Proseminar 2013
"Ein-/Ausgabe - Stand der Wissenschaft"
Universität Hamburg

September 30, 2013

Lustre File System

Paul Bienkowski
Author
2bienkow@informatik.uni-hamburg.de

Michael Kuhn
Supervisor
michael.kuhn@informatik.uni-hamburg.de

1 Introduction

File systems are a vital component in any computing system, having the task of storing and organizing data files on block devices, such as hard drives. File systems (FS) are generally a piece of software, often implemented as system drivers. There is a variety of different file systems for different purposes.

Lustre is a parallel file system. As such, it distributes the data across multiple nodes, usually independent hardware machines, and allows the access over network. The concept of Lustre is to be a well-scaling solution for High Performance Computing (HPC) clusters with a high amout of clients.

In this article I will explain the general idea of Lustre, its core architecture, as well as some performance measurings and recent improvements.

Contents

2 The Project

Lustre was started as a research project in 1999 by Peter Braam, who then founded **Cluster File Systems** to continue development on the project. The first version was released in 2003. Four years later, **Sun Microsystems** aquired Cluster File Systems, and in 2010, Sun itself was bought by the **Oracle Corporation**.

Oracle was not interested in continuing development or support of the project, so they discontinued it. Since Lustre is released under the terms of the GNU General Public License v2, many new organizations could continue development, including **Xyratex** and **Whamcloud** (part of **Intel** since 2012).

Several other organizations took over community management and funding of the project, including **OpenSFS** (Open Scalable File Systems), whose slogan is "keeping Lustre open", and **EOFS** (European Open File Systems), managing community collaboration. Other companies, such as **Xyratex**, **Dell** and **Terascala**, sell hardware storage solutions bundled with Lustre software. Braam and other contributors joined some of these companies and continue working on the project.

Lustre is actively being used in the HPC environment. According to hcpwire.com, more that 50 percent of the top 50 supercomputers manage their data using Lustre. This includes the **Titan**, today's fastest supercomputer by Cray.

Since Lustre is now being used in huge clusters, the priorities of the project have changed. While it started out as a science project, for production environments, the focus had to be shifted to stability instead of features and performance.

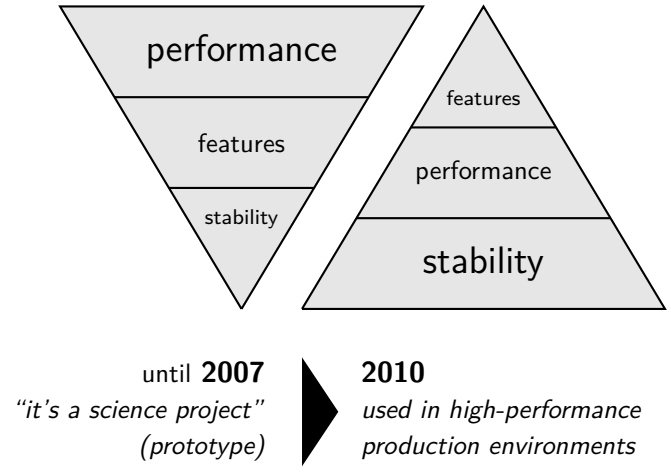


Figure 1: The change in project priorities

3 Lustre Architecture

3.1 Network Architecture

The Lustre architechtecture aims to connect a number of clients with the data servers in an efficient and failsafe way. The following graph shows an example structure of the network setup in a Lustre environment.

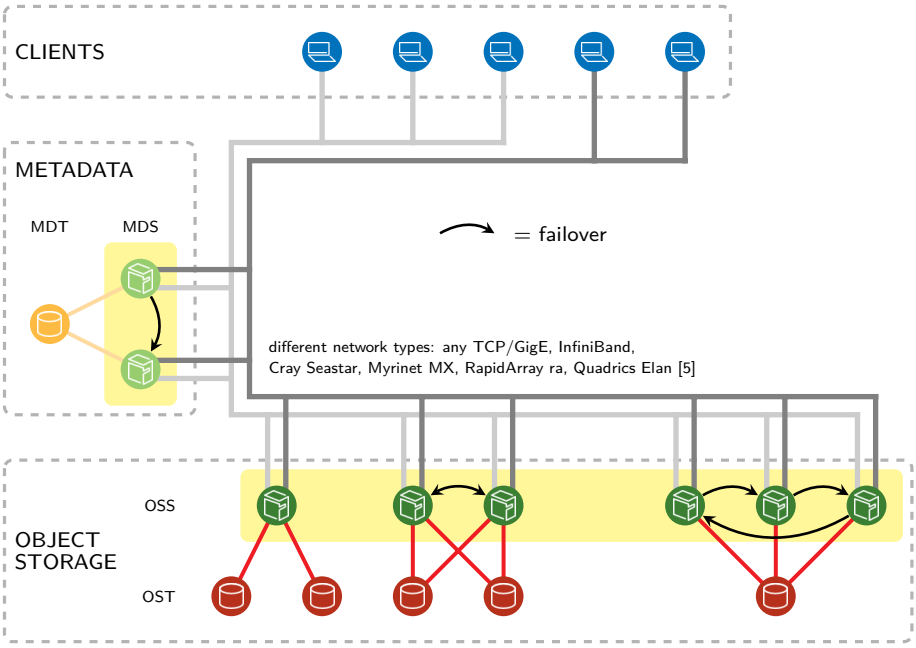


Figure 2: The Lustre network architecture

There are three types of nodes required for a Lustre network. The **clients** are the nodes that will receive access to the data storage. They have to be connected via any network (e.g. Ethernet or InfiniBand) to all the server nodes. Multiple network types are supported simultaneously, as long as every client can reach every server somehow.

The **metadata servers** (MDS) are responsible for every metadata management. They store file information in a structure very similar to the Inodes known from other file systems such as ext3 on a storage device attached to them, the **metadata target** (MDT). At least one metadata server is required for a Lustre system, but there can be multiple failover systems.

The **object storage servers** (OSS) are finally the nodes responsible for storing the actual file contents in so-called **objects** on the attached **object storage targets** (OST). Again, at least one OSS is required, but the number of possible nodes is not limited, with more nodes yielding a greater amount of attached storage and combined data throughput. In very large clusters, Lustre systems with more than 10,000 OSS have been created.

Both metadata and object storage targets can be any type of block device for raw byte

storage. This includes common hard disk drives (HDD), flash drives, solid state drives (SSD) as well as other, more advanced storage systems such as enterprise-class storage arrays. Each target will be formatted for use with Lustre, in older versions with a fork of the ext3/4 file system named **ldiskfs**, in more recent versions support for ZFS has been implemented.

To ensure avaiabiliy, a failover mechanism is used. When one server fails or is unreachable, another server may take over serving its targets. There is a number of different configurations, providing different levels of redundancy and performance. The failover mechanism can be utilized to upgrade the system while on-line, by taking one server off the network, upgrading it, and then upgrading the failover system.

3.2 File access

To access a file, the client first has to request the metadata for the file from the metadata server. This metadata contains the information, which parts of the file are stored on which OST. To read or write the actual file content, the client then accesses these OSTs directly, without the need to route the data through other nodes like the MDS.

Subsystem	**Clients**	**Object Storage**	**Metadata Storage**
System count	1-100,000	1-1,000	1-2
Performance	1 GB/s IO 1000 metadata ops	500 MB/s - 2.5 GB/s	3k-15k metadata ops
Required storage	–	total capacity / OSS count	1-2% of file system capacity
Desirable hardware	–	good bus bandwidth	fast CPU, plenty of memory

Table 1: Lustre subsystem characteristics

3.3 Striping

Striping is the technology used to distribute any data across multiple targets. The striping mechanism resembles a RAID-0 striping system, where a single file is being split into blocks, which are then evenly distributed across the target objects.

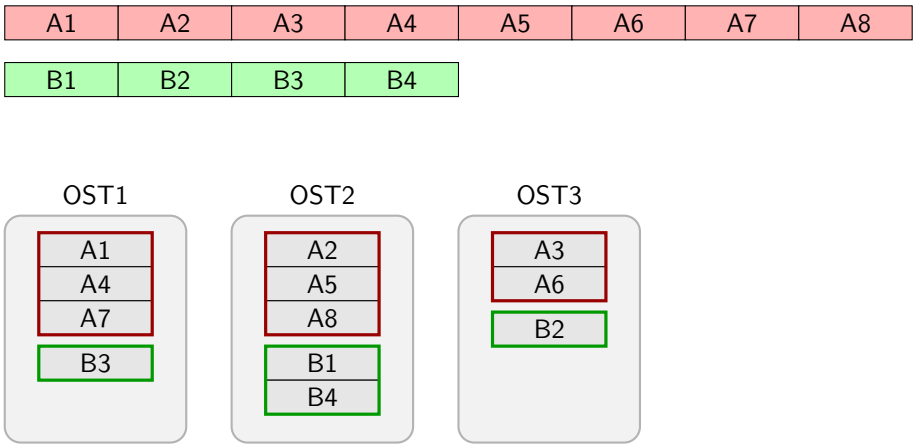


Figure 3: Striping

Figure 3 indicates how files are being split and distrubuted. The red and green blocks on top represent file data, which is split into eight and four blocks, respectively. The red file (file A) starts on OST1, and the blocks are placed on OST 1 through 3 in order. The resulting red block on OST1 consists of the data blocks a1, a4 and a7. This target block is called the **object** of file A on OST1. Each object is internally saved as a single file on the OST, using the filesystem present on that target.

The placement of file B (green) in this example starts on OST2, again creating three objects on the OSTs. Only the first object contains more than one data blocks, since the file is only four blocks in length.

Striping has multiple advantages for the Lustre architecture. Most importantly, it describes a mechanism to distribute the data across the OSTs. Thus, the client can write/read simultaneously to/from all OSSs involved, accumulating the bandwith and disk speed limits for a higher total throughput. By striping across multiple targets, the file size is also not limited to the size of a single target anymore, allowing the user to store huge amounts of data in a single file.

The block size and number of target objects is adjustable. In Lustre, these attributes can be set as metadata on a per-tree or per-file level, giving the administrator great control of the striping process and the ability to fine-tune the setup for the user's needs.

3.4 Data Storage and Access

Traditional filesystems like ext2 use so-called **inodes** (from "index node") to manage their file structure. The nodes are stored in a flat, non-hierarchical array, each accessed with its index. The nodes contain all the metadata of a file, as well as the location on the block device (disk). Every node corresponds to exactly one file, creating a bijective mapping.

Lustre uses similar nodes for metadata storage. The Lustre inodes contain information about the OSTs on which the file objects are located, as well as the object location inside the target. This information is partially stored in the **xattr** ("extended attributes") section of the inode, allowing for many referenced objects across all targets. The inode array itself is stored on the MDT.

3.5 Data Safety & Integrity

The safety and integrity of the stored data is an important aspect of HPC storage solutions. While the striping mechanism itself does not backup any data, other means of data protection can be implemented. The easiest available solution is to protect the targets, for example using RAID technology. Because Lustre supports any type of block device, the targets can not only be simple hardware drives, but may also be hardware or software RAID systems or even professional high-end storage solutions. These provide their own data safety mechanism, so the Lustre system does not require to manage data safety.

To ensure high availability, Lustre has two main features. The measure against server failure is the simple failover mechanism. This makes sure that each target is available in case of a single server failing. To counter network failures, caused for example by broken switches or routers, Lustre has support for multiple simultaneous network types. If a Lustre server is connected to a client via two or more networks, e.g. Ethernet and Infiniband, either one of them may fail without impeding the reachability.

The third problem in large, parallel computing environments is data consistency. Lustre implements simultaneous write protection using the Lustre Distributed Lock Manager (LDLM), which runs on the OSSs, allowing only one client to write a single file at any time.

For further consistency and integrity, the Lustre log, which is similar to the ext4 journal, can restore the file state in case of write failure, caused for example by a defect client or connection failure.

4 Software Architecture

4.1 Server Software

The Lustre servers have a variety of requirements and limitations to consider before installation.

The object and metadata storage targets are formatted using `mkfs.lustre`, which is using the ldiskfs filesystem. This filesystem is based on ext4, and requires some kernel patches only available for a limited number of Enterprise Linux 2.6 kernels, for example Red Hat.

This makes the server software very platform dependent, a normal Linux user cannot simply run a Lustre storage cluster "at home". This is not that much of a problem in HPC environments, though, since the kernel that runs on the storage servers is independent of the actual computing environment, allowing the clients to run the kernels required/desired for the computations.

Starting with Lustre 2.4, ZFS support has finally been implemented. This is a new and promising feature, that may not yet perform as well as the old ldiskfs versions, but supports a wider range of systems, more data integrity features [1] and is actively being developed and improved.

4.2 Client Software

For Lustre clients, there is a broad variety of solutions available to access the data. The basic implementation is a "patchfree" kernel module, available for any Linux 2.6 kernel. This module lets the user mount the remote file system using a native implementation, yielding the best performance results.

Other implementations include a userspace library (liblustre), that can be used to develop custom applications that directly interact with the storage system, a userspace filesystem driver (FUSE), which is important for OSX support, and NFS access for MS Windows support and legacy versions, as well as any operating system not supporting any of the above solutions. Thus, the Lustre client is available cross-platform, with best performances using the native filesystem module on Linux 2.6 or later.

4.3 Interversion Compatibility

The Lustre project usually supports some level of version **interoperability** [6]. This means for example that clients of version 1.8 can connect to servers running version 2.0 and vice versa. This feature is mainly used to be able to upgrade a Lustre storage system without taking it off-line, using the failover system, and upgrading the versions sequentially. During this process, some devices still run the old version while others run the new, in which case interoparability comes in handy.

[1] See `http://zfsonlinux.org/lustre.html` for detailled description on ZFS features

5 Performance

> *A well designed Lustre storage system can achieve 90% of underlining hardware bandwidth.*
>
> — Zhiqi Tao, Sr. System Engineer, Intel [3]

5.1 Theoretical Limits

In the following table, I present a theoretical Lustre environment setup with some key specifications, and the resulting, theoretical limits. These calculated values of course do not contain any performance limitations such as overhead, metadata access, or network capabilities, but rather show what would be the absolute limit.

System specifications		Theoretical performance	
OSS count	160	**Total storage**	2.5 PiB (Pebibyte)
OST/OSS	16	**Throughput per OSS**	800 MiB/s
OST size	2 TiB	**Total throughput**	125 GiB/s
OST throughput	50 MiB/s		

Due to the Lustre architecture, the resulting values come close to the presented limits. There are only a few bottlenecks that still remain, and these are actively being improved.

5.2 Recent Improvements

5.2.1 ZFS support

ZFS support comes with many improvements. Mainly, it allows for new kernels to run the servers without patching required. Furthermore, ZFS is more widely used by the open source community, thus being more actively developed and supported by a wider user base.

More ZFS features include checksums, support for up to 256 ZiB (Zebibyte, 10^21 Bytes) storage per OST, compression support and copy-on-write. All of these assist the performance and possibilities of a Lustre system.

5.2.2 Wide Striping

Until recently, the number of OSTs that stored a single file was limited to 160, due to the number of records available in an inode. This limit has been breached by storing the information in the extended attributes ("xattrs") array of the inode. ZFS also does not have this limit.

5.2.3 Multiple MDS

Since metadata access used to be the main performance bottleneck, work has been focused on improving this. Since Lustre 2.4, more than one simultaneous MDS are supported. This is implemented by either striping the metadata across multiple targets, similar to object striping, or namespacing the metadata. In a namespaced environment, some subtree (directory) of the file system would be handled by a different MDS, thus splitting the load.

These two methods provide a good way of fine-tuning the metadata system by the administrator according to the needs of the clients.

5.2.4 Metadata overhead

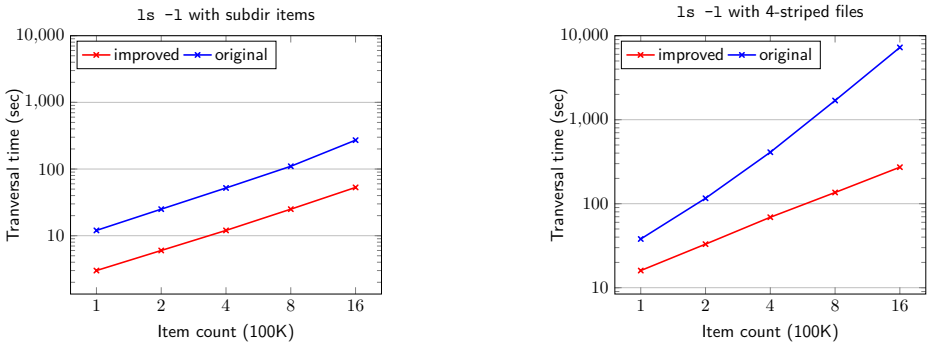
A common task in file system operation is directory traversal (`readdir`) and metadata readout (`stat`). This combination is used, for example, in the UNIX command `ls -l`.

The problem with this approach is that for each file in the directory, a separate RPC (`stat`) has to be sent and processed. This creates a lot of network overhead and I/O wait for directories with many files.

One solutions is the non-standard `readdirplus` command from the POSIX HPC I/O Extensions [11]. Lustre implements this extensions, and sents a single response including the metadata of the files.

Since not every client supports the HPC I/O Extensions, the Lustre MDS also detects a combination of `readdir` and `stat` calls and requests all stats from OSSs in advance, parallelized, then sending a combined RPC reply that can be up to 1 MiB in size.

The following graphs visualize the metadata performance of both the original POSIX directory traversal, and the improved Lustre detection system. One can see that the improved version is up to 27x faster (y-axis in log scale) than the original one on files that are striped across 4 OSSs, since the metadata is fetched in parallel.



graph data from [4]

5.2.5 SSDs as MDT

Since metadata performance is still the major bottleneck, suggestions have been made to better support solid state drives (SSDs) as metadata targets. These have a higher throughput, but most importantly an order of magnitude more IOPS, which is important for metadata seeks. Furthermore, the metadata storage requires relatively low capacity, which makes using the more expensive SSDs for MDTs affordable.

The graphs in Figure 4 visualize the analysis of different drive types for metadata storage for the three main metadata operations (create, stat, unlink). They were measured using both ldiskfs and ZFS (in development) versions, using 8 processes per client node.

The data shows that generally, the ZFS branch achieved the lowest performances. Since this is an early development branch (Orion-Lustre), recent improvements have to be expected. Furthermore, the ldiskfs-RAM line (green) displays the limit of what can be achieved independent of the drive used. One can notice that SSDs are better than HDDs, but not by the expected order of magnitude. Some research can be expected in this area. Also quite noticable is the apparent limit of stat calls per second, not rising above ~50,000, not even in RAM. This seems to be another bottleneck, where metadata striping or namespacing may yield better results due to distribution of the load across multiple servers.

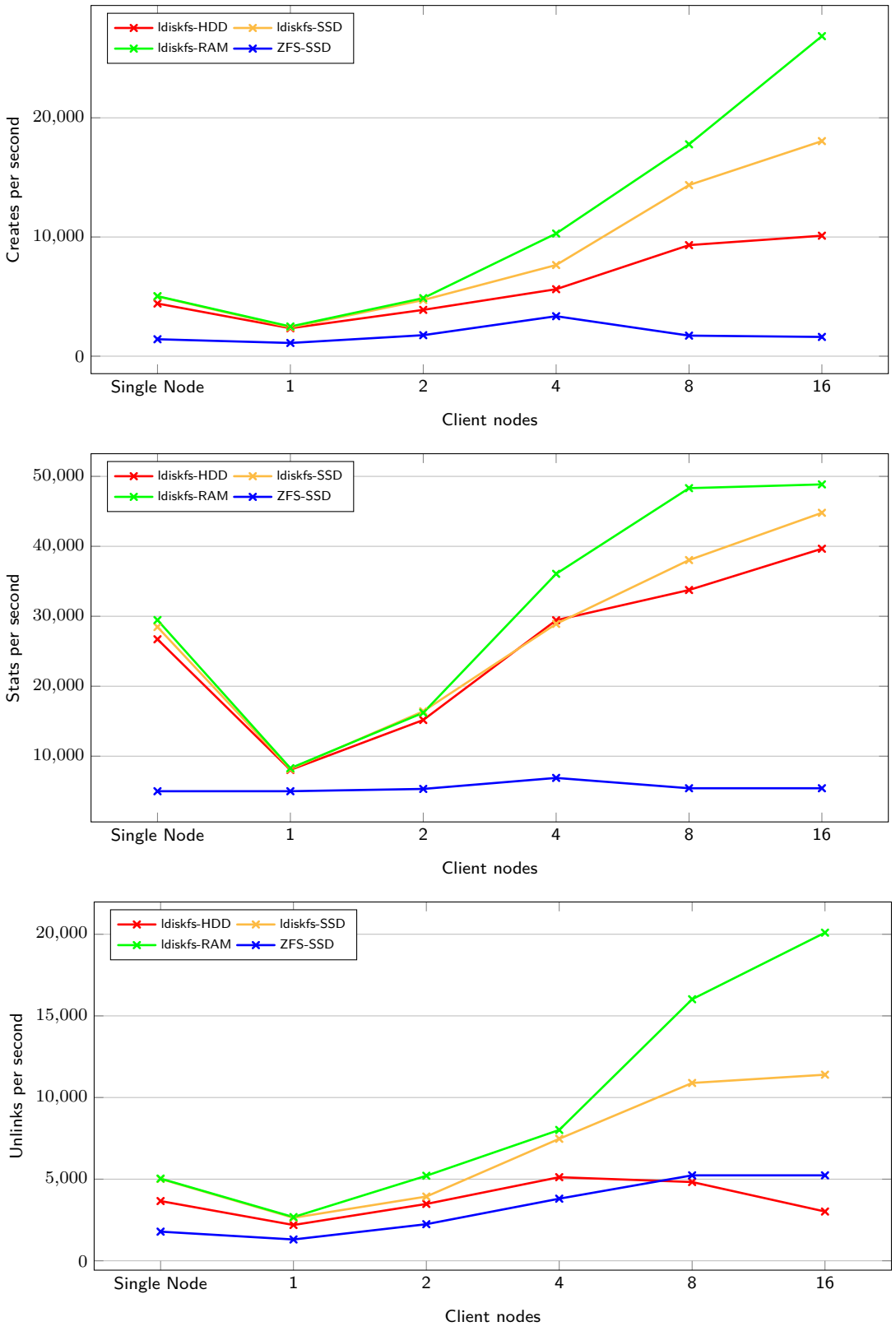


Figure 4: Metadata performance by number of clients using different drive types [10]

6 Conclusion

As a conclusion, the Lustre project is a promising and well-established storage solution for high performance computing environments. It is actively being developed by the Open Source Community as well as many interested companies that provide funding and employ developers.

It is used in many of the TOP-500 HPC clusters, scales exceptionally well in both size and throughput, only depending on the network architecture to support the required speeds.

There is still room for improvements in metadata performance though, but these bottlenecks are actively being tackled, leading to an even better product in the foreseeable future.

References

[1] http://www.raidinc.com/assets/documents/lustrefilesystem_wp.pdf 2013-05-17

[2] http://www.opensfs.org/wp-content/uploads/2011/11/Rock-Hard1.pdf 2013-05-17

[3] http://www.hpcadvisorycouncil.com/events/2013/Switzerland-Workshop/Presentations/Day_3/10_Intel.pdf 2013-05-21

[4] http://storageconference.org/2012/Presentations/T01.Dilger.pdf 2013-05-21

[5] http://wiki.lustre.org/images/3/35/821-2076-10.pdf 2013-05-28

[6] http://wiki.lustre.org/index.php/Lustre_Interoperability_-_Upgrading_From_1.8_to_2.0 2013-05-25

[7] http://wiki.lustre.org/index.php/FAQ_-_Installation 2013-05-12

[8] https://wiki.hpdd.intel.com/display/PUB/Why+Use+Lustre 2013-05-21

[9] http://www.hpcwire.com/hpcwire/2008-11-18/suns_lustre_file_system_powers_top_supercomputers.html 2013-05-28

[10] http://www.isc-events.com/isc13_ap/presentationdetails.php?t=contribution&o=2119&a=select&ra=sessiondetails 2013-05-31

[11] http://www.pdl.cmu.edu/posix/docs/POSIX-IO-SC05-ASC.ppt 2013-05-31

[12] http://zfsonlinux.org/lustre.html 2013-05-31

[13] http://www.olcf.ornl.gov/wp-content/themes/olcf/titan/images/gallery/titan1.jpg 2013-06-02