LStore: Logistical Storage
Scalable, Fast, Cheap storage

Alan Tackett, Matthew Heller, Mathew Binkley
Vanderbilt University, Nashville, TN 37203

Yet another Parallel file system…

LStore provides a flexible logistical storage framework for distributed and scalable access to data for a wide spectrum of users. It contains:

- **Virtually unlimited scalability in raw storage**
- **Support for arbitrary metadata associated with each file**
- **User controlled fault tolerance and data reliability on a file and directory level**
- **Scalable performance in raw data movement**
- **A FUSE-based file system interface with both a native mount in Linux (exportable via NFS and CIFS to other platforms)**
- **High performance command line interface**
- **Support for the geographical distribution and migration of data**

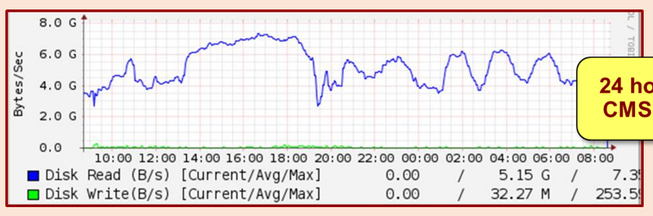
These features are accomplished by segregating directory and metadata services from data transfer. LStore clients use the LServer only for metadata operations, removing an important bottleneck.

Just a Typical Day

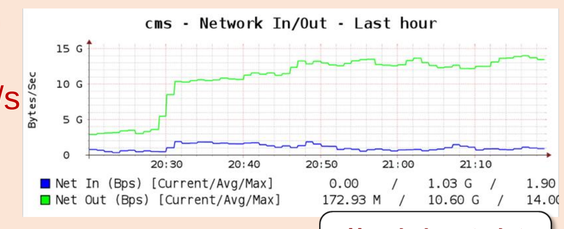
CMS-HI has 10PB of native disk space (6+PB usable)

Routinely read over 2PB/week using the FUSE mount for production jobs and sustain 80-100Gb/s read rates.

Vanderbilt TV News Archive has 400+ TB of space



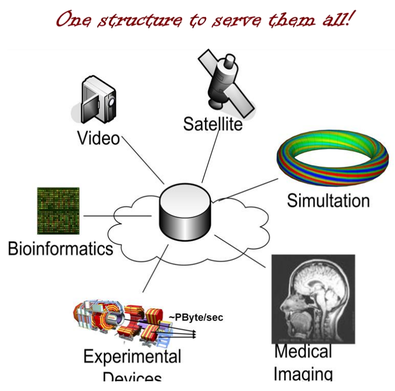
24 hour plot of CMS-HI usage



Hourly burst plot of CMS-HI usage

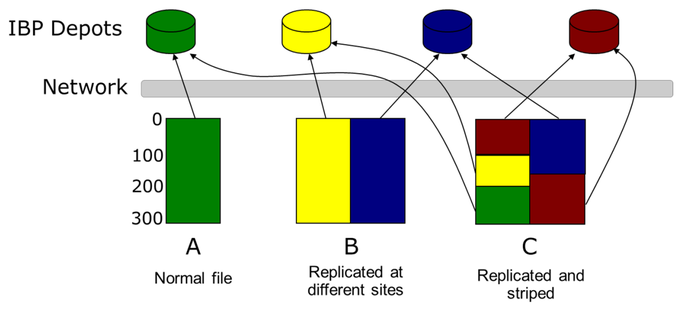
Bits are Bits: Logistical Networking

- Standardize on what we have an adequate common model for
  - Storage/buffer management
  - Coarse-grained data transfer
- Leave everything else to higher layers
  - End-to-end services: checksums, encryption, error encoding, etc.
- Enable autonomy in wide area service creation
  - Security, resource allocation, QoS guarantees
- **Gain the benefits of interoperability!**



eXnode

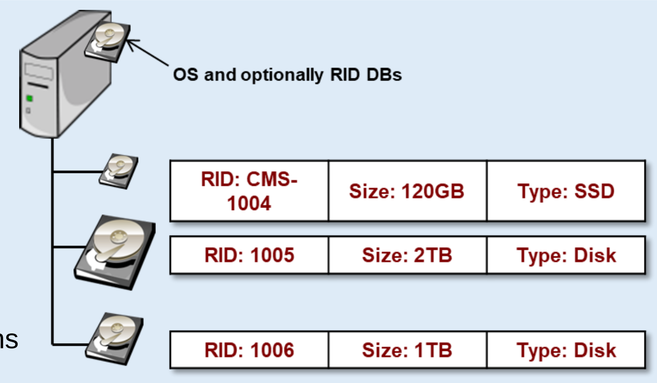
LN uses a generalized Abstraction of the inode, a container called the exNode. Information about a distributed file's data mappings is placed in the exNode. In addition to the file's structural metadata, Arbitrary non-structural metadata, called attributes, can be stored in the exNode. The generality of this abstraction allows us to combine storage resources from many sources, implemented by many different technologies. Files stored in this distributed mix of hardware, along with their metadata, can be manipulated by users through widely used interfaces with familiar semantics.



Cheap Bulk Storage: Internet Backplane Protocol

The fundamental unit of storage in LN is the depot. A depot can be a single disk or a collection of disks in a server. In LN, the pieces or blocks of a file are called allocations.

As currently implemented in LStore, exNodes use one basic storage access protocol —the Internet Backplane Protocol (IBP). IBP provides a generic, best-effort service that allocates, reads, writes and manages allocations On network-addressable storage (the depots). The exNode data mappings capture the way in which the (possibly replicated) segments of a data extent are mapped to IBP allocations. By factoring the data movement and storage and data-mapping aspects of the file abstraction, the LN architecture exposes the structural metadata in a form that can be safely and directly managed and manipulated by client processes and by services acting on behalf of the client. IBP allows for third party transfer of data between depots relieving users of the burden of moving data around the Internet, as other middleware tools can do this on their behalf.



The LStore software stack implements a complete virtual file system. L-Store uses IBP as the underlying abstraction of distributed storage. It manages, monitors, and maintains data distributed to IBP depots, and supports RAID-like mirroring and striping of data for performance and fault tolerance.

Mix and Match Storage: Lifecycle management

LStore supports the complete lifecycle management of hardware. No downtime is required to add disk storage, which can be added on the fly with the space becoming immediately available. As hardware is added, allocations can be redistributed to maintain uniformity in data distribution. Hardware can be retired, resulting in the automatic migration of data off the retiring hardware. IBP supports heterogeneous disk sizes and storage hardware. As a result, LStore can grow based on demand, using the best technology. This has become a routine and common occurrence in the multi-petabyte L-Store storage element used at the Vanderbilt CMS Tier2 center

Surviving Bit Rot: Data Integrity

What are the challenges of wide-area network and data management at the petabyte scale?

Any time one changes a parameter by an order of magnitude new challenges arise. Managing multi-petabyte data sets is no different. Moving a few terabytes of data between remote sites can be done in under a day with existing tools and hardware. Ten Gb/s or greater networking is not required, and one can most likely use traditional hardware or software based RAID systems to insure data integrity. At the petabyte scale, reliable high performance networks become vital and standard data integrity solutions such as RAID5 and even RAID6 become less viable.

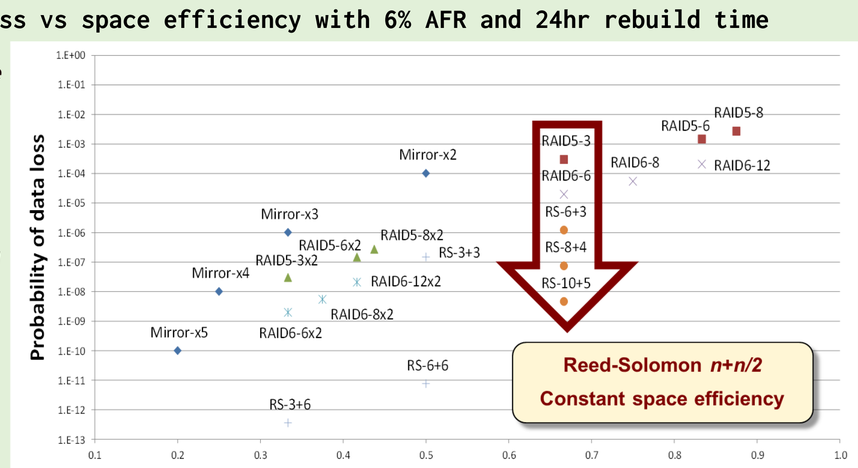
Why is data integrity such a big issue at petabyte scales?

100% chance of or read error at 1PB scale
Drive errors rate ~10^{15} bits
1PB = ~10^{16} bits

Data integrity becomes a much more significant issue at petabyte scales due to increasing probabilities of data loss and array **rebuild times taking days**. At these scales, **unrecoverable read errors are guaranteed**! **Manufacturing defects** or **firmware bugs** lead to systematic and correlated problems. In addition, because rack space is typically a precious commodity one tends to use dense disk enclosures which can suffer from **mechanical vibrations** and **broken circuit traces** on connectors.

Probability of data loss vs space efficiency with 6% AFR and 24hr rebuild time

Space efficiency is defined as (unique data)/(data+parity). The ideal configuration is in the lower right corner and would have a low chance of data loss and make good use of space. Mirror-xN signifies N replicas of data. RAID5-N and RAID6-N correspond to N total drives, data and parity, used in the array. The RAID6-Nx2 correspond to RAID6 arrays using N total drives that are replicated. RS-D+P represents generic Reed-Solomon using D data disk and P parity disks.



Data Integrity Schemes

Data integrity schemes typically employ either data **mirroring** or some form of **RAID**. Data mirroring is nothing more than keeping multiple copies of the data stored on different devices at different locations. **Mirroring is great for read dominated workloads** since all replicas can be used to service user requests. Write performance is slowed by the making of all the replicas. **Mirroring is not very space efficient** which can be quite costly at the petabyte scale. It is also not very good at providing data integrity as can be seem from the above figure.

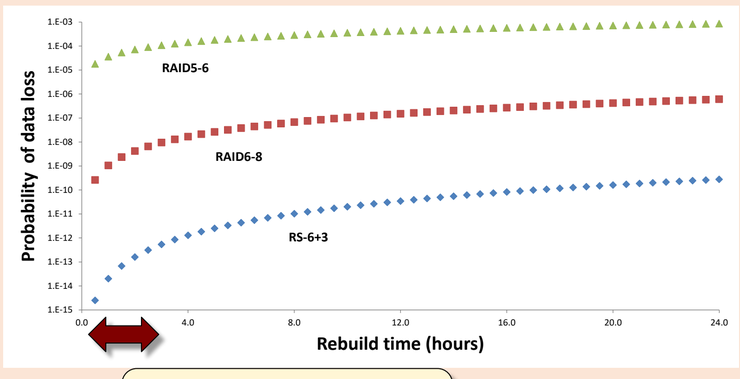
The other standard approach is to employ some form of RAID. In this case one stores the data and additional parity information that can be used to reconstruct unrecoverable bit errors or drive failures. The two most common forms for RAID are RAID5 and RAID6. RAID5 uses one additional disk for storing parity and is can survive a single device failure. RAID6 has two additional disks for parity and can survive two drive failures. Reed-Solomon encoding is the generalization or RAID5 and RAID6 to support arbitrary numbers of drive failures.

LStore has implemented generic Reed-Solomon encoding along with several other data integrity schemes. We typically recommend 6+3 Reed-Solomon (RS-6+3) encoding — 6 data disks and 3 parity disks. More generally we recommend the use of the RS-d+(d/2) family of configurations which can be seen in the figure above as RS-6+3, RS-8+4, and RS-10+5. This family uses 2/3 of the total space for data with the remainder for parity and has excellent reliability. RS-6+3 has the same reliability as keeping 3 copies of the data but uses only half the space.

Quick Disk Rebuilds

Recovering from the failure of a multi-terabyte disk drive can take days! The possibility of correlated drive failures increases the probability that additional drive failures could occur, resulting in unrecoverable data loss. The above figure shows the probability of data loss due to the loss of additional disks as a function of the rebuild time for three RAID configurations. RAID5-6 provides very little protection in this scenario. The RADI6-8 configuration is better since two additional drives are required in order to lose data. Even so, the chances of data loss are uncomfortably high if it takes a few days for the repair. The RS-6+3 is substantially more reliable even if repair takes a few days but keeping the rebuild times down to a few hours helps minimize the chance of data loss.

Probability of data loss vs Rebuild Time and 6% AFR



Try to keep the rebuild time to a few hours!

Distributed RAID Arrays

Traditional RAID arrays completely reconstruct a single failed drive on a single replacement drive. The use of a single replacement drive is a major factor in the rebuild time. Traditionally the entire drive is reconstructed with no regard for used vs. unused space. If the array is active with user I/O requests, this will greatly increase the rebuild time.

Distributed RAID arrays are designed to overcome these limitations. Instead of using the whole disk the disk is broken up into many smaller blocks. These blocks are combined with blocks on other disks creating many small logical RAID arrays utilizing a large subset of the available drives as shown in the figure. These distributed logical RAID arrays are based on space that is actually used. The free space on each drive can be used to store the newly reconstructed data. This allows for a large number of drives being read and written to simultaneously providing significantly faster rebuild times. For LStore each file and associated parity is placed on a random collection of drives based on the fault tolerance scheme used and data placement criteria. We routinely rebuild a single 8TB in a couple of hours using a single host to perform the data reconstruction. Adding more hosts causes the rebuild time to proportionately decrease.