

HPC File Systems

Timothy Brown
June 24, 2015

Overview

HPC Environment

File Systems

Lustre

Best Practices

File Striping

File I/O

HPC Environment

The typical HPC system has multiple file systems intended for different uses.

- ▶ **Home**

Individual user home directories. Backed-up with quotas.

- ▶ **Scratch**

Fast temporary access, not backed-up.

- ▶ *Shared* file system across all nodes.

- ▶ *Local* each node has it's own.

(A lot of clusters do not put disks in the nodes)

- ▶ **Mass Storage**

Long term storage, typically a tape system.

File Systems

- ▶ NFS – Network File System
- ▶ CXFS – Clustered XFS
- ▶ PanFS – Panasas ActiveScale File System
- ▶ GPFS – General Parallel File System
- ▶ PVFS – Parallel Virtual File System
- ▶ Lustre

Why Parallel?

- ▶ Spinning disks are slow.
- ▶ Serial I/O is even slower.

Key Features

- ▶ Scalability.
Can scale out to tens of thousands of nodes and petabytes of storage.
- ▶ Performance.
Throughput of a single stream ~GB/s and parallel I/O ~TB/s.
- ▶ High availability.
- ▶ POSIX compliance.
- ▶ Supports ROMIO.

Lustre

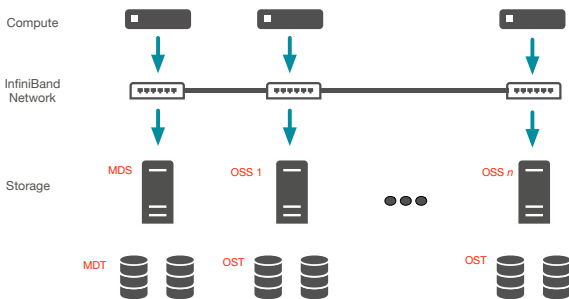
Approximately 50% of top 100 HPC's use Lustre.
It consists of four components:

MDS Metadata Server

MDT Metadata Target

OSS Object Storage
Server

OST Object Storage
Target



Metadata Server and Target

The MDS is a single service node that assigns and tracks all of the storage locations associated with each file in order to direct file I/O requests to the correct set of OSTs and corresponding OSSs.

The MDT stores the metadata, filenames, directories, permissions and file layout.

Object Storage Servers and Targets

An OSS manages a small set of OSTs by controlling I/O access and handling network requests to them.

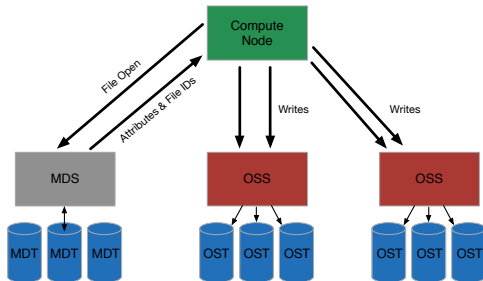
An OST is a block storage device. Often several disks in a RAID configuration.

Typical Setup

- ▶ All nodes (login, compute, compile, ...) have the lustre file-system mounted at `/lustre/`.
- ▶ The number of servers and targets.
 - ▶ 2 MDSs (active, standby)
 - ▶ 10's OSSs
 - ▶ 100's OSTs
- ▶ Often giving a total of ~1PB of usable disk space.

File Operations

- ▶ When a compute node needs to create or access a file, it requests the associated storage locations from the MDS and the associated MDT.
- ▶ I/O operations then occur directly with the OSSs and OSTs associated with the file bypassing the MDS.
- ▶ For read operations, file data flows from the OSTs to the compute node.



User Commands

Lutre provides a utility to query and set access to the file system. They are all sub commands to the program `lfs`.

- For a complete list of available options.

```
login01 $ lfs help
```

- To get more information on a specific option.

```
login01 $ lfs help option
```

Checking Diskspace

The `lfs df` command displays the file system disk space usage. Additional parameters can be specified to display inode usage of each MDT/OST or a subset of OSTs. The usage for the `lfs df` command is:

```
login01 $ lfs help df
Usage: df [-i] [-h] [--lazy|-l] [--pool|-p <fsname>[.<pool>] [path]
```

Example, get a summary of the disk usage:

```
login01 $ lfs df -h /lustre/tbrown/ | \
    grep -v ^scratch
UUID                bytes  Used   Available Use% Mounted on
filesystem summary: 852.8T 607.5T 236.8T    72% /lustre/
```

Finding Files

The `lfs find` command is more efficient than the GNU `find`.
Example, finding fortran source files accessed within the last day.

```
login01 $ lfs find . -atime -1 -name '*.f90'
```

Other lfs Commands

- ▶ `lfs cp` – to copy files.
- ▶ `lfs ls` – to list directories and files.

These commands are often quicker as they reduce the number of `stat` and `rpc` calls needed.

Avoid Wild Cards

- ▶ `tar` and `rm` are inefficient when operating on a large class of files on lustre.
- ▶ The reason lies in the time it takes to expand the wild card.
- ▶ `rm -rf *` on millions of files could take days, and impact all other users.
- ▶ Generate a list of files to be removed or tar-ed, and to act them one at a time, or in small sets.

```
login01 $ lfs find old/ -t f -print0 | xargs -0 rm
```

Avoid Colons

MPI-IO will split a path and file name on a colon to obtain a file system hint.

- ▶ If the colon is used in the time format specification, the interpretation of the hint will cause an error.

```
/lustre/tbrown/wrfout_2015_06_23_00:00:00.nc
```

- ▶ Provide a hint that the underlying file system is lustre.

```
lustre:/lustre/tbrown/wrfout_2015_06_23_00_00_00.nc
```


Limit Files Per Directory

- ▶ It is best to limit the number of files per directory.
- ▶ Writing thousands of files to a single directory produces a massive load on the MDSs, this often takes the file system offline.
- ▶ If you need to create a large number of files. Use a directory structure.
- ▶ A suggested approach is a two-level directory structure with \sqrt{N} directories each containing \sqrt{N} files, where N is the number of tasks.

Read Only Access

- ▶ If a file is only going to be read, open it as `O_RDONLY`.
- ▶ If you don't care about the access time, open it as `O_RDONLY | O_NOATIME`.
- ▶ If you need access time information and your doing parallel IO, let the master open it as `O_RDONLY` and all other ranks as `O_RDONLY | O_NOATIME`.

Broadcast Stat

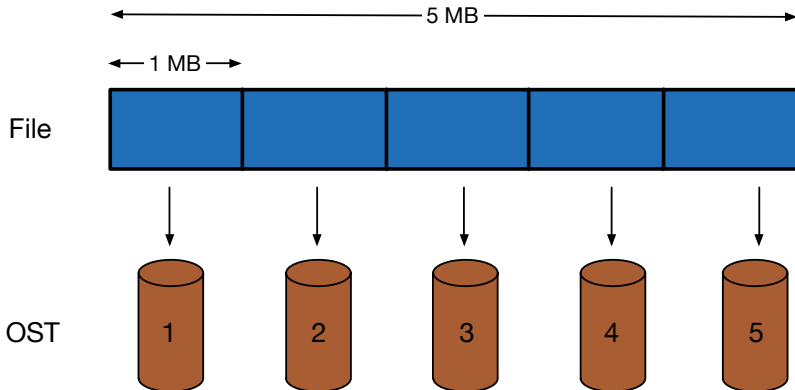
- ▶ If many processes need the information from `stat()`.
- ▶ Have the master process perform the `stat()` call.
- ▶ Then broadcast it to all processes.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <err.h>
#include <mpi.h>
int
main(int argc, char **argv)
{
    int rank = 0;
    int len = 0;
    struct stat sbuf = {0};

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    if (rank == 0) {
        if (stat(argv[0], &sbuf)) {
            warn("Unable to stat %s", argv[0]);
        }
    }
    len = sizeof(sbuf);
    MPI_Bcast(&sbuf, len, MPI_BYTE, 0, MPI_COMM_WORLD);
    MPI_Finalize();
    return(EXIT_SUCCESS);
}
```

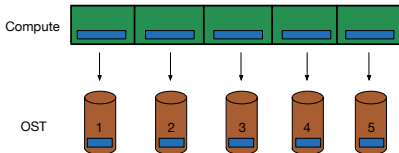
File Striping

- ▶ A file is split into segments and consecutive segments are stored on different physical storage devices (OSTs).

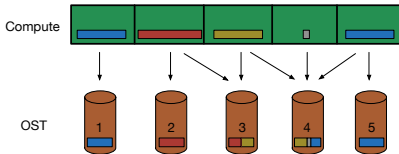


Aligned vs Unaligned Stripes

- ▶ Aligned stripes is where each segment fits fully onto a single OST. Processes accessing the file do so at corresponding stripe boundaries.



- ▶ Unaligned stripes means some file segments are split across OSTs.



Stripe Sizes

- ▶ You can get/set the stripe size, number of OSTs and which OST to start at.
- ▶ The stripe size must be a multiple of the maximum page size (64 KB).
- ▶ The typical default is
 - ▶ stripe count: 1
 - ▶ stripe size: 1048576 (1 MB)
 - ▶ stripe offset: -1 (MDS selects)

```
login01 $ lfs getstripe .
```

```
login01 $ lfs setstripe -s 32m -c 4 .
```

Large File Stripe Sizes

- ▶ Set the stripe count of the directory to a large value.
- ▶ This spreads the reads/writes across more OSTs, therefore balancing the load and data.

```
login01 $ lfs setstripe -c 30 \  
         /lustre/tbrown/large_files/
```

Small File Stripe Sizes

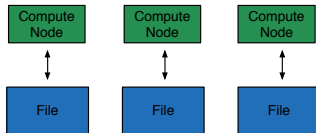
- ▶ Place small files on a single OST.
- ▶ This causes the small files not to be spread out/fragmented across OSTs.

```
login01 $ lfs setstripe -s 1m -c 1 \  
         /lustre/tbrown/small_files/
```


File I/O

Three cases of file I/O:

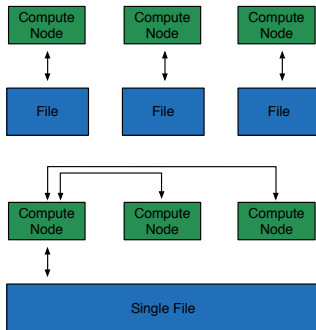
- Single stream.



File I/O

Three cases of file I/O:

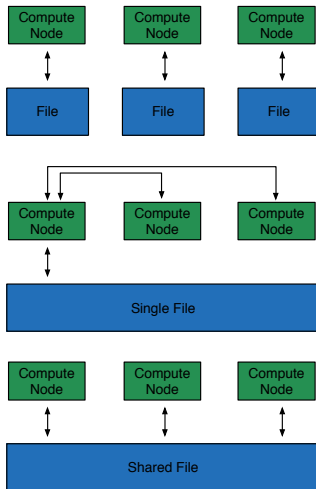
- Single stream.
- Single stream through a master.



File I/O

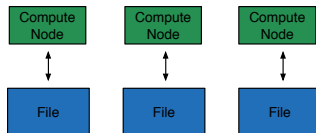
Three cases of file I/O:

- Single stream.
- Single stream through a master.
- Parallel.



Single Stream IO

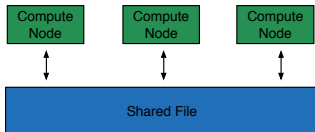
- ▶ Set the stripe count to 1 on a directory.
- ▶ Write all files in this directory.
- ▶ Otherwise set the stripe count to 1 for the file.



```
login01 $ lfs setstripe -s 1m -c 1 \  
         /lustre/tbrown/serial/
```

Parallel IO Stripe Count I

- ▶ Single shared files should have a stripe count equal to the number of processes which access the file.
- ▶ If the number of processes is >160 , set the count to -1, this will stripe across all OSTs (lustre has a max of 160).
- ▶ The stripe size should be set to allow as much stripe alignment as possible.
- ▶ Try to keep each process accessing as few OSTs as possible.



```
login01 $ lfs setstripe -s 32m -c 24 \  
/lustre/tbrown/parallel/
```

Parallel IO Stripe Count II

You can specify the stripe count and size programmatically, by creating an MPI info object.

```
use mpi
use hdf5
implicit none
integer          :: info                ! MPI IO Info
integer          :: ierr                ! Error status
integer(kind=hid_t) :: p_id, f_id      ! Property and file id
character(len=256) :: filename,lcount,lsize ! Filename

! Init the HDF5 library
call h5open_f(ierr)
! Create an MPI object setting the strip size and count
call mpi_info_create(info, ierr)
write(lcount, '(I4)') 4
write(lsize, '(I8)') 4 * 1024 * 1024
call mpi_info_set(info, "striping_factor", trim(lcount), ierr)
call mpi_info_set(info, "striping_unit", trim(lsize), ierr)
! Set up the access properties
call h5pcreate_f(H5P_FILE_ACCESS_F, p_id, ierr)
call h5pset_fapl_mpio_f(p_id, MPI_COMM_2D, info, ierr)

! Open the file
call h5fcreate_f(filename, H5F_ACC_TRUNC_F, f_id, ierr, &
    access_prp = p_id)
```

Questions? Survey

License

© University of Colorado Boulder, 2015

This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by/4.0/>

When attributing this work, please use the following text:
“HPC File Systems”, Research Computing, University of Colorado Boulder, 2015. Available under a Creative Commons Attribution 4.0 International License.

