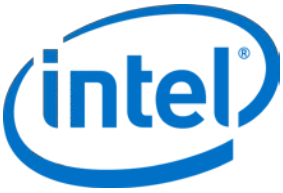
WHITE PAPER
Intel® Enterprise Edition for Lustre* Software
High Performance Data Division



Architecting a High Performance Storage System

January 2014

Contents

Summary

Designing a large-scale, high-performance data storage system presents significant challenges. This paper describes a step-by-step approach to designing such a system and presents an iterative methodology that applies at both the component level and the system level. A detailed case study using the methodology described to design a Lustre storage system is presented.

Introduction

A good data storage system is a well-balanced: each individual component is suited for its purpose and all the components fit together to achieve optimal performance. Designing such a system is not straightforward. A typical storage system consists of a variety of components, including disks, storage controllers, IO cards, storage servers, storage area network switches, and related management software. Fitting all these components together and tuning them to achieve optimal performance presents significant challenges.

Experienced storage designers may employ a collection of practical rules and guidelines to design a storage system. Such rules are usually based on individual experience; however they may not be generally

applicable, and may even be outdated due to recent advances in storage technology. For example, some designers consider it a poor practice to mix different manufacturer's hard disks in one RAID group, and that continues to be true. Another common rule says to fill only 80 percent of the available space in a disk enclosure, since the extra space may not be needed and the controller may not have the bandwidth to support the added capability. This latter rule may only apply in specific circumstances.

It is not always possible to design one system to perfectly meet all requirements. However, if we choose to start with one aspect of the design and gradually incorporate more aspects, it is possible to find the best balance between performance, availability, and cost for a particular installation.

A typical design process starts with a requirements analysis. The designer determines requirements in a top-down process that creates a complete view of the system. Once the design constraints are understood, the performance requirements can be determined at the component level. The design can then be built, one component at a time.

A Systematic Approach to Storage System Design

A high-performance storage system is part of a larger compute resource. Such a compute resource is generally a cluster of computers (compute nodes - CNs) connected by a high-speed network (HSN) to a group of disks that provide long-term storage for data. Applications running on the CNs either consume data (input) or produce data (output). The disks storing this data are generally organized in groups and served by one or more servers. Various architectures connect the hardware components in different ways and provide different software mechanisms for managing the data and access to it.

The designer planning the storage system for such a compute resource has the task of identifying the general structure of the storage system, specifying the components that will go into that general structure, and determining how those components will interact with the compute and network components.

Storage system design begins with creating a list of requirements that the system is to fulfill. This list may have several diverse requirements, such as:

- a fixed budget, with prioritizations on requirements, such as performance or capacity
- limits on power or space
- minimum acceptable performance (aggregate data rate)
- minimum aggregate storage space
- fault tolerance
- The ability to support a specific application workload

This will be a list of fixed and more flexible requirements, and many others are possible. One fixed requirement might set the specific minimum bandwidth that the design must meet. Then other, more flexible requirements may be adjusted in order to meet fixed requirements and meet the overall performance and cost goals.

The overall storage system design will specify the kinds of components to be employed and how they will be connected. Creating this design can be a challenging task. Design choices may be constrained by practical considerations respecting the needs of the customer or vendor partner.

This paper begins by selecting an overall design structure, although other structures are possible. How one chooses among these basic design structures is beyond the scope of this paper, but here are a few ways one might do so:

- An experienced designer may have guidance about the best structure to meet the primary requirements.
- A reference system may have already been deployed and found to meet a set of similar requirements.
- A review of case studies such as the study in the second half of this paper may provide guidance to the novice designer.

For this paper, we've selected a relatively common reference design structure for our storage system. Our task is to create from that reference design, a complete design for our target storage system. Figure 1 depicts our selected reference design structure. Other structures are possible.

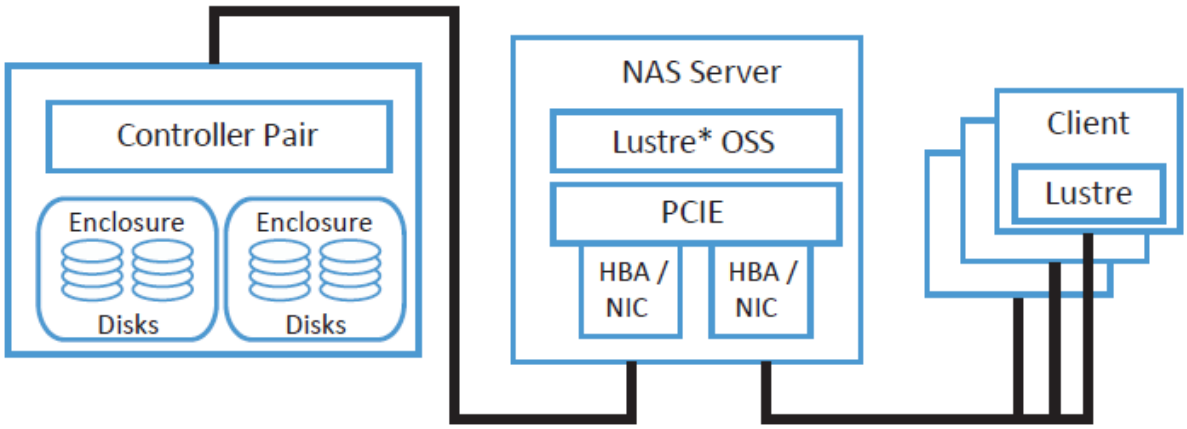


Figure 1. A General Storage Architecture.

Before the design is complete, it needs to specify the number and type of every component, and identify to what extent the design meets the requirements. As design choices are made, a choice may lead to a design that does not meet the requirements and/or impacts other choices. In such a case, one will need to iterate over the choices to improve the design. The following design methodology uses a step-by-step "pipeline" approach for examining and selecting each component.

Evaluating Components - the Pipeline Approach

Our design methodology uses a "pipeline" approach for examining each component. This approach evaluates components in order, by following the path of a byte of data as it flows from a disk, through the intervening components, to the application. Other orderings are possible, but this paper confines itself to this read pipeline.

The entire pipeline's performance is governed by the performance of its individual components, and system performance is limited by the slowest component. Exceptions will only be brief, transient departures from what is otherwise a steady flow of data, limited by the slowest component. Thus, we need to consider each component individually.

First, we examine the storage media. Next, the storage controller is examined together with the disks as a composite. The performance of these two components taken together will not be better than the performance of the individual components, and generally will be worse due to inevitable inefficiencies in their operation.

We continue this process, adding one component at a time to the composite, until the pipeline is complete.

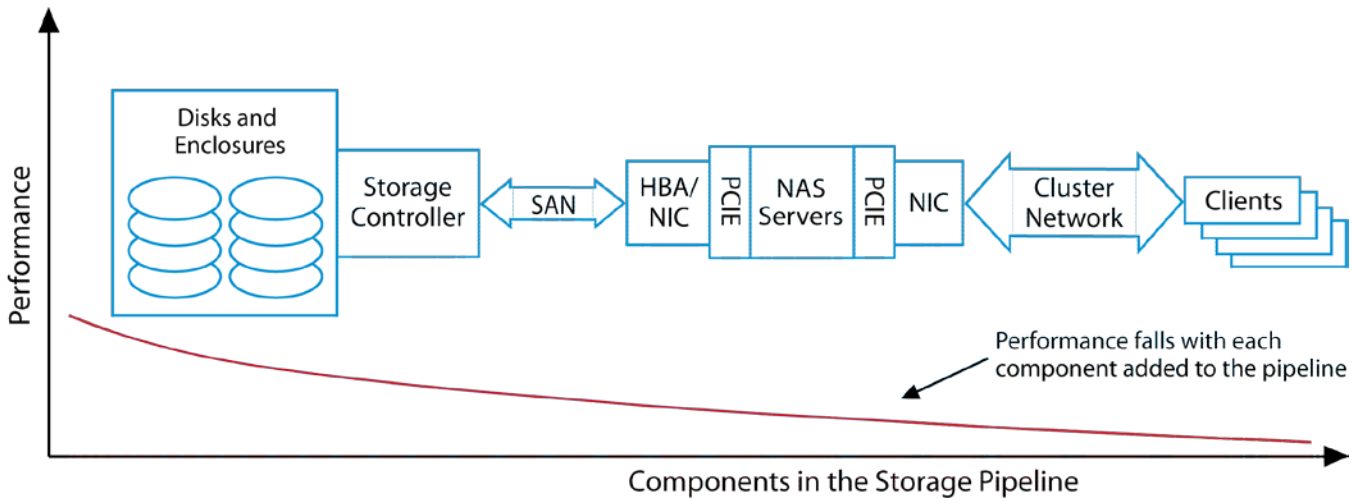


Figure 2. A storage pipeline

Figure 2 arranges the components from Figure 1 in order, from left to right, following the read pipeline. The performance line, beginning on the left, represents the performance as each successive component is added to the pipeline. For example, when the storage controller is added, some small inefficiency may cause the two components (disks and controller) to perform a little below the value for the disks alone. This is represented by the small step down for the line below the controller. The line shows a decrease in performance (or ideally, stays about the same) with the addition of each new component to the design.

One caveat to this approach is that the introduction of a new component may cause us to rethink the design of a previous component. For example, if the number of disks just satisfies the performance and capacity requirements, but together with the controller the performance drops below the requirement, we may need to backtrack and redesign the disk portion. Further if, we know that the controller can easily handle more disks, this may motivate us to consider provisioning more disks, in anticipation of performance bottlenecks that may occur later in the design. For the designer new to this activity, this may lead to significant backtracking to get the end-to-end design just right. An experienced designer may modify the design in anticipation of such backtracking, and the case study in the second half of this paper shows an example of that.

This paper does not address performance benchmarking. Possible targets for benchmarking and relevant applications are mentioned in passing. Some individual components cannot be tested in isolation, but a systematic approach to benchmarking methodology can allow the designer to infer the capability of an individual component. A component's performance can be acquired by testing, or by finding such results documented by others.

Using an Iterative Design Process

There are many components that go into a storage system. Accordingly, the design process needs to be methodical. Breaking the process into discreet steps makes it a straightforward activity of iterating a simple procedure that incorporates successively more components to best meet requirements.

This design process introduces a component, selects

its properties, combines it with the previously designed pipeline of components, and evaluates to what extent, the new pipeline meets the system requirements. Several cycles of selection, combination, and evaluation (S-C-E) will be needed before the design is complete.

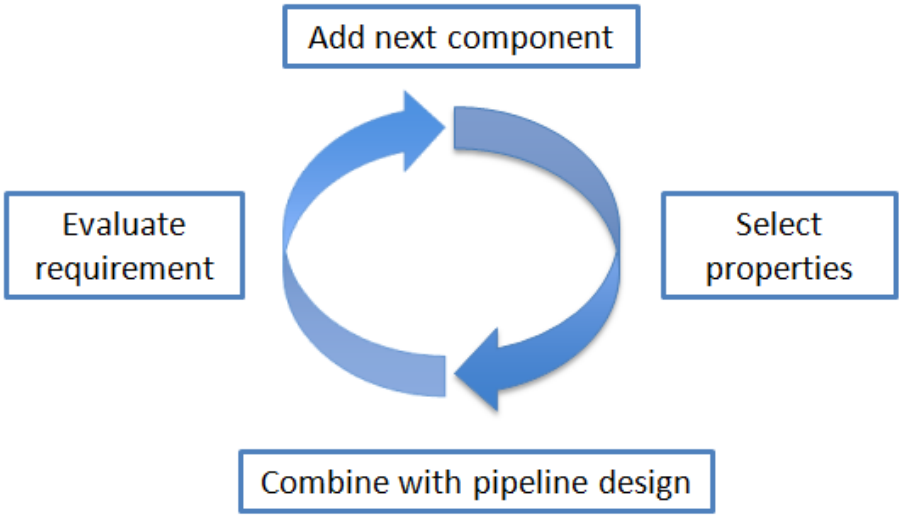


Figure 3. The iterative design approach

Figure 3 depicts the S-C-E cycle. At the *Add next component* step, we introduce the next component in the pipeline. Next, we *select properties* of the component that may satisfy the requirements. The third step for the component is to add it to the pipeline (*Combine with pipeline design*). Finally, we *evaluate the requirement* to see if the design thus far meets that requirement.

It may be that choices in previous iterations locked the pipeline into a design that cannot meet the requirements. At such a point it is usually apparent where the faulty choice was, so the process can backtrack to select a better component that will meet system requirements. This will come up in the case study presented next.

A Case Study Using the Lustre File System

In the following case study, we'll design a storage system for a high performance compute cluster.

Analyzing the Requirements

Analysis of this hypothetical storage system identified the following requirements:

- A single namespace
- 10 PB (10 X 10245 bytes) of usable space
- 100 GB/s (100 X 10243 bytes per second) aggregate bandwidth
- Ability to support access by 2000 clients in parallel
- No single point of failure

Table 1 summarizes the capabilities offered by Lustre.

The requirements fall well within the capabilities of Lustre, so Lustre is a good choice for this system.

Table 1. Suitable Use Cases for Lustre*

Storage System Requirements	Lustre File System Capabilities
Large file system	Up to 512 PB for one file system.
Large files	Up to 32 PB for one file.
Global name space	A consistent abstraction of all files allows users to access file system information heterogeneously.
High throughput	2 TB/s in a production system. Higher throughput being tested.
Many files	Up to 10 million files in one directory and 2 billion files in the file system. Virtually unlimited with Distributed Name Space.
Large number of clients accessing the file system in parallel	Up to 25,000+ clients in a production system.
High metadata operation rate	Support for 80,000/s create operations and 200,000/s metadata stat operations.
High Availability	Works with a variety of high availability (HA) managers to support automated failover to meet no-single-point-of-failure *(NSPF)* requirements.

Designing and Building the Pipeline

Starting at the storage end of the pipeline shown in Figure 2, the disks and disk enclosures are designed to meet system requirements. Then the next component, the storage controller, is added to the design and adjustments made to ensure the two components together meet requirements. Then the next component is added, adjustments are made again, and so on, until the pipeline is complete.

Disks and Disk Enclosures

Our example storage system requires a disk configuration that delivers 10 PB of usable storage and 100 GB/s of aggregate bandwidth. Usable storage means the storage that the client sees when the file system is mounted. The usable storage capacity of a disk is less than its physical capacity, which is reduced by such factors such as RAID, hot spares, etc. In a Lustre file system, an IO operation may access metadata storage or object storage. Each storage type has its own characteristic workload, which must be taken into account in the design for that storage.

Metadata storage stores information about data files such as filenames, directories, permissions, and file layouts. Metadata operations are generally small IOs that occur randomly. Metadata requires a relatively small proportion, typically only 1-2 percent, of file system capacity.

Object storage stores the actual data. Object storage operations can be large IOs that are often sequential.

A reasonable starting point for designing metadata storage or object storage is to consider the capacity and performance characteristics of the available storage devices. For this example, we consider the Seagate* hard disk drives shown in Table 2.

Table 2. Comparing Hard Disks

Specifications	Seagate 15K.7 HDD (600 GB, 15,000 RPM)	Seagate ES.2 HDD (3 TB, 7200 RPM)	Seagate ES HDD (1 TB, 7200 RPM)
Average IO/s	178	76	76
Sustained data transfer rate (MB/s)	208	155	147

As shown, the rotational speed makes a significant difference in disk performance. The 15,000-RPM hard disk offers twice the input/output operations per second (IO/s) and about 30 percent more bandwidth than the 7,200-RPM hard disk. The 7,200-RPM 3 TB hard disk and the 7,200-RPM 1 TB hard disk have roughly similar performance characteristics.

The number of spindles can also affect performance. For example, although three 1-TB disks and one 3-TB disk offer the same capacity, three 1-TB disks have three times the bandwidth and support three times the IO/s compared to one 3-TB disk.

For the example in this paper, two types of disk enclosures are considered, a 12-disk enclosure and 60-disk enclosure. Both disk enclosures use the same type of SAS-2 interface for connecting to the storage controller. Because the metadata storage requires smaller space but high IO/s, the 12-disk enclosure will be considered for the metadata storage. The 60-disk enclosure appears more appropriate for the object storage if it provides enough bandwidth.

To address the fault tolerance requirement, both the metadata storage and the object storage will be configured into appropriate RAID groups. Capacity and performance are the two primary determining factors when selecting a RAID level.

In this example, the object storage will be designed to provide usable space for data storage, and then the metadata storage will be designed to support the object storage.

Configuring Object Data Storage

For the object storage, capacity is of primary interest because the capacity of the object storage determines the usable space for storing data. However, performance is also an important consideration because the IO throughput directly impacts the performance of the file system. This study prioritizes capacity over performance; however in other cases, performance may be the driving factor.

For object data storage, we'll consider HDD disks (4-TB, 7200-RPM) that will be located inside enclosures capable of holding sixty disk drives (see Table 2). First, the RAID configuration for the disks must be determined. The configuration must be designed for high disk utilization first, and then optimized for performance. A ten-disk, RAID-6 group (eight data disks + two parity disks) allows 80 percent disk utilization while accommodating two disk failures.

The RAID-6 configuration can result in a write performance penalty when the data size does not match the stripe size. In this case, the controller must read old parity and old data from the disk and then use that information in combination with the new data to calculate the new parity. This is operation is known as "read-modify-write".

To mitigate this performance penalty, "full stripe write" is often used, in which the data size matches the stripe size so that parity can be calculated directly without reading information from the disk first. Because Lustre reads and writes data in 1-MB segments, the RAID segment size is set to 128 KB, so that 1 MB of data is striped across all the data disks.

The disks in each 60-disk enclosure can be configured into six, 10-disk, RAID-6 groups. Each RAID-6 group provides approximately 32 TB of storage, so each enclosure provides approximately 192 TB of storage.

Note that disk capacity is stated in decimal GB (where 1 GB = 1,000,000,000 bytes) while file system capacity is stated in binary GB (where 1 GiB = 1,073,741,824 bytes). This needs to be taken into account in these calculations.

$$\frac{TB}{TB} = \frac{10^{12}\ B}{2^{40} B} \cong 0.91$$

Plus, the root file system will reserve 5%. Therefore, each enclosure can provide 164 TB of usable space.

$$192 \times 0.9 \times 0.95 \cong 164\ TB$$

The number of enclosures needed for 10 PB of usable space is:

$$\frac{10 \times 1024\ TB}{164\ TB\ /enclosure} = 62.43$$

To satisfy the requirement for 10 PB of usable space, 63 disk enclosures will be used for this design.

The next step is to consider performance requirements starting with this initial design. To reach 100 GB/s aggregated bandwidth, the bandwidth each disk enclosure must contribute is:

$$\frac{100\ GB/sec}{63\ enclosures} \cong 1.6\ GB/sec$$

Since each enclosure has six RAID-6 groups, with each containing eight data disks, and the Seagate ES.2 disks in the enclosure support a theoretical sustained data transfer rate of 155 MB/s, the total theoretical bandwidth of the 48 data disks is:

$$155\ MB/s\ x 48 = 7.4\ GB/s$$

However, the theoretical bandwidth is difficult to achieve in actual use because the efficiency of the RAID controller and the bandwidth of the disk controller must also be taken into account. The actual throughput of a RAID-6 storage enclosure may be less than half of the theoretical throughput per disk. For example, 3 GB/s per enclosure if the SAS expander is the limiting factor, which is enough to exceed the 1.6 GB/s minimum requirement.

As most disk enclosures connect to the storage controller via a 4-lane Serial Attached SCSI-2 (SAS-2) cable with a performance of 6.0 Gb/s per lane, the maximum possible bandwidth out of a single disk enclosure is calculated as follows:

$$6\ GB/s\ \times\ 4\ lanes\ \times \frac{1\ B}{8\ b} = 3\ GB/s$$

Thus, the connection between the disk enclosure and the storage controller determines the aggregate bandwidth that can be achieved in this design. This bandwidth exceeds the bandwidth contribution of

1.6 GB/s required by each of the 62 enclosures to achieve the 100 GB/s aggregate bandwidth for the system.

The bandwidth of the disks themselves is twice that of the enclosure connection to the storage controller, so disk performance is not a limiting factor in this design.

The number of disk enclosures is also not a limiting factor.

Note, however, that if the system bandwidth requirement were tripled to 300 GB/s, each of the 63 disk enclosures would need to provide 4.76 GB/s bandwidth, which is higher than what the 3 GB/s disk enclosures used in this design can support. In that case, the disk layout would need to be redesigned to add more disk enclosures. For example, 2-TB, 7200-RPM disks could be substituted for the 4-TB disks used in this design to achieve better performance. Doing so would spread the same storage capacity over more enclosures.

Configuring Metadata Storage

For metadata storage, performance is of primary interest. This is because for each IO operation, file attributes must be obtained from the inode for the file in the metadata storage before the file can be accessed in the object storage. Thus, metadata access time affects every IO operation. However, metadata storage capacity requirements are typically small relative to capacity requirements for object storage, so capacity is unlikely to be a significant concern.

Since the majority of metadata operations are queries (reads), a RAID-10 configuration is best suited for metadata storage. RAID-10 combines features of RAID-1 and RAID-0. RAID-1 provides data mirroring by configuring disks in pairs to mirror each other. RAID-0 enhances performance by striping data across two RAID-1 pairs. Because two disks in the RAID-10 set contain the same content, data can be alternately read from both of disks, effectively doubling read performance, which significantly contributes to overall system performance. Unlike RAID-5 or RAID-6, RAID-10 has no parity drives and, thus, no "read-modify-write" penalty. The segment size of the metadata storage can be set to 4 KB, which matches the metadata IO size.

To properly determine the size of the metadata storage, it is important to understand what the average file size will be. For this example, the average file size is assumed to be 5 MB. The minimum number of inodes that will be stored can be calculated by dividing the required object storage capacity by the average file size:

$$10\ PB / 5\ MB\ per\ inode = 2\text{x}10^9\ inodes$$

For future expansion, space should be reserved for twice the minimum number of inodes. Since each file's metadata in Lustre 2.1 or a newer release can need up to 2 KB of space, metadata storage must be at least:

$$\frac{2\ KiB}{inode}\ x\ \frac{2x10^9\ inodes}{2^{30}}\ x\ 2 \cong\ 7.5\ TB$$

For this design, Seagate 15000–RPM, 600-GB disks will be used for the metadata storage. See Table 2. The required number of disks can be determined by dividing the required metadata storage by the disk capacity, and multiplying by two (because the disks will be configured as RAID-10 mirrored pairs):

$$\frac{2 \times 7.5TB}{600\ GB \times \frac{10^9}{2^{30}} \times 0.95} \cong 29\ disks$$

Recall that disk capacity is measured in decimal GB (10^9 bytes) while file system capacity is measured in binary GB (2^{30} bytes).

As 4 billion inodes was the estimate based on assumed average file size (2 billion inodes X 2), for the sake of architecture simplicity, 30 disks are used for the rest of this discussion. If there were the exact file count requirements, the same calculation applies.

Three, 12-disk enclosures will be needed to make 30 disks available for inode storage, leaving the remaining six disks slots for hot spare devices (two per enclosure).

Figure 4 shows the storage configuration for the metadata storage.

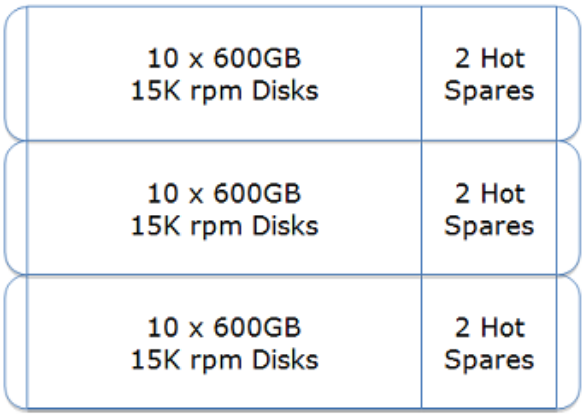


Figure 4. Metadata storage system

Storage Controllers

A storage controller manages disks and disk enclosures. Its basic function is to make disk space available as block devices to the storage area network (SAN). The storage controller serves as a gateway, connecting to the disk enclosures via SAS-2 cables.

The controller provides one or more types of storage network interfaces to the SAN, such as Fiber Channel (FC), 10GbE, or InfiniBand (IB). As a gateway, a storage controller aggregates the bandwidth of the backend disks/enclosures as it passes data between the backend disks and the storage servers. Therefore, it is important to understand the performance characteristics of the storage controller to be used.

For this example, a general-purpose storage controller is used. The controller has eight SAS-2 ports to connect the backend disk enclosures. It also has eight 8-Gbps Fibre Channel ports and four SAS-2 ports, one or the other of these two port types can be used at a time, to connect the storage servers.

To calculate the number of storage controllers needed to support the backend disk enclosures, use these considerations:

- Each of the 63 object storage enclosures must provide a bandwidth of at least 1.6 GB/s.
- One 4-lane, SAS-2 cable can support 3 GB/s.

Controller hardware or software can fail due to external or internal causes. Thus, it is common for a controller to be composed of two sub-controller modules that serve as backups for each other, as shown in Figure 5.

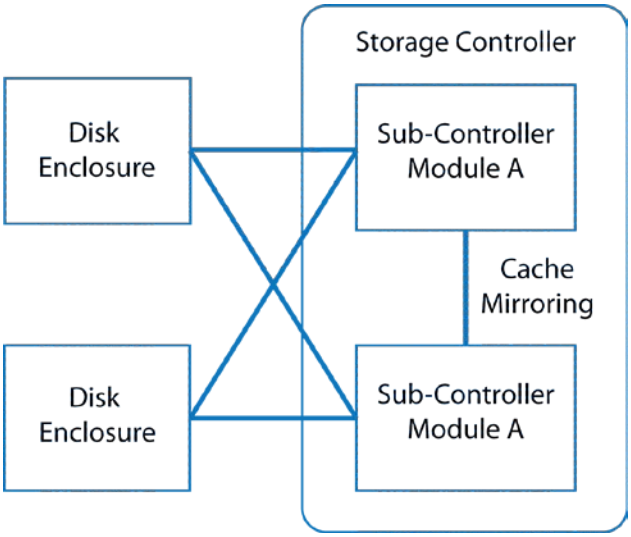


Figure 5. Cache mirroring between sub-controller modules

To optimize performance, the two sub-controller modules are normally in an Active-Active configuration. Many controllers have built-in cache mechanisms to improve performance. Caches must be coherent between sub-controller modules so that either sub-controller module can transparently take over the I/O from the other sub-controller module. A cache-mirroring link often exists between sub-controllers for this purpose.

This cache-mirroring link can be a limiting factor because data must be mirrored to the cache of the second sub-controller module before the first sub-controller can return an acknowledgement indicating a successful IO operation.

Cache-mirroring links are implemented using a variety of means, from gigabit Ethernet cables to a dedicated PCIe bus. Because the performance degradation due to cache mirroring has become more widely

acknowledged, many vendors have enhanced the cache-mirroring link to support a bandwidth that is at least equivalent to that of a quad-data-rate Infiniband cable. Nevertheless, it is important to examine the storage controller architecture as part of the design process.

Note that in the case where there is no cache-mirroring link between the controllers, the cache on the controllers must be disabled entirely to ensure that the file system does not become corrupted in the event of a controller failure.

Our example assumes that the cache-mirroring link between the sub-controllers in the selected storage controller limits the overall bandwidth of the storage controller to 6 GB/s.

Each enclosure can support 1.6 GB/s bandwidth. Three enclosures will deliver 4.8 GB/s bandwidth, while four enclosures will deliver 6.4 GB/s, which exceeds the limit of 6 GB/s. So in this case, the bandwidth provided by three enclosures will best match each storage controller's bandwidth. The number of storage controllers required is 21.

Storage Servers

A storage server serves files for access by clients. A storage server differs from a storage controller in that the storage server exports file system space rather than block devices. The content in the shared file system space on a storage server is visible to all clients and the storage server coordinates parallel, concurrent access by different clients to the files. In contrast, a storage controller cannot coordinate access by multiple clients to the block devices it exports due to limitations of the block device protocols. Potential conflicts must be managed at the client level, making parallel access by clients to stored data more complex.

Many storage servers consolidate the functions of a storage server and a storage controller into one physical machine. This section discusses the limitations that storage servers can introduce, regardless of whether they reside on physically separate hardware or not.

Designing the Lustre Metadata Server

A Lustre file system includes two types of servers, a metadata server (MDS) and one or more object storage servers (OSSs). The metadata server must be able to quickly handle many remote procedure calls (RPCs) because the MDS is the starting point for all POSIX file system operations, such as open, close, read, write, and unlink. Whenever a Lustre client needs to access a file from a Lustre file system, it queries the metadata target (MDT) via the MDS to obtain the POSIX attributes for that file (e.g., owner, file type, access permission) and the file data layout (e.g.: how many OSTs the file is striped over and the specific objects that make up this file).

Therefore, the MDS needs powerful CPUs to handle simultaneous inquiries, and a large amount of RAM to cache the working set of files and avoid high-latency disk operations.

Because IO operations on the MDT are mostly small and random, the more data that can be cached into memory, the faster the MDS will respond to client queries. As a result, the number of clients and the number of files the clients are accessing in their working set, determine the amount of memory required by the MDS [5].

Apart from 1 GB required by the operating system and 4 GB required for the file system journal, about 0.1% of the MDT size is needed for the MDT file system's metadata cache. The remaining RAM is available for caching the file data for the user/application file working set. The working set cached in RAM is not always actively in use by clients, but should be kept "hot" to reduce file access latency and avoid adding extra read IO/s to the MDT under load.

Approximately 2 KB of RAM is needed on the MDS for the kernel data structures, to keep a file in cache without a lock. Every client that accesses a file also needs approximately 1 KB of RAM for the Lustre Distributed Lock Manager (LDLM) lock, resulting in about 3 KB of RAM required for each file in cache. A typical HPC working set might be 100 files per CPU core.

Table 3 shows how required MDS memory is calculated for the Lustre file system in this example. This file system has:

- one MDT on an active MDS
- 2,000 8-core compute clients
- 64 user-interactive clients (with a considerably larger working set)
- a hot cache supporting an additional working set of 1.5 million files.

Table 3: MDS RAM Calculation

Memory Consumer	Required Memory
Operating system overhead	1,024 MB
File system journal	4,096 MB
MDT file system metadata (0.1% of 8192 GB)	8,192 MB
2000, 8-core clients X 100 files per core X 3 KB/file	4,687 MB
64 interactive clients X 10,000 files X 3 KB/file	1,875 MB
2-million-file working set X 1.5 KB/file	2,929 MB
Total	**22,803 MB**

The minimum memory for the MDS server for a file system with this configuration is 24 GB RAM. However, the example shows that with larger numbers of interactive clients and larger working sets of files, additional RAM will be required for optimal performance. 128 GB RAM or more is often used for better performance.

Designing the Object Storage Servers

Storage servers are usually equipped with IO cards that either communicate with the back end storage controllers or with clients at the front end. Commonly-used storage protocols are SAS, FC, and iSCSI, while client network protocols are usually IB or 10GigE. Typically, a SAS or FC host bus adapter (HBA), and an IB host channel adapter (HCA) or 10GbE NIC are installed on storage servers. Even if both the storage and client networks are run over Infiniband, separate physical networks should be used to avoid contention during IO, when both channels will be used at the same time.

For the example in this case study, to calculate how many OSS nodes are required and how the object storage targets (OSTs) will be distributed among them, consider that the storage controller needs to aggregate at least 4.8 GB/s from the backend disks. To accommodate this bandwidth, two SAS-2 ports on each storage server can be used to connect the storage server to the storage controller.

Because the OSS requires a high-availability design, the number of IO cards at the backend must be doubled to be able to support failover. The SAS cards are more efficient than the FC cards in terms of PCIe slot utilization, so they will be used for this design.

Figure 6 shows how active-active pairs of OSSs are connected to two each of the 21 storage controllers. Here, 21 OSSs will be needed at a minimum.

However, because we should also consider failover between each two OSS nodes, 22 OSSs are required.

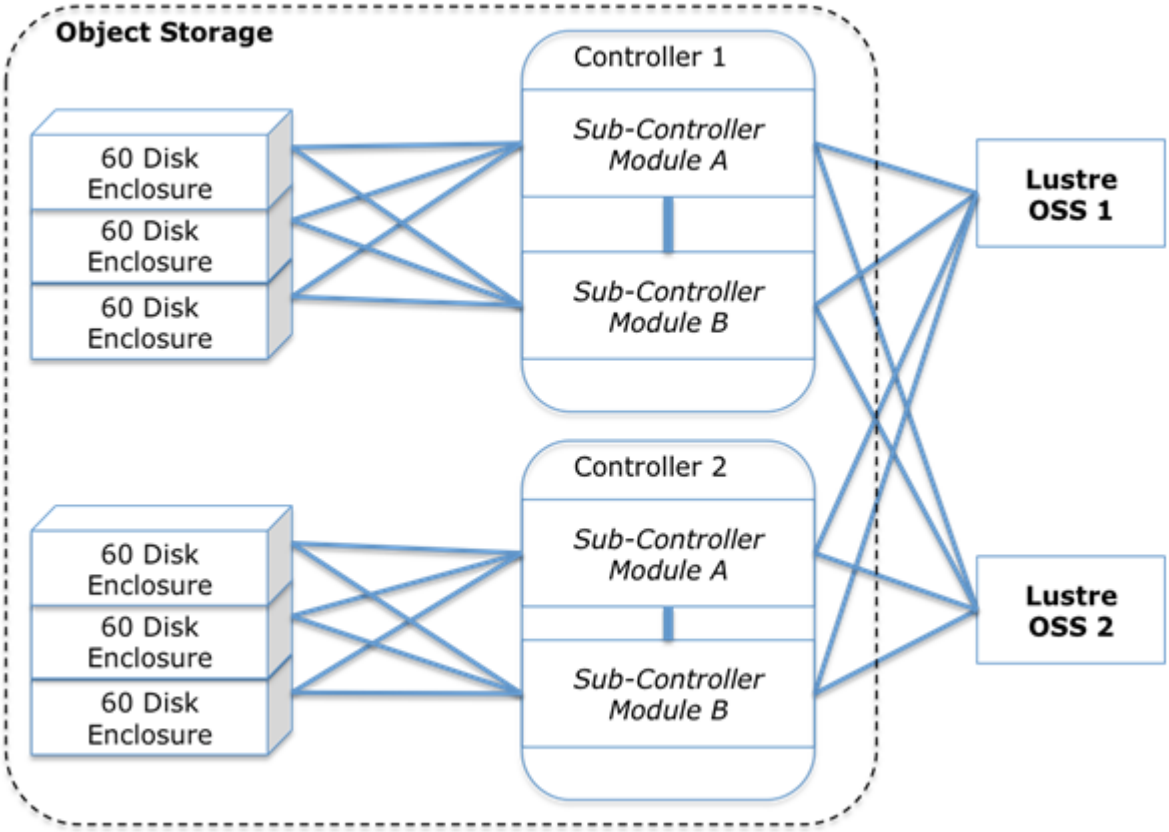


Figure 6. Lustre OSS high availability configuration

Determining OSS Memory Requirements

Like the MDS, the OSS uses memory to cache file system metadata and for LDLM locks held by the clients. In addition to the metadata memory requirements described for the MDS above, the OSS needs additional memory for the 1-MB RDMA I/O buffer needed for each object storage target (OST) IO service thread. The same calculation applies for files accessed from the OSS as for those accessed from the MDS (see Table 4), but the load is spread over 22 OSS nodes in this case, so the amount of RAM required for inode cache, locks, etc., is spread out over the OSS nodes [4].

While an OST has less file system metadata than the MDT (due to smaller inodes, few directories, and no extended attributes), there is considerably more OST storage space per OSS in this configuration (492 TB vs. 8 TB). This means an adequate amount of RAM must be reserved for the OST metadata.

Table 4 calculates the absolute minimum RAM required in an OSS node. These calculations take into account a failover configuration with 18 primary OSTs on each OSS node, and 18 backup OSTs on each OSS node. When an OSS is not handling any failed-over OSTs, the extra RAM is used as read-cache. The OST thread count defaults to 512, which is close to the 32 IO threads per OST that have been found to work well in practice. In this case, 64 GB RAM is minimal and 128 GB RAM is recommended for better performance.

Table 4: OSS RAM Calculation

Memory Consumer		Required Memory
Operating system overhead		1,024 MB
Ethernet/TCP send/receive buffers (1 MB X 512 threads)		512 MB
400 MB journal X (18 + 18) OST devices		14,400 MB
1.5 MB RDMA per OST IO thread X 512 threads		768 MB
OST file system metadata cache (0.05% of 492 TB)		25,190 MB
800 MB data read cache X 18 OSTs		14,400 MB
2000 8-core clients X 100 files per core X 3 KB/file	4,687 MB / 40	118 MB
64 interactive clients X 10,000 files X 3 KB/file	1,875 MB / 40	47 MB
2 million file working set X 1.5 KB/file	2,929 MB / 40	74 MB
Total		**56,533 MB**

Selecting IO Cards for the Interface to Clients

An Infiniband host channel adapter (HCA), or a 10 Gb or 1 Gb Ethernet NIC is typically provided on the storage server, and this provides connection to the network on the front end, depending on the type of storage area network.

For example, if the OSTs on an OSS only need to deliver 1 GB/s bandwidth, one 10 Gb Ethernet card is sufficient to relay the bandwidth to Lustre clients. However, if the OSTs on an OSS can deliver 5 GB/s bandwidth, one fourteen data rate (FDR) Infiniband HCA would be needed to make the bandwidth available to Lustre clients. The numbers in Table 5 can be used as a guide.

Table 5: IO Card Bandwidth

IO Card	Bit Rate	Theoretical Peak Bandwidth	Peak Bandwidth as Measured by Author
1 FDR Infiniband	54 Gbps	6.75 GB/s	6,000 MB/s
1 10 Gb Ethernet	10 Gbps	1.25 GB/s	1.1 GB/s

Because each OSS needs to provide at least 4.8 GB/s from the storage controller, an FDR Infiniband port would be the best match.

Cluster Network

Lustre clients connect to the Lustre file system via a network protocol. Popular choices are InfiniBand and 10 Gb Ethernet. A network architecture that has not been optimized can be a limiting factor. The most common issue encountered is that the network switches do not support interconnections of sufficient bandwidth between storage servers and clients, due to oversubscription, as shown in Figure 7.

In Figure 7, twelve ports from each of two 36-port IB switches are used to bridge two IB fabrics, while the other 24 ports on each switch support 24 internal nodes on the IB fabric. The aggregated bandwidth of the 12 IB links is not sufficient to support optimal communication between the 24 nodes on each of the switches.

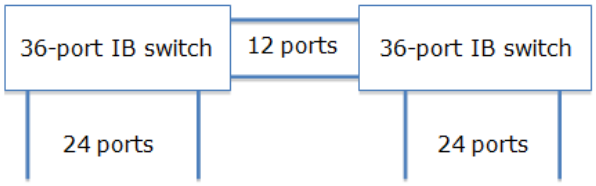


Figure 7. Oversubscribed IB fabric

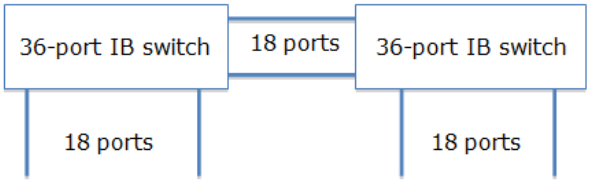


Figure 8. Non-oversubscribed IB fabric

Conversely, Figure 8 is an example of a non-oversubscribed IB fabric. The bandwidth between any of the nodes on either IB switch is the same.

The Lustre file system in this example has 22 OSS nodes and two MDS nodes. Each OSS requires one port on the FDR IB switch. The backend storage can be connected to these nodes directly with cables because only block devices are presented to the Lustre servers. Thus, 24 ports are required on the Lustre storage side of the OSS servers.

However, 2,000 client nodes will be accessing the Lustre file system and all of these clients need to be connected to the same IB fabric. To the best of our knowledge, an IB switch does not exist that supports 2,040 ports. Multiple IB switches must be bridged to satisfy the capacity requirements. It is important for optimal performance that these IB switches be bridged without oversubscription.

Another consideration is that, although Lustre servers and Lustre clients are often both on a high speed network such as InfiniBand, some Lustre clients may be on a slower network such as 1 Gb Ethernet. Clients on a slower network can access the Lustre file system using Lustre LNET routers. A Lustre LNET router is a special Lustre client with multiple network interfaces, for example, one InfiniBand interface and one 1 Gb Ethernet interface. The Lustre LNET router bridges these interfaces, offering flexibility when designing the network configuration.

Reviewing the Storage System

The iterative design process is now applied to all the aggregated building blocks comprising the complete storage system.

As Figure 9 shows, the storage system consists of:

- 3 12-disk enclosures
- 1 storage controller for the MDT
- 2 MDS servers
- 63 enclosures, each containing 60 disks
- 21 storage controllers for OSTs
- 22 OSSs

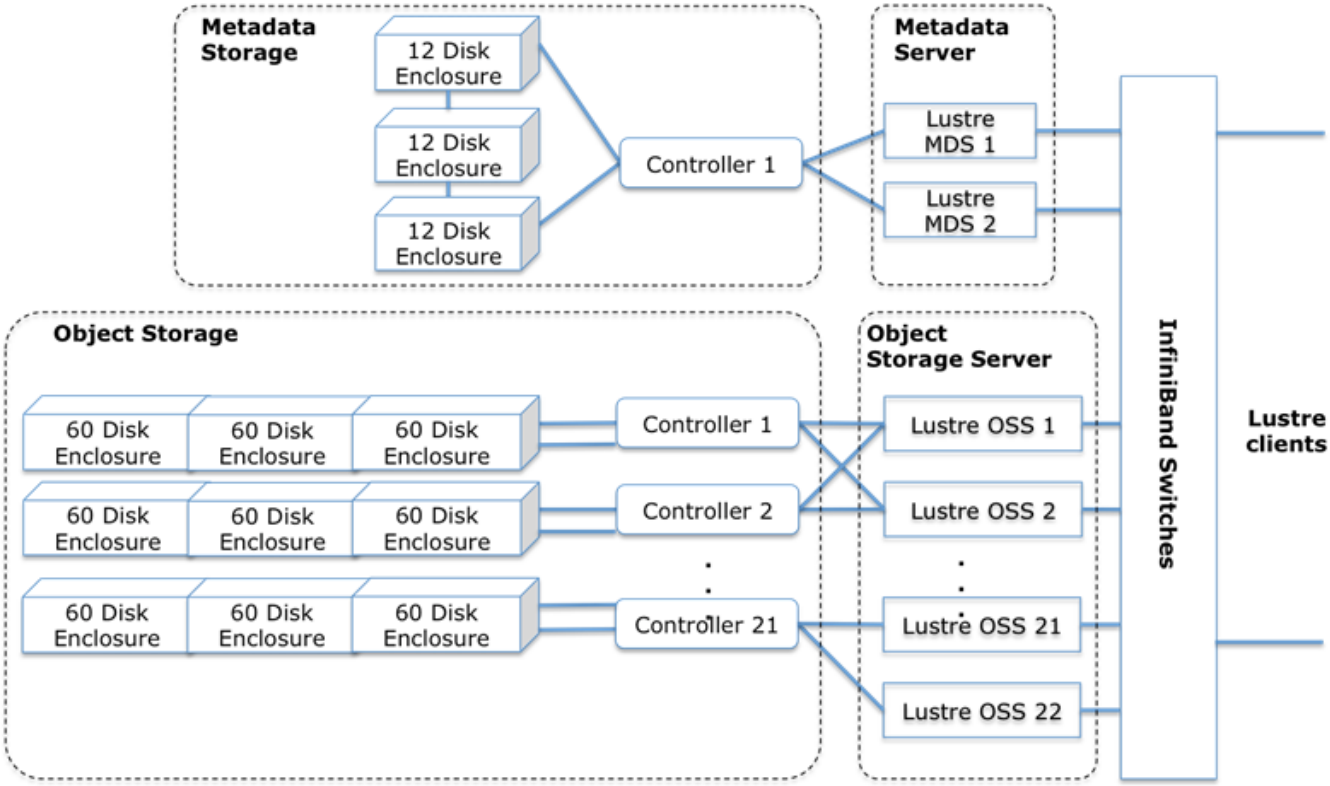


Figure 9. The complete Lustre storage system

Each of the 63 disk enclosures contains 60, 4-TB, 7,200 RPM disks. The disks in each enclosure are structured into six RAID-6 groups of ten disks each, with eight of the ten disks available for data. Each RAID-6 group presents as a LUN, which is formatted by the Lustre OSS as an OST. Thus, each disk enclosure contains 6 OSTs.

The capacity of the designed system is calculated as:

$$4 \text{ x } 1012 \text{ x } 10 \text{ x } 8/10 \text{ x } 0.9 \text{ x } 0.95 \text{ x } 6 \text{ x } 63 \cong 10.1 \ PB$$

Thus, this configuration meets the requirement for 10 PB of usable space.

Each backend disk enclosure offers approximately 3 GB/s bandwidth; the bandwidth of the storage pipeline is limited to 6 GB/s by the storage controller. Because Lustre aggregates all the OSS bandwidths linearly and can achieve up to 90% of hardware bandwidth, the performance of the designed system is calculated as:

$$6 \ GB/s \text{ x } 90\% \text{ x } 21 \ storage \ controllers = 113.4 \ GB/s$$

Thus, the total bandwidth supported by this Lustre file system is 113.4 GB/s. This performance meets the aggregate bandwidth requirement of 100 GB/s.

Conclusion

The process of designing a storage system is not straightforward, as many different aspects must be considered. The step-by-step approach to designing a high-performance storage system, and resolving the common issues that were described in this paper, is based on two general design methods:

- Design the backend disk storage first, and then gradually work up the storage "pipeline" to the client.
- Iteratively review the design and incrementally factor in more requirements.

We demonstrated our approach with a case study showing the design of a Lustre file system. Starting with selecting disks and ending with designing the storage area network, we applied the pipeline approach and iterative design method to gradually arrive at a storage architecture that met the system requirements.

More information

For more information, contact your Intel® Lustre reseller, or email the Lustre team at Intel® at: hpdd-info@intel.com.

References

1. Intel® Corp., Lustre Product Brief (2010).
   URL: https://wiki.hpdd.intel.com/display/PUB/Why+Use+Lustre
2. DataDirectNetworks, SFA10K-X and SFA10K-E: Breaking Down Storage Barriers by Providing Extreme Performance in Both Bandwidth and IOPS. White Paper (2010).
   URL: http://www.ddn.com/pdfs/SFA10K-X10K-E.pdf
3. Z. Liang. Lustre SMP Scaling Improvements. Lustre User Group presentation (2010).
   URL: http://wiki.lustre.org/images/6/66/Lustre_smp_scaling_LUG_2010.pdf
4. Y. Fan, Z. Liang. Lustre Metadata Performance Improvements. Lustre User Group presentation (2011).
   URL: http://www.opensfs.org/wp-content/uploads/2012/12/200-230_Fan_Yong_LUG11_Lustre-MDPI_v2.pdf
5. Lustre Software Release 2.X Operations Manual.
   URL: http://build.whamcloud.com/job/lustre-manual/lastSuccessfulBuild/artifact/lustre_manual.pdf

Disclaimer and legal information

Copyright 2014 Intel Corporation. All Rights Reserved.

The document is protected by worldwide copyright laws and treaty provisions. No part of this document may be used, copied, reproduced, modified, published, uploaded, posted, transmitted, distributed, or disclosed in any way without Intel's prior express written permission. No license under any patent, copyright, trade secret or other intellectual property right is granted to or conferred upon you by disclosure of information in this document, either expressly, by implication, inducement, estoppel or otherwise. Any license under such intellectual property rights must be express and approved by Intel in writing.

NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: http://www.intel.com/design/literature.htm.

Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and/or other countries.

* Other names and brands may be claimed as the property of others.