ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

CSCS
Swiss National Supercomputing Centre

File Systems for HPC Machines

Course Outline – Background Knowledge

- Why I/O and data storage are important
- Introduction to I/O hardware
- File systems
- Lustre specifics
- Data formats and data provenance

Course Outline – Parallel I/O

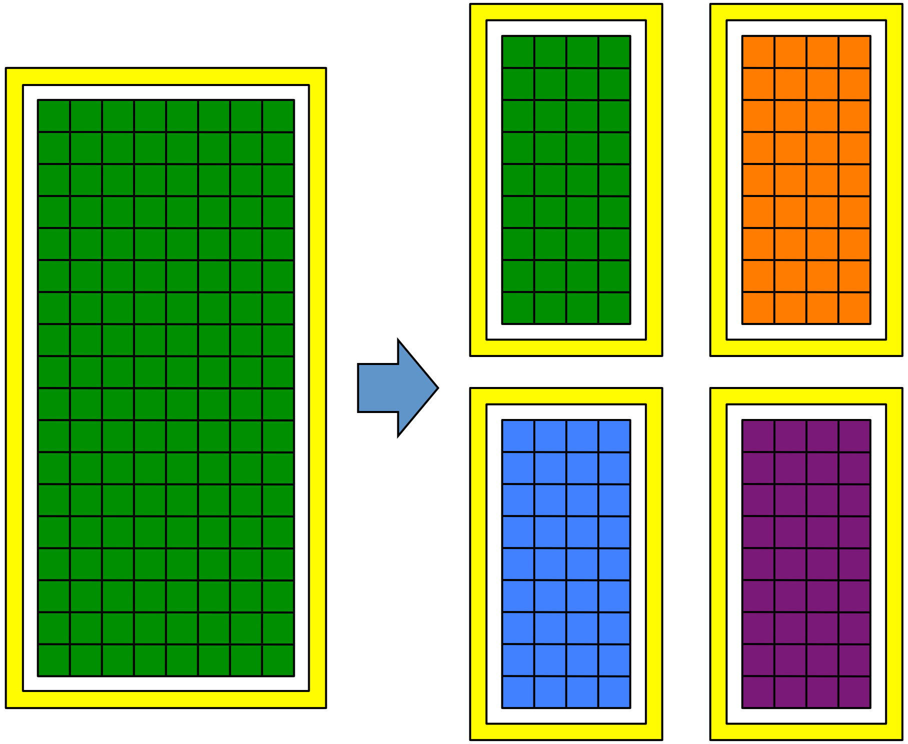
- MPI-IO
  - The foundation for most other parallel libraries
- HDF5
  - A portable data format widely used in HPC
- NetCDF
  - Portable data format commonly used in climate simulations
- ADIOS
  - A library that builds on other data formats

Data and I/O in Applications

- Checkpoint/restart files
  - Must use full precision of simulation data
  - Amount of data needed for restart will determine frequency of output
- Input of initialisation data
  - Normally not parallel
- Output of data for analysis
  - Might be output in reduced precision
  - Might store only a subset of full resolution
- Output of data for job monitoring
  - Typically from process 0
  - Normally ASCII

Recap –why I/O is such a problem

- Limiting factor for many data intensive applications
- Speed of I/O subsystems for writing data is not keeping up with increases in speed of compute engines



P processors, each with …
MxN Grid points
2M+2N Halo points

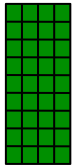
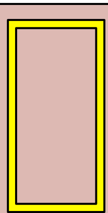
4P processors, each with …
(M/2)x(N/2) Grid points
M+N Halo points

Idealised 2D grid layout for data decomposition with halos:

Increasing the number of processors by 4 leads to each processor having
- *one quarter* the number of grid points to compute
- *one half* the number of halo points to communicate

The same amount of total data needs to be output at each time step.

I/O reaches a scaling limit

Computation:		Scales O(P) for P processors	*Minor scaling problem – issues of halo memory bandwidth, vector lengths, efficiency of software pipeline etc.*
Communication:		Scales O(√P) for P processors	*<u>Major scaling problem</u> – the halo region decreases slowly as you increase the number of processors*
I/O (mainly "O"):		No scaling	***<u>Limiting factor in scaling</u>*** *– the same amount of total data is output at each time step*

Scalability Limitation of I/O

- I/O subsystems are typically very slow compared to other parts of a supercomputer
  - You can easily saturate the bandwidth
- Once the bandwidth is saturated scaling in I/O stops
  - Adding more compute nodes increases aggregate memory bandwidth and flops/s, but not I/O
- I/O scaling is a problem independent of the method used
  - Structured grid with domain decomposition, particle methods, unstructured grids

Storing Data

- Long-term data storage can be expensive
  - Storage systems need to have good I/O bandwidth to read back the data
- Only a few restart files are normally stored
- Most data is for analysis
- Storing data in reduced precision saves space
  - *If the simulation can output in reduced precision then this also saves on bandwidth*
  - Providing sufficient servers for the compute power required for compression is typically more economical than buying large amounts of data storage
- Data should be compressed before storage
  - … and reduced precision data can be compressed more than full precision data

Scientific decisions to be made:

- Which output fields from a simulation are needed, which can be ignored?
  - Re-examine output request before each simulation
- Can output frequency be reduced without compromising the scientific analysis?
- Can you store less data in general from the simulation ?
  - Grid based domain decomposition
    - Can some fields be on reduced resolution grid?
    - Can you store a subset of the grid?
  - Particle simulations
    - Can you store a subset of particles?
- Can you reduce the precision of variables
  - Do you need double precision or is single precision enough for analysis
  - Can you pack data even further?
    - E.g. NetCDF offers a reduced precision called NC_SHORT (16 bit)?

These questions require a scientific value judgement, but could reduce the data storage or I/O requirements substantially

The case for reduced precision (packing)

- Packing data is an efficient method to decrease data volumes at the cost of less precision
- Packed data is already employed in some communities
  - E.g. For weather and climate, the GRIB format is already a packed (16-bit) format, and packing is supported in NetCDF
- In NetCDF it is easy to use packed data
  - NCO tools have support functions to pack data
  - Many tools support packed data format
- Packed data compresses much better with lossless compression algorithms compared to non-packed data
- It should be easy to carry out data packing in parallel during a simulation

Packed Data

1. For a data set in single precision, find the minimum *min* and maximum *max* of the whole data set.
2. Store the offset *min* and range *(max-min)* of the dataset in single precision
3. For each element *Y* of the dataset, store a 16-bit integer *X* to represent its position within the range

Each reduced precision element $Y_{reduced}$ of the data can then be recovered from the formula:-

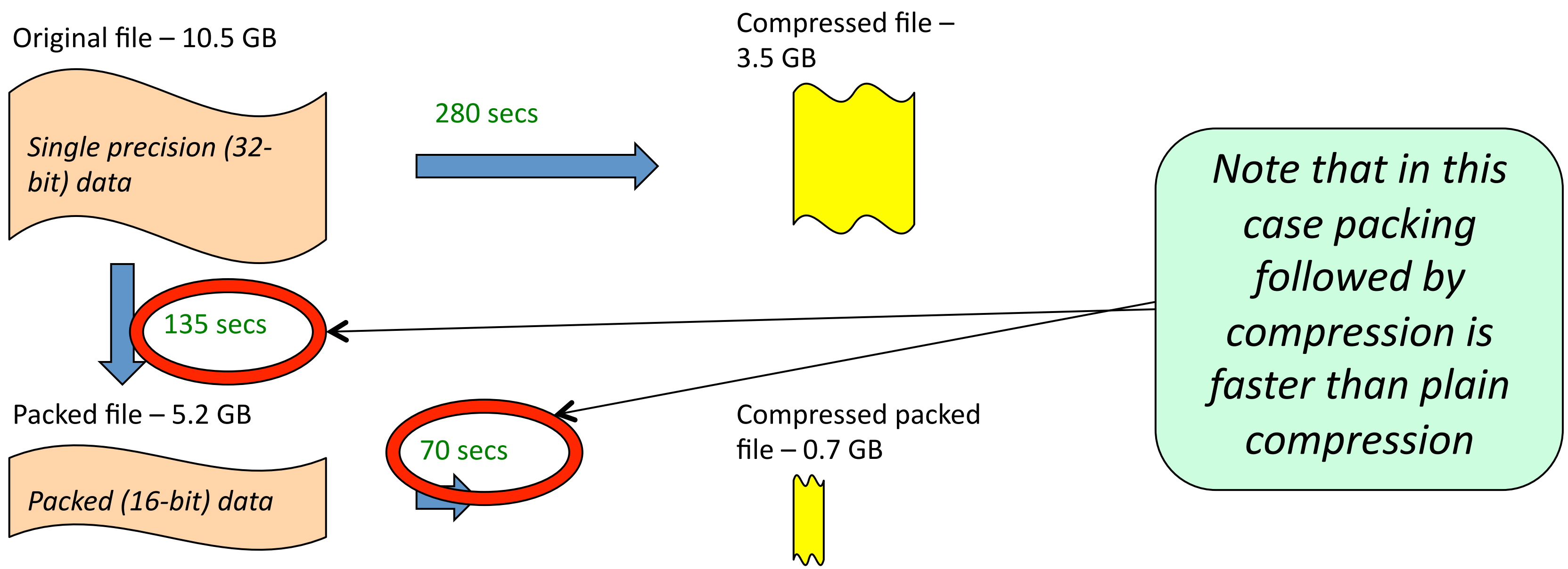
$$Y_{reduced} = offset + (X/65535) * range$$

The case for lossless compression

- Lossless compression reduces data volumes without losing precision
  - No scientific judgement is required
    - When uncompressed, the data is back to its original form
  - Standard LZ compression algorithms are widely used (e.g. gzip)
- It is relatively free
  - Some compute time is required to compress/uncompress
- Lossless compression reduces bandwidth requirements for data transfer to longer-term storage
- It might not be straightforward to introduce lossless compression with parallel file I/O from within your application
  - Most lossless compression will be carried out with tools (e.g. gzip) after the simulation has output the data
  - Using compression in *parallel* HDF5 for example is not yet supported

Compression example

- Testing of packing (`ncpdq`) and compression (`gzip –fast`) of a 10 Gigabyte high-resolution COSMO NetCDF output file on Ela



HPC Systems

- Machines consist of three main components
  - Compute nodes
  - High-speed interconnect
  - I/O infrastructure
- Most optimisation work on HPC applications is carried out on

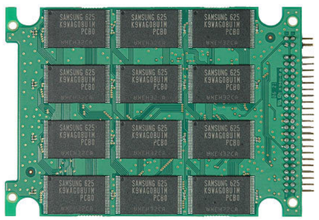
1. Single node performance
2. Network performance
3. *… and I/O only when it becomes a real problem*

Storage Hardware

- Modern storage hardware at CSCS consists of
  - Disk drives to provide storage
    - Optionally SSDs for fast storage
  - Disk controllers to manage the disks
  - File servers to offer applications access to the disks through a file system
  - Network infrastructure to connect the components
  - Tape backup systems for failover or long-term storage
    - CSCS uses tape as a backup of the /users file system and as a failover for the /project file system

Disk Drives and SSDs

- Disk drives come in several varieties
  - Fibre channel disks
    - Fast, expensive, low capacity
      - Often used for metadata operations
  - SATA disks
    - Slow, cheap, high capacity
  - SAS disks
    - SAS controllers with SATA drives are being mentioned as a potential replacement for standard SATA
- Solid state drives are gaining acceptance
  - NAND based flash
    - Fast, expensive, low capacity
      - Huge number of IOPs



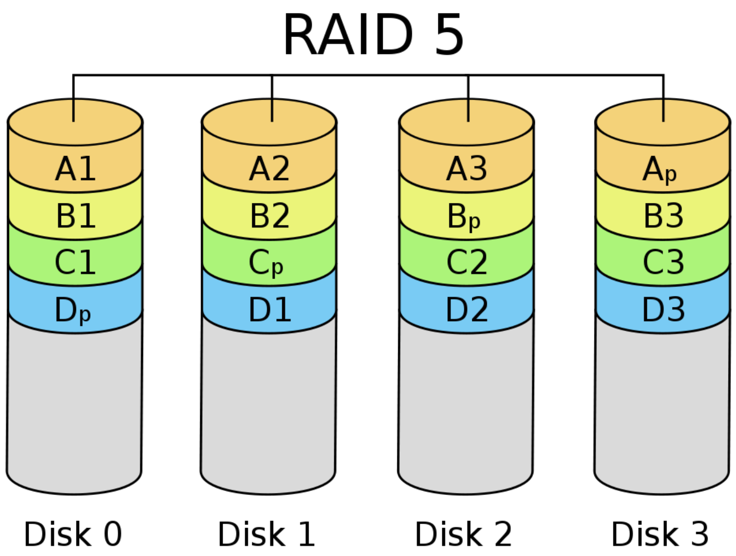
RAID Arrays

- For reliability, disks are collected together to form RAID arrays
  - RAID – Redundant Array of Independent Disks
- RAID arrays provide protection against disk failures and data corruption
- Typical RAID attached to HPC systems is RAID 5 or RAID 6
  - Newer systems use RAID 6
- RAID array configurations try to find a compromise between reliability, speed and volume of storage
- A RAID array typically delivers much lower bandwidth than the aggregate possible from the disks in the array

N.B. RAID 0 does not provide protection, it is only a mechanism for improved performance

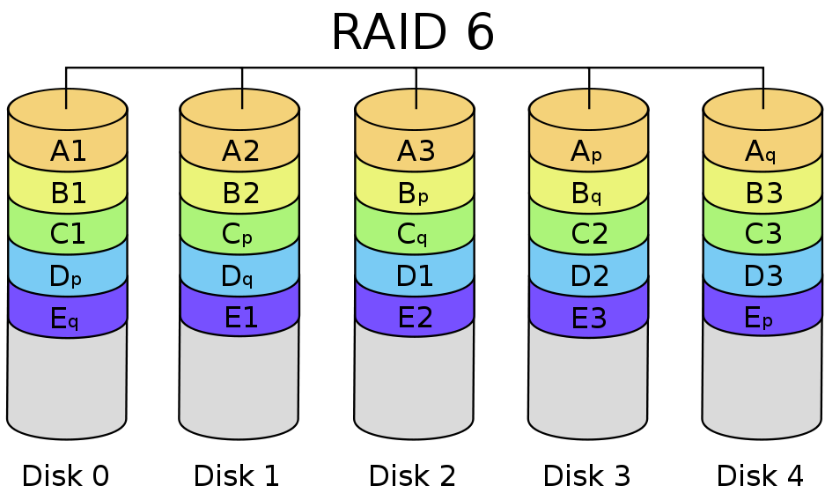
RAID 5

- A RAID 5 array consists of N +1 disks
  - Each N blocks of data have 1 extra block for parity
  - The parity blocks are distributed across the disks
  - N+1 Terabytes of raw disk storage provide N Terabytes of usable storage
  - RAID 5 arrays are typically described in terms of being a N+1 array
    - For example 4 disks would provide a 3+1 array designating 3 blocks of data to 1 block of parity



RAID 6

- A RAID 6 array consists of N+2 disks
  - Each N blocks of data have 2 extra block for parity
  - The parity blocks are distributed across the disks
  - N+2 Terabytes of raw disk storage provide N Terabytes of usable storage
  - RAID 6 arrays are typically described in terms of being a N +1+1 or an N+2 array
    - For example 5 disks would provide a 3+2 array designating 3 blocks of data to 2 blocks of doubly distributed parity
- The /scratch file systems on Rosa and Palu use 8+2 RAID 6



Multiple RAID arrays and LUNs

- On a typical parallel file system there are hundreds or thousands of disks
  - Palu /scratch file system uses 280 2-Terabyte disks
  - Rosa /scratch file system uses 800 512-Megabyte disks
- These disks are partitioned into RAID arrays
- The RAID arrays are then described as logical units (LUNs), and software will then see each LUN as if it were a single disk
- Palu's /scratch file system has 28 LUNs
  - Each LUN is a 8+1+1 RAID 6 array
- Rosa's /scratch file system has 80 LUNs

RAID Rebuild

- If a disk fails in a LUN, the LUN can continue to be used
  - In RAID 5 there is no parity check if a disk is down
  - In RAID 6 data corruption and recovery can still be carried out
- When a disk fails, the system will rebuild the missing disk from the remaining sector
- Disk rebuild is a slow process
  - The larger the individual disk drive, the longer the rebuild
- During the rebuild the remaining disks are in use
  - Still in use by the filesystem itself
  - Heavy read activity occurs on the remaining disks as the data is read to reconstruct the failed disk
- RAID rebuild issues have led to research into alternatives to RAID infrastructure for use in high performance file systems
- For a /scratch file system it would be preferable to have a larger number of smaller disks
  - More disks means more potential bandwidth
  - Smaller disks mean quicker rebuild times
  - … but we tend to be offered only larger disks
    - As in all aspects of modern HPC we rely on commodity components

Disk Controllers

- Provision of access to the collection of disks in a file system is through a disk controller
- High performance disk controllers control the RAID arrays in a file system
- Disk controllers organise the RAID arrays and present them to the outside world as LUNs
- Disk controller infrastructure also includes disk enclosures
- Modern disk controllers can each control hundreds of disks and deliver up to 10 GB/s of I/O bandwidth to those disks
  - Palu uses a single DDN SFA10K controller for its /scratch file system
    - Total peak sustained write ~8 GB/s
  - Rosa uses 5 LSI 7900 disk controllers for its /scratch file system
    - Total peak sustained write ~14 GB/s

File Servers

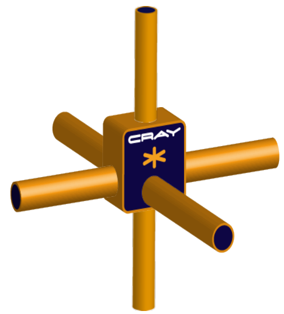
- The storage hardware is presented to the application as a file system, which is mounted from a number of file servers
- File servers are typically standard server nodes with connectivity to the disk controller hardware
  - Often they are simple x86 based servers
- On the Cray XT3/4/5 systems the file servers are nodes on the Cray 3D-Torus
- The file servers are where the file system server software runs

Internal and External File Systems on the Cray XT/E

- The Cray XT3/4/5 systems had the file systems mounted with internal file servers
  - The file servers were nodes on the Cray machine itself
  - The only supported file systems were Lustre and remote NFS mounted file systems
- The Cray XT5/6 and Cray XE6 systems support file systems that are mounted with external file servers
  - The file servers are remote nodes
  - The file servers are reachable from the compute nodes through router nodes on the Cray
  - The Cray XE6 Palu has an external Lustre /scratch file system
  - Additional DVS servers on the Cray provide fast access to other file systems
    - The Cray XE6 Palu has reasonably fast access to the GPFS based file system /project

Network Connectivity

- In order to get data from your application to the storage devices, the data has to be transmitted over networks
- The data first has to travel from the compute node to the server
  - On a Cray it passes across the high-speed network of the Torus
- On a Cray internal file system this is the only network
- For external file systems we typically use Infiniband
- For a Cray XE6 external file system the data has to pass over two networks
  - First internally through the Torus to a router node
  - Then across a PCI-express bus and onto the Infiniband network



File Systems for HPC

- There are a number of vendors who supply file systems for HPC
- The most commonly used file systems in large HPC installations
  - Several vendors supply and support Lustre
  - GPFS from IBM
  - PanFS from Panasas
- A number of other file systems are available for distributed storage
  - Ceph
  - Fraunhofer Parallel File System
  - Parallel Virtual File System
- CSCS uses GPFS for its global file systems such as /project and /users, and Lustre for the /scratch file systems on the Cray

Design Criteria for Parallel File Systems

- Parallel file systems for HPC are designed to handle large volumes of data
- It is expected that users of these file systems write data in large blocks at a time
- Small file writes are not optimised, and may be slow

GPFS Overview

- The General Parallel File System (GPFS) is a high performance file system from IBM
- GPFS delivers high resiliency and failover as well as high performance
  - GPFS can be upgraded live without needing to take the file system out of service
  - GPFS has maximum redundancy to avoid file system failures
  - GPFS has a distributed metadata model to avoid a single point of failure
- GPFS hides most of the complexity of the file system from the user
- GPFS is used at CSCS for the central file systems (/project, /users, /apps and /store)

Lustre Overview

- Lustre is a high performance file system that is used on most of the world's fastest computers
- Lustre was designed to deliver high scalability in numbers of clients and performance in terms of high sustained bandwidth
  - Many resiliency features were not included in the earlier versions
- Lustre allows users to exercise a great degree of control on the mapping of their files onto the underlying hardware
  - The mapping or striping of files onto the underlying hardware is exploited by Cray's MPI library to give fast MPI-IO performance

Common Features of Parallel File Systems

- Most file systems adhere to POSIX standards for I/O
  - GPFS and Lustre are POSIX compliant
- Parallel file systems separate metadata and data storage
- Parallel file systems spread data from an individual file across multiple RAID arrays
  - Writing to a single file can deliver the full performance of the file system hardware
    - Very important for parallel I/O libraries

Metadata and Data

- Parallel file systems such as GPFS and Lustre separate information in a file into Metadata and Data
- Metadata is information about the file itself
  - Filename, where it is in the file system heirarchy
  - Timestamps, permissions, locks
- The data for the file is then separated from the metadata

Compare this to searching for journal papers in a database.

- *Metadata such as journal, data, authors, title, citation count etc. are searchable in the database without needing to see the actual paper itself*
- *The data (text) is then only accessed when the journal paper is required*

N.B. In this case much of the metadata is also normally present in the journal paper itself

Multiple Data Servers

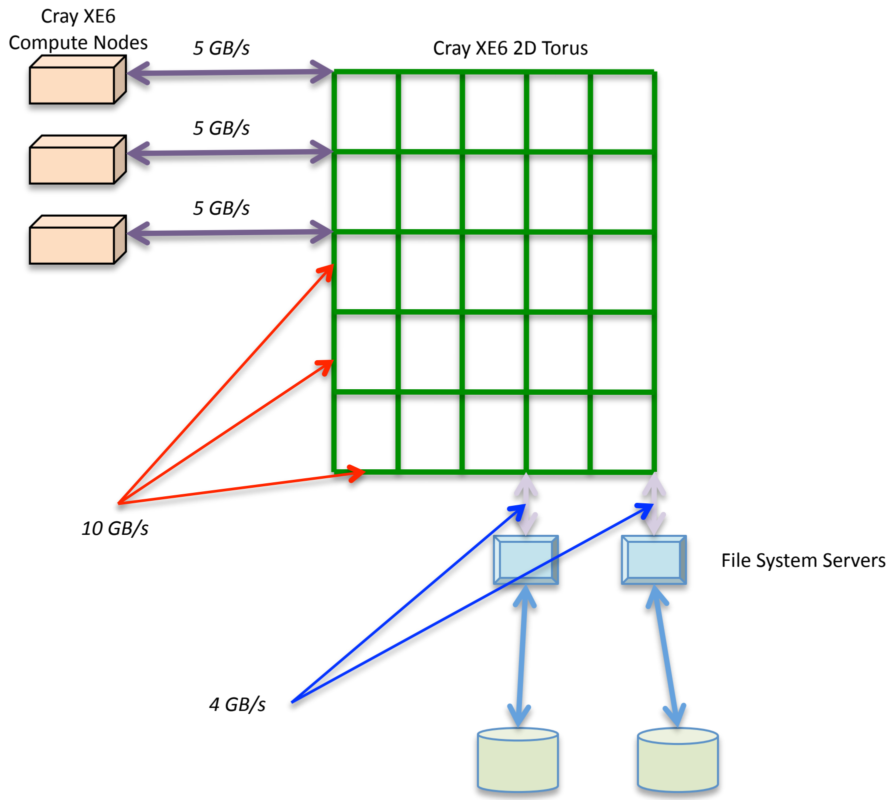
- In large parallel file systems multiple servers are used to look after the disks
- Each server will be responsible for several RAID arrays
- Each RAID array may have multiple servers that *can* read/write to it
  - Typically one server is the primary server, with other servers acting only in the case of primary server failure
- Using multiple servers increases the bandwidth potential of the file system and improves reliability if failover is enabled

Hardware Bottlenecks

- At each stage of the data transfer there are a number of potential bottlenecks
  - Interconnect bottleneck to the metadata or data servers
  - Number of file servers that are responsible for a particular file
  - Capacity of individual servers
  - Number of disk controllers used by each file
  - Capacity of each disk controller
  - Number of RAID arrays used by each file
  - Speed of each RAID array

Contention on Networks

- The networks are shared resources
  - Shared by multiple users
  - Shared by MPI traffic and file system traffic
  - Shared by multiple processes on a node
- Contention can occur at several stages on networks
- For example on a Cray XE6
  - Network injection bandwidth of a compute node is ~ 5 GB/s
  - Link bandwidth on the Torus is approximately 10 GB/s
  - The router nodes can receive data at ~ 5 GB/s
  - The router nodes use PCI-express and QDR Infiniband to access the external file servers
  - The file servers use QDR Infiniband to talk to the disk controller



Contention on the Data Targets

- The data targets can have contention when multiple writes try to access the same RAID arrays
  - This could be multiple writes from the same process
  - Could be multiple writes from multiple processes
    - From one job, multiple jobs or multiple users
- It is important to not just use a few RAID arrays if I/O bandwidth is the main bottleneck

Contention for Metadata Operations

- File metadata operations can also be a bottleneck
- Repeatedly opening and closing files can put a strain on the file system
  - Even for parallel I/O libraries, each process needs to issue a metadata operation to get its own file handle in order to work on the file
- On the Cray XT5, poor usage patterns from some applications with metadata operations have led to dramatic slowdowns of the whole file system

Deep Dive Lustre

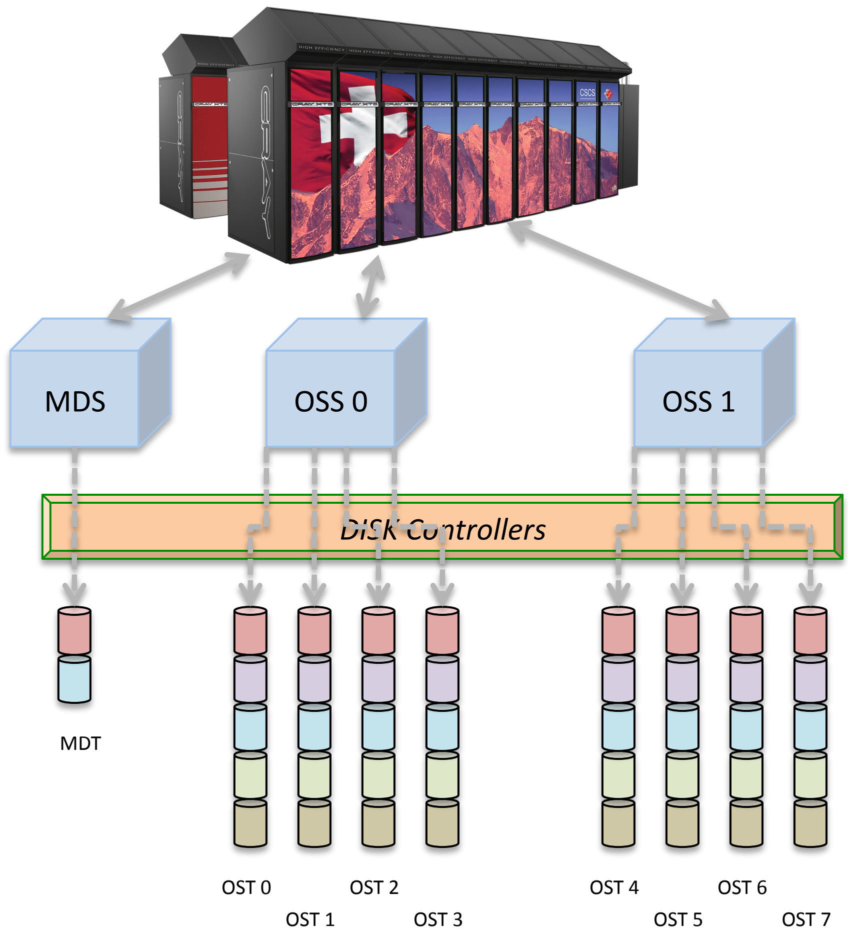
- The Lustre file system is the most widely used file system for HPC
- Lustre was designed to have very good parallel I/O performance from the standpoint of sustained bandwidth
- Lustre was designed to be highly scalable and to be able to serve thousands of clients

Lustre History and Future

- Lustre began development in 1999
- First Lustre release was in 2003
- Current version being shipped with most systems is Lustre 1.8.X
- Lustre was originally developed by Cluster File Systems
  - … who were bought by Sun Microsystems in 2007
  - … who were bought by Oracle in 2009
- In April 2010 Oracle announced that future development of Oracle would only be supported on their hardware
- … but since Lustre was released under the GNU public licence, a consortium of companies and groups continue development themselves
  - Whamcloud, Xyratex, OpenSFS etc.

Lustre Data Layout

- Lustre implements a separation of data and metadata
- The metadata is stored on a Metadata Target (MDT)
- The data is stored on a number of Object Storage Targets (OSTs)
- A Metadata Server (MDS) serves all file system requests for metadata, and it looks after the MDT
- A number of Object Storage Servers (OSS) each look after several OSTs and serve requests for data on those OSTs

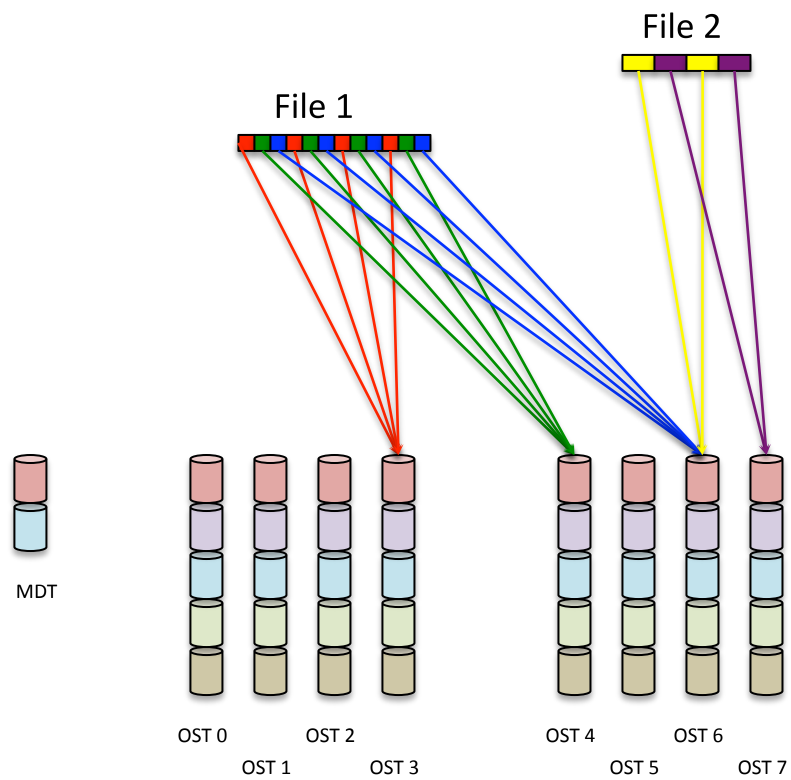


Striping in Lustre

- Lustre allows the user to have explicit control over the how a file is *striped* over the OSTs
- A default is configured by the system adminstrator
  - On Rosa the default is 4 OSTs per file, 1 Mbyte stripes
  - On Palu the default is 8 OSTs per file, 1 Mbyte stripes
- When data is written to a file, it is split into chunks equivalent to the stripe size
- Chunks are then sent to the different OSTs to improve disk bandwidth
- The file system tries to balance the load across all OSTs, but through dynamic file deletion and creation there is usually some imbalance
- Note that small files will typically be only have real data on 1 OST
  - E.g. files less than one megabyte on Rosa and Palu /scratch file systems

Striping Example

- A 12MB file “File 1” is configured with a stripe count of 3, and a stripe size of 1MB
  - The file system assigns OSTs 3, 4 and 6 to the file
- A 8 MB file “File 2” is configured with a stripe count of 2 and a stripe size of 2 MB
  - The file system assigns OSTs 6 and 7 to the file
- Note that both files are being striped onto OST 6



Understanding the File System Layout

```
lfs df [-i|-h]
```

- lfs df gives you information about the Lustre file system to which you have access
  - Number of OSTs
  - Size of OSTs
  - Usage of individual OSTs
- The "-h" option gives output in a more readable format
  - The default is to report numbers in bytes

```
zsh$ lfs df -h
UUID                       bytes        Used Available  Use% Mounted on
scratch-MDT0000_UUID      976.1G        5.0G    915.3G    0% /scratch/rosa[MDT:0]
scratch-OST0000_UUID        3.6T        2.4T      1.0T   66% /scratch/rosa[OST:0]
scratch-OST0001_UUID        3.6T        2.4T    998.7G   67% /scratch/rosa[OST:1]
scratch-OST0002_UUID        3.6T        2.3T      1.0T   65% /scratch/rosa[OST:2]
scratch-OST0003_UUID        3.6T        2.4T      1.0T   65% /scratch/rosa[OST:3]
scratch-OST0004_UUID        3.6T        2.3T      1.1T   64% /scratch/rosa[OST:4]
scratch-OST0005_UUID        3.6T        2.3T      1.1T   65% /scratch/rosa[OST:5]
scratch-OST0006_UUID        3.6T        2.3T      1.1T   65% /scratch/rosa[OST:6]
scratch-OST0007_UUID        3.6T        2.3T      1.1T   65% /scratch/rosa[OST:7]
scratch-OST0008_UUID        3.6T        2.4T      1.0T   66% /scratch/rosa[OST:8]
.
.
.
scratch-OST004b_UUID        3.6T        2.5T    956.3G   68% /scratch/rosa[OST:75]
scratch-OST004c_UUID        3.6T  [illegible] [illegible] [illegible] /scratch/rosa[OST:76]
scratch-OST004d_UUID        3.6T        2.3T      1.1T   63% /scratch/rosa[OST:77]
scratch-OST004e_UUID        3.6T        2.3T      1.1T   63% /scratch/rosa[OST:78]
scratch-OST004f_UUID        3.6T        2.4T   1005.5G   67% /scratch/rosa[OST:79]

filesystem summary:       286.2T      188.6T     83.1T   65% /scratch/rosa
```

The MDT is always first and will normally have very little usage

Reports for individual OSTs will show any imbalance in the system (here 5% difference between OST 75 and OST 77)

The status of the whole file system is reported at the end

Default Striping

```
lfs getstripe file
```

- lfs getstripe tells you the status of striping of a given file or directory
- If you issue “lfs getstripe” on a directory you will get output about all files contained in that directory
- Adding the –v flag gives information about stripe size

```
zsh$ lfs getstripe -v myfile
OBDS:
0: scratch-OST0000_UUID ACTIVE
1: scratch-OST0001_UUID ACTIVE
2: scratch-OST0002_UUID ACTIVE
3: scratch-OST0003_UUID ACTIVE
4: scratch-OST0004_UUID ACTIVE
5: scratch-OST0005_UUID ACTIVE
6: scratch-OST0006_UUID ACTIVE
7: scratch-OST0007_UUID ACTIVE
8: scratch-OST0008_UUID ACTIVE
9: scratch-OST0009_UUID ACTIVE
myfile
lmm_magic:          0x0BD10BD0
lmm_object_gr:      0
lmm_object_id:      0x6bc8517
lmm_stripe_count:   8
lmm_stripe_size:    1048576
lmm_stripe_pattern: 1
        obdidx          objid          objid            group
             1       46301377      0x2c280c1                0
             2       46255933      0x2c1cf3d                0
             3       45929151      0x2bcd2bf                0
             4       46453416      0x2c4d2a8                0
             5       46000715      0x2bdea4b                0
             6       46200177      0x2c0f571                0
             7       46004652      0x2bdf9ac                0
             8       46333367      0x2c2fdb7                0
```

Control of Striping

```
lfs setstripe -s <stripesize> -c <stripecount> -i <firststripe> file
```

- lfs setstripe is used to define the striping of a directory or file
- The <stripesize>must be a multiple of 64 Kbytes
- You cannot change the striping of an existing file
- If the file does not exist it will be created (as a file)
- Setting the <firststripe> is only really useful if the file has just one stripe
  – Or it will fit on one stripe because it will be smaller than <stripesize>
- Setting the striping on a directory then affects all future files and subdirectories created within it

Optimising for Striping

- Different usage characteristics of the file system can benefit from different striping strategies
- Typically if you want to us a parallel I/O library to write a single file then you want to stripe everywhere
- When writing to a set of individual files it may be best to stripe on a small number of OSTs
- In most cases it is best to try different striping strategies and see what works best

PRACTICAL

File Write Example

This example uses a simple MPI code that writes a 1 Gigabyte file per process.

The files are contained in the directory

`/project/csstaff/IO_Course/simplewrite`

The source code is in the file writedata.f90 and there are precompiled executables for Palu and Rosa

On Palu, make a directory under /scratch and copy the executable:

```
$ mkdir $SCRATCH/testwrite
$ cd $SCRATCH/testwrite
$ cp /project/csstaff/IO_Course/simplewrite/writedata_palu .
```

Now start an interactive session on a compute node and run the executable:

```
$ salloc -N 1 --time=00:10:00
salloc: Granted job allocation XXXX
$ aprun -n 1 ./writedata_palu
Writing file took      YYYYYY seconds
Bandwidth was    ZZZZZZ MB/s
```

Repeat the run to confirm the timings (typically I/O results are much more variable than other benchmarks)

Repeat the run with different numbers of processes (e.g. 1 to 16 in powers of 2)

Don't forget to exit the interactive session when you have finished

File Write Analysis

1. What is the bandwidth on 1 process
2. How does the bandwidth vary with the number of processes ?
3. What is the maximum bandwidth you achieve ?
4. Where are your bottlenecks ?
5. *Optional: Repeat the exercise on Rosa*
6. What can you do to improve bandwidth?

The maximum sustained write bandwidth that we have seen on the Palu /scratch file system is ~8 GB/s
The maximum sustained write bandwidth that we have seen on the Rosa /scratch file system is ~14 GB/s