I/O Optimization

Gordon McPheeters
ALCF

www.anl.gov

Acknowledgments

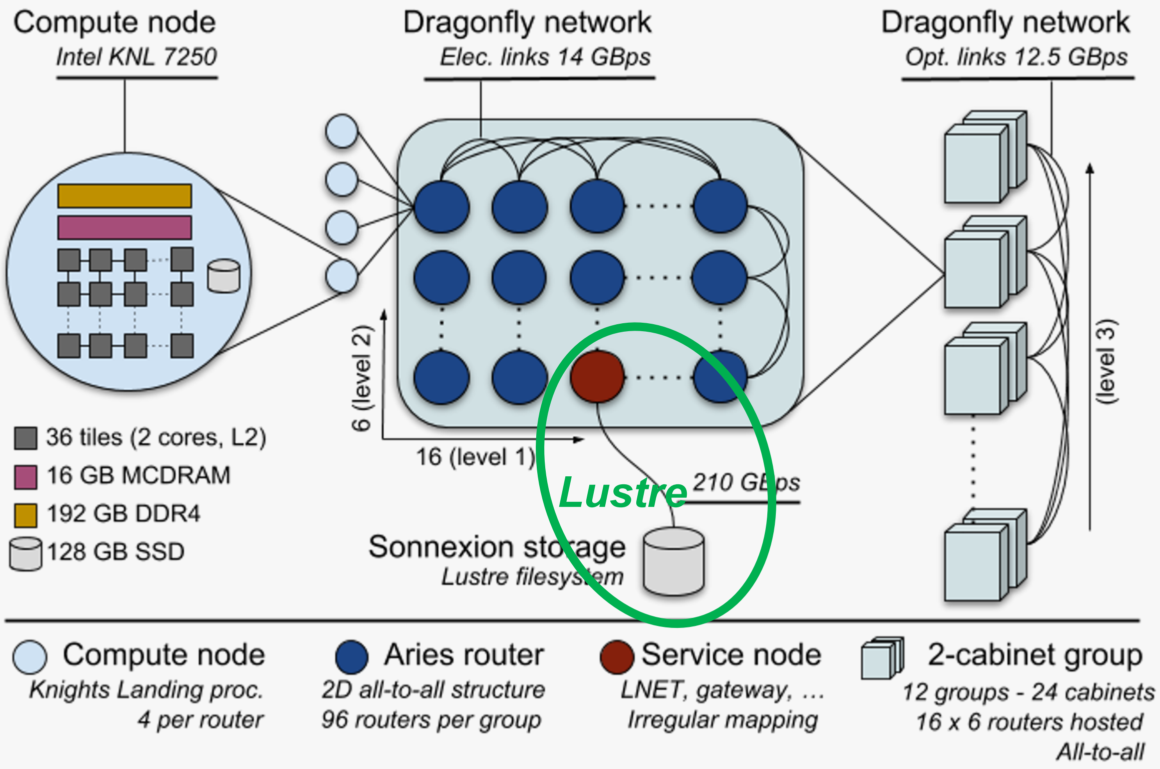
Content Contributed by multiple people @ ALCF

— Kevin Harms, harms@anl.gov

— Alex Kulyavtsev, alexku@anl.gov

— Jack O’Connell, joconne@alcf.anl.gov

Theta Overview



Architecture: Cray XC40

Processor: 1.3 GHz Intel Xeon Phi 7230 SKU

Peak performance of 11.69 petaflops

Racks: 24

Nodes: 4,392

Total cores: 281,088

Cores/node: 64

Memory/node: 192 GB DDR4 SDRAM
(Total DDR4: 843 TB)

High bandwidth memory/node: 16 GB MCDRAM
(Total MCDRAM: 70 TB)

10 PB Lustre file system

SSD/node: 128 GB
(Total SSD: 562 TB)

Aries interconnect - Dragonfly configuration

Lustre Terminology

Client = Lustre software running on compute node

lnet = Lustre Network (LNET) Router, I/O forwarding

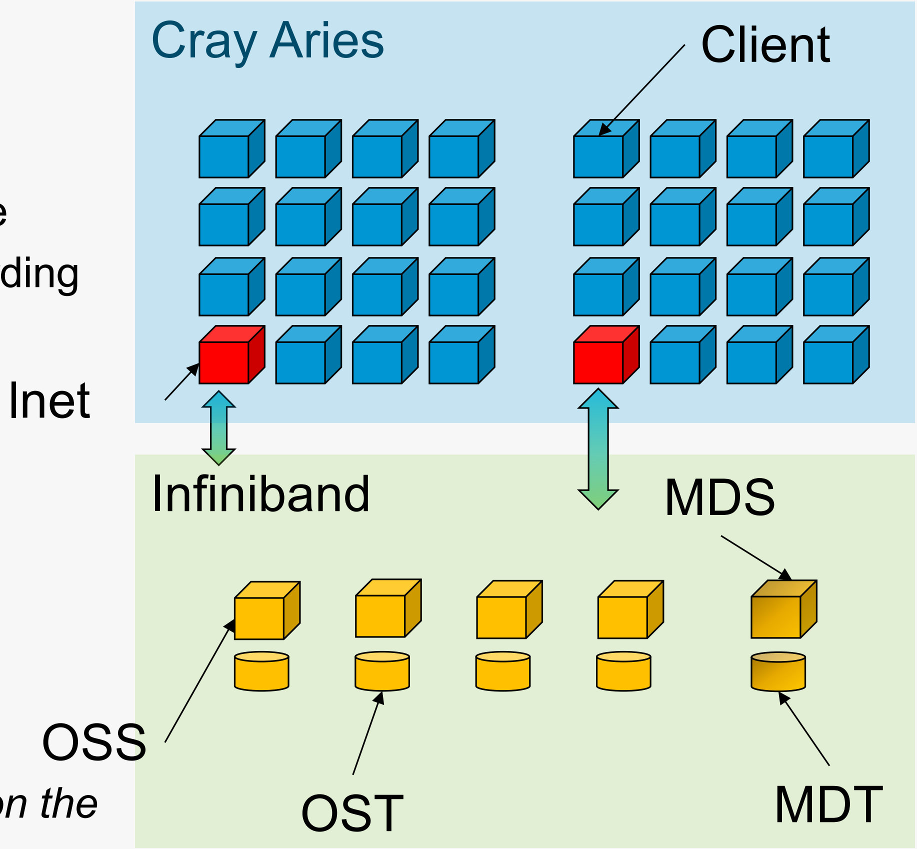
MDS = Metadata Server, manages metadata
Each MDS may serve multiple MDTs

MDT = Metadata Target, metadata storage

OSS = Object Storage Server, manages data
Each OSS may serve multiple OSTs

OST = Object Storage Target, data storage

Each file is distributed over 1+ OSTs, depending on the size and striping settings for the specific file.



Theta – File Systems 1/2

/projects/$PROJECT aka /lus/theta-fs0

- Lustre 2.12
- **Hardware**
  - 10 PB usable RAID storage
  - 56 OSS (1 OST per OSS)
- **Performance (when new):**
  - Total Write BW **172 GB/s**
  - Total Read BW **240 GB/s**
  - Peak Performance of 1 OST is 6 GB/s
  - **Lustre client-cache effects may allow higher apparent BW**
  - **Single client ~ 1GB/s**

/home/$USER aka /gpfs/mira-home

- GPFS 4.0
- Accessed via DVS service
- **Hardware**
  - 1 PB usable RAID storage
- **Performance**:
  - Not for performance of bulk I/O
  - Optimized for storage efficiency of code and binaries

Theta – File Systems 2/2

/grand/projects/$PROJECT
/eagle/projects/$PROJECT

- Lustre 2.12
- **Hardware (Each file system)**
  - 100 PB usable RAID storage
  - 40 OSS (4 OST per OSS) (HDD)
  - 20 MDS (2 MDT per MDS) (NVMe Flash)
- **Performance**:
  - Total Write BW **650 GB/s**
  - Total Read BW **650 GB/s**
  - Peak Performance of 1 OST is 6 GB/s
  - **Lustre client-cache effects may allow higher apparent BW**

Theta – File Systems - Tools

All Lustre file systems maintain quotas at the project level.

```
gmcpheet@thetalogin1:~> myprojectquotas

Lustre : Current Project Quota information for projects you're a member of:

Name                           Type     Filesystem          Used          Quota           Grace
===============================================================================================
Acceptance                  Project     theta-fs0         6.609M           100T               -
Operations                  Project     theta-fs0         359.9T           700T               -
Maintenance                 Project     theta-fs0         2.241G           100T               -
CFS_UX_TEST                 Project     theta-fs0             4k             1T               -
Operations                  Project     grand                 0k             0k               -
Maintenance                 Project     grand                 0k             0k               -
RAN                         Project     grand             661.2G             5T               -
CFS_UX_TEST                 Project     grand                12k             1T               -
Operations                  Project     eagle             1.514T             0k               -
CFS_UX_TEST                 Project     eagle               524k             1T               -
gmcpheet@thetalogin1:~>
```

I/O Models

I/O Interfaces

POSIX I/O

- Standard API and fully supported by Lustre
- Lowest level API for the system

MPI-IO

- Designed to support parallel I/O
- Independent MPI-IO
  - Each MPI task is handles the I/O independently using *non-collective* calls
    - Ex. `MPI_File_write()` and `MPI_File_read()`
    - Similar to POSIX I/O, but supports derived datatypes (useful for non-contiguous access)
- Collective MPI-IO
  - All MPI tasks participate in I/O, and must call the same routines.
    - Ex. `MPI_File_write_all()` and `MPI_File_read_all()`
  - Allows MPI library to perform collective I/O optimizations (often boosting performance)

I/O Libraries

Cray PE offers several pre-built I/O libraries

- module avail provides list of available libraries
- HDF5
  - cray-hdf5-parallel/1.10.6.1
  - `hid_t xferPropList = H5Pcreate(H5P_DATASET_XFER);`
  - `H5Pset_dxpl_mpio(xferPropList, {H5FD_MPIO_INDEPENDENT, H5FD_MPIO_COLLECTIVE});`
  - Metadata collectives
    - `H5Pset_all_coll_metadata_ops`, `H5Pset_coll_metadata_write` as of release 1.10.0
- NetCDF
  - cray-netcdf/4.7.3.3(default)
- PNetCDF
  - cray-parallel-netcdf/1.12.0.1(default)

- ALCF strongly recommends the use of high-level I/O libraries
  - Provide portability
  - Baseline performance should be good out-of-the-box

Files

File per process

- Scales to O(10000) files
- System default settings work well for FPP
- Can run into issues when MDS is busy

Single shared file

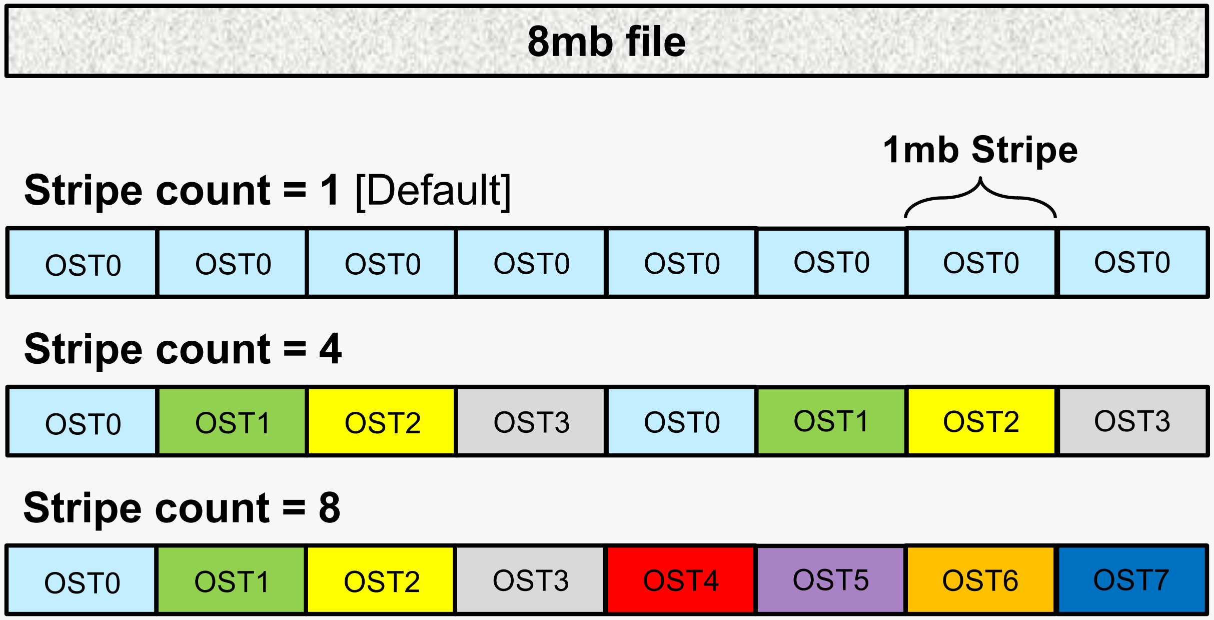
- Write scaling limited by lock contention – read does not have this concern
- Server side does not enable read cache so each IO needs to access disk
- Use MPI-IO
- Need to consider independent versus collective I/O

- ALCF does not recommend any particular approach

Optimization

Lustre File Striping Basics
Key to Parallel Performance

Example: Consider a single **8mb file** with **1mb stripe size…**



8mb file

1mb Stripe

Stripe count = 1 [Default]

OST0	OST0	OST0	OST0	OST0	OST0	OST0	OST0

Stripe count = 4

OST0	OST1	OST2	OST3	OST0	OST1	OST2	OST3

Stripe count = 8

OST0	OST1	OST2	OST3	OST4	OST5	OST6	OST7

Basic Idea
Files are *striped* across OSTs using a predefined striping pattern (pattern = count & size)

Stripe count
The number of OSTs (storage devices) used to store/access the file
`[Default = 1]`

Stripe size
The width of each contiguous OST access
`[Default = 1m]`

Note: **`1m = 1048576`**

Important Notes about File Striping

- Files and directories inherit striping patterns from the parent directory
- Default Striping is `stripe_count=1` and `stripe_size=1048576`
- Don't set the `stripe_offset` yourself (let Lustre choose *which* OSTs to use)
- Stripe count cannot exceed number of OSTs (56 – theta-fs0, 160 – eagle/grand)
- `stripe_count=-1` Use all available OSTs
- Striping cannot be changed once file created
  - Need to re-create file – copy to directory with new striping pattern to change it

Suggestions

- File Per Process
  - Use default stripe count of 1
  - Use default stripe size of 1MB
- Shared File
  - Use 48 OSTs per file for large files > 1 GB
  - Experiment with larger stripe sizes between 8 and 32MB
  - Collective buffer size will set to stripe size
- Small File
  - Use default stripe count of 1
  - Use default stripe size of 1MB

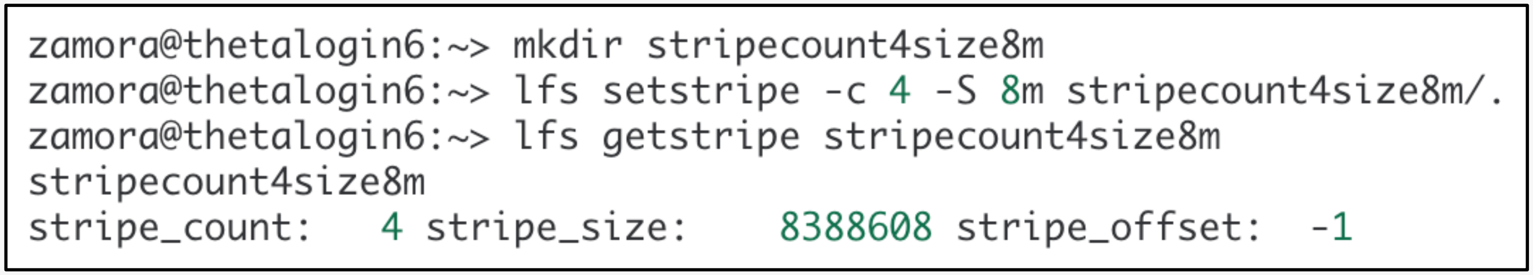
Example: `lfs setstripe`

The stripe settings are critical to performance

- Defaults are **<u>not</u>** optimal for large files

Command syntax:

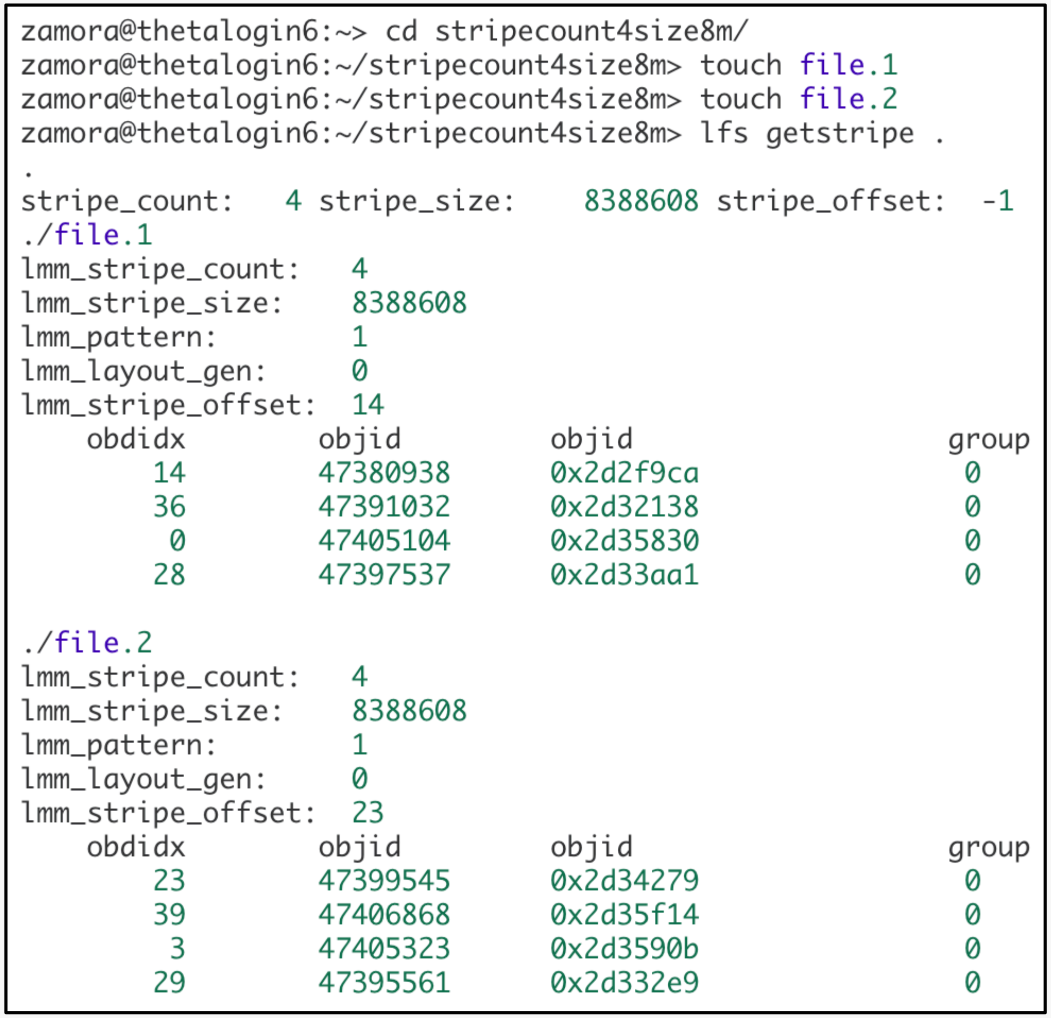
```
lfs setstripe --stripe-size <size> --count <count> <file/dir name>
lfs setstripe –S <size> -c <count> <file/dir name>
```



```
zamora@thetalogin6:~> mkdir stripecount4size8m
zamora@thetalogin6:~> lfs setstripe -c 4 -S 8m stripecount4size8m/.
zamora@thetalogin6:~> lfs getstripe stripecount4size8m
stripecount4size8m
stripe_count:   4 stripe_size:    8388608 stripe_offset:  -1
```

Example:

lfs getstripe



```
zamora@thetalogin6:~> cd stripecount4size8m/
zamora@thetalogin6:~/stripecount4size8m> touch file.1
zamora@thetalogin6:~/stripecount4size8m> touch file.2
zamora@thetalogin6:~/stripecount4size8m> lfs getstripe .
.
stripe_count:   4 stripe_size:    8388608 stripe_offset:  -1
./file.1
lmm_stripe_count:   4
lmm_stripe_size:    8388608
lmm_pattern:        1
lmm_layout_gen:     0
lmm_stripe_offset:  14
    obdidx       objid          objid                group
        14       47380938       0x2d2f9ca             0
        36       47391032       0x2d32138             0
         0       47405104       0x2d35830             0
        28       47397537       0x2d33aa1             0

./file.2
lmm_stripe_count:   4
lmm_stripe_size:    8388608
lmm_pattern:        1
lmm_layout_gen:     0
lmm_stripe_offset:  23
    obdidx       objid          objid                group
        23       47399545       0x2d34279             0
        39       47406868       0x2d35f14             0
         3       47405323       0x2d3590b             0
        29       47395561       0x2d332e9             0
```

Cray MPI-IO Optimizations

Lustre Striping

- Can set stripe settings in **Cray MPI-IO** (striping_unit=*size*, striping_factor=*count*)
  - Ex: `MPICH_MPIIO_HINTS=*:striping_unit=<SIZE>:striping_factor=<COUNT>`

Collective Optimization

- Number of aggregator nodes (`cb_nodes` hint) defaults to the striping factor (*count*)
  - `cray_cb_nodes_multiplier` hint will multiply the number of aggregators
  - Total aggregators = `cb_nodes` x `cray_cb_nodes_multiplier`
- Collective buffer size defaults to the stripe size
  - `cb_buffer_size` hint (in ROMIO) is ignored by Cray
  - ROMIO's collective buffer is allocated (according to this setting), but not used
  - `MPICH_MPIIO_HINTS=*:cray_cb_nodes_multiplier=<N>`

Documentation

- `man intro_mpi`

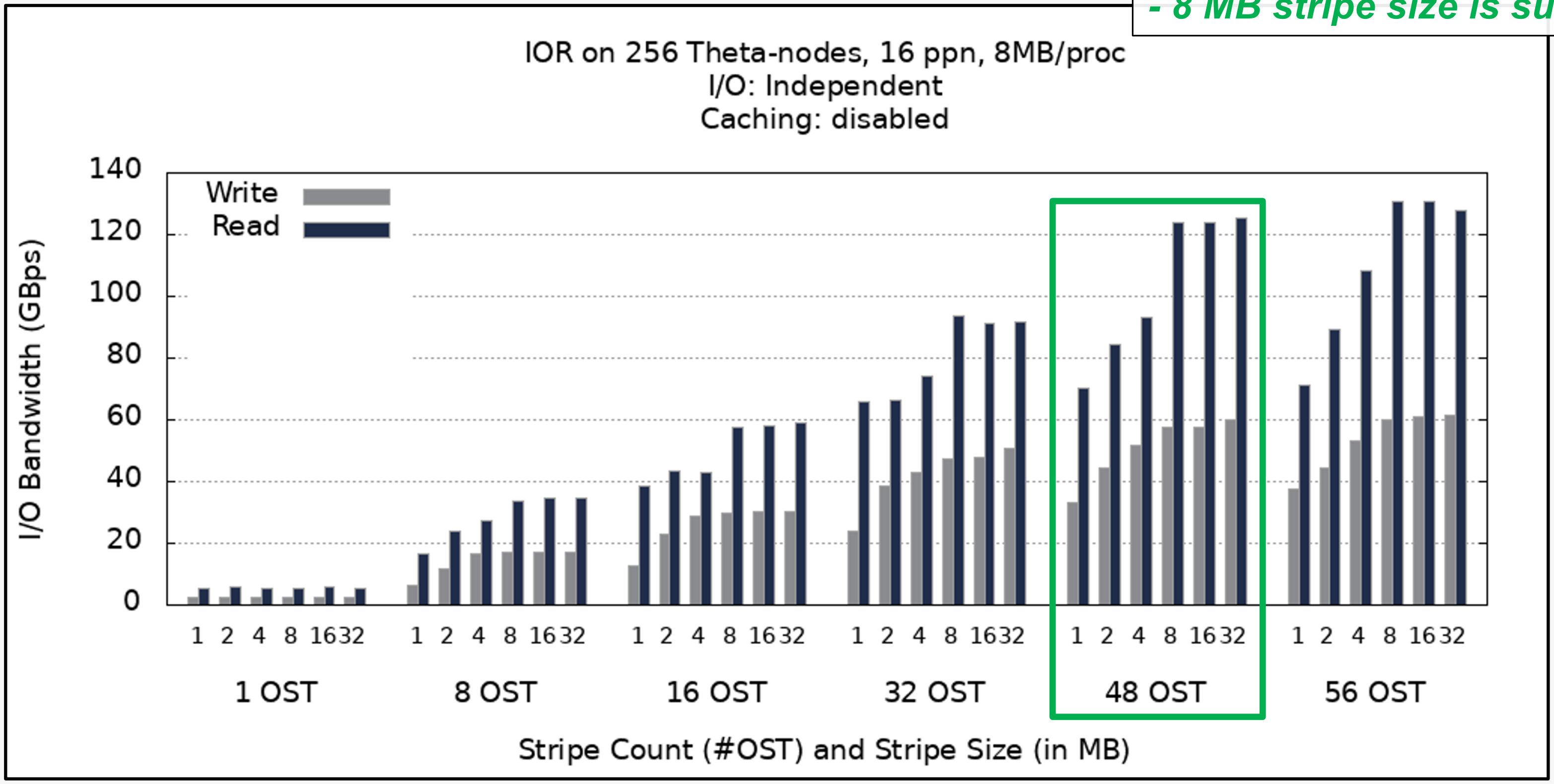
Weigh cost of collective aggregation against optimization of access

- For small discontiguous chunk data, collective faster
- For larger contiguous data, independent read has no lock contention and may be faster
- If rank data is stripe aligned, independent writes may also be faster
- Experiment – implement collective calls (MPI_File_*_all) and then turn off collective aggregation via romio_cb_write and romio_cb_read hints to see which performs better

Shared File – 8MB/proc – Independent I/O

Client-side Caching DISABLED

- More OSTs is better
- 8 MB stripe size is sufficient



Profiling

Darshan I/O Profiling

Open-source statistical I/O profiling tool (https://www.alcf.anl.gov/user-guides/darshan)

- No source modifications, lightweight and low overhead
  - Finite memory allocation (about 2MB) - Overhead of 1-2% total

USE:

- Make sure postscript-to-pdf converter is loaded: `module load texlive`
  - `darshan` module should be loaded by default
- I/O characterization file placed here at job completion:

```
/lus/theta-fs0/logs/darshan/theta/<YEAR>/<MONTH>/<DAY>
```

Format: `<USERNAME>_<BINARY_NAME>_id<COBALT_JOB_ID>_<DATE>-<UNIQUE_ID>_<TIMING>.darshan`

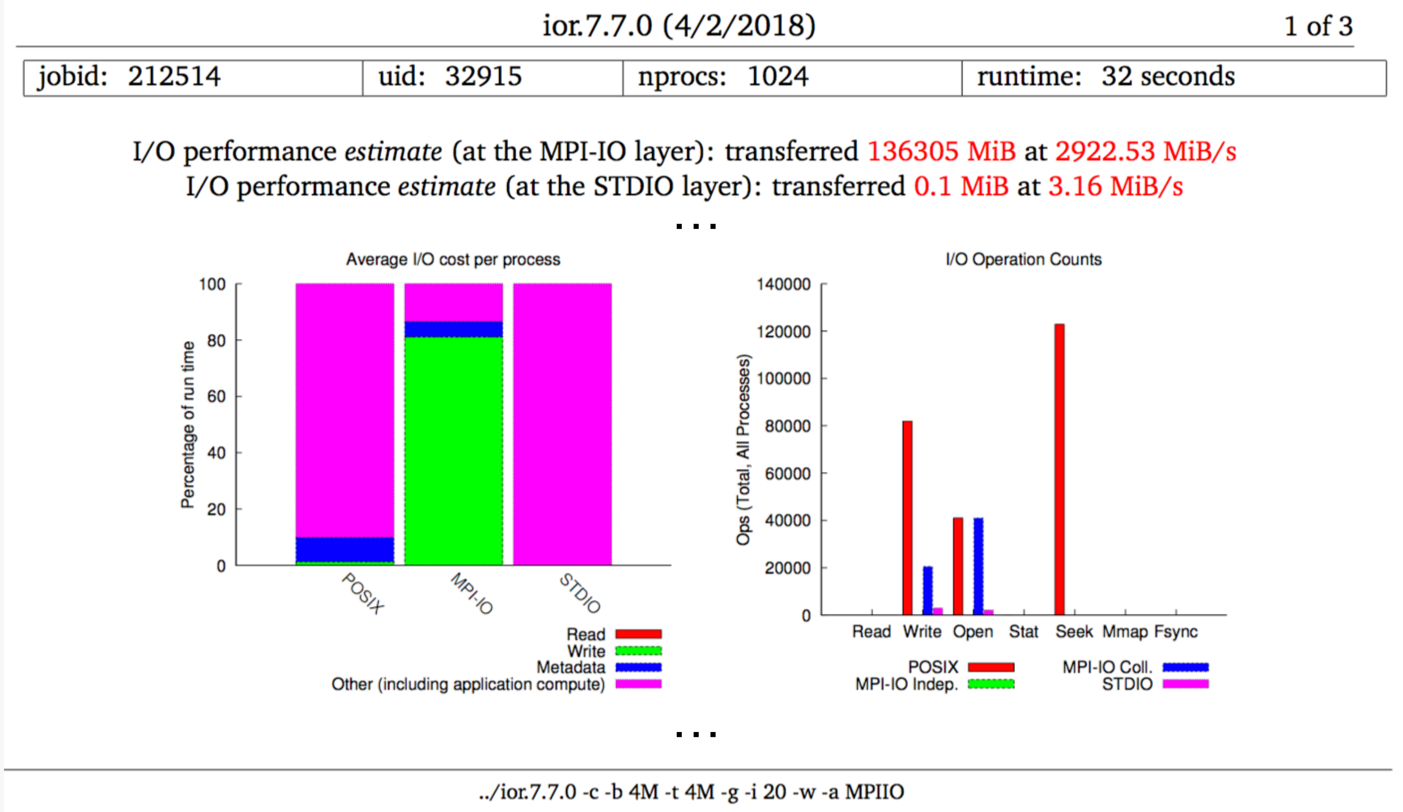
- Use `darshan-job-summary.pl` command for charts, table summaries

```
darshan-job-summary.pl <darshan_file_name> --output darshansummaryfilename.pdf
```

- Use `darshan-parser` for detailed text file

```
darshan-parser <darshan_file_name>  > darshan-details-filename.txt
```

Darshan Output Example



ior.7.7.0 (4/2/2018) 1 of 3

jobid: 212514	uid: 32915	nprocs: 1024	runtime: 32 seconds

I/O performance *estimate* (at the MPI-IO layer): transferred 136305 MiB at 2922.53 MiB/s

I/O performance *estimate* (at the STDIO layer): transferred 0.1 MiB at 3.16 MiB/s

. . .

. . .

```
../ior.7.7.0 -c -b 4M -t 4M -g -i 20 -w -a MPIIO
```

Cray-MPI: Environment Variables for Profiling

- `MPICH_MPIIO_STATS=1`
  - MPI-IO access patterns for reads and writes written to stderr by rank 0 for each file accessed by the application on file close
- `MPICH_MPIIO_STATS=2`
  - set of data files are written to the working directory, one file for each rank, with the filename prefix specified by the `MPICH_MPIIO_STATS_FILE` env variable
- `MPICH_MPIIO_TIMERS=1`
  - Internal timers for MPI-IO operations, particularly useful for collective MPI-IO
- `MPICH_MPIIO_AGGREGATOR_PLACEMENT_DISPLAY=1`
- `MPICH_MPIIO_AGGREGATOR_PLACEMENT_STRIDE`
- `MPICH_MPIIO_HINTS=<file pattern>:key=value:…`
- `MPICH_MPIIO_HINTS_DISPLAY=1`

CrayPat for I/O

- CrayPat uses binary instrumentation
- `module load perftools`
- `pat_build –w –g io –g mpi <binary name>`
- `pat_report –s pe=ALL <pat-dir-name>`

```
Table 5: File Input Stats by Filename
Time | Read MBytes | Read Rate | Reads | Bytes/ Call | File PE
0.645434 | 1,280.719242 | 1,984.275263 | 9,952.0 | 134,940.86 | Total
|-----------------------------------------------------------------------------
| 0.585577 | 1,280.000000 | 2,185.878594 | 1,280.0 | 1,048,576.00 | testFile
||----------------------------------------------------------------------------
|| 0.076877 | 160.000000 | 2,081.242606 | 160.0 | 1,048,576.00 | pe.16
|| 0.074686 | 160.000000 | 2,142.314659 | 160.0 | 1,048,576.00 | pe.17
```

Node-Local

Node Local SSDs on Theta

Node Local SSD

- **128 GB** capacity
- Read ~**1000 MB/s**
- Write ~**500 MB/s**
- Node-limited scope
- Requires explicit manual programming

Use Cases

- Store local intermediate files (scratch)
- Legacy code initialization with lots of small data files – every rank reads
  - Untar into local ssd
- Need to be granted access – PI contact support@alcf.anl.gov

https://www.alcf.anl.gov/user-guides/running-jobs-xc40#requesting-local-ssd-requirements

Using the SSDs on Theta

To access the SSD, add the following in your `qsub` command line:

- **`--attrs ssds=required:ssd_size=128`**
  - This is in addition to any other attributes that you need
  - `ssd_size` is optional

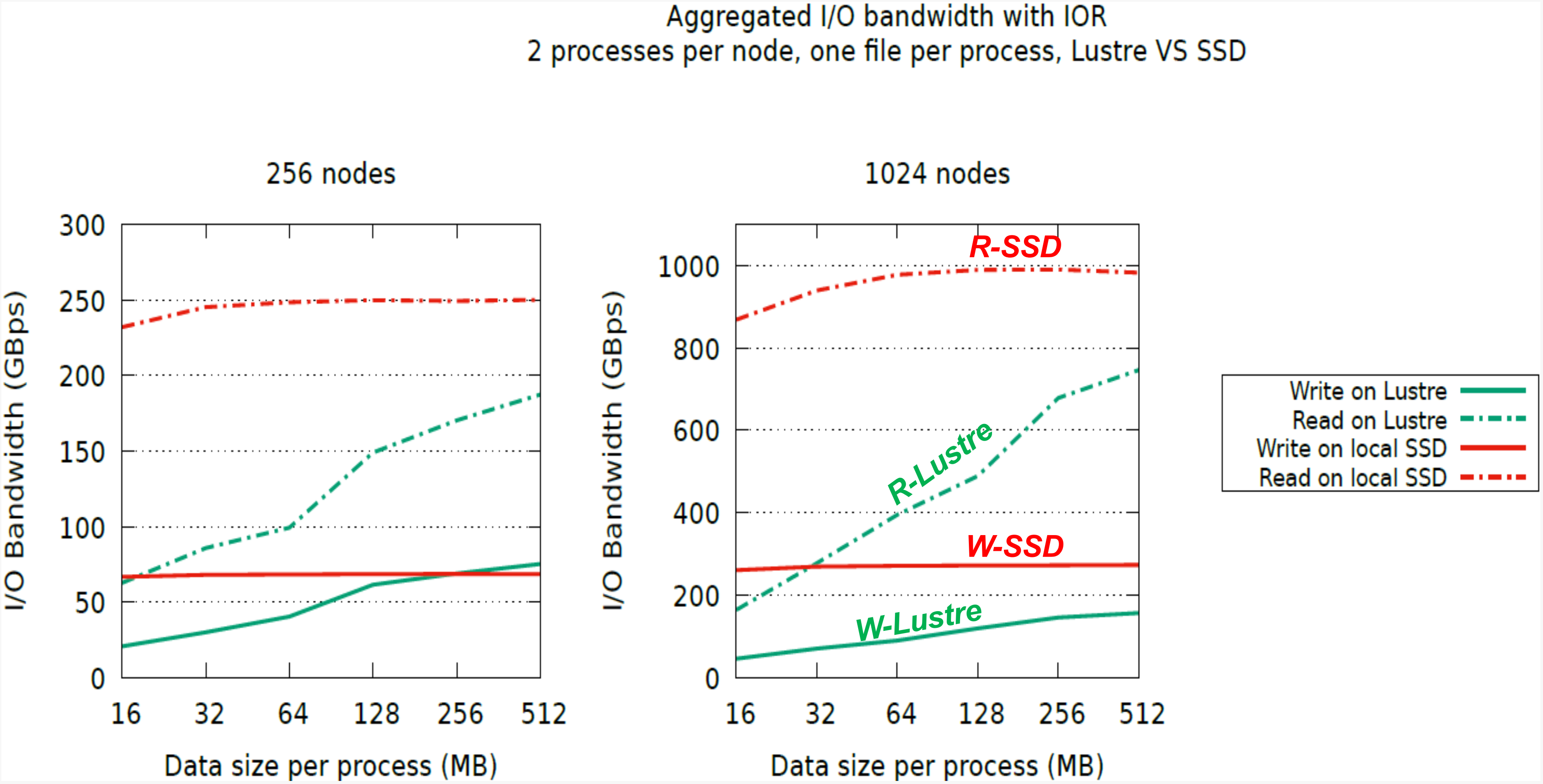
The SSD are mounted on `/local/scratch` on each node

- Data deleted when cobalt job terminates

SSD I/O Performance

- A few SSDs will be slower overall than Lustre, but ...
- Outperforms Lustre at scale based on aggregated bandwidth

Node-Local SSD Performance



Summary

- Use Lustre project file system for best performance
- Set stripe count and stripe size according to usage
- Use I/O libraries or MPI-IO libraries for best performance
- Use Darshan or other profiling tools to investigate current I/O behavior

ALCF Staff is available to help with I/O performance and analysis

Thank You

Questions?