# Lustre: The intergalactic file system

*Peter J. Braam*
*Philip Schwan*
Cluster File Systems, Inc.
*braam@clusterfs.com, http://www.clusterfs.com*

## 1 Introduction

This is a short overview of Lustre, a new open source cluster file system. The name Lustre embodies "Linux" and "Cluster." Lustre focusses scalability for use on large compute clusters, but can equally well serve smaller commercial environments. Lustre runs over different networks, including at present Ethernet and Quadrics.

Lustre originated from research done in the Coda project at Carnegie Mellon. It has seen interest from several companies in the storage industry that contributed to the design and funded some implementation. Soon after the original ideas came out, the USA NationaL Laboratories and the DOD started to explore Lustre as a potential next generation file system. During this stage of the project we received a lot of help and insight from Los Alamos and Sandia National Laboratories, most significantly from Lee Ward.

Lustre provides many new features and embodies significant complexity. In order to reach a usable intermediate target soon, Mark Seager from Lawrence Livermore pushed forward with Lustre Lite. We are hopeful that Lustre Lite will be the shared file system on the new 800 node MCR Linux cluster during 2002.

This paper provides a high level overview of Lustre.

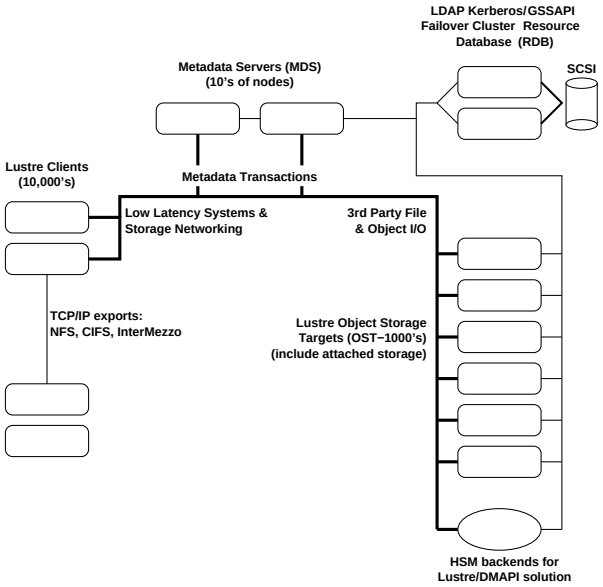



Figure 1: A Lustre Cluster

## 2 Lustre Components

In Lustre clusters there are three major types of systems, the Clients, the object storage targets (OST's) and metadata server MDS systems. Each of the systems internally has a very modular layout. Many modules, such as locking, the request processing and message passing layers are shared between all systems. Others are unique, such as the Lustre Lite client module on the client systems. Figure 1 gives a first impression of the interactions that are to be expected.

Lustre (see http://www.lustre.org) provides a

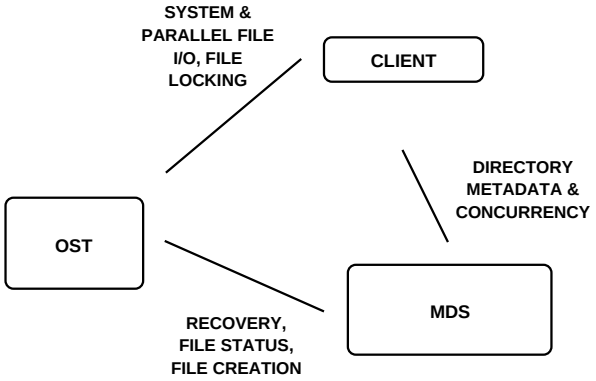



Figure 2: outline of interactions between systems

clustered file system which combines features from scalable distributed file systems such as AFS [1], Coda [2] and InterMezzo (see www.inter-mezzo.org), and Locus CFS [3], with ideas derived from traditional shared storage cluster file systems like Zebra [4], Berkeley XFS, which evolved to Frangipani Petal [5], GPFS [6], Calypso [7], InfiniFile [8] and GFS [9]. Lustre clients run the Lustre file system and interact with object storage targets (OST's) for file data I/O and with metadata servers (MDS) for namespace operations. When client, OST and MDS systems are separate, Lustre appears similar to a cluster file system with a file manager, but these subsystems can also all run on the same system, leading to a symmetric layout. The main protocols are described in figure 2.

## 3 Object Storage Targets

At the root of Lustre is the concept of object storage see [10]. Objects can be thought of as inodes and are used to store file data. Access to these objects is furnished by OST's which provide the file I/O service in a Lustre cluster. The name space is managed by metadata services which manages the Lustre inodes. Such inodes can be directories, symbolic links or special devices in which case the associated data and metadata is stored on the metadata servers. When a Lustre inode represents a file, the metadata merely holds references to the file data objects stored on the OST's.

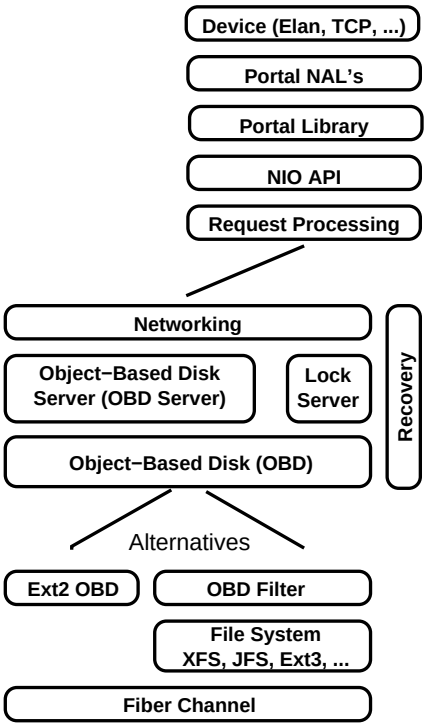



Figure 3: Object Storage Targets (OST)

Fundamental in Lustre's design is that the OST's perform the block allocation for data objects, leading to distributed and scalable allocation metadata. The OST's also enforce security regarding client access to objects. The client - OST protocol bears some similarity to systems like DAFS in that it combines request processing with remote DMA. The software modules in the OST's are indicated in figure 3.

Object storage targets provide a networked interface to other object storage. This second layer of object storage, so-called direct object storage drivers, consists of drivers that manage objects, which can be thought of as files, on persistent storage devices. There are many

choices for direct drivers which are often interchangeable. Objects can be stored as raw ext2 inodes by the *obdext2* driver, or as files in many journal file systems by the filtering driver, which is now the standard driver for Lustre Lite. More exotic compositions of subsystems are possible. For example, in some situations an OBD Filter direct driver can run on an NFS file system (a single NFS client is all that is supported).

In the OST figure we have expanded the networking into its subcomponents. Lustre request processing is built on a thin API, called the Portals API which developed at Sandia. Portals interoperates with a variety of network transports through Network Abstraction Layers (NAL). This API provides for the delivery and event generation in connection with network messages and provides advanced capabilities such as using remote DMA (RDMA) if the underlying network transport layer supports this.

## 4 Metadata Service

The metadata servers are perhaps the most complex subsystem. The provide backend storage for the metadata service and update this transactionally over a network interface. This storage presently uses a journal file system, but other options such as shared object storage will be considered as well.

The MDS contains locking modules and heavily exercise the existing features of journal filesystems, such as ext3 or XFS. In Lustre Lite the complexity is limited as just one single metadata server is present. The system still avoids single points of failure by offering failover metadata services, based on existing solutions such as Kimberlite.

In the full Lustre system metadata processing will be load balanced, which leads to significant complexity related to the concurrent access to persistent metadata.

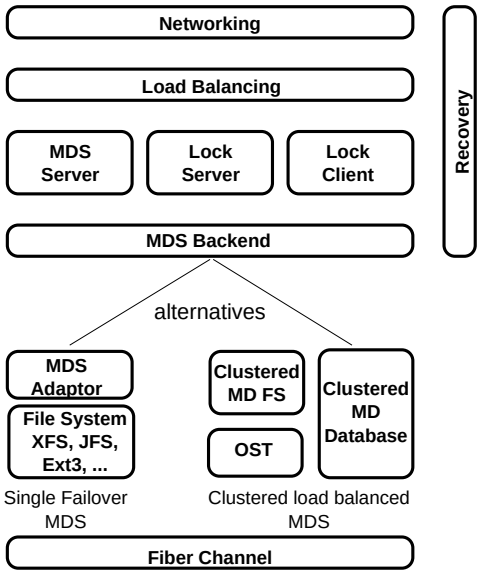



Figure 4: Meta Data Servers (MDS)

## 5 The client file system

The client metadata protocols are transaction-based and derive from the AFS, Coda and InterMezzo file systems. The protocol features authenticated access, and write-behind caching for all metadata updates. The client again has multiple software modules as shown in figure 5.

Lustre can provide UNIX semantics for file updates. Lock management in Lustre supports coarse granularity locks for entire files and subtrees, when contention is low, as well as finer granularity locks. Finer granularity locks appear for extents in files and as pathname locks to enable scalable access to the root directory of the file system. All subsystems running on Lustre clients can transparently fail over to other services.

The Lustre and Lustre Lite file system provide explicit mechanisms for advanced capabilities

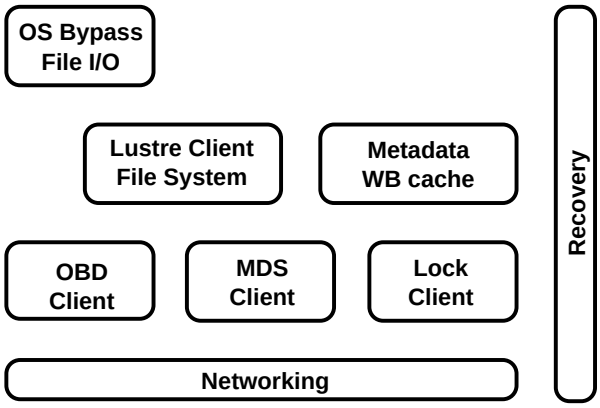



Figure 5: client software modules

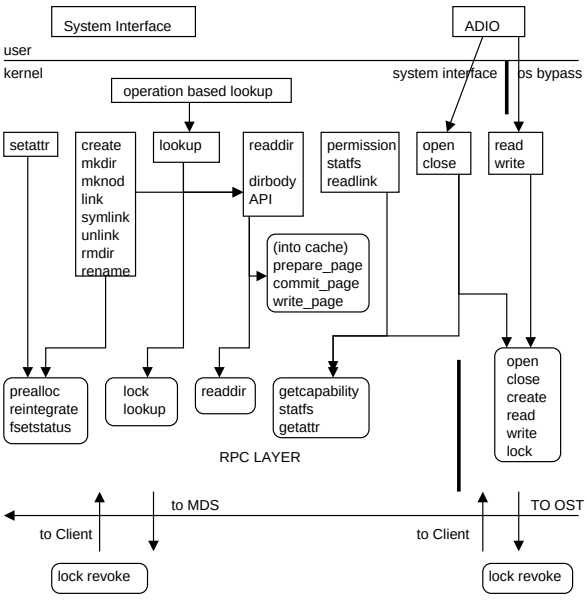
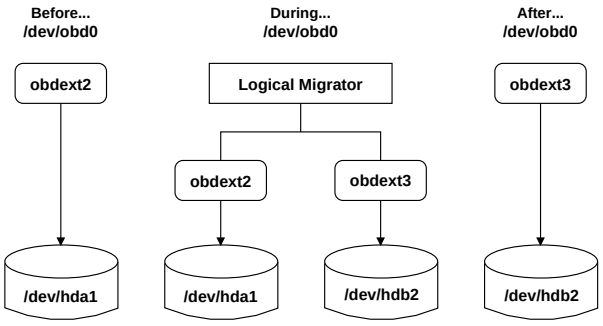



Figure 6: client file system internals




**Key Principle: dynamically switch object device types**

Figure 7: Hot Data Migration using Logical Object Volumes

such as scalable allocation algorithms, security and metadata control. In traditional cluster file systems, such as IBM's GPFS many similar mechanisms are found, but are not independent abstractions, but instead part of a large monolithic file system.

## 6 Storage Management

Lustre provides numerous ways of handling storage management functions, such as data migration, snapshots, enhanced security and quite advanced functions such as active disk components for data mining. Such storage management is achieved through stacks of object modules, interacting with each other. A general framework is provided for managing and dynamically changing the driver stacks.

An example of stacking object modules is shown in figure 7 for the case of hot data migration from one storage target to another.

## 7 Conclusions

Lustre provides a novel approach to storage. I heavily leverages existing techniques and software, yet breaks many patterns with new pro-

tocols, heavy modularization. The next few years will show if Lustre will establish itself as a mainstream element of the storage industry or will remain an exciting exploration in file system design.

## References

[1] J. H. Howard, "An overview of the andrew file system," in *In Proceedings of the USENIX Winter Technical Conference*, 1988.

[2] J. J. Kistler and M. Satyanarayanan, "Disconnected operation in the coda file system," in *Thirteenth ACM Symposium on Operating Systems Principles*, Asilomar Conference Center, Pacific Grove, U.S., 1991, vol. 25 5, pp. 213–225, ACM Press.

[3] Bruce J. Walker Gerald Popek, *The LOCUS Distributed System Architecture*, MIT Press, 1986.

[4] John H. Hartman and John K. Ousterhout, "The Zebra striped network file system," *ACM Transactions on Computer Systems*, vol. 13, no. 3, pp. 274–310, 1995.

[5] Chandramohan A. Thekkath, Timothy Mann, and Edward K. Lee, "Frangipani: A scalable distributed file system," in *Symposium on Operating Systems Principles*, 1997, pp. 224–237.

[6] IBM, "Gpfs," *FAST proceedings*, 2002.

[7] Murthy Devarakonda, Bill Kish, and Ajay Mohindra, "Recovery in the Calypso file system," *ACM Transactions on Computer Systems*, vol. 14, no. 3, pp. 287–310, 1996.

[8] Yoshitake Shinkai, Yoshihiro Tsuchiya, Takeo Murakami, and Jim Williams, "Alternatives of implementing a cluster file system," pp. 163–178, 2000.

[9] Steven R. Soltis, Thomas M. Ruwart, and Matthew T. O'Keefe, "The Global File System," in *Proceedings of the Fifth NASA Goddard Conference on Mass Storage Systems*, College Park, MD, 1996, p. ??, IEEE Computer Society Press.

[10] Garth A. Gibson, David Nagle, Khalil Amiri, Fay W. Chang, Eugene M. Feinberg, Howard Gobioff, Chen Lee, Berend Ozceri, Erik Riedel, David Rochberg, and Jim Zelenka, "File server scaling with network-attached secure disks," in *Measurement and Modeling of Computer Systems*, 1997, pp. 272–284.

# Proceedings of the
# Ottawa Linux Symposium

June 26th–29th, 2002
Ottawa, Ontario
Canada

## Conference Organizers

Andrew J. Hutton, *Steamballoon, Inc.*
Stephanie Donovan, *Linux Symposium*
C. Craig Ross, *Linux Symposium*

## Proceedings Formatting Team

John W. Lockhart, *Wild Open Source, Inc.*