more services and storage management in the object stack. An alternative approach is to allow clients to manage allocations on traditional storage controllers. While the latter provides more backward-compatibility, we regard the ASCI setting as an opportunity to innovate.

A **metadata control cluster** is, perhaps, the most tentative of our choices. The primary motivation is that VAX-Clusters were very successful systems. Hashed directories will provide load balancing for single object updates. The reductions of our system that occur by co-locating metadata control, client and target components lead to a number of known good solutions. Alternatives in this space are scarce. It has been suggested to run a large redundant SQL database server as the metadata service in Lustre, and idea that leads to many secondary opportunities such as name space indexing. We expect to do further research on each of these two alternatives.

3 Related Technologies

To achieve wide spread adoption and portability the Lustre project will need to integrate a variety of complex technology components such as file systems, networking, clustering and storage target execution environments. We recognize that in each of these domains significant technological evolution is ongoing, and we are actively participating in numerous industry standards activities [15, 16, 21].

This section outlines how we expect to leverage many of the needed software components for Lustre from existing open standards and open source efforts.

1. *NFS v4*: Lustre clients bear some resemblance to NFS v4 clients [8]. However, Lustre performs much more aggressive client side caching of data and meta-data and will use a directory format that enables extensible hashing across a cluster. As a result, Lustre clients aggregate I/O commands to a much greater extent than NFS v4 clients, which improves performance. Our clients do not participate as full cluster nodes but as satellites and we expect the Lustre client implementation to be similar to an NFS v4 client.

2. *Clustering*: Several groups have contributed significant clustering infrastructure components in to the open source. We expect Lustre's clustering infrastructure to integrate components from IBM's open source distributed lock manager [26], Cornell University's Ensemble group membership protocols [27], and Mission

Critical Linux' Kimberlite cluster recovery daemon [28].

3. *Cluster File Systems*: The Lustre metadata control systems are responsible for managing object meta-data coherency. Within the metadata control systems, we expect to use a novel locking mechanism based on hashed directories with sub-dividable extensible hashes. Some aspects, such as cluster transitions will be handled as in traditional block-based cluster file system similar to GFS [25], GPFS [24], or the VAX cluster file system [29,30], but we will use object storage and will replace inode data with object metadata.

4. *File System Scalability*: The current Lustre code is based on the standard Linux ext2 file system. We definitely need a journaling file system and we expect that for file I/O the Silicon Graphic's XFS file system [31] is most scalable. We will incorporate one or more of XFS, ReiserFS, JFS, Ext3 into our storage targets.

5. *Storage Networking*: We expect to incorporate several recent low-overhead networking advancements in to the Lustre design. Remote DMA (RDMA) [10] will significantly reduce network protocol processing overheads on clients and targets on VI-architecture based networks. The WARP protocol [11] will also enable RDMA writes over standard TCP/IP networks. The DAFS initiative [15] has introduced an improved RPC data layout. Combined with WARP-like RDMA writes, the improved DAFS RPC data layout is a very attractive solution for low-overhead Lustre client-target communication, even over TCP/IP.

6. *Object-Storage Protocols*: The Lustre object storage protocol bears many resemblances to currently evolving storage standards. Compared to the DAFS file system storage access protocol [15], Lustre eliminates the need for server-side name space handling, and provides several extensions over DAFS. While iSCSI [16] is primarily focused on low-cost SAN replacement using block-based semantics, the T10 standards group is defining an extended object-based SCSI command set called OSD [21]. Although Lustre defines a more substantial set of capabilities than the T10/OSD, using the Lustre clustering and meta-data architecture to aggregate a set of T10/OSD/iSCSI compatible targets is desirable.

7. *Security and management:* Both the cluster of all systems and the file system require a substantial amount of configuration information. LDAP style directories have become a widely used global infrastructure for such data. Kerberos/X509 security is widely in use now and run-time configuration is successfully done with DHCP-style services. We expect to draw on all of these.

While we expect to leverage code, capabilities and techniques from the above components, the Lustre design also adds significant novel capabilities. Lustre incorporates novel locking mechanisms and directory structures, an advanced object-based storage access protocol that supports byte-granular scatter gather I/O, object pre-allocation and command aggregation and integrated journal recovery support. To meet the required performance and scalability targets, Lustre also exposes a direct object access interface and provides specialized performance hints. The ability to execute code on the Lustre storage targets allows them to actively participate in storage management functions.

4 Lustre Software Architecture

4.1 Overview

The Lustre software architecture [5] is easiest described as a stack of layered object-based storage (OBD) modules. Key software layers in Lustre are:

a) *POSIX Compliant File System*: An object-based file system exposes a POSIX compliant VFS interface to client applications, and two protocol interfaces to software layers below: (1) a meta-data coherency protocol to handle name space operations, and (2) an object storage protocol to handle file I/O operations. Typically, the object-based file system is the top-most layer in a stack of object-based storage modules. It is intended to be stacked on top of a logical or a direct OBD driver.

b) *Direct-Access Object Storage Interface*: This is an alternative top-level module that enables high-performance computing where applications take responsibility of object data coherency. Lustre provides a direct access interface that allows such applications to bypass the above-mentioned file system layer. By relaxing cluster file system synchronization and coherency mechanisms, we expect to achieve higher performance from specialized direct-access applications.

c) *Logical Object Modules*: Logical OBD drivers are stackable modules that have the same OBD interface going in (to the top) and coming out (of the bottom). Logical OBD interact with other logical drivers, or layer on top of a direct OBD driver. Logical object modules typically perform functions such as mirroring, RAID, data migration, or versioning, and can be used to execute downloaded active disk computations.

d) *Direct OBD Drivers*: Direct OBD drivers provide an OBD interface (at the top) to an actual underlying disk, that is a set of blocks (at the bottom). These drivers perform block allocation for an object storage device, and provide a persistent data repository that is exported through an OBD interface.

e) *Client/Target Driver Pairs & Networking*: are use to encapsulate the object protocol over a network. Client/target driver pairs can be specialized for a variety of transport protocols, e.g. plain TCP/IP or transport layers that support RDMA such as VI architecture [14] supporting adapters [12, 13] that run over Infiniband, Fibre Channel, or even TCP/IP [10,11]. Development of high-performance transport drivers is a key component of the overall project.

The figure below illustrates the stacking possibilities for these drivers.

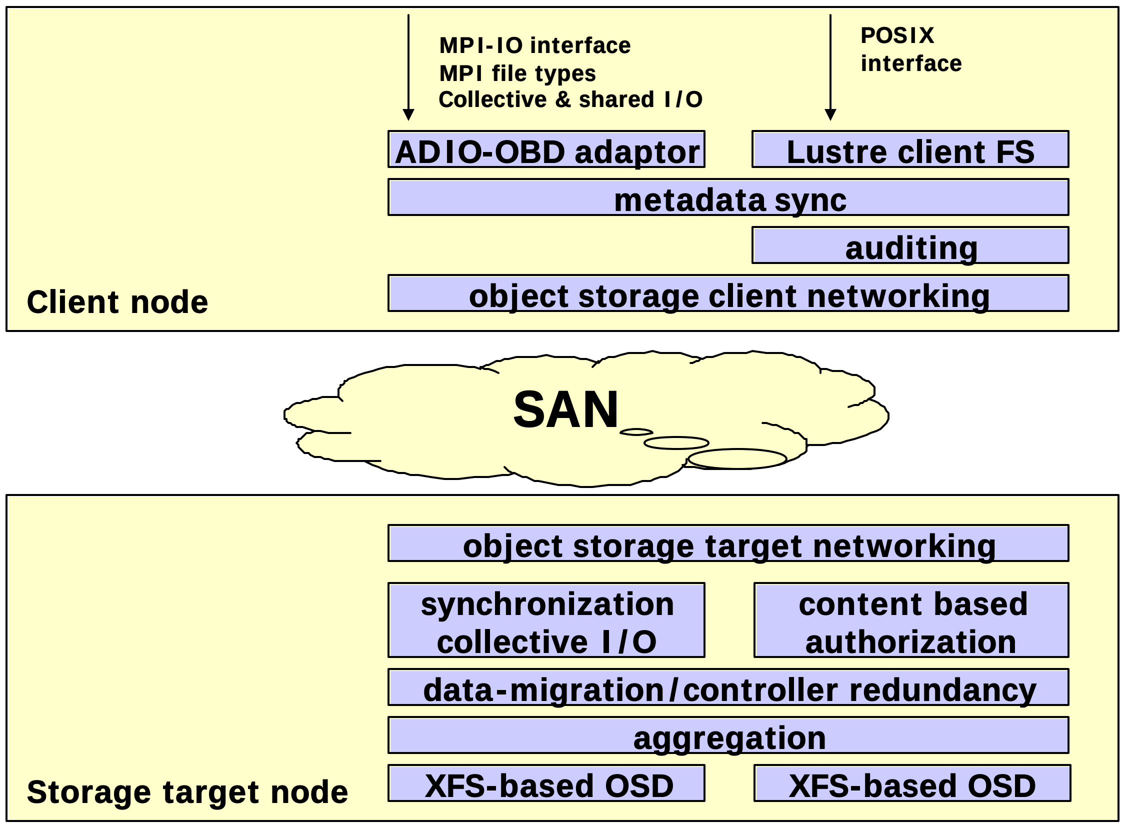


Figure 2: Driver stacking

In each of these areas, the Lustre project requires further work to meet the requirements of the Tri-Labs/NSA. Key steps we will undertake are:

1. *Class drivers*
   - Enable remote management
   - Add the capability for profile based automatic configuration as in enterprise management systems.
2. *High performance direct drivers* for data and metadata storage
   - An journal file system (XFS, ReiserFS, JFS, Ext3) based direct object driver is a candidate for a file storage driver
   - Journaling, recovery, concurrency and pre-allocation requires API enhancements
   - Extent based object locking APIs
   - To support redundancy and recovery we need support for replication logs, and orphan removal
3. Implement NASD-style *capabilities based security* between clients and targets.
4. *Client/target pairs*
   - High-performance networking including remote DMA capability
   - Incorporation of new networking and object-storage enhancements from WARP, DAFS, and iSCSI/T10/OSD.

- Scalability in terms of number of clients

5. *Logical drivers*
    - Distributed striping drivers
    - Synchronization mechanisms for the above (if needed)

4.2 The Lustre Cluster File System

One of the most challenging RFP requirements is that of metadata scalability and performance. This problem has several aspects, such as high performance in clients in situations of low contention for resources, as well as that of throughput when many clients are operating on the same resource. Generally we will avoid synchronous writes to the maximum extent feasible.

A key design aspect of the Lustre cluster file system is the integration of caching, cluster coherency and recovery. To achieve best performance from a client perspective, objects should only be evicted from client caches only when a (possibly timeout based) lock revocation is processed, and eviction should flush groups of object operations. To ensure recoverability, object eviction should handle groups of related object operations and direct these to persistent storage in a transactional manner. Our basic technique here will be to built a write-back log of operations in the clients memory and migrate that to metadata control nodes for replay upon timeouts of locks or eviction.

In the case of high contention for resources, such as all clients creating files in the same directory, we will employ quite different techniques. A first problem is how to avoid large amount of lock revocation traffic. Client locks should time out, but also, when lock requests are coming in a rate exceeding a certain threshold, it will be more advantageous to not grant client locks, but instead to perform operations on the metadata cluster in a client server RPC model similar to AFS. We see here how Lustre can run in write-back mode as well as in RPC mode, dynamically adapting to the level of contention in the cluster.

Another problem in this situation is how to spread the update load over the metadata cluster. The key to a solution here is to use hashed directories that can be subdivided over the cluster in conjunction with operation based object locks [3].

Efficient recovery of the cluster under membership changes is another key problem. In our RFI response [3], we proposed that Lustre clients remain satellite nodes with respect to cluster coherency and locking. This is critical for scalable recovery, which will have to rely on the scalable flood fill algorithm described in [34] for notifications to clients. In current open source cluster file systems, e.g., GFS [25], file system concurrency semantics has received relatively little attention. A literature study of commercial cluster file system implementations, e.g. Frangipani/Petal [32], Calypso [33], and VAX/VMS clusters [29], [30], reveals that these systems have gone through much detailed refinement and performance tuning. A detailed design and prototyping to assess concurrency performance and recoverability will be very important for Lustre.

There are two cases where avoiding synchronous writes is not appropriate. First, to implement NFS semantics, metadata updates have to reach stable storage before system calls return. Our system can support this by operating in the RPC model and requiring the RPC's to sync data. Operating in RPC mode also provides an opportunity for integration with the DiFFS file system [38]. This system avoids locking and allows metadata transactions to span multiple storage targets. DiFFS also avoids journal recovery mechanisms that span multiple nodes, but requires synchronous writes and detection of RPC failures to order updates [51].

The following list outlines key components of the Lustre cluster file system:

1. *Protocols*
    - A file I/O protocol between clients and storage targets
    - A metadata update protocol between clients and metadata control
    - Recovery protocols for all system failures
    - Separation of protocol and service implementation
2. A *file system*
    - VFS operations with VFS extensions for locking
    - Scalable directory formats based on extensible hashes or invisible subdirectories.
    - Operation based lock acquisition and lock version management [3]
    - Grouping of operations into file system transaction groups
    - Data and attribute retrieval and storage through an object storage API
    - GSSAPI compliant security
    - Resource awareness for constituent object allocation
3. *Single namespace*s
    - A filtering file system layer for namespace management
    - File sets, and Unix mount points can be grafted into a namespace
    - Cross-complex file set location databases with global identities
4. *Third party file I/O* to object storage controllers
    - Directory entries pointing to inodes describing large granularity (device/object/extent) triples
    - Controller based locking (flock) and collective operation support
5. A *write-back file system cache* integrated with existing caches in the OS comprising:
    - Cached extents in files
    - Cached metadata
    - Managing dependencies within and between transaction groups
    - Flushing and unpinning
    - Lock revocation support
6. *Recovery* support:
    - Coherency and recovery between targets, clients and controllers
    - Log replay and distributed log dependencies
    - I/O fencing
7. Exportable interfaces
    - Notification and ACL support for correct NFS/CIFS exports

- InterMezzo support

4.3 Clustering Infrastructure

To successfully build a cluster file system, a set of basic cluster infrastructure is needed. A sample architecture designed by the Linux Cluster Cabal [34] incorporates critical scalability enhancements of lock managers. Through the correct distribution of resources, locking scalability will be mostly an issue between metadata control nodes and satellite clients and are unlikely to affect file I/O. Components we expect to use for basic clustering support are listed below:

1. *Connection and live-ness support, membership and quorum*: We may base these on the Ensemble project [27].
2. An *event delivery* mechanism: IBM has detailed an implementation as part of high-availability CMP [37]. Tweedie's design [34] is similar. We plan to build on these designs.
3. A *cluster resource database***:** Replicated LDAP servers have been suggested.
4. A *recovery manager* for staged distributed recovery: The daemon which is part of Kimberlite [28] or Compaq's open source cluster manager (to be released) will be a good starting points. Barrier support will be implemented.
5. A *distributed lock manager*: IBM's open source DLM will form the basis, with scalability enhancements as proposed by the Cluster Cabal in [34], and additional DLM communication performance improvements.
6. Basic IP and shared storage fail-over services where appropriate. LVS and Kimberlite provide excellent starting points.

4.4 Programmable Object Storage Targets

We will design and implement a high-performance object-based storage target based on standard high-volume hardware running a standard Linux operating system. Basis for high-performance target run-time system is the Linux in-kernel TUX web server architecture [17, 18, 19]. This will provide a highly optimized multi-threaded network communication system and additionally a kernel-level safe execution environment. TUX is well integrated with the Linux I/O subsystem and already provides many of the storage and networking hooks required to support high-performance object data I/O. Reuse of TUX provides one of the basic Lustre building bock to achieve the Tri-Labs/NSA SOW section 3.3 (no single point of failure) and section 4.2 (scalable infrastructure for clusters and the enterprise) requirements.

We will define and develop an "active disk" execution environment on the storage target that allows safe execution of downloadable logical object modules. Active disk functions fall into two categories: (1) an active element in the object metadata path can perform storage management functions such as backup, load balancing or on-line data migration, and (2) an active element in the object data read/write path can perform data manipulation and aggregation functions useful, for instance, for transcoding of multi-media streams, customized content-based security or low-overhead data mining [XX]. The Lustre object protocol includes an iterator method that allows a user-defined function to be applied to an entire set of objects. This primitive allows the storage target to optimize disk block traversal used by data mining or disk indexing operations [YY].

Safe execution of downloadable logical object modules hinges on two factors. First, we need to constrain downloaded code to see (or be able to update) only a subset of all stored objects. Second, we need to ensure that downloaded code cannot subsume all storage target resources, be they compute cycles, network bandwidth or disk space. Lustre meets the first safety constraint by enforcing NASD capabilities-based security on individual objects, in other words, an active disks request will only be able to see/update objects for which it has been granted the appropriate capabilities. The second safety constraint requires storage targets to constrain the amount of resources available to unknown downloadable logical modules. One sandboxing possibility is to execute the logical module in user-space, where resource limitations such as time-slices and disk quotas can easily be enforced. Another is to pre-allocate a downloaded code module's required maximum resources, and fail it when it oversteps it allocation.

We expect downloadable logical modules to be used for providing management services such as backup, archiving, data balancing, on-line data migration, as well as security (content based authorization and encryption) or performance (e.g. prefetching) functions etc. that are required by SOW sections 4.2.4 (archive driven performance), 4.2.5 (adaptive prefetching), 4.4.1 (minimize human management effort), 4.4.3 (dynamic tuning), 4.4.4 (diagnostic reporting), 4.4.16 (backup), and 4.5 (security).

4.5 Storage Management & Configuration

To fulfill the capacity and bandwidth scalability requirements specified in the Tri-Labs/NSA RFP [35], Lustre storage clusters would probably consist of thousands of storage targets. Configuration and administration of such a large number of devices must be completely automated. Additionally substantial storage management infrastructure is required for archiving and backup.

While we will not develop management tools per-se, we will provide the basic infrastructure for monitoring, recovery and replacement and discovery of nodes. A basic directory scheme controlling grouping and configuration of nodes will be addressed by this proposal. On the storage management front we will provide snapshots and data migration API's upon which further tools can be built.

Although the Lustre *file system* will provide a single name end-user visible name space, for management purposes, *the systems in* a Lustre cluster will be subdivided into *organizational units*. These units describe collections of systems with different characteristics that will typically manage file sets for a particular purpose. The systems in an organizational unit are managed by associating the organizational unit with a management *profile* which describes a file set's backup configuration, security and performance attributes, as well as required storage target "active disk" code execution modules. Enterprise management systems (Novell's NDS, Windows 2000, Tivoli and CA Unicenter) provide such infrastructure.

We will provide:

1. Cluster resource database (LDAP) storage for management data
2. Directory schemas for organizational units and management profiles
3. API's to apply management profiles to systems

At a more detailed level **storage management** requires a substantial collection of modules. Among these we will deliver basic implementations of the following:

1. Data migration API's for backup/restore and HSM
2. File system snapshot
3. Monitoring of hardware and users

5 Coverage of Tri-Labs/NSA Requirements

In the following, we refer to the section numbers in the Tri-Labs/NSA SGS File System statement of work (SOW).

5.1 SOW Section 3 - Minimum Requirements

5.1.1 SOW Section 3.1 POSIX-like interface

Our file system will have the standard POSIX interfaces. The system will be able to run in Unix semantics mode and in an optimized mode. In Unix-semantics mode the systems calls will have the expected behavior, while in optimized mode significant performance-gains are possible – this will likely affect the *stat and fstat* calls, which will involve RPCs to storage targets.

There are a number of special system calls such as *mmap* and calls associated with *asynchronous I/O*. On single systems we expect these calls to have the normal semantics, but their behavior across the cluster will depend on the OS.

5.1.2 SOW Section 3.2 Integration Compatibility

The Lustre storage stack is highly modular, and we will retain maximum modularity at the file system level as well. Most management functions, such as DMAPI, backup support, encryption can be built as modules that can be downloaded into the storage stack. We have indications from existing Lustre modules that these do not introduce a performance penalty.

5.1.3 SOW Section 3.3 No single point of failure

Failures in the cluster can affect clients, storage targets and cluster control nodes and communications. The state of the cluster is represented by data in memory including cache file system data as well as cluster state such as lock resources and membership, data in transit over communication links and persistent data on systems. Under normal operation the cluster will move forward from one consistent state to another through groups of operations, which typically involve multiple state changes. Recovering from failures involves halting normal operations on affected nodes, restoring state to a consistent form and continuing the operation.

We will introduce a systematic recovery framework based on the work of Tweedie, Braam, Callahan and McVoy (the Linux Cluster Cabal) [34], which in turn was heavily influenced by the VAX Cluster literature.

There is very little literature about the recovery issues for cluster file systems, but our experience with Coda and InterMezzo [20] will lead us to build a framework that integrates synchronization, journal file system transactions and cache behavior.

We expect that the clusters may require some scalability enhancements to deal with large numbers of nodes. Such enhancements were discussed in [34].

To limit the scope of our work, we will use industry standard redundancy mechanisms such as backup Kerberos KDC and replicated LDAP databases wherever read-mostly cluster resources are involved.

5.2 SOW Section 4 Desired performance features

5.2.1 SOW Section 4.1 Global Access

As indicated in section 2, the sites addressed by this proposal are bigger than traditional clusters and our solution needs to span site boundaries. A full solution in this space goes well beyond the SOW requirements and may draw on InterMezzo [20] to address replication. However, our solution will provide basic infrastructure in the area of file set support, management, security and networking.

5.2.1.1 SOW Section 4.1.1 Global Scalable Namespace

Global name spaces were perhaps first introduced into AFS, followed by similar implementations in Coda, DCE/DFS, InterMezzo and Microsoft dfs and NFS v4. We will follow a similar strategy and construct a name space module that combines file sets and file system mount points into a single name space. We will combine a cluster resource database which the file system queries for transparent traversal of the name space with new techniques known as struts or pseudo file systems in the InterMezzo and NFS v4 efforts. The latter provide higher availability by bridging temporarily unavailable sections of the name space.

The global namespace will be exportable through NFS and CIFS. However, the detailed Windows semantics (such as Window’s exclusive open) offered through CIFS (Samba) servers running on different clients requires hooks in the server for which we will only provide notification interfaces.

A new aspect of namespace management arises from the allocation of object storage targets for file data (object placement). The choice of appropriate controllers to handle segments or stripes of files should be based on a resource management scheme in the subsystems responsible for metadata.

5.2.1.2 SOW Section 4.1.2 Client Software

Our client software is based on the Lustre object file system currently available for Linux. The key aspects enabling easy portability of the client software to operating systems other than Linux are:

1. There will be an open source Linux reference implementation
2. This file system bears a striking resemblance to NFS
3. Our locking mechanisms for client nodes will be refinements and extensions of those employed in NFS v4. The refinements will address finer granularity of locks, and more aggressive caching and synchronization of meta-data and directory data.
4. Our security mechanisms will be largely analogous to those used in NFS v4, combined with NASD object security, see section 5.2.5.
5. Our transport between clients and storage controllers, i.e. *the Lustre object storage protocol,* has considerable similarity to DAFS. Lustre does not require any DAFS calls related to directory lookups and file names and brings extensions to DAFS for more aggregate commands and vectored I/O, pre-allocation and file system journal transaction support.

The systems we will be employing, be it with modifications, form the foundation of the next generation storage and file system infrastructure for the industry. We feel there is a great likelihood that ports will not be problematic at all. In addition to the portability it provides this approach allows us to benefit from and contribute to related efforts.

5.2.1.3 SOW Section 4.1.3 Exportable Interfaces

Our file system will be exportable to NFS V4 and CIFS. However, there is a caveat to be aware of. Neither the CIFS nor the NFS server is stateless. Exporting CIFS and NFS v4 from a single file server already requires synchronization of state between the Samba server and the file system (similar issues apply even to NFS v2/3). Exporting load-balancing instances of these servers, exporting CIFS and NFS from multiple client systems requires synchronization of state among all these servers for full support of the semantics. In addition the servers would need to be aware of cluster membership transitions to handle the addition or disappearance of one of the file servers.

Our cluster infrastructure will enable NFS and Samba to be modified to provide such state synchronization and the Samba Team (at VA Linux) has expressed interest in doing so with us.

5.2.1.4 SOW Section 4.1.4 Coexistence with other file systems

This is not a problem.

5.2.1.5 SOW Section 4.1.5 Transparent global capabilities

The Lustre file system will do an extremely aggressive form of caching. Lookup and attribute acquisition will be based on cached data. Writes (except when especially requested) will never be synchronous and large caches will flush numerous update requests periodically.

Our security and resource management will be global as addressed under other requirements (global namespace 6.2.1.1, global identities 6.2.3.2 and WAN security 6.2.3.3).

We expect good performance and have included a deliverable to perform tuning and provide additional adaptation to our networking and file system layers for WAN performance.

We will develop specialized Lustre client/target driver pairs for a variety of common storage transport protocols, e.g. standard TCP/IP for Ethernet, VI-architecture based for Fibre Channel and other high-performance storage area networks. While these client/target driver pairs will be optimized for their particular transport layer, Lustre's "active disk" concept allows object storage code modules to be loaded into both storage client and target systems. Specialized prefetching or buffer resizing and caching module pairs can be used to transparently pipeline large requests over long latency high bandwidth wide-area communication links. We will explore a range of adaptive self-tuning and prefetching performance enhancing target modules. We will also develop a set of specialized client-side direct object storage APIs that will allow knowledgeable clients applications to specify performance hints to directly to the object store.

5.2.1.6 SOW Section 4.1.6 Integration into a SAN environment

We expect most Lustre storage targets to run on high performance commodity hardware that use traditional block storage for persistent data. Since Lustre direct drivers (the lowest level of the Lustre storage abstraction as described in section 4 above) are inherently capable of handling block-based storage, Lustre already works with existing block-based storage area network (SAN) technologies such as Fibre Channel or iSCSI. This is most attractive in the context where the storage controllers are also responsible for RAID so that a simple JBOD and fail-over, commodity (SMP) system can act as the storage target.

In this setting we expect to incorporate explicit support of RDMA into the Lustre object storage protocol. We have carefully studied the following networking technologies and outline below how we expect to use them in the Lustre context:

1. *WARP* is a recent protocol proposal for encoding an interleaved *send* and *RDMA* packet stream on top of TCP/IP [11]. WARP *RDMA packets* are self-describing RDMA write "chunks" that contain destination buffer-ids and offsets (this allows receiving network interface cards to determine destination address for each packet). WARP *send packets* do not specify an address; instead they specify a Send Sequence Number and an offset. The WARP proposal includes a mapping

of these functions on to TCP, SCTP, and also discusses how iSCSI protocol data units can be mapped on to WARP.
Conclusion: WARP is of interest to Lustre as it increases the probability for an RDMA write mechanism to exist on top of TCP/IP. This indicates that Lustre should be able to take advantage of RDMA writes not only on VI-architecture based transports, but, in the future, on TCP/IP based transports as well.

2. *DAFS* is the Direct Access File System access protocol [15]. DAFS is focused on providing NFS v4 file access capability over a reliable system area network. The RPC data layout style used by DAFS for marshalling (described in section 6.1.1 of [15]) is of particular interest to Lustre. The key element of the DAFS layout places all fixed-size structure elements at their naturally aligned boundaries, and moves all variable size buffers to the end of the message. Pointers in the fixed-size structure that reference variable size objects are converted into relative offsets in the on-the-wire format. This allows receivers to simply re-instantiate pointers by adding the offset to the message's base address. Additionally, if receivers place incoming messages at the proper alignment in memory then the received data structure can simply be typecast to the appropriate data type in place, without further copying or reassembly.
   Conclusion: The DAFS RPC data layout, combined with WARP-like RDMA writes is a very attractive solution for low-overhead Lustre client-target communication, even over TCP/IP.

3. *iSCSI/T10 OSD*: iSCSI is a SCSI encapsulation protocol layer on top of TCP/IP [16]. iSCSI has gained significant industry momentum over the past few months. While iSCSI's primary focus is low-cost SAN replacement with block-based semantics, work in T10 has been ongoing to define an extended object-based SCSI command set [21]. Furthermore, the WARP effort [11] discussed above has already proposed an encapsulation scheme for iSCSI commands in their protocol.
   Conclusion: iSCSI has a lot of momentum that make it an interesting transport layer for data transport in Lustre. Even though Lustre defines a more substantial set of capabilities than the T10/OSD devices provide (namely preallocation, file system recovery support, vectored byte granular I/O and execution of loadable modules on the storage target) using the Lustre clustering and meta-data architecture to aggregate a set of T10/OSD/iSCSI targets is a very desirable goal.

We expect to spend a significant amount of design and implementation effort to bring new technology developments to bear on transport layer performance and robustness. This includes not only high performance in storage area networks based on VI-architecture abstractions [12,13], but also improved performance on standard TCP/IP based local and wide area networks.

Support for direct 3rd-party I/O in "legacy" block-based SAN

Existing commercial SAN deployments may want to re-use expensive block-based storage infrastructure (such as an EMC Symmetrix, HP XP512 or similar systems) in the context of an object storage cluster. An object storage target can utilize such block storage, but I/O would incur an extra "hop" at the object target. Lustre can also support direct 3rd-party I/O in such "legacy" SANs installations by introducing "Proxy Object Targets" (POTs) that perform block-allocation for each of the block-based SAN storage targets. POTs execute all object commands, translate objects into SAN device and block information, and then trigger the appropriate set of 3rd-party direct block I/O transfers between the client and SAN storage targets. Compared to native object storage this introduces an additional message, and hence latency, for each read and each write operation, but does not pass the data through the proxy.

When clients send write commands to a POT, they do not include data for writes, but instead expect to receive SAN device and block information from the POT. Then clients perform the I/O directly with the SAN target. When the I/O has completed the client must confirm the completion to the POT. Also the client must handle the case of "write-and-free" buffers and "write-and-retain" buffers to support journal transactions.

The POT will be careful to build the same transactional update logs as Lustre object targets to avoid the introduction of a second recovery mechanism.

5.2.2 SOW Section 4.2 Scalable infrastructure for clusters

Generally speaking our approach to scalability is two-tiered. First we aggressively *limit the footprint of shared resources*, secondly we introduce new *sub-dividable resources*. A few examples will illustrate our approach.

Traditional cluster file systems have delegated file block allocation to a single metadata server (for non-symmetric cluster file systems such as CXFS) or to every cluster node (in symmetric cluster file systems). In the former case a bottleneck can easily arise, while in the latter a substantial amount of synchronization mechanisms surrounds the update of allocation bitmaps.

5.2.2.1 SOW Section 4.2.1 Parallel I/O bandwidth

To achieve minimal interference between systems for scalable I/O bandwidth we made two important design decisions. The first is to use object storage targets, which offload block allocation from the file system clients and avoid unnecessary sharing of allocation metadata – this is one of the items that falls under footprint reduction. Similarly storage controllers will implement file extent locking (for striped files we have some open questions in our design). Again this limits the file locking resources to precisely those clients and controllers that are involved.

N by M mapping is a good example of our on-controller computing environment. Logical storage modules can interpret MPI-IO views of structured data on the controller and, for example, deliver columns when storage order is in rows. Our strategy here is to implement the ADIO interface for which the Lustre protocol has already been adapted with scatter-gather byte level compound write commands, including hints.

A very substantial portion of our deliverables is focused on not merely implementing but also finalizing, testing and debugging the scalable I/O infrastructure, including everything between high performance storage networking based on VI-like protocols, locking relaxations and other performance hints.

5.2.2.2 SOW Section 4.2.2 Support for very large file systems

We will support very large file systems as mentioned in the requirements. Object storage brings new opportunities to bypass limitations found in some current operating systems related to the address space of blocks in storage devices. By striping over multiple devices Linux systems will be able to support file systems of practically unlimited size and files of up to 2^{64} bytes.

5.2.2.3 SOW Section 4.2.3 Scalable File Creation and Metadata operations

To support scalable file creation it is important that a creation operation for file *foo* in directory *bar* can be executed on a single cluster node. Two key aspects in our solution to this problem (see [3]) are to acquire a lock, not just on bar as is common practice at present, but on a combination of the resources involved in the operation. In this way a client can locate a system responsible for that part of the directory file that will hold bar – this is needed both for the insertion of a new directory entry and for the accompanying check of non-existence preceding the creation. To split directory handling across multiple cluster nodes an extensible hashing scheme is needed. Interestingly, GFS and the ext2 and ext3 file systems have just seen the introduction of this. While we will likely need to change the directory format, there is much to be learned from the algorithms, which have shown good results with directories with millions of entries.

An open question at present is if we should allow clients to modify directory data (as is customary in cluster file systems) or if it is more prudent to have an RPC style interaction, possibly with write back caching, as is done in InterMezzo (with write back caching) & Coda/AFS/DCE-DFS (without caching). Caching of directory data for the purpose of updates may result in much higher performance but mandates that clients are systems entrusted with enforcing authorization – in a WAN environment this is probably undesirable.

To scale the performance of “ls” an interesting read-ahead operation on directory objects will be needed, which may span multiple metadata controller nodes.

The most pressing issue for scalability is the recovery mechanism, since it will also need to be invoked when there is a cluster transition between the metadata control nodes (metadata servers). We will carefully integrate cache flushes, synchronization and journal transactions so that cached data is migrated from clients to control nodes in transactional units. Our system will rely on abstractions found in the Spiralog File Server [7].

As for I/O scalability, we have reserved ample room to build, test and improve the metadata scalability issues in our project.

5.2.2.4 SOW Section 4.2.4 Archive driven performance

The Lustre protocol provides serverless remote copy commands for extents of objects, as well as object iteration functions to process entire volumes without server interaction. While we do not expect to have resources for fully-fledged NDMP and DMAPI interfaces, the complete infrastructure to build these easily – as logical storage modules, – will be available.

5.2.2.5 SOW Section 4.2.5 Adaptive prefetching

Like most file systems, Lustre provides standard read-ahead and Lustre will have more aggressive write-behind techniques than currently found in cluster file systems to improve performance for sequential access patterns. In addition to the standard POSIX file system access, Lustre will provide access to stored objects through a specialized direct object storage API. The direct access API will allow applications to bypass standard file system prefetching behavior, and will provide a relaxed consistency model for file data to support applications that are able to take advantage of it. Additionally, a set of performance hints (specifiable on a per object/file basis) will allow users to persistently record the preferred access methods with each object. Finally, special purpose logical object modules can be plugged into the Lustre client and target stacks allow flexible and, possibly application programmable prefetching strategies to be deployed.

5.2.3 SOW Section 4.3 WAN Access

Fundamentally we see three components to the WAN access issues raised by the RFP: a global name space, a global security model and an adaptive multi-channel WAN transport infrastructure.

5.2.3.1 SOW Section 4.3.1 WAN access to files

Our file system will perform extremely aggressive write-back caching with read-ahead. This will eliminate many latency-induced bottlenecks in WAN environments.

Use of multiple transport channels and connection trunking will improve throughput and latency in wide-area networks and is one of the key research items for the object storage and networking team. For WAN access to the Lustre object store, we expect to create adaptive load balancing strategies in which multiple channels are automatically created and torn-down based on dynamic feedback. Differentiated use of a set of channels for short control and coherence messages and another channel for bulk data transfers can significantly improve perceived latency. We further expect this capability to be very useful for storage management functions such as backup and on-line data migration that can benefit from background operation.

InterMezzo will be able to export parts of the Lustre namespace to clients, even mobile, disconnected clients. Such data can be presented on the client in secure private namespaces and modifications are reintegrated with log replay that exhibits the full security features desirable.

5.2.3.2 SOW Section 4.3.2 Global Identities

We will use global identities with cross realm authentication and authorization. Such global identities require mappings to integer user identities on clients to enable export of NFS. It may also be efficient to maintain such mappings on control nodes, but *equality of Unix user identities on different client is not required.* DCE/DFS and Coda have solutions here that we may adopt.

5.2.3.3 SOW Section 4.3.3 WAN security integration

We will take the following steps to ensure wide area security integration. First our ACL support will use principals not user-id's to accommodate global definitions of authorization. Second we will use Kerberos with ANL/Globus X509 patches for cross-cell authentication. Our volume location service, part of the cluster resource database, will be stored in (replicated) LDAP directories with referrals which is itself secured with Kerberos authentication and ACL's.

5.2.4 SOW Section 4.4 Scalable management & operational facilities

Our target is to implement a basic management infrastructure, which will enable other entities to port existing software to Lustre. For example, there are API's for secure remote management and MIB's for monitoring. There are data migration and hole punching API's and snapshots. However our focus is on file systems and storage networking and we merely plan to provide sufficient infrastructure to enable further development of management tools by others.

5.2.4.1 SOW Section 4.4.1 Minimize the human effort

We will lay the foundation for configuration of devices based on a profile description in the cluster resource database. When a new device is added, it needs to know to what organization unit it belongs and auto-configure based on a profile.

5.2.4.2 SOW Section 4.4.2 Integration with other management tools

Will not be addressed, but should be easily possible.

5.2.4.3 SOW Section 4.4.3 Dynamic tuning and reconfiguration

We will allow for dynamic resizing and on-line migration of data, as well as dynamic tuning of system parameters. We hope to automate many of the networking transport configuration issues by making the networking layer automatically adapt to changes in workload and available bandwidth. We expect storage management functions such as backup and on-line data migration will benefit from this.

5.2.4.4 SOW Section 4.4.4 Diagnostic reporting

Will be present through SNMP MIBs and driver-exported information about problems. We have already created a logical object module that monitors the number, size and latency of object transactions executed by the file system. This will be valuable in analyzing performance problem and reporting overall performance statistics to administrators and end-users.

5.2.4.5 SOW Section 4.4.5 Support for configuration management

We will develop precise configuration information of software and hardware running on all systems.

5.2.4.6 SOW Section 4.4.6 Problem determination GUI

Will not be included.

5.2.4.7 SOW Section 4.4.7 User statistics reporting

Basic infrastructure in the file system clients, cluster controllers and targets will be made available.

5.2.4.8 SOW Section 4.4.8 Security management

Command line tools will be available to control all aspects of security.

5.2.4.9 SOW Section 4.4.9 Improved characterization and retrieval of files

Will not be covered. Our object based target drivers allow modular addition of storage management modules that could be capable of prioritizing object access as well as selecting network drivers based on object attributes.

At the file system level InterMezzo style filtering (which we will use for auditing) can easily be adapted to maintain file system attribute databases that remain consistent in a transactional manner with updates of the system.

Running a SQL based metadata cluster engine would provide unique opportunities here.

5.2.4.10 SOW Section 4.4.10 Full documentation

A basic set of system, operation and user manuals will be delivered.

5.2.4.11 SOW Section 4.4.11 Fault tolerance, Reliability, Availability, Serviceability (RAS)

Clients

Much of our design work so far has centered about scalable tolerance mechanisms for failures. Given the enormous number of clients it is necessary for these clients to not provoke cluster transitions involving large counts of other systems.

As to locking they will be so called *satellite nodes* that have full use of lock mechanisms but do not participate in resource mastering. Should such nodes leave or enter the cluster there will be minimal disruption. If we allow memory-to-memory data transfer between client nodes the recovery problems remain involved since the cluster control nodes may have to STOMITH systems that depend on a dying system flushing its write-back log. If we synchronize by flushing through cluster control nodes such problems do not occur.

Metadata control nodes

Our metadata control nodes will follow mechanisms in VAX Cluster Style to deal with cluster transitions. The SOW lists the possibility of giving such systems redundant fail-over hardware, but this does not address memory state and is not usable for metadata cluster transitions. The failure handling in metadata control nodes is complicated, similar to that found in traditional cluster file systems. Traditional cluster file systems typically exploit synchronous I/O when a revocation of write locks occurs. We want to avoid synchronous writes to the maximum extent (but will allow this to be an option). As a result the failure of a metadata control nodes may force clients to re-flush their cached operations to a newly erected metadata control node or otherwise face a failure themselves to maintain application consistency. Clients will detect the failure of a metadata control node and will have to await a message from the cluster before continuing to execute meta-data transactions. A top-down “flood-fill” scalable cluster structure put forward by the cluster cabal will be used to transmit such messages [8].

The storage industry has developed many solutions for redundancy at the target level. Our object-based infrastructure makes many of these issues easier and allows for the following solutions. In each case the redundancy is introduced as an independent storage management module, allowing the same mechanisms to be used with different types of storage networking and with different back-ends.

Storage Targets

Active/active redundant storage targets enable two independent targets that can concurrently process requests. For block devices there is no shared state among requests but with object targets, all allocation data is shared. Doing active/active pairs of controllers would require a “mini cluster file system” between the controllers – a no-no.

A partially active/active controller allows one controller to write to one disk partition and another one to address another disk partition concurrently - load sharing at a coarse granularity is possible. Such targets can be built as follows: first let's use shared storage between the controllers. Using Kimberlite style fail-over clustering we can get redundancy for the storage networking and controller hardware. Such shared storage itself needs to be RAID and requires partially active/active block raid controllers on the two object storage controllers - possibly this could be software raid and commodity SMP systems attached to JBODs with some processors responsible for the target and some for the backend raid is attractive, possibly with some shared solid-state memory to speed up bitmap maintenance for RAID restoration.

The initiators have to detect that the first controller has failed and then retry on the other controller. The other controller needs to STOMITH the first controller, do journal recovery on the shared storage. Then using a simple form of an InterMezzo style shared operation log it needs to figure out what the last operation was that made it to the disk. It then tells the initiator to resume at the next operation.

Without the possibility of doing full active/active we need a different mechanism to exploit multiple channels. For this an SMP storage target with multiple incoming storage-network interfaces, multiple busses, probably each handled by a dedicated CPU and with more CPU's at the backend for doing RAID.

We also remain very interested in full redundancy in the case of 100% commodity hardware. Now we have two object storage controllers, each with one IDE and one interface disk, and we want to build a replicated redundant configuration out of them. The servers can blast out two object commands, one to each storage controller (this requires support in the file system metadata (as in Cheops [22]) since the object id's on the two controllers might have diverged). If one of these fails and recovers, the question is how to resynchronize the two. An InterMezzo style log maintained on each of these would be able to replay "missed object commands" to a recovering node.

When the log becomes too large data migration between the controllers (aka "lan" free backup) should rebuild an entire object store. In this case very innovative solutions exist which use the active nature of our controllers to run rsync to re-synchronize all objects, a dramatic improvement over classical RAID drive restoration.

Interactions

The creation and removal of files include file data objects and causes metadata updates in Lustre that span multiple nodes: metadata control and possibly many storage targets. Recovery from failures in this situation requires special care. If file data objects are lost we don't care: that is the usual behavior of NOT journaling file data.

If the container is lost due to system failure, it should probably take the file data objects into its grave - that is done using an orphan list (see Ext3 [49,50] or the XFS literature). The file data objects are orphan listed until the confirmation comes to the client that the container has reached persistence. The client now includes a "deorphanize" message in the next I/O operation.

If the container is lost (typically because the client and meta data controller die), the best way to remove the orphans is for the storage controller from time to time to query the metadata control cluster about orphan listed objects. If the metadata cluster guarantees to flush buffers every 30 seconds, then clients will learn that containers are persistent in little more than 30 secs. Storage targets would learn soon afterwards. Therefore, an orphan that is more than a minute old, and has not been de-orphanized is suspect and the target should contact the metadata controller to find out if the object was possibly lost. [Such contact from targets to metadata control cluster is needed anyway to update the summary metadata held in the container.]

The unlink case exploits an InterMezzo style replay log between metadata control nodes and object storage targets, which remains present until all storage targets have executed the object destruction requested.

5.2.4.12 SOW Section 4.4.12 Integration with tertiary storage

We will not implement a DMAPI interface but will write a design document how this can easily be done, for other entities to step in. Our object interface provides strong infrastructure for XDSM/DMAPI tertiary storage interaction, such as remote copy and hole-punching APIs. DMAPI like other storage management can be implemented as a loadable storage module. This will automatically be inter-complex. We will make sure to afford room for dtime support.

5.2.4.13 SOW Section 4.4.13 Standard POSIX and MPI-IO

POSIX interfaces will be the default and will be part of the file system.

We intend to implement an ADIO interface on top of the object file system. The Lustre API was adapted to deal easily with most of the requirements raised by ADIO. A new interface for collective operations is needed which is probably easily implemented using the infrastructure which for default file I/O arranges synchronization.

Hint parameters will be available, but we require extensive discussion with the Tri-Labs/NSA to learn more about specific requirements. We feel that to maximize portability the system APIs should gear towards standard calls and not exotic features.

5.2.4.14 SOW Section 4.4.14 Special API semantics for increased performance

We will definitely introduce special semantics to allow relaxed locking schemes to dramatically improve the performance of parallel I/O. We will probably apply this at the file-set level, and the semantics will likely be such that the applications are responsible for synchronization.

5.2.4.15 SOW Section 4.4.15 Time to build a file system

We will be using something like the XFS file system for backend storage. This system has extremely good characteristics for file system building and resizing and we will make sure to bring support for these to the Lustre file system.

5.2.4.16 SOW Section 4.4.16 Backup/Recovery

We will provide sufficient hooks for easy integration with standard enterprise backup software, including hooks for advanced features such as LAN free backup. We will not be delivering backup clients or servers for Lustre, but build design guidelines for others to build such systems.

5.2.4.17 SOW Section 4.4.17 Snapshot Capability

Lustre already has a prototype implementation of fully featured snapshots, using a logical object module. This has been transformed into a production quality file snapshot file system by Mountain View Data and we will similarly the Lustre code to provide robust and efficient snapshots, with support for database flushes and ".snap" directory support.

5.2.4.18 SOW Section 4.4.18 Flow control and QOS

Except for flow control in our high-performance and WAN storage networking modules this will not be addressed.

5.2.4.19 SOW Section 4.4.19 Benchmarks

A very significant part of our deliverables focuses on benchmarks and performance improvements. In these areas, we require extensive collaboration from the Tri-Labs/NSA.

5.2.5 SOW Section 4.5 Security

We expect to deliver an almost complete security implementation vis-à-vis the requirements It will be based on industry standards, in outline form a lightweight AFS, DCE/DFS file system security model, combined with NASD style security for storage controllers. Additionally we will show how storage modules can be used for content-based security and encryption. A variety of issues were not mentioned by the Tri-Labs/NSA document, most notably PKI for storage and retrieval of encrypted data. We are sensitive here to adjusting our designs in such a fashion that these issues can be addressed at a later stage.

5.2.5.1 SOW Section 4.5.1 Authentication

We will use a GSS-API compliant authorization mechanism. We will use Kerberos with the Globus-ANL X509 extensions as the token acquisition mechanism. The TGT's obtained in this fashion will enable users session keys but also enable shared secrets ("daily keys" in NASD speak) between storage targets and metadata control nodes.

At the kernel level we need identities such as AFS-style Process Authentication Groups (PAG) for realistic security enforcement. Such identities can have tickets and session keys associated with them after authentication.

Groups are a separate issue of extreme importance. We will seek a scalable integration between Unix groups and users and Kerberos principals. Additionally groups are needed to provide an infrastructure for encryption by clients. LDAP based implementations might provide a good secure and scalable solution here.

Authentication can involve remote authentication servers using cellular Kerberos and LDAP referrals.

5.2.5.2 SOW Section 4.5.2 Authorization

Our authorization is a two level approach. NFS v4 access control lists will be used at the file system level to authorize access to files and directories. Of particular concern to us is a detailed discussion with the Tri-Labs/NSA regarding the case where multiple users on a client are accessing the same files – in this case the NFS mechanisms are very inefficient and complex and key expiration – already a known hairy issue – can become even more involved.

The primary means of authorizing client access to objects on the object storage targets are capabilities. A capability is a token issued by metadata control nodes to the client, which describes the object it applies to and the access rights that are granted. The client presents the capability to the target on each operation, and the drive can cryptographically verify the authenticity of the capability without contacting the file manager. If the capability is invalid, the drive returns an error to the client, and the client must contact the file manager to receive a new capability.

5.2.5.3 SOW Section 4.5.3 Content based authorization

Content-based authorization is an obvious attractive application of controller based computing. We will build a sample module to demonstrate how this can be done. An interesting aspect here will be the API to transfer such authorization information from clients to storage controllers since the NASD capabilities may not suffice for this purpose.

5.2.5.4 SOW Section 4.5.4 Logging and auditing

The InterMezzo file system can audit file access at the file system level. It is a low overhead file system filter, which can easily be modified to track access to storage objects on targets as well. The management of 10,000's of clients and 1000's of storage controllers providing auditing information is an issue by itself.

5.2.5.5 SOW Section 4.5.5 Encryption

While we will not implement an encryption module, we will write a design specification for other entities to step in and also provide the key infrastructure required to handle client-based encryption. Several companies have approached us to build hardware supported encryption modules for Lustre.

A loadable logical object storage module is an ideal vehicle for encryption and can optionally be run at the target as well.

5.2.5.6 SOW Section 4.5.6 Trust analysis

A trust analysis will be delivered as part of our design specifications.

6 References

[1] Terry Jones & Anne Huber, "Request for Information: GFS/DFS File Systems", National Labs RFI, October 30, 2000.

[2] W. de Jonge, M.F. Kaashoek, and W.C. Hsieh, "The Logical Disk: A New Approach to Improving File Systems", http://www.pdos.lcs.mit.edu/ld/, Published in Proceedings of the Thirteenth Symposium on Operating Systems Principles, 1993.

[3] http://www.lustre.org.

[4] Peter Braam, "Lustre and SGPFS", Mountain View Data, Inc., Response to RFI, November 30, 2000.

[5] Peter J. Braam and Andreas E. Dilger, "Object Based Storage", http://www.lustre.org/docs/obdspec.pdf, Stelias Computing, Inc., 1999.

[6] Peter J. Braam & Michael J. Callahan, "Lustre: A SAN File System for Linux", http://www.lustre.org/docs/luswhite.pdf, Stelias Computing, Inc., 1999.

[7] Christopher Whitaker, J. Stuart Bayley, Rod D. W. Widdowson, "Design of the Server for the Spiralog File System", http://research.compaq.com/wrl/DECarchives/DTJ/DTJM02/DTJM02HM.HTM, October 1996.

[8] S. Shepler et.al., "Request for Comments: 3010 - NFS version 4 Protocol", http://www.faqs.org/rfcs/rfc3010.html, December 2000.

[9] W. Richard Stevens, and Gary R. Wright, "TCP/IP Illustrated Volume 1: The Protocols", Addison Wesley, November 1993.

[10] C. Sapuntzakis, A. Romanow, and J. Chase, "The Case for RDMA", http://www.cs.duke.edu/~chase/draft-csapuntz-caserdma-00.txt, December 2000

[11] J. Pinkerton et.al., "WARP Architectural Requirements Summary", http://www.ece.cmu.edu/~ips/archive/draft-jpink-warp-summary-00.txt, January 2001.

[12] Emulex Corp., "GN9000/VI - VI/IP PCI Host Bus Adapter", http://wwwip.emulex.com/ip/products/gn9000VI.html, April 2001.

[13] QLogic, Inc., "QLA2300 Series - 2 Gigabit Fibre Channel", http://www.qlogic.com/products/qla2300.html, January 2001.

[14] Compaq, Intel, Microsoft, "Virtual Interface Architecture Specification", http://www.viarch.org/html/collateral/san_10.pdf, Version 1.0, December 16, 1997.

[15] DAFS Collaborative, "DAFS: Direct Access File System Protocol", http://www.dafscollaborative.org/tools/spec_v055.pdf, Version 0.55 February 26, 2001.

[16] Julian Satran et.al., "iSCSI", http://www.globecom.net/ietf/draft/draft-ietf-ips-iscsi-02.html, December 30, 2000.

[17] Red Hat, Inc., "TUX", Version2.0, http://www.redhat.com/support/manuals/TUX-2.0-Manual/index.html, March 2001.

[18] Ingo Molnar, "TUX patches", http://people.redhat.com/mingo/TUX-patches/, April 2001.

[19] Chuck Lever et.al., "An Analysis of the TUX web server", Center for Information Technology Integration, University of Michigan, http://citeseer.nj.nec.com/386260.html, November 16, 2000.
[20] InterMezzo, http://www.inter-mezzo.org/, December 2000.
[21] SNIA/T10, "SCSI OSD Command Set Proposal", http://www.snia.org/English/Work_Groups/OSD/WG_OSD_Docs.html, Revision 3, October 2000.
[22] Dave Nagle and Joan Digney, "Network Attach Secure Disks (NASD)", http://www.pdl.cs.cmu.edu/NASD/, July 2000.
[23] ASCI I/O SGPFS, http://www.llnl.gov/asci/sc99fliers/sgpfs_pg1.html, September 1999.
[24] IBM, "GPFS Primer", http://www.rs6000.ibm.com/resource/technology/paper2.html, December 1998.
[25] Sistina, Inc., "Global File System (GFS)", http://www.sistina.com/gfs/, 2001.
[26] IBM, "Distributed Lock Manager", http://oss.software.ibm.com/developer/opensource/linux/projects/dlm/?dwzone=linux, February 2001.
[27] Mark Hayden, "Ensemble Membership Service" http://www.cs.cornell.edu/Info/Projects/Ensemble/Maestro/groupd.htm, Cornell University, 1997 (?).
[28] Mission Critical Linux, "Kimberlite Clustering Technology" http://oss.missioncriticallinux.com/projects/kimberlite/, 2000.
[29] Roy Davis, VAX Cluster Principles, Digital Technical Press.
[30] Kirby Mccoy, VMS File System Internals, Digital Press 1990.
[31] Silicon Graphics, Inc., "Project XFS Linux", http://oss.sgi.com/projects/xfs/, May 2001.
[32] C. A. Thekkath, T. Mann, and E. K. Lee, "Frangipani: A Scalable Distributed File System", Proc. 16th SOSP, pp. 224-237, October 1997.
[33] M. Devarakonda, B. Kish and A. Mohindra, "Recovery in the Calypso file system", ACM Trans. on Computer Systems, Vol. 14, No. 3, 1996.
[34] S. Tweedie, P.J. Braam, M.Callahan and L. McVoy, "The Linux Cluster Cabal Papers" (authored by Tweedie and Braam), http://www.linux-ha.org/PhaseII/WhitePapers/, 1999.
[35] DOE National Nuclear Security Administration & the DOD National Security Agency, "STATEMENT OF WORK: SGS File System", Attachment A of RFP B514193, April 25, 2001.
[36] A. D. Birrell and R. M. Needham, "A universal file server", IEEE Transactions on Software Engineering, SE-6(5):450–453, September 1980.
[37] IBM, "Programming Locking Applications", http://www.rs6000.ibm.com/software/downloads/ha44clients.pdf, 2000.