

Advanced Message-Passing Programming

Efficient use of the Lustre Filesystem



Reusing this material



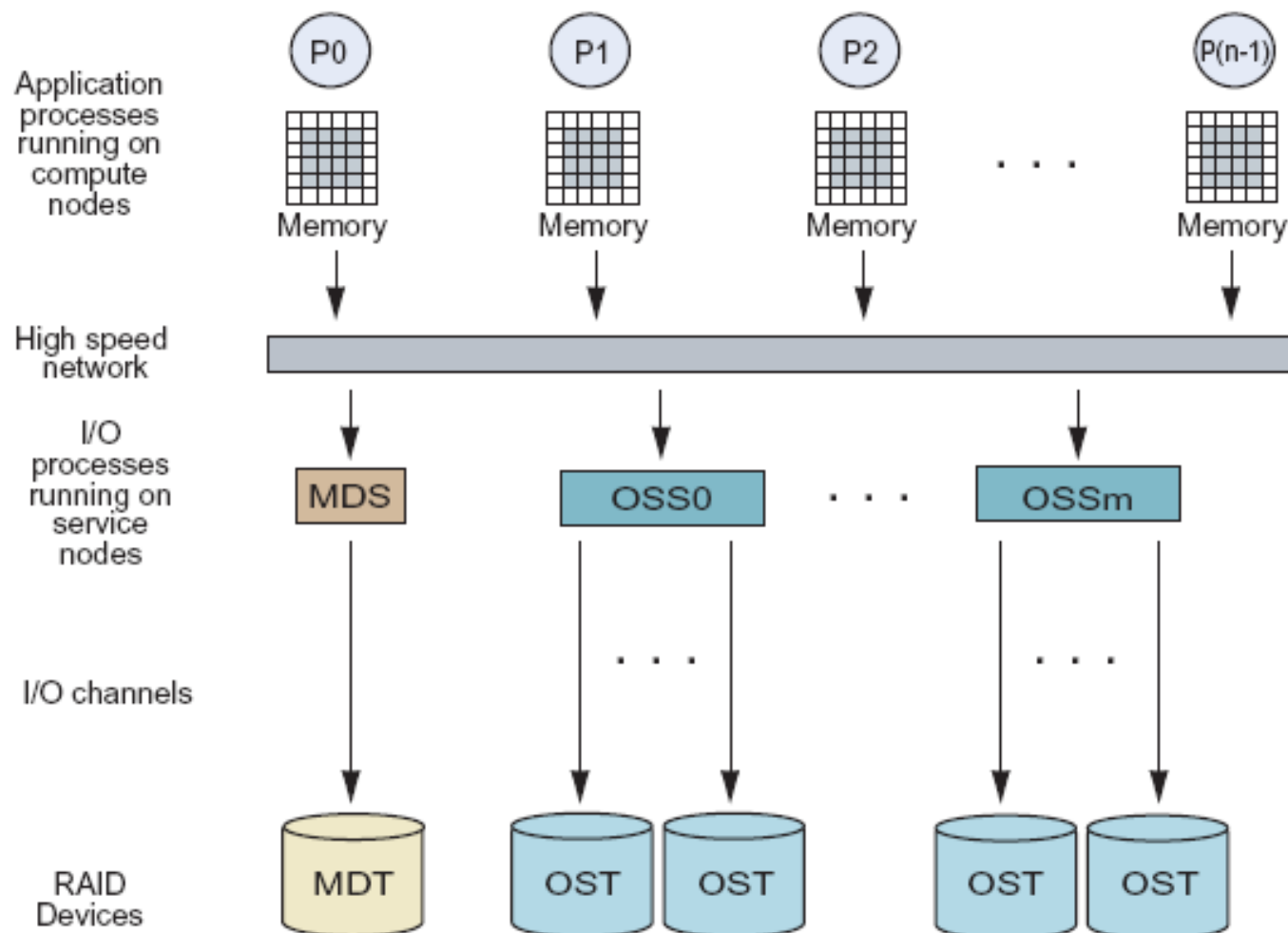
This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

<http://creativecommons.org/licenses/by-nc-sa/4.0/>

This means you are free to copy and redistribute the material and adapt and build on the material under the following terms: You must give appropriate credit, provide a link to the license and indicate if changes were made. If you adapt or build on the material you must distribute your work under the same license as the original.

Note that this presentation contains images owned by others. Please seek their permission before reusing these images.

Lustre: where are the bottlenecks?



Caution!

- IO benchmarking is very difficult and can be non-reproducible
 - you are sharing the ARCHER2 system with other users
 - someone else may also be writing to the same OSTs
 - someone else may be using the same network links
- Caching can give very high IO rates
 - especially with small files
- Ensure benchmarks run for a reasonable time
 - e.g. a few seconds (i.e. large amounts of data)
 - and repeat them several times

ARCHER2 hardware

- Three separate /work filesystems
 - each has 12 OSTs and one MDS
 - consortia assigned to different partitions to share the load
- Additional filesystem with SSDs not spinning disks
 - still to be configured
 - expect better performance for small I/O transactions
 - better latency and maybe better bandwidth
- Each has around 3.3 PiB of storage
 - total of 13.2 PiB (which is 14.5 PB !)

Serial IO

- You are not using parallel libraries
 - single file with controller IO (a single writer)
 - file-per-process (many independent writers)
- Little point in striping the file
 - single file: performance bottlenecks appear to be elsewhere
 - multiple files: already parallel as we are using many disks
- This is why a single stripe is the default
- Ballpark figure: single process can achieve about 1 GiB/s
 - in the absence of caching, i.e. writing very large files

MDS performance

- The MDS can become overloaded
 - e.g. opening and closing a file requires MDS access
 - this is therefore a serial operation
 - and you share the MDS with all users on the same filesystem
 - do not do multiple “open/seek/close” operations on the same file
- Tricks
 - try not to write too many files
 - a simple trick is file-per-node rather than file-per-process
- If you must have lots of files, consider multiple directories
 - e.g. a directory per node
 - decreases lookup times for files

Stripe size

- Default is 1 MiB
 - seems quite small for very large datasets
 - experiment with larger settings:
`lfs setstripe -S 4m <dir/file>`
- Do not set too large
 - as you won't be using all of the OSTs!

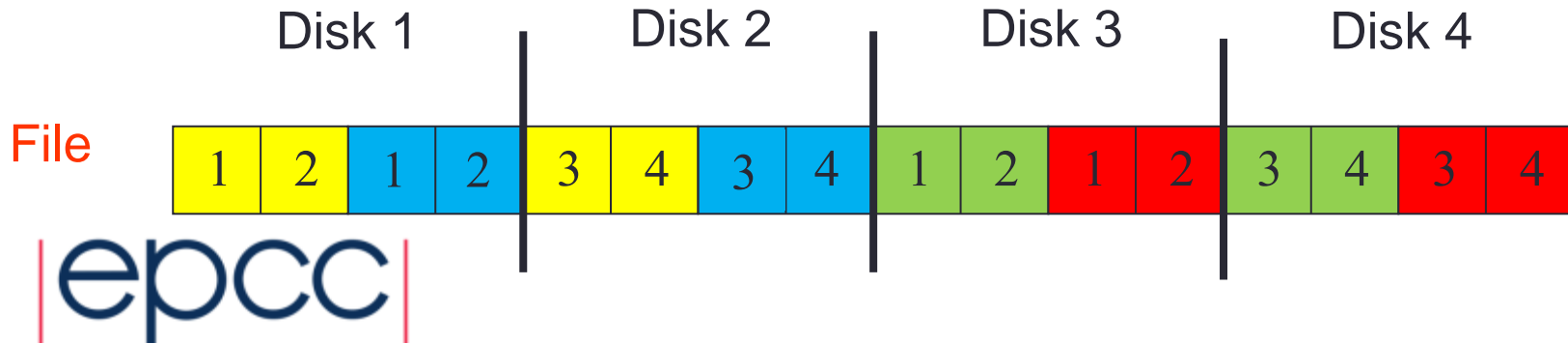
How does parallel IO work?

- MPI-IO auto-configures to the Lustre settings
- Identifies a small number of aggregator processes
 - spread across nodes if possible
- Uses MPI collectives (e.g. scatter/gather) to aggregate data to these processes
 - then performs a small number of large write operations
 - minimises overheads of locking etc.
- Only possible with collective IO calls
- Can get useful runtime statistics by setting
 - `export MPICH_MPIIO_STATS=1`
 - in your SLURM script

4x4 array on 2x2 Process Grid

Parallel Data

2	4	2	4
1	3	1	3
2	4	2	4
1	3	1	3



Aggregators

- By default, Cray MPI uses one aggregator per stripe
 - does not seem optimal as single-process IO is slow
- This default can be changed
- E.g. to use four times as many aggregators (four per node)

```
export MPICH_MPIIO_HINTS=:cray_cb_nodes_multiplier=4
```

- This causes multiple processes to write to the same OST
 - default locking approach is very inefficient
 - try using `:cray_cb_write_lock_mode=2;:cray_cb_nodes_multiplier=4`

Switching MPI versions

- Default low-level network layer: OFI, Open Fabrics Interface
 - can sometimes have poor MPI collective performance
 - fix (which is already in the default script) is:

```
export FI_OFI_RXM_SAR_LIMIT=64K
```

- Can switch MPI to use UCX, developed by Mellanox
 - UCX seems to be better for collectives with heavy comms loads
 - no need to recompile – just alter your SLURM script

```
module swap craype-network-ofi craype-network-ucx  
module swap cray-mpich cray-mpich-ucx
```

Common mistakes

- Serial IO mistakes
 - text data files
 - multiple accesses to a large number of small files
 - using striping for serial IO
- Parallel IO mistakes
 - using a parallel IO library but not striping the file
 - I would recommend full striping across all OSTs
 - using non-collective (i.e. individual) IO to a single file
 - you must quantify performance in terms of GiB/s
 - you should be able to achieve in excess of 10 GiB/s for parallel write

Conclusions

- For parallel IO to a single shared file
 - use a high-level parallel IO library
 - use collective calls
 - stripe file across all OSTs: `lfs setstripe -c -1 <output_dir>`
- For serial IO (single or multiple files)
 - do not stripe the files (e.g. accept defaults on ARCHER2)
 - consider writing file per node
 - controller IO: only uses a single disk
 - file-per-process: may overload the MDS, especially with many users
 - consider file-per-node, or multiple files-per-node to utilise full bandwidth