I/O and Data Management

Shane Snyder
Argonne National Laboratory

ALCF Hands-on HPC Workshop, Day 2
October 30, 2024

Managing scientific data

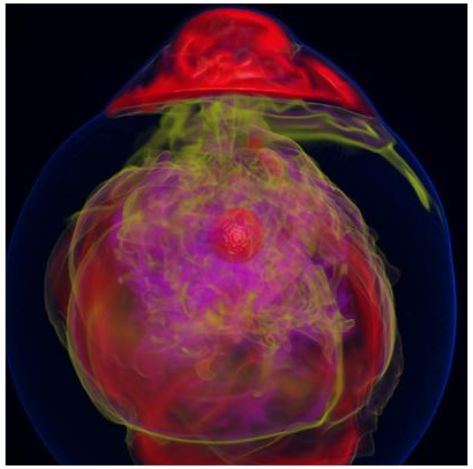
HPC applications spanning various scientific disciplines have a range of diverse data management needs.

- An explosion of scientific data (both in terms of volume and in diversity of access patterns) is compounding the I/O bottleneck, a longstanding performance impediment on HPC systems.

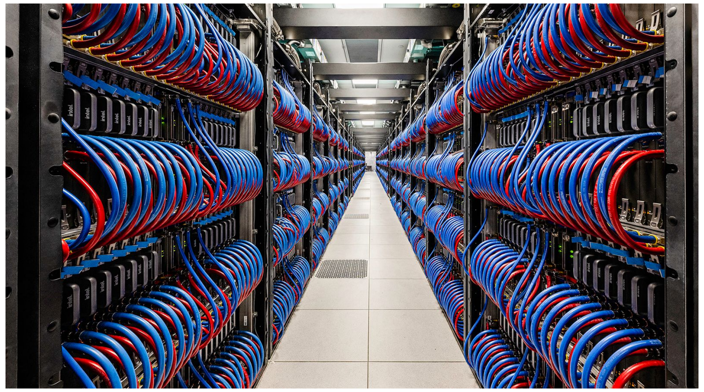
Meanwhile, hardware trends have enabled novel, high-performance storage system designs that promise increased productivity to HPC apps.

ALCF and other HPC facilities deploy vast amounts of storage resources to help meet the I/O needs of HPC applications.

- Today, we'll introduce the basics of the HPC data management stack at ALCF and try to establish some best practices for using it effectively.



Visualization of entropy in Terascale Supernova Initiative application. Image from Kwan-Liu Ma's visualization team at UC Davis.



HPE/Cray Aurora system at the ALCF

Parallel file systems

Parallel file systems (PFSes) have been the traditional tool for storing users' data at HPC facilities for decades now.

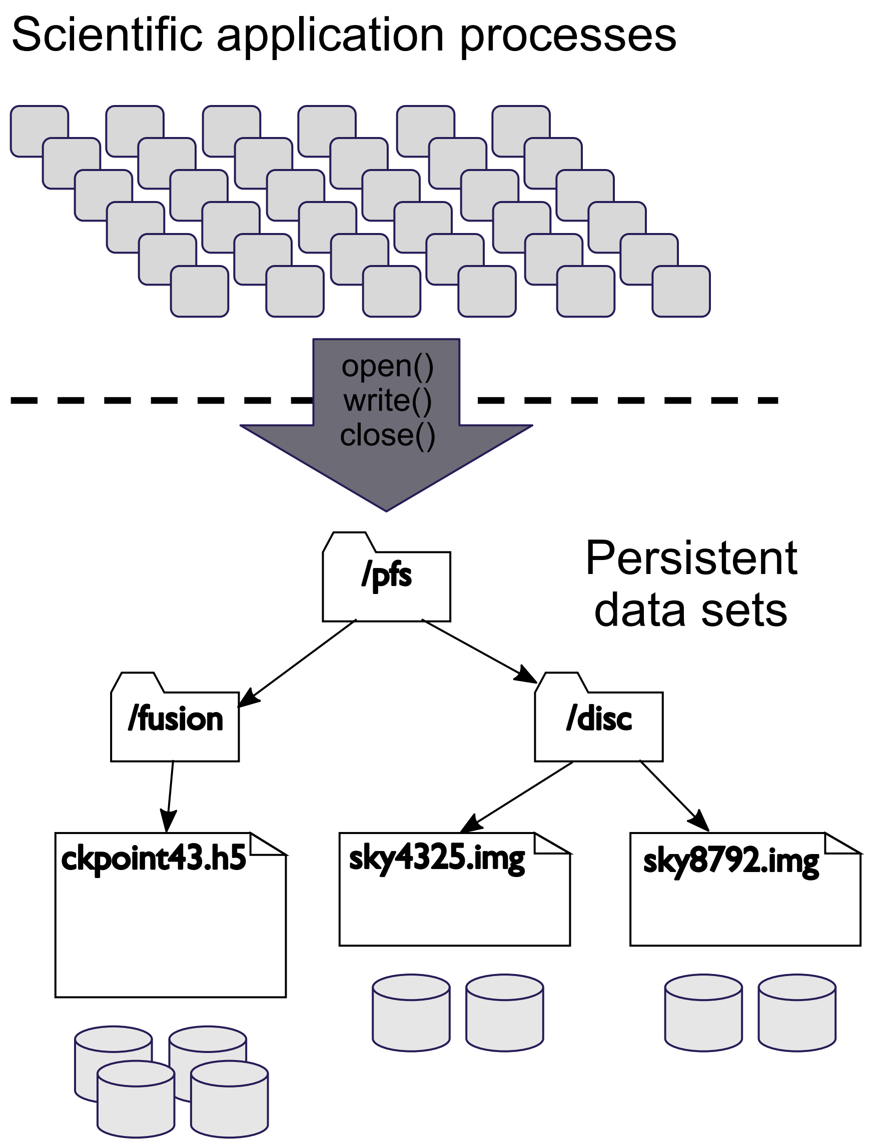
- Users store data in a familiar file/directory hierarchy, but with much more aggregate capacity and performance relative to a local FS.

PFSes offer a number of attractive characteristics that have led to their widespread usage in HPC:

- **High performance** - parallel I/O paths enabling aggregate performance of many storage resources using high speed interconnects
- **Scalability** - storage resources scaled to meet demands of current and future applications
- **Reliability** - failover mechanisms to ensure availability of data in face of failures

Popular PFSes available on modern HPC systems include:

- Lustre
- GPFS (a.k.a Spectrum Scale)
- BeeGFS



Parallel file systems: Lustre

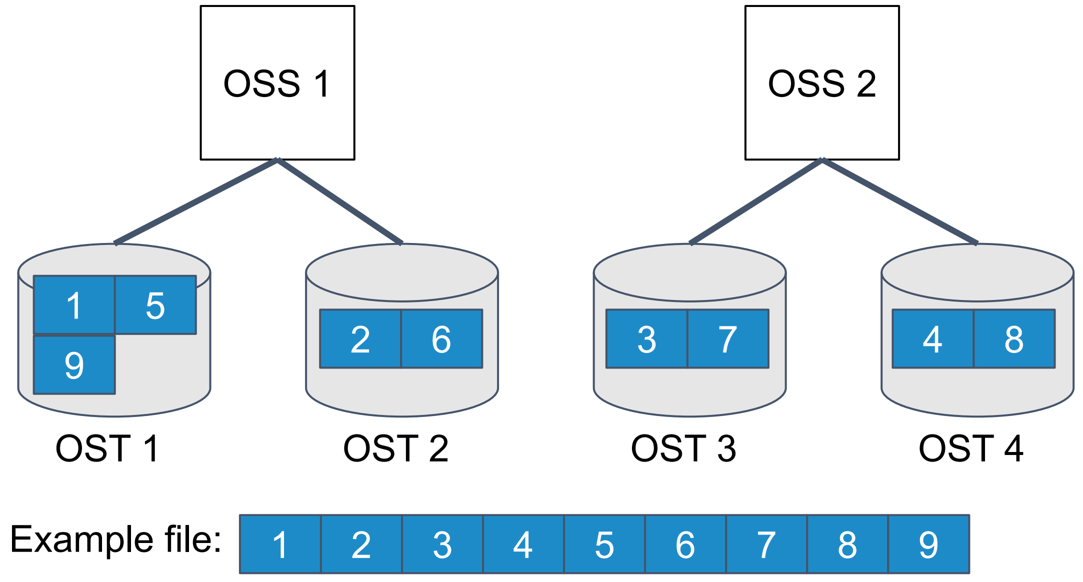
Lustre is currently the preferred scratch file system in production at the ALCF (as well as at many other HPC facilities).

Lustre's design is centered around an object storage service and a metadata storage service.

- Metadata servers (MDSes) manage sets of metadata targets (MDTs).
  - Maintain filesystem namespace and key file metadata
- Object storage servers (OSSes) manage sets of object storage targets (OSTs).
  - Provide bulk storage for file contents

Lustre clients coordinate with metadata servers to set/query file layout, but then interact strictly with storage servers for reading/writing data.

Lustre files are broken into *stripes*, with file stripes round-robin distributed over 1 or more OSTs.



ALCF Polaris file systems

Users have a few different storage options on Polaris:

- Eagle/Grand (Lustre)
  - Temporary storage of I/O intensive data
  - Mounted at `/eagle` and `/grand`, subdirs for each project
  - Community sharing with Globus
  - 100 PB aggregate capacity, 650 GB/sec transfer rate
  - 160 OSTs, 40 MDTs
- Home (Lustre)
  - General-purpose storage of data that is not I/O-intensive (e.g., binaries, source code, etc.)
  - Mounted at `/home`, subdirs for each user
  - Regular backups to tape
  - Default quota of 50 GB
- Node local storage (XFS)
  - Temporary, compute node-local storage for jobs
  - Mounted at `/local/scratch` on each compute node
  - 2 SSDs with total capacity of 3.2 TB, ~6 GB/sec rate
  - Users must copy data somewhere persistent at job end

ALCF Polaris file systems

Users have a few different storage options on Polaris:

- Eagle/Grand (Lustre)
  - Temporary storage of I/O intensive data
  - Mounted at `/eagle` and `/grand`, subdirs for each project
  - Community sharing with Globus
  - 100 PB aggregate capacity, 650 GB/sec transfer rate
  - 160 OSTs, 40 MDTs
- Home (Lustre)
  - General-purpose storage of data that is not I/O-intensive (e.g., binaries, source code, etc.)
  - Mounted at `/home`, subdirs for each user
  - Regular backups to tape
  - Default quota of 50 GB
- Node local storage (XFS)
  - Temporary, compute node-local storage for jobs
  - Mounted at `/local/scratch` on each compute node
  - 2 SSDs with total capacity of 3.2 TB, ~6 GB/sec rate
  - Users must copy data somewhere persistent at job end

Users should always carefully consider whether their usage of production storage resources matches their intended use.

➢ Maximize app performance
➢ Maximize system efficiency
➢ Ensure data integrity

Using Lustre file systems

Achieving the best performance with Lustre often requires thoughtful file striping settings.

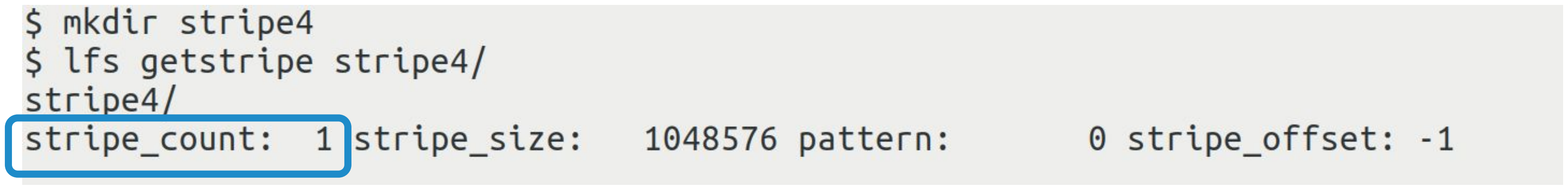
Default Lustre stripe settings may not be optimal! On ALCF systems, files default to a stripe width of 1, meaning they are stored on a single OST. The onus is on users to ensure stripe settings are set appropriately.

Stripe settings can be modified using the `lfs` tool:

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

Using Lustre file systems

Achieving the best performance with Lustre often requires thoughtful file striping settings.



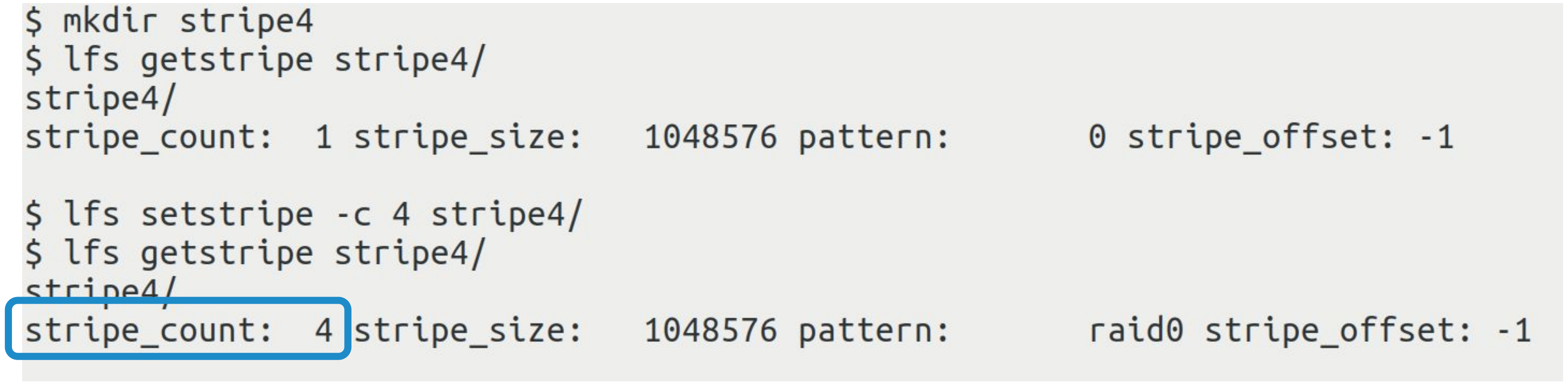
```
$ mkdir stripe4
$ lfs getstripe stripe4/
stripe4/
stripe_count:  1 stripe_size:   1048576 pattern:       0 stripe_offset: -1
```

By default (on this file system), new files/directories are set to use a stripe count of 1.

NOTE: Stripe settings applied to a directory are inherited by all files within it.

Using Lustre file systems

Achieving the best performance with Lustre often requires thoughtful file striping settings.

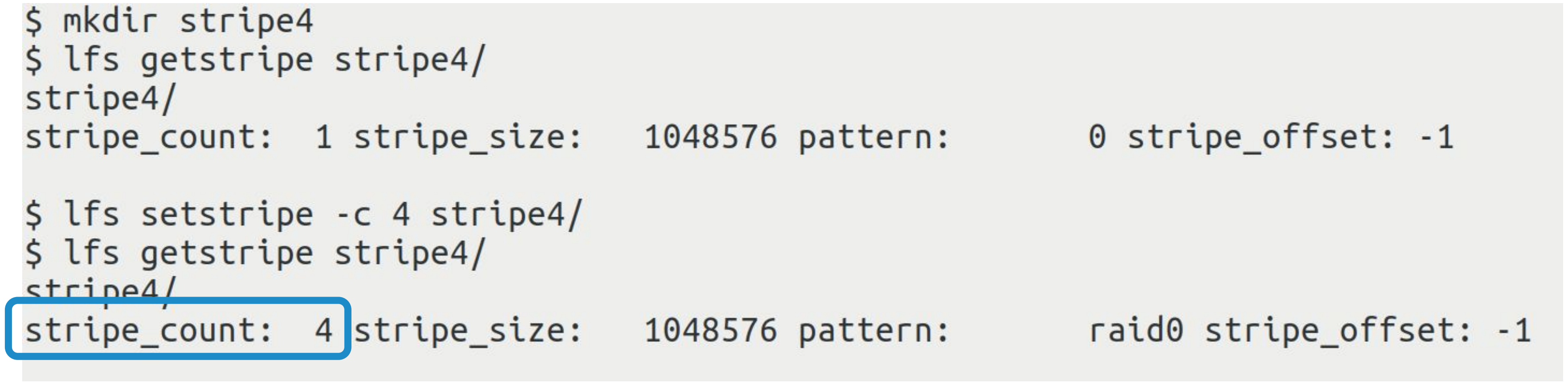


```
$ mkdir stripe4
$ lfs getstripe stripe4/
stripe4/
stripe_count:  1 stripe_size:   1048576 pattern:       0 stripe_offset: -1

$ lfs setstripe -c 4 stripe4/
$ lfs getstripe stripe4/
stripe4/
stripe_count:  4 stripe_size:   1048576 pattern:       raid0 stripe_offset: -1
```

Using the `setstripe` command we can override the default striping and request more storage resources.

Using Lustre file systems

Achieving the best performance with Lustre often requires thoughtful file striping settings.



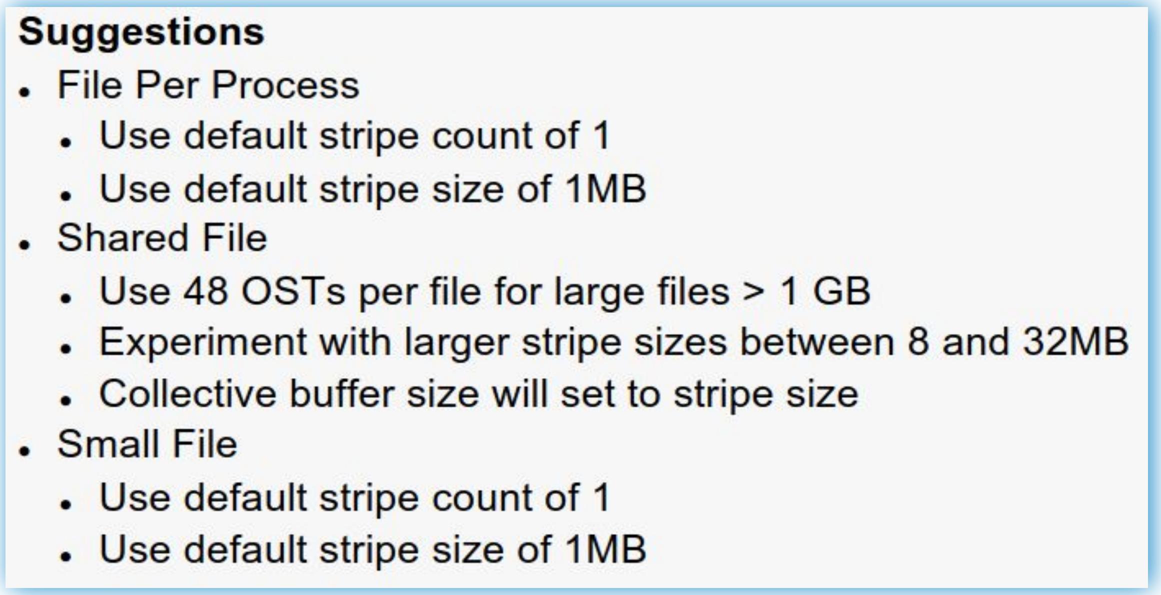
```
$ mkdir stripe4
$ lfs getstripe stripe4/
stripe4/
stripe_count:  1 stripe_size:   1048576 pattern:       0 stripe_offset: -1

$ lfs setstripe -c 4 stripe4/
$ lfs getstripe stripe4/
stripe4/
stripe_count:  4 stripe_size:   1048576 pattern:       raid0 stripe_offset: -1
```

For a demo on how to set file stripe settings and the impact on performance, see the 'lustre-striping' example in the 'file-systems' directory in the hands-on repo.

https://github.com/argonne-lcf/ALCF_Hands_on_HPC_Workshop

Using Lustre file systems

Achieving the best performance with Lustre often requires thoughtful file striping settings.



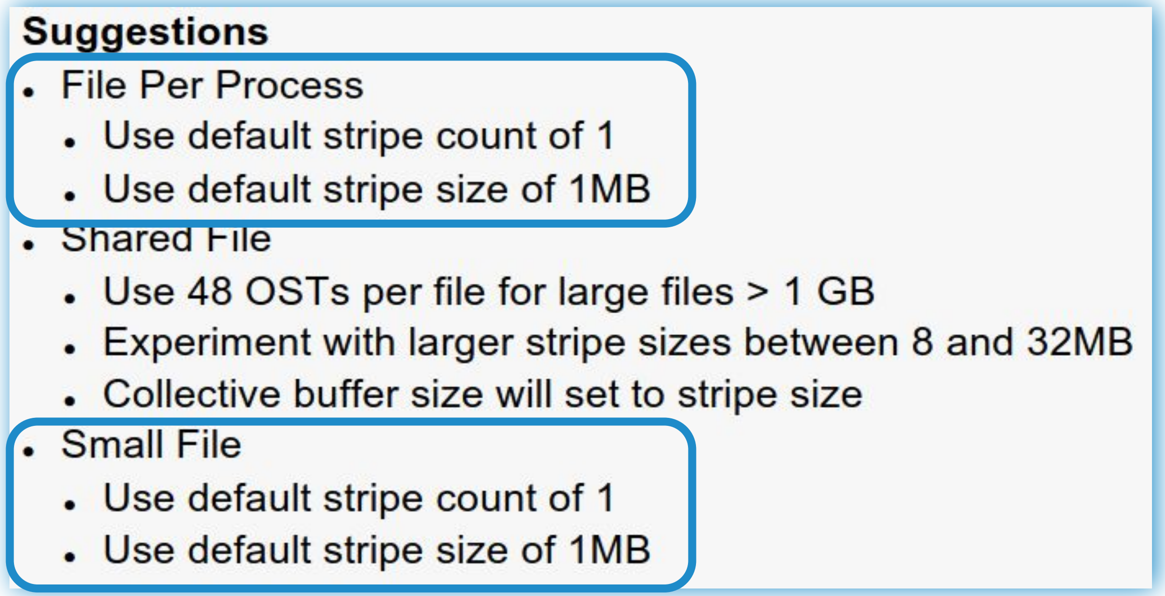
Suggestions

- 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

ALCF suggestions on Lustre file striping settings for achieving optimal I/O performance.

Using Lustre file systems

Achieving the best performance with Lustre often requires thoughtful file striping settings.



Suggestions

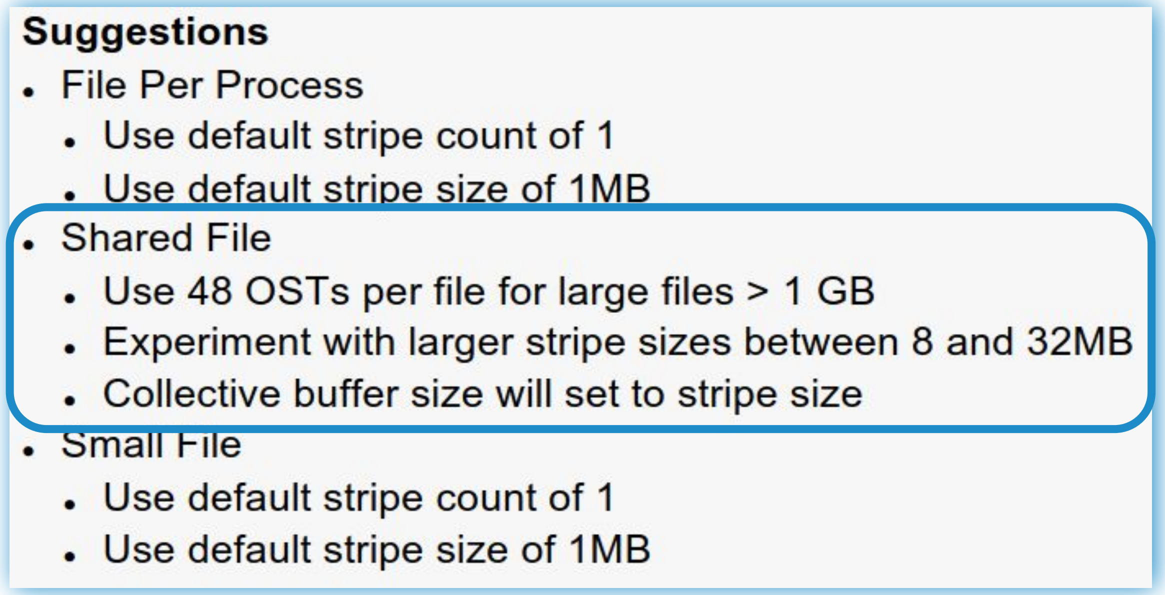
- File Per Process
  - Use default stripe count of 1
  - Use default stripe size of 1MB
- Shared File
  - Use 48 OSTs per file for large files > 1 GB
  - Experiment with larger stripe sizes between 8 and 32MB
  - Collective buffer size will set to stripe size
- Small File
  - Use default stripe count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

ALCF suggestions on Lustre file striping settings for achieving optimal I/O performance.

File per process and small file workloads benefit most from the default settings of striping to a single storage server.

Using Lustre file systems

Achieving the best performance with Lustre often requires thoughtful file striping settings.



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

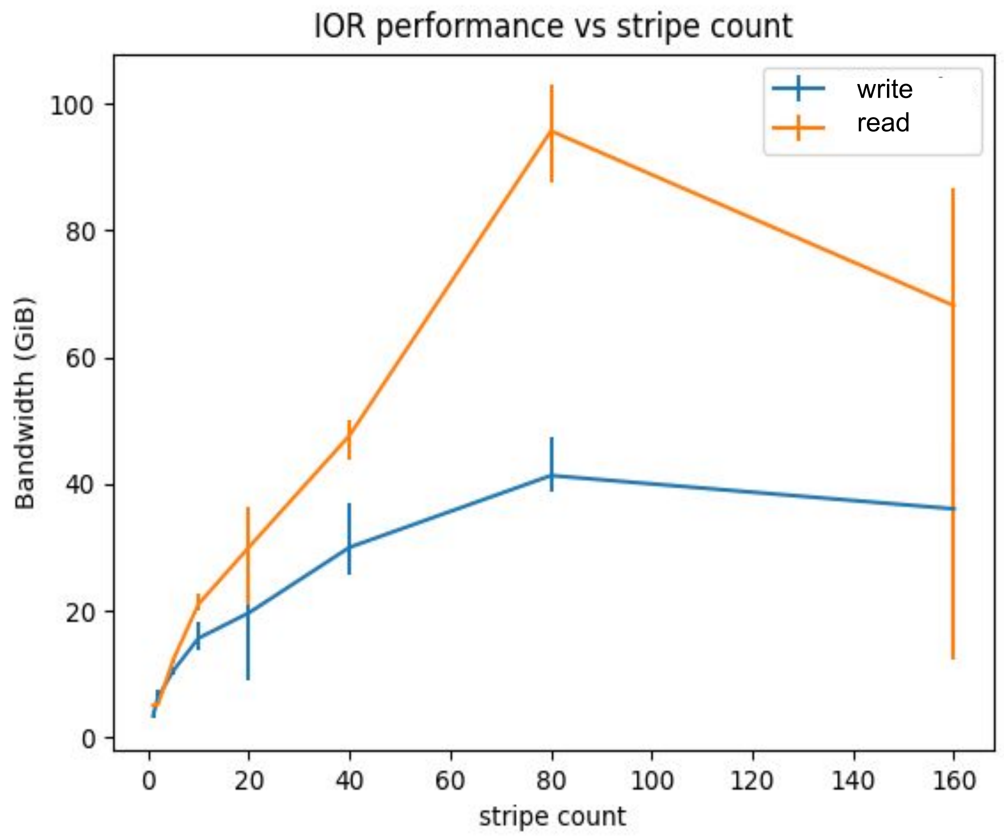
ALCF suggestions on Lustre file striping settings for achieving optimal I/O performance.

Large shared file workloads benefit most from striping across many storage servers and using larger stripe sizes, e.g.:

```
$ lfs setstripe -c 48 -S 16M \
      data.txt
```

Using Lustre file systems

Achieving the best performance with Lustre often requires thoughtful file striping settings.



128-node run of the IOR benchmark using a single shared file and varying the Lustre stripe size on Polaris.

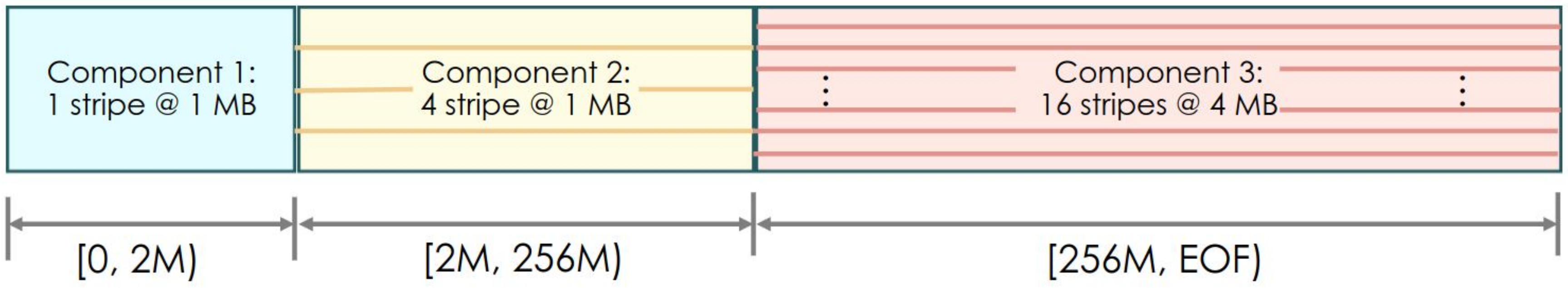
More storage servers drastically improves I/O performance, to a point…

Using Lustre file systems

Recent Lustre versions have introduced a new feature called progressive file layouts (PFL) that enables a more flexible file striping strategy.

- Utilizes a dynamic striping approach that increases the stripe size as the file offset increases
- Small files stored on a single OST, large files grow to stripe across many OSTs

```
$ lfs setstripe -E 2M -c 1 -S 1M \
      -E 256M -c 4 -S 1M \
      -E -1 -c 16 -S 4M \
      data.txt
```



https://wiki.lustre.org/images/5/5a/LUG2022-Advanced_File_Layouts_Tutorial-Mohr.pdf

Using Lustre file systems

Recent Lustre versions have introduced a new feature called progressive file layouts (PFL) that enables a more flexible file striping strategy.

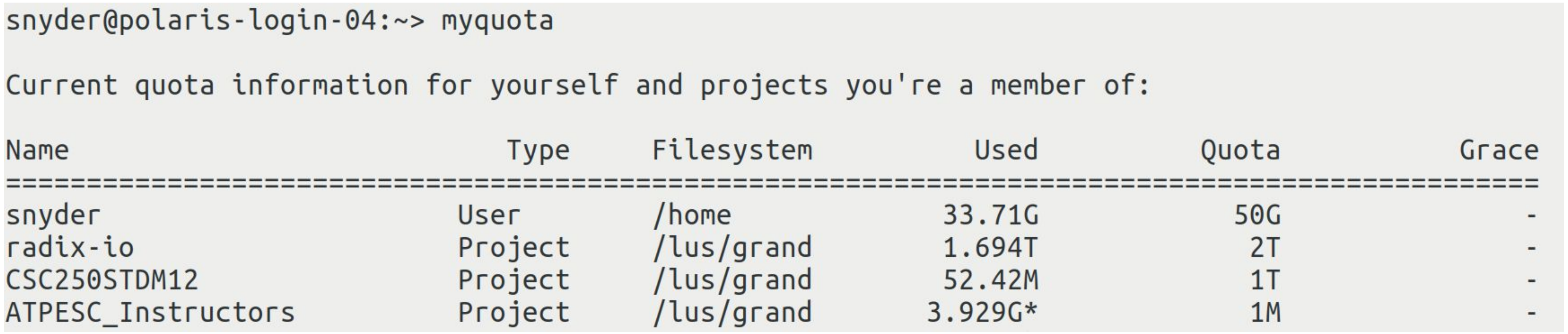
- Utilizes a dynamic striping approach that increases the stripe size as the file offset increases
- Small files stored on a single OST, large files grow to stripe across many OSTs

PFL not enabled by default on ALCF Lustre volumes yet, but something to keep in mind.

PFL can provide default stripe settings that achieve reasonable performance for a variety of I/O workloads, but knowledgeable users can still achieve the best performance by thoughtful tuning of striping.

Using Lustre file systems

ALCF’s Lustre file systems impose disk quotas that may be inspected with the **`myquota`** tool.



```
snyder@polaris-login-04:~> myquota

Current quota information for yourself and projects you're a member of:

Name                          Type      Filesystem          Used          Quota          Grace
==============================================================================================
snyder                     User        /home              33.71G           50G              -
radix-io                   Project     /lus/grand         1.694T            2T              -
CSC250STDM12               Project     /lus/grand         52.42M            1T              -
ATPESC_Instructors         Project     /lus/grand        3.929G*            1M              -
```

`/home` quota is tied to individual users, while project directory quotas (e.g., on `/eagle`) encompass all project users.

Using the node-local SSDs

For some workloads, I/O efficiency can be improved using local SSDs on Polaris as a cache.

- Much lower latency relative to Lustre, but also much less aggregate bandwidth and capacity

Two important things to keep in mind:

1. Data is only visible to processes running on the same node!
   - For MPI jobs spanning multiple nodes, this has obvious data visibility implications.
2. Data staging must be explicitly managed by the user (e.g., in the job script).

Using the node-local SSDs

For some workloads, I/O efficiency can be improved using local SSDs on Polaris as a cache.

- Much lower latency relative to Lustre, but also much less aggregate bandwidth and capacity

Two important things to keep in mind:

1. Data is only visible to processes running on the same node!
   - For MPI jobs spanning multiple nodes, this has obvious data visibility implications.
2. Data staging must be explicitly managed by the user (e.g., in the job script).

```
# one process on each node copies the input file
# from the Eagle file system
mpiexec -n $NNODES --ppn 1 cp \
    /eagle/my_project/input.dat /local/scratch/

# run application using local input file
mpiexec -n $NPROCS --ppn $NPROCS_PER_NODE \
    my_data_reader --input /local/scratch/input.dat
```

SSD stage-in for job read data.

Using the node-local SSDs

For some workloads, I/O efficiency can be improved using local SSDs on Polaris as a cache.

- Much lower latency relative to Lustre, but also much less aggregate bandwidth and capacity

Two important things to keep in mind:

1. Data is only visible to processes running on the same node!
   - For MPI jobs spanning multiple nodes, this has obvious data visibility implications.
2. Data staging must be explicitly managed by the user (e.g., in the job script).

```
# run application writing a local file per process
mpiexec -n $NPROCS --ppn $NPROCS_PER_NODE \
    my_data_writer --output-dir /local/scratch/

# one process on each node (recursively) copies the
# output directory contents to the Eagle file system
mpiexec -n $NNODES --ppn 1 cp -R \
    /local/scratch/ /lus/my_project/job_output/
```

SSD stage-out for persisting job write data.

Using the node-local SSDs

For some workloads, I/O efficiency can be improved using local SSDs on Polaris as a cache.

- Much lower latency relative to Lustre, but also much less aggregate bandwidth and capacity

Two important things to keep in mind:

1. Data is only visible to processes running on the same node!
   - For MPI jobs spanning multiple nodes, this has obvious data visibility implications.
2. Data staging must be explicitly managed by the user (e.g., in the job script).

For a demo on how to stage data to and from the node-local SSDs on Polaris, see the 'ssd-stage-out' and 'ssd-stage-in' examples in the 'file-systems' directory in the hands-on repo.

https://github.com/argonne-lcf/ALCF_Hands_on_HPC_Workshop

I/O libraries: interacting with file systems

HPC apps, of course, need an interface to interact with file systems and manage their data.

- Most file systems (including PFSes) expose a POSIX-like interface that should be familiar to many programmers.

POSIX (Portable Operating System Interface)

- Standard interfaces for portably interacting with file systems, e.g.:
  - `open()`, `read()`, `lseek()`, `close()` operations
- Semantics guaranteed by each operation, e.g.:
  - *successful writes to a file must be immediately visible to subsequent reads*

I/O libraries: interacting with file systems

HPC apps, of course, need an interface to interact with file systems and manage their data.

- Most file systems (including PFSes) expose a POSIX-like interface that should be familiar to many programmers.

POSIX (Portable Operating System Interface)

- Standard interfaces for portably interacting with file systems, e.g.:
  - `open()`, `read()`, `lseek()`, `close()` operations
- Semantics guaranteed by each operation, e.g.:
  - *successful writes to a file must be immediately visible to subsequent reads*

This semantic is tricky to enforce for PFSes where potentially hundred of thousands of clients collectively access and cache file contents.

I/O libraries: interacting with file systems

HPC apps, of course, need an interface to interact with file systems and manage their data.

- Most file systems (including PFSes) expose a POSIX-like interface that should be familiar to many programmers.

POSIX (Portable Operating System Interface)

- Standard interfaces for portably interacting with file systems, e.g.:
  - `open()`, `read()`, `lseek()`, `close()` operations
- Semantics guaranteed by each operation, e.g.:
  - *successful writes to a file must be immediately visible to subsequent reads*

POSIX was never designed or necessarily intended for the large-scale parallel file access.

- Inflexible, strong consistency requirements often lead PFSes to implement elaborate locking protocols or to eschew strong consistency entirely.

I/O libraries: interacting with file systems

HPC apps, of course, need an interface to interact with file systems and manage their data.

- Most file systems (including PFSes) expose a POSIX-like interface that should be familiar to many programmers.

POSIX (Portable Operating System Interface)

- Standard interfaces for portably interacting with file systems, e.g.:
  - `open()`, `read()`, `lseek()`, `close()` operations
- Semantics guaranteed by each operation, e.g.:
  - *successful writes to a file must be immediately visible to subsequent reads*

POSIX was never designed or necessarily intended for the large-scale parallel file access.

- Inflexible, strong consistency requirements often lead PFSes to implement elaborate locking protocols or to eschew strong consistency entirely.

To avoid performance or consistency issues, best practice in the HPC community typically involves avoiding concurrent access of overlapping regions of a file. However, "false sharing" can still lead to performance inefficiencies.

I/O libraries: parallel I/O capabilities

The **MPI-IO** interface was designed to address needs for parallel I/O support by HPC apps.

- Allow MPI programs to read/write data using various parallel I/O strategies (e.g., single shared file)

MPI is actually an attractive environment for providing parallel I/O support:

- Collective operations enabling all processes to participate in some task (e.g., reading/writing)
- MPI datatypes support for describing layout of data in both memory and file

MPI-IO offers numerous I/O capabilities, allowing flexible and performant I/O strategies:

- Independent operations
- Collective operations
- Non-blocking operations
- Optimizations
  - General and system-specific

I/O libraries: parallel I/O capabilities

The **MPI-IO** interface was designed to address needs for parallel I/O support by HPC apps.

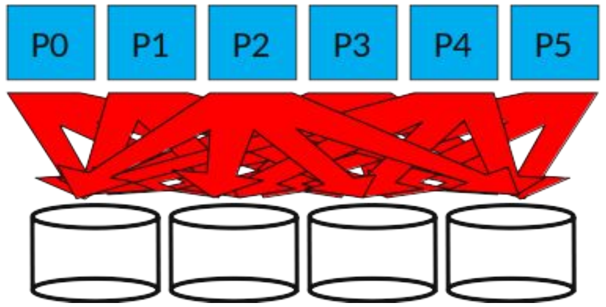
- Allow MPI programs to read/write data using various parallel I/O strategies (e.g., single shared file)

MPI is actually an attractive environment for providing parallel I/O support:

- Collective operations enabling all processes to participate in some task (e.g., reading/writing)
- MPI datatypes support for describing layout of data in both memory and file

MPI-IO offers numerous I/O capabilities, allowing flexible and performant I/O strategies:

- **Independent operations**
- Collective operations
- Non-blocking operations
- Optimizations
  - General and system-specific



I/O libraries: parallel I/O capabilities

The **MPI-IO** interface was designed to address needs for parallel I/O support by HPC apps.

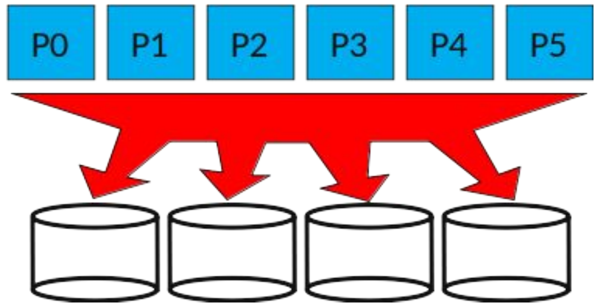
- Allow MPI programs to read/write data using various parallel I/O strategies (e.g., single shared file)

MPI is actually an attractive environment for providing parallel I/O support:

- Collective operations enabling all processes to participate in some task (e.g., reading/writing)
- MPI datatypes support for describing layout of data in both memory and file

MPI-IO offers numerous I/O capabilities, allowing flexible and performant I/O strategies:

- Independent operations
- **Collective operations**
- Non-blocking operations
- Optimizations
  - General and system-specific



I/O libraries: parallel I/O capabilities

The **MPI-IO** interface was designed to address needs for parallel I/O support by HPC apps.

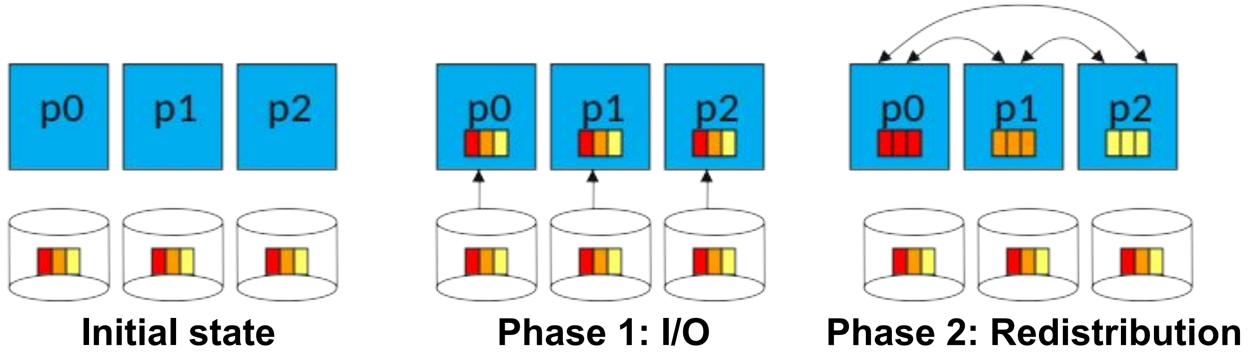
- Allow MPI programs to read/write data using various parallel I/O strategies (e.g., single shared file)

MPI is actually an attractive environment for providing parallel I/O support:

- Collective operations enabling all processes to participate in some task (e.g., reading/writing)
- MPI datatypes support for describing layout of data in both memory and file

MPI-IO offers numerous I/O capabilities, allowing flexible and performant I/O strategies:

- Independent operations
- Collective operations
- Non-blocking operations
- **Optimizations**
  - General and system-specific



Two-phase collective I/O algorithm

I/O libraries: parallel I/O capabilities

The **MPI-IO** interface was designed to address needs for parallel I/O support by HPC apps.

- Allow MPI programs to read/write data using various parallel I/O strategies (e.g., single shared file)

MPI is actually an attractive environment for providing parallel I/O support:

- Collective operations enabling all processes to participate in some task (e.g., reading/writing)
- MPI datatypes support for describing layout of data in both memory and file

MPI-IO offers numerous I/O capabilities, allowing flexible and performant I/O strategies:

- Independent operations
- Collective operations
- Non-blocking operations
- Optimizations
  - General and system-specific

More on MPI-IO in tomorrow's session (Rob Latham, ANL)

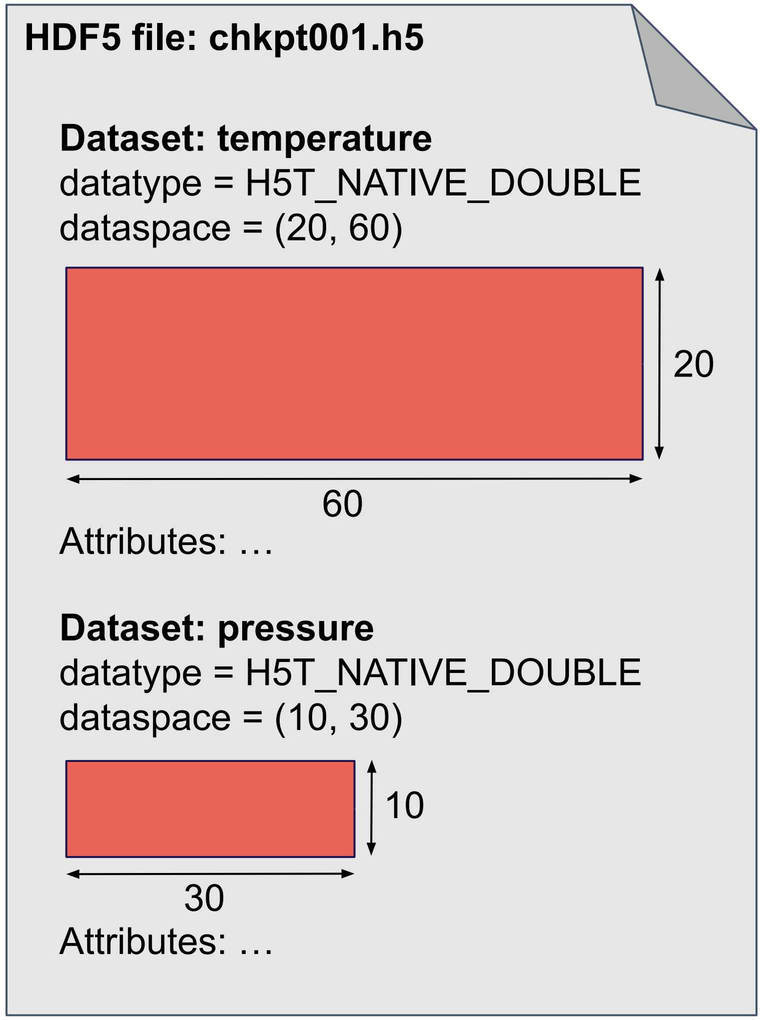
I/O libraries: scientific data management abstractions

MPI-IO is a step in the right direction, but application scientists often prefer richer data management abstractions than simple files.

- Storing independent data products in unique files or manually serializing collections of data products into a single file is often untenable.

HDF5 is a popular data management library and file format that specializes in storing large amounts of scientific data.

- Enables storage of multi-dimensional datasets, attributes, etc. in an HDF5 file (more like a "container")
- Interfaces allow for access of individual dataset elements, subarrays, or entire datasets
- Support for collective I/O (using MPI-IO) or independent I/O (using MPI-IO or POSIX)
- VOL layer allows abstract implementation of storage for HDF5 objects
  - e.g., using async operations, using log-structured storage, using an object store rather than file system



I/O libraries: scientific data management abstractions

MPI-IO is a step in the right direction, but application scientists often prefer richer data management abstractions than simple files.

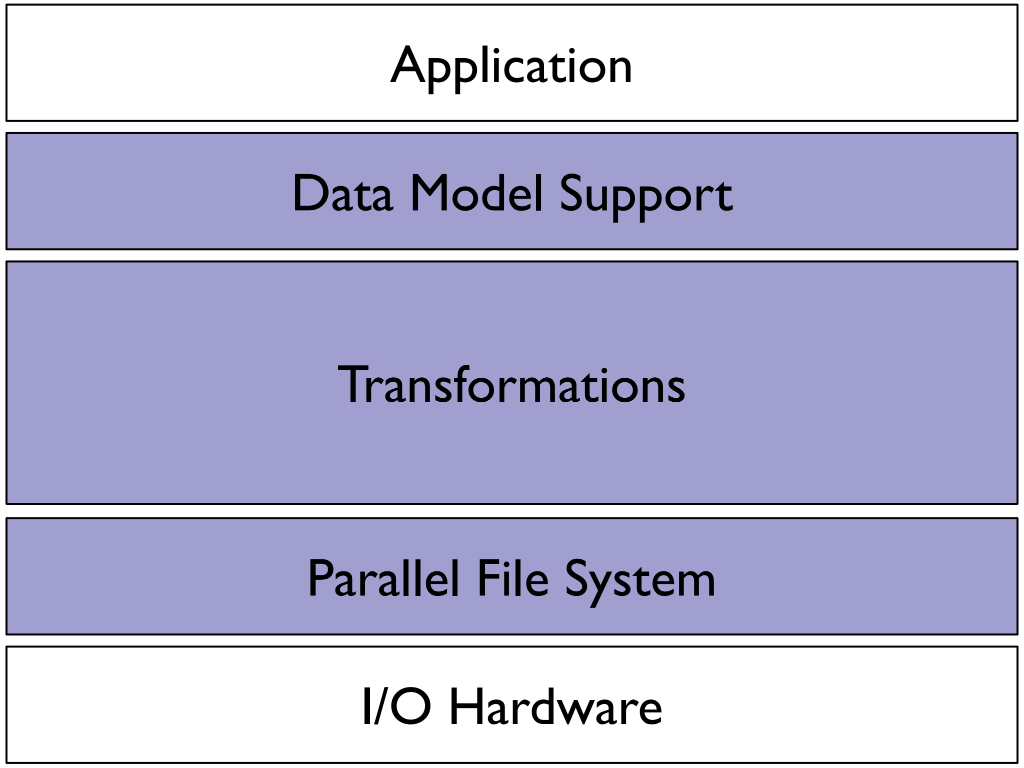
- Storing independent data products in unique files or manually serializing collections of data products into a single file is often untenable.

HDF5 is a popular data management library and file format that specializes in storing large amounts of scientific data.

- Enables storage of multi-dimensional datasets, attributes, etc. in an HDF5 file (more like a “container”)
- Interfaces allow for access of individual dataset elements, subarrays, or entire datasets
- Support for collective I/O (using MPI-IO) or independent I/O (using MPI-IO or POSIX)
- VOL layer allows abstract implementation of storage for HDF5 objects
  - e.g., using async operations, using log-structured storage, using an object store rather than file system

More on HDF5 in tomorrow’s session (Rob Latham, ANL)

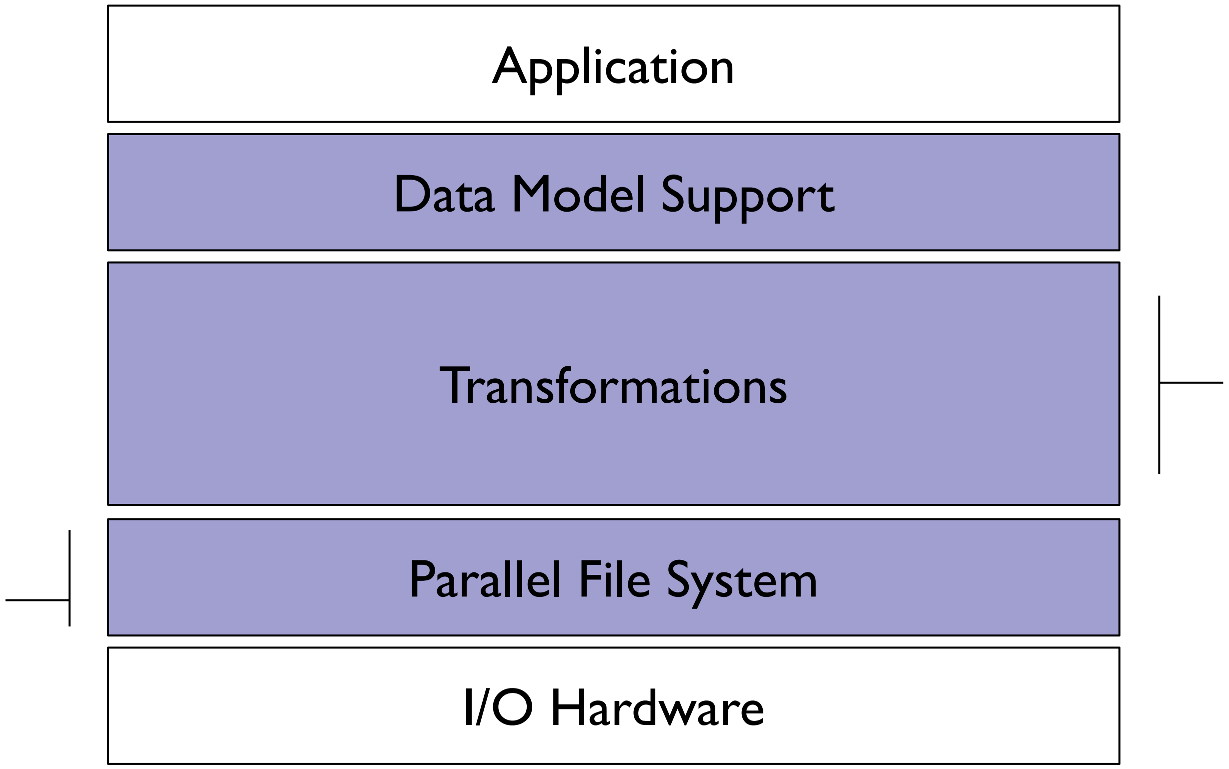
Putting it all together: the HPC I/O stack



Parallel file system maintains logical file model and provides efficient access to data using a POSIX-like interface.

Lustre, GPFS

Putting it all together: the HPC I/O stack



Parallel file system maintains logical file model and provides efficient access to data using a POSIX-like interface.

Lustre, GPFS

I/O Middleware organizes and transforms accesses from many processes, especially those using collective I/O.

MPI-IO

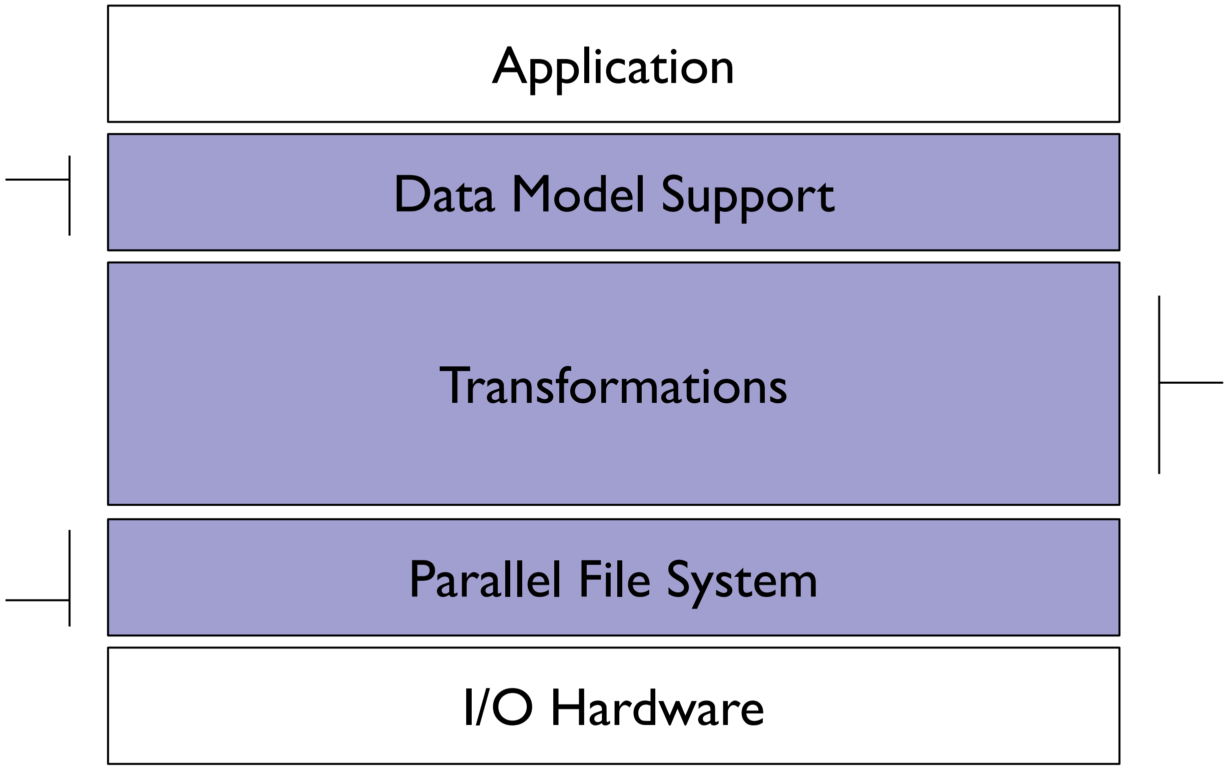
Putting it all together: the HPC I/O stack

Data Model Libraries map application abstractions onto storage abstractions and provide data portability.

HDF5, Parallel netCDF, ADIOS

Parallel file system maintains logical file model and provides efficient access to data using a POSIX-like interface.

Lustre, GPFS



I/O Middleware organizes and transforms accesses from many processes, especially those using collective I/O.

MPI-IO

ALCF Community Data Co-Op (ACDC)

ACDC enables data-driven research by providing a platform for data access and sharing, and services for data discovery and analysis.

- Share data with collaborators on Eagle directly without the need for creating new ALCF accounts
- Project-specific data portals that enable search and discovery of data hosted on Eagle
- Based on the Django Globus Portal Framework

More info here: https://acdc.alcf.anl.gov/

ALCF Community Data Co-Op (ACDC)

ACDC enables data-driven research by providing a platform for data access and sharing, and services for data discovery and analysis.

- Share data with collaborators on Eagle directly without the need for creating new ALCF accounts
- Project-specific data portals that enable search and discovery of data hosted on Eagle
- Based on the Django Globus Portal Framework

More info here: https://acdc.alcf.anl.gov/

More on Globus in the next session (Greg Nawrocki, Globus)

Tools: analyzing application I/O behavior

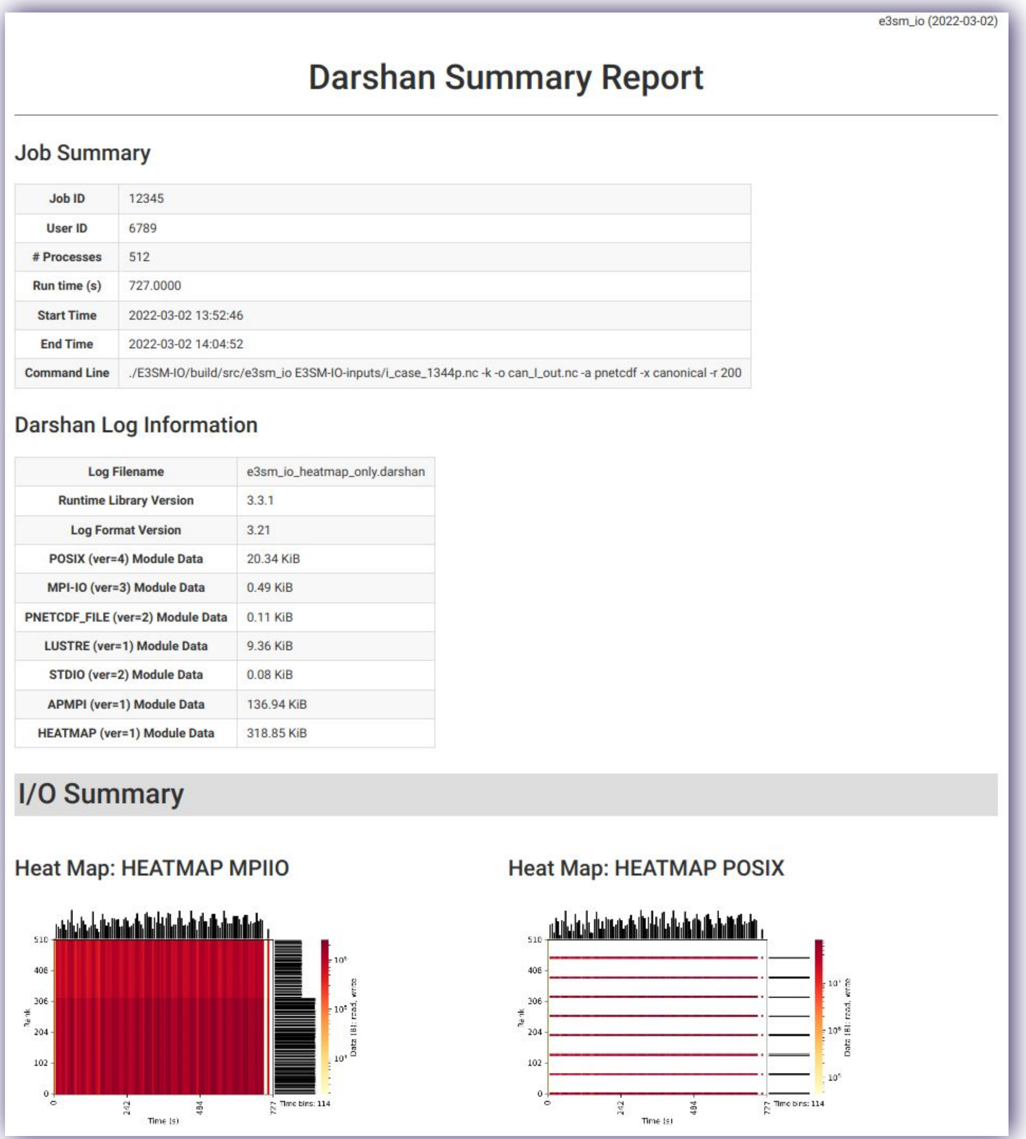
Application-level analysis tools are critical to *better understanding I/O behavior* and *informing potential tuning decisions.*

Darshan is a lightweight I/O characterization tool commonly deployed at HPC facilities, including ALCF systems.

- Transparent, low-overhead instrumentation of multiple layers of the HPC I/O stack
- Detailed counters/timers/statistics for each file accessed by the app stored in a condensed log
- Analysis tools for inspecting and presenting key information about I/O behavior (e.g., the Darshan job summary tool, **right**)

https://www.mcs.anl.gov/research/projects/darshan/

https://github.com/darshan-hpc/darshan


e3sm_io (2022-03-02)

Darshan Summary Report

Job Summary

Job ID	12345
User ID	6789
# Processes	512
Run time (s)	727.0000
Start Time	2022-03-02 13:52:46
End Time	2022-03-02 14:04:52
Command Line	./E3SM-IO/build/src/e3sm_io E3SM-IO-inputs/i_case_1344p.nc -k -o can_I_out.nc -a pnetcdf -x canonical -r 200

Darshan Log Information

Log Filename	e3sm_io_heatmap_only.darshan
Runtime Library Version	3.3.1
Log Format Version	3.21
POSIX (ver=4) Module Data	20.34 KiB
MPI-IO (ver=3) Module Data	0.49 KiB
PNETCDF_FILE (ver=2) Module Data	0.11 KiB
LUSTRE (ver=1) Module Data	9.36 KiB
STDIO (ver=2) Module Data	0.08 KiB
APMPI (ver=1) Module Data	136.94 KiB
HEATMAP (ver=1) Module Data	318.85 KiB

I/O Summary

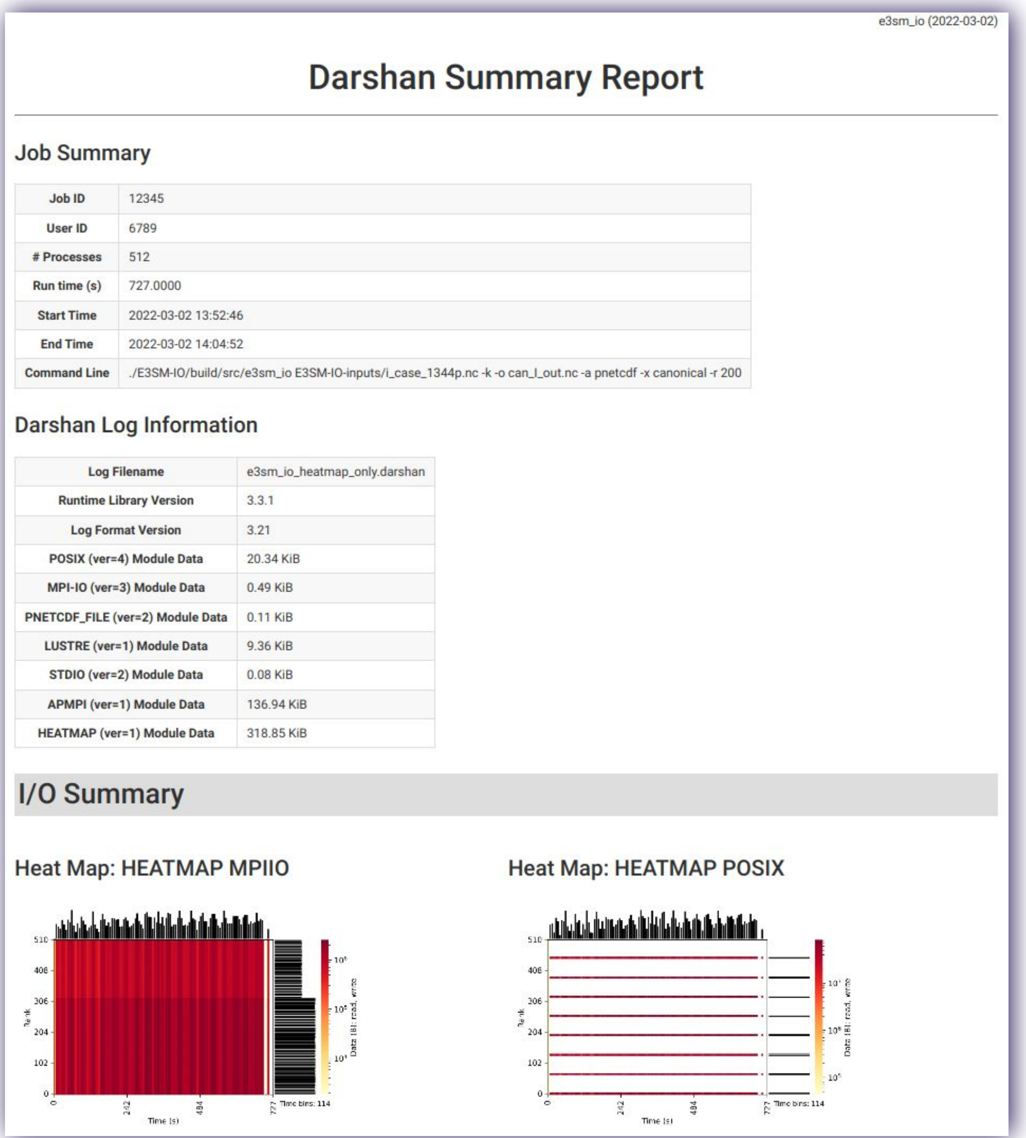
Heat Map: HEATMAP MPIIO

Heat Map: HEATMAP POSIX

Tools: Darshan job summary

The Darshan job summary tool can be a useful starting point for application I/O performance analysis.

It generates an HTML report providing key details on I/O performance and access characteristics that can be used to better understand the application's I/O behavior.



e3sm_io (2022-03-02)

Darshan Summary Report

Job Summary

Job ID	12345
User ID	6789
# Processes	512
Run time (s)	727.0000
Start Time	2022-03-02 13:52:46
End Time	2022-03-02 14:04:52
Command Line	./E3SM-IO/build/src/e3sm_io E3SM-IO-inputs/i_case_1344p.nc -k -o can_I_out.nc -a pnetcdf -x canonical -r 200

Darshan Log Information

Log Filename	e3sm_io_heatmap_only.darshan
Runtime Library Version	3.3.1
Log Format Version	3.21
POSIX (ver=4) Module Data	20.34 KiB
MPI-IO (ver=3) Module Data	0.49 KiB
PNETCDF_FILE (ver=2) Module Data	0.11 KiB
LUSTRE (ver=1) Module Data	9.36 KiB
STDIO (ver=2) Module Data	0.08 KiB
APMPI (ver=1) Module Data	136.94 KiB
HEATMAP (ver=1) Module Data	318.85 KiB

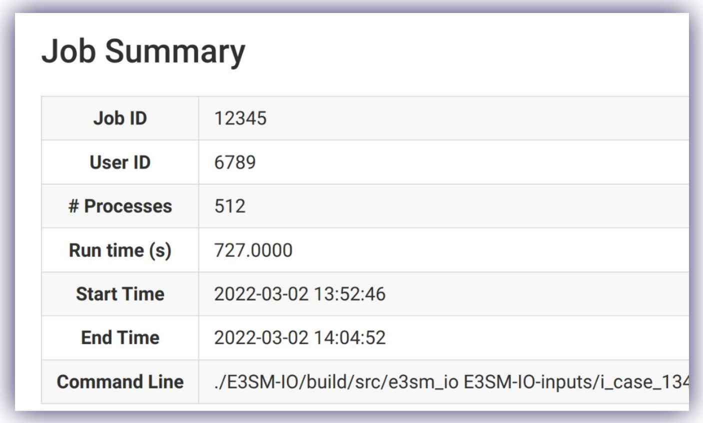
I/O Summary

Heat Map: HEATMAP MPIIO

Heat Map: HEATMAP POSIX

Tools: Darshan job summary

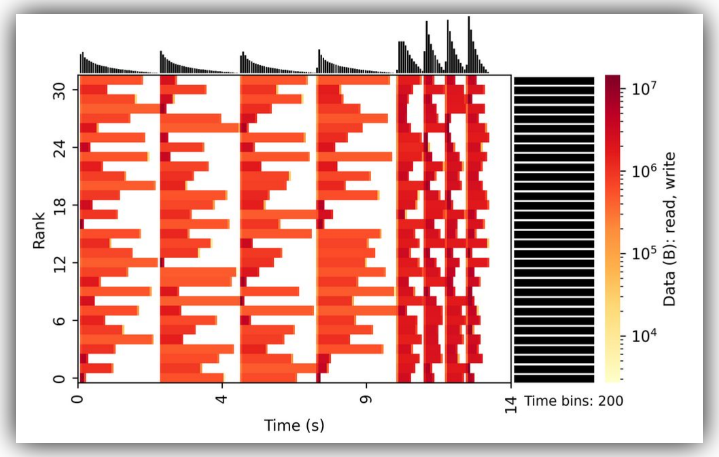
Detailed job metadata

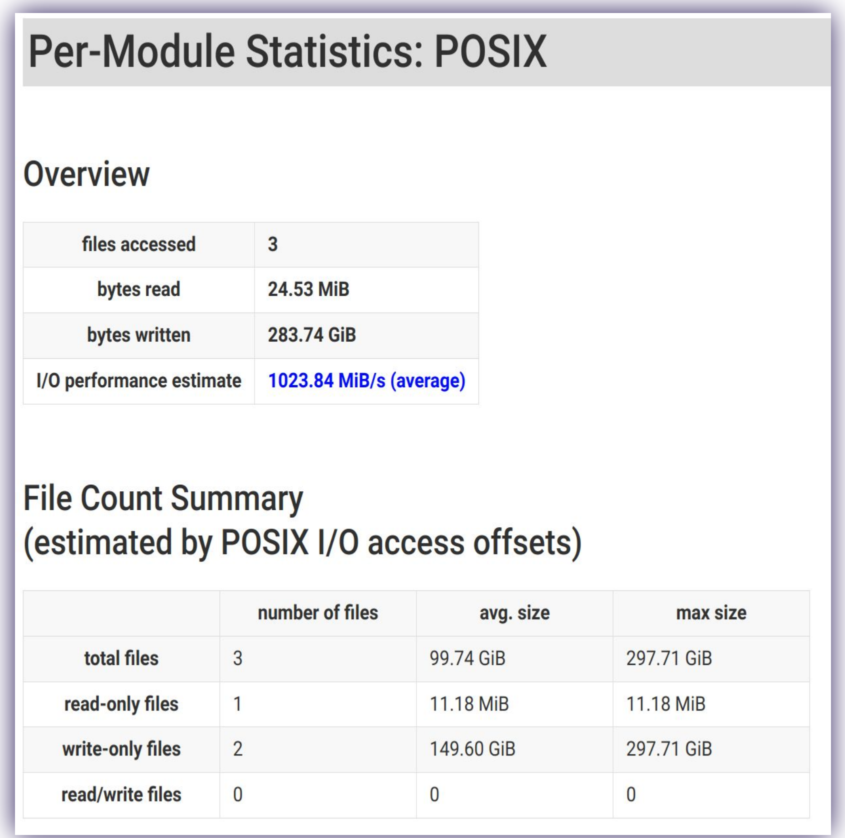


Job Summary

Job ID	12345
User ID	6789
# Processes	512
Run time (s)	727.0000
Start Time	2022-03-02 13:52:46
End Time	2022-03-02 14:04:52
Command Line	./E3SM-IO/build/src/e3sm_io E3SM-IO-inputs/i_case_134

Heatmaps of I/O activity





Per-Module Statistics: POSIX

Overview

files accessed	3
bytes read	24.53 MiB
bytes written	283.74 GiB
I/O performance estimate	1023.84 MiB/s (average)

File Count Summary
(estimated by POSIX I/O access offsets)

	number of files	avg. size	max size
total files	3	99.74 GiB	297.71 GiB
read-only files	1	11.18 MiB	11.18 MiB
write-only files	2	149.60 GiB	297.71 GiB
read/write files	0	0	0

Comprehensive I/O statistics for multiple layers of the I/O stack

A recap

Today we have covered various software technologies that comprise the HPC I/O stack, which is used to manage large-scale scientific datasets.

- *Parallel file systems* offer high-performance, scalable file storage.
- *I/O libraries* provide interfaces for managing data at different abstraction layers.
  - POSIX provides a portable, performant low-level file system interface
  - MPI-IO introduces capabilities for parallel access of files
  - HDF5 provides a data management interface more closely aligned with app data abstractions
- *Tools* like Darshan are available for better understanding and, ideally, improving I/O performance.

Always consider facility documentation and other resources to help understand general best practice and reach out on support channels for help if needed!

Thank you!