Understanding and tuning HPC I/O performance

ATPESC 2020

Shane Snyder
Mathematics and Computer Science Division
Argonne National Laboratory

July 31, 2020

Hands on exercises:
https://xgitlab.cels.anl.gov/ATPESC-IO/hands-on

Surveying the HPC I/O landscape

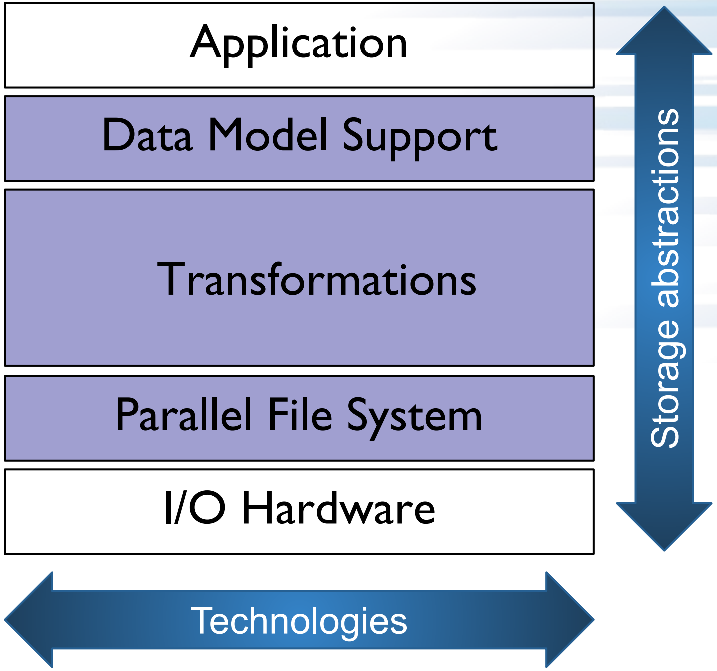
A complex data management ecosystem

As evidenced by today's presentations, the HPC I/O landscape is deep and vast

- High-level data abstractions: HDF5, PnetCDF
- Parallel file systems: Lustre, GPFS
- Storage hardware: HDDs, SSDs, NVM

Application developers tend to prefer high-level data models for convenience, but these APIs often obfuscate the behavior of lower level interfaces that drive I/O performance

Understanding I/O behavior in this environment is difficult, much less turning observations into actionable I/O tuning decisions



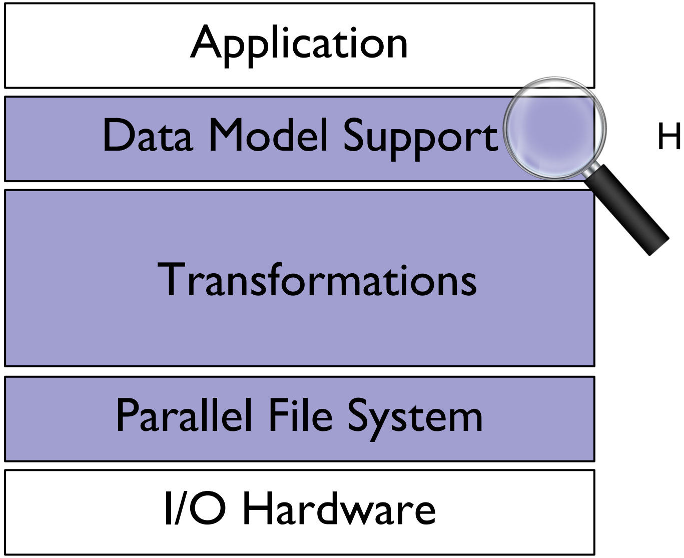
Hands on exercises:
https://xgitlab.cels.anl.gov/ATPESC-IO/hands-on

Characterizing HPC I/O workloads with Darshan

A look under the hood of an HPC application

You have already heard some basics about Darshan, a powerful tool for users to better understand and tune their I/O workloads

Darshan provides many helpful stats across multiple layers of the I/O stack that are critical to understanding application I/O behavior



HDF5 file stats*:

- Metadata operation counts (open, flush)
- MPI-IO usage
- Metadata timing

*Note: Detailed HDF5 instrumentation can be optionally enabled only for Darshan versions 3.2.0+

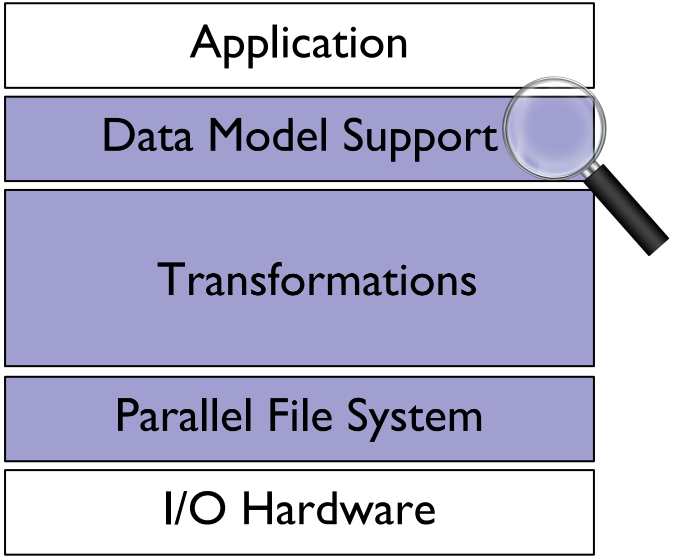
Hands on exercises:
https://xgitlab.cels.anl.gov/ATPESC-IO/hands-on

Characterizing HPC I/O workloads with Darshan

A look under the hood of an HPC application

You have already heard some basics about Darshan, a powerful tool for users to better understand and tune their I/O workloads

Darshan provides many helpful stats across multiple layers of the I/O stack that are critical to understanding application I/O behavior



HDF5 dataset stats*:

- Data operation counts (read, write)
- Metadata operation counts (open, flush)
- Total I/O volumes (read, write)
- Common access info (size, hyperslab parameters)
- Chunking parameters
- Dataspace total dimensions, points
- MPI-IO collective usage
- Data & metadata timing

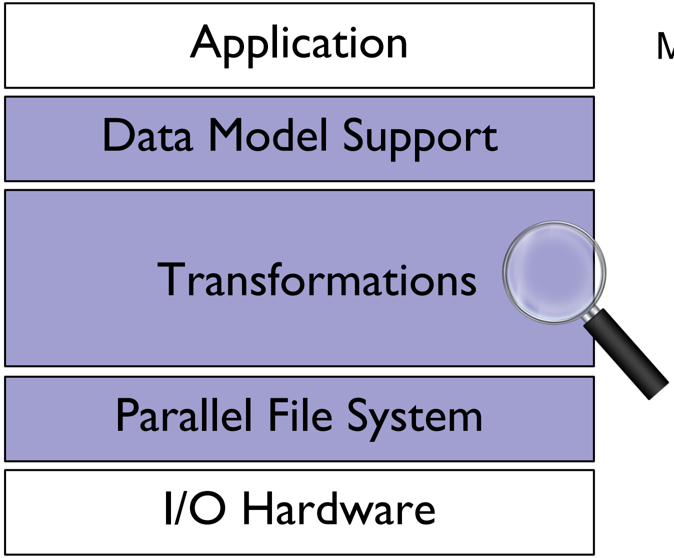
*Note: Detailed HDF5 instrumentation can be optionally enabled only for Darshan versions 3.2.0+

Characterizing HPC I/O workloads with Darshan

A look under the hood of an HPC application

You have already heard some basics about Darshan, a powerful tool for users to better understand and tune their I/O workloads

Darshan provides many helpful stats across multiple layers of the I/O stack that are critical to understanding application I/O behavior



MPI-IO file stats:

- Data operation counts (read, write, sync)
- Metadata operation counts (open)
- Collective and independent
- Total I/O volumes (read, write)
- Access size info
    - Common values
    - Histograms
- Data & metadata timing

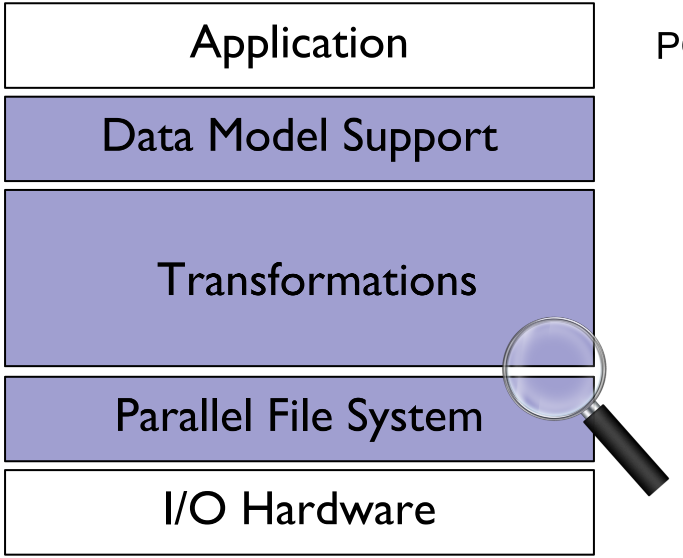
Hands on exercises:
https://xgitlab.cels.anl.gov/ATPESC-IO/hands-on

Characterizing HPC I/O workloads with Darshan

A look under the hood of an HPC application

You have already heard some basics about Darshan, a powerful tool for users to better understand and tune their I/O workloads

Darshan provides many helpful stats across multiple layers of the I/O stack that are critical to understanding application I/O behavior



POSIX file stats:

- Data operation counts (read, write, sync)
- Metadata operation counts (open, seek, stat)
- Total I/O volumes (read, write)
- File alignment
- Access size/stride info
  - Common values
  - Histograms
- Data & metadata timing

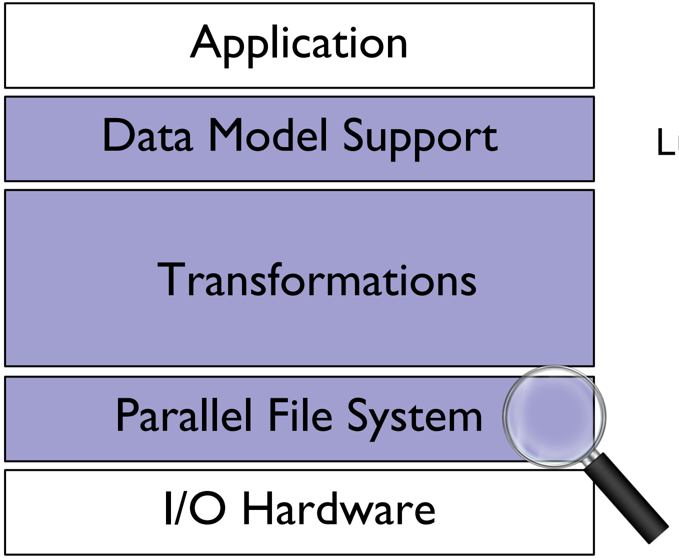
Hands on exercises:
https://xgitlab.cels.anl.gov/ATPESC-IO/hands-on

Characterizing HPC I/O workloads with Darshan

A look under the hood of an HPC application

You have already heard some basics about Darshan, a powerful tool for users to better understand and tune their I/O workloads

Darshan provides many helpful stats across multiple layers of the I/O stack that are critical to understanding application I/O behavior



Lustre file stats:

- Data server (OST) and metadata server (MDT) counts
- Stripe size/width
- OST list serving a file

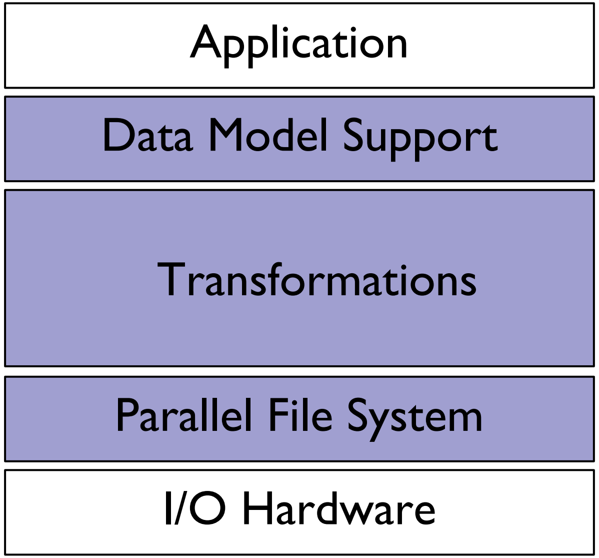
Hands on exercises:
https://xgitlab.cels.anl.gov/ATPESC-IO/hands-on

Characterizing HPC I/O workloads with Darshan

A look under the hood of an HPC application

You have already heard some basics about Darshan, a powerful tool for users to better understand and tune their I/O workloads

Darshan provides many helpful stats across multiple layers of the I/O stack that are critical to understanding application I/O behavior



Let's see how Darshan can be leveraged in some practical use cases that demonstrate some widely held best practices in tuning HPC I/O performance

Tuning the parallel file system

Ensuring storage resources match application I/O needs

For some parallel file systems like Lustre, users have direct control over file striping parameters

Bad news: Users may have to have some knowledge of the file system to get good I/O performance

Good news: Users can often get higher I/O performance than system defaults with thoughtful tuning -- file systems aren’t perfect for every workload!

Hands on exercises:
https://xgitlab.cels.anl.gov/ATPESC-IO/hands-on

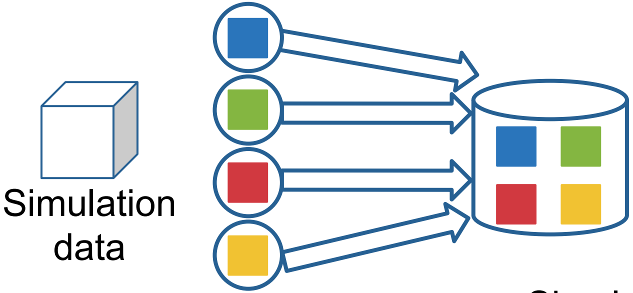
Tuning the parallel file system

Ensuring storage resources match application I/O needs

For some parallel file systems like Lustre, users have direct control over file striping parameters

Bad news: Users may have to have some knowledge of the file system to get good I/O performance

Good news: Users can often get higher I/O performance than system defaults with thoughtful tuning -- file systems aren't perfect for every workload!



Simulation clients write data to 1 storage server

Hands on exercises:
https://xgitlab.cels.anl.gov/ATPESC-IO/hands-on

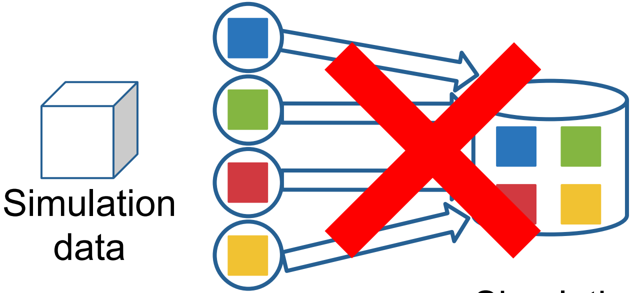
Tuning the parallel file system

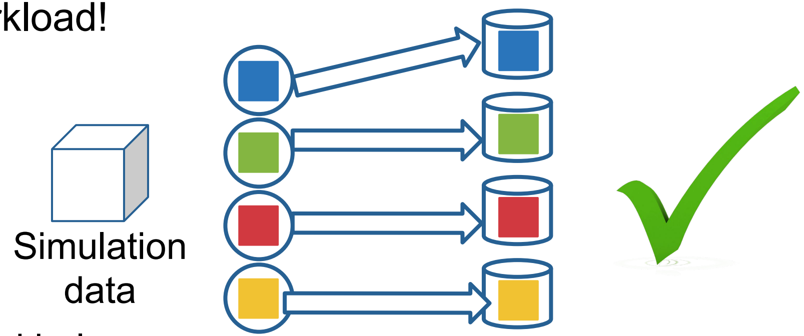
Ensuring storage resources match application I/O needs

For some parallel file systems like Lustre, users have direct control over file striping parameters

Bad news: Users may have to have some knowledge of the file system to get good I/O performance

Good news: Users can often get higher I/O performance than system defaults with thoughtful tuning -- file systems aren't perfect for every workload!





Simulation clients load balance writes across multiple servers

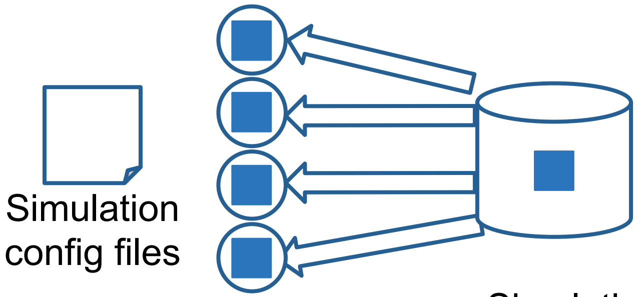
Hands on exercises:
https://xgitlab.cels.anl.gov/ATPESC-IO/hands-on

Tuning the parallel file system

Ensuring storage resources match application I/O needs

Tuning decisions can and should be made independently for different file types

While large application datasets should ideally be distributed across as many storage resources as possible, smaller files tend to benefit from being contained to a single server



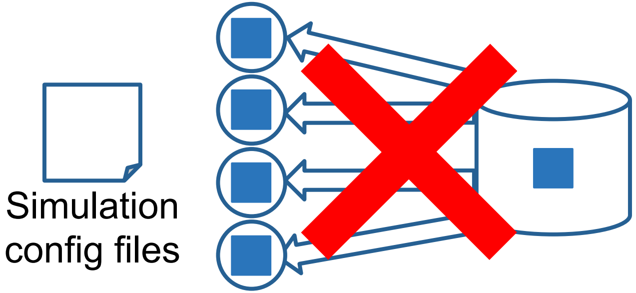
Simulation clients read config data from 1 storage server

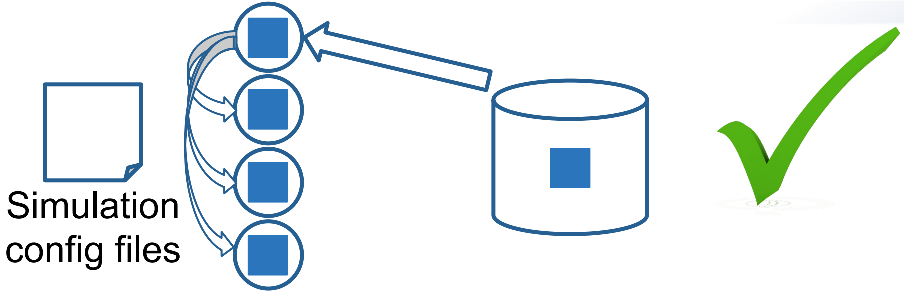
Tuning the parallel file system

Ensuring storage resources match application I/O needs

Tuning decisions can and should be made independently for different file types

While large application datasets should ideally be distributed across as many storage resources as possible, smaller files tend to benefit from being contained to a single server





Better yet, limit storage contention by having 1 client read data and distribute using communication (e.g., MPI)

Hands on exercises:
https://xgitlab.cels.anl.gov/ATPESC-IO/hands-on

Tuning the parallel file system

Ensuring storage resources match application I/O needs

Be aware of what file system settings are available to you and don't assume system defaults are always the best… you might be surprised what you find

- NERSC's Cori default Lustre stripe width is 1

Darshan output from a simple 10-process (10-node) POSIX I/O workload to shared file on a Cori's Lustre scratch volume:

jobid: 32840482	uid: 69628	nprocs: 10	runtime: 6 seconds

I/O performance *estimate* (at the POSIX layer): transferred 1000.0 MiB at 210.38 MiB/s

```
LUSTRE_STRIPE_SIZE  1048576 /global/cscratc
LUSTRE_STRIPE_WIDTH 1    /global/cscratch1/s
LUSTRE_OST_ID_0 100 /global/cscratch1/sd/ss
```

Hands on exercises:
https://xgitlab.cels.anl.gov/ATPESC-IO/hands-on

Tuning the parallel file system

Ensuring storage resources match application I/O needs

jobid: 32840482	uid: 69628	nprocs: 10	runtime: 6 seconds

I/O performance *estimate* (at the POSIX layer): transferred 1000.0 MiB at 210.38 MiB/s

```
> lfs setstripe -c 10 testFile # change stripe width to 10
```

jobid: 32840482	uid: 69628	nprocs: 10	runtime: 3 seconds

I/O performance *estimate* (at the POSIX layer): transferred 1000.0 MiB at 562.48 MiB/s

```
LUSTRE_STRIPE_SIZE  1048576 /global/cscrat
LUSTRE_STRIPE_WIDTH 10  /global/cscratch1/
LUSTRE_OST_ID_0 220 /global/cscratch1/sd/s
LUSTRE_OST_ID_1 146 /global/cscratch1/sd/s
LUSTRE_OST_ID_2 107 /global/cscratch1/sd/s
LUSTRE_OST_ID_3 181 /global/cscratch1/sd/s
LUSTRE_OST_ID_4 47  /global/cscratch1/sd/s
LUSTRE_OST_ID_5 209 /global/cscratch1/sd/s
LUSTRE_OST_ID_6 244 /global/cscratch1/sd/s
LUSTRE_OST_ID_7 112 /global/cscratch1/sd/s
LUSTRE_OST_ID_8 36  /global/cscratch1/sd/s
LUSTRE_OST_ID_9 154 /global/cscratch1/sd/s
```

~200% performance boost

Tuning low-level (POSIX) file I/O

Making efficient use of a no-frills I/O API

Users may also need to pay close attention to file system alignment when crafting I/O accesses to a file

- Accesses that cross alignment boundaries likely perform worse than nicely aligned I/O

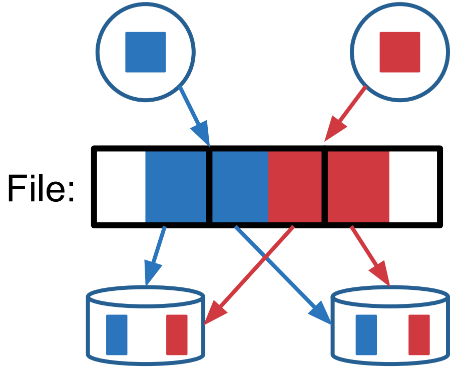
Tuning low-level (POSIX) file I/O

Making efficient use of a no-frills I/O API

Users may also need to pay close attention to file system alignment when crafting I/O accesses to a file

- Accesses that cross alignment boundaries likely perform worse than nicely aligned I/O

For Lustre, performance can be maximized by aligning I/O to stripe boundaries:



Unaligned I/O requests can span multiple servers and introduce inefficiencies in storage protocols

Hands on exercises:
https://xgitlab.cels.anl.gov/ATPESC-IO/hands-on

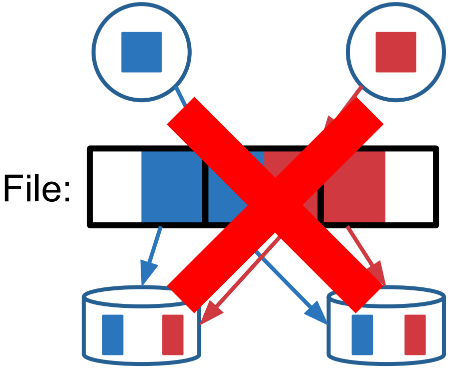
Tuning low-level (POSIX) file I/O

Making efficient use of a no-frills I/O API

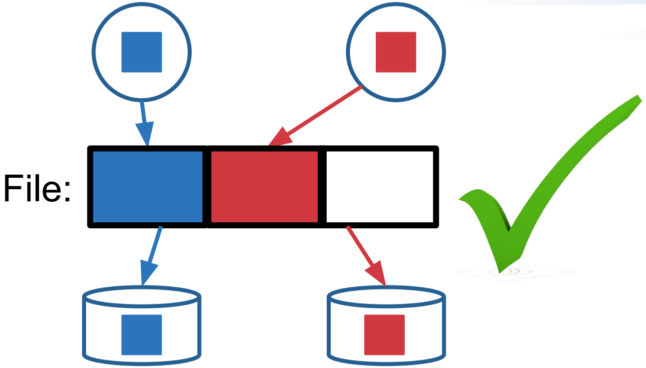
Users may also need to pay close attention to file system alignment when crafting I/O accesses to a file

- Accesses that cross alignment boundaries likely perform worse than nicely aligned I/O

For Lustre, performance can be maximized by aligning I/O to stripe boundaries:



Instead, ensure client accesses are well-aligned to avoid Lustre server contention



Hands on exercises:
https://xgitlab.cels.anl.gov/ATPESC-IO/hands-on

Tuning low-level (POSIX) file I/O

Making efficient use of a no-frills I/O API

Repeating our simple 10-client example striping a single file across 10 Lustre OSTs

Unaligned: transferred 1000.0 MiB at 310.14 MiB/s

# Module	Rank	Wt/Rd	Segment	Offset	Length	Start(s)	End(s)	[OST]	
X_POSIX	0	write	0	524288	1048576	0.0065	0.0594	[32]	[197]
X_POSIX	1	write	0	1572864	1048576	0.0065	0.0538	[197]	[237]
X_POSIX	2	write	0	2621440	1048576	0.0070	0.0440	[237]	[26]
X_POSIX	3	write	0	3670016	1048576	0.0067	0.0485	[26]	[213]

Hands on exercises:
https://xgitlab.cels.anl.gov/ATPESC-IO/hands-on

Tuning low-level (POSIX) file I/O

Making efficient use of a no-frills I/O API

Repeating our simple 10-client example striping a single file across 10 Lustre OSTs

Unaligned: transferred 1000.0 MiB at 310.14 MiB/s

# Module	Rank	Wt/Rd	Segment	Offset	Length	Start(s)	End(s)	[OST]	
X_POSIX	0	write	0	524288	1048576	0.0065	0.0594	[32]	[197]
X_POSIX	1	write	0	1572864	1048576	0.0065	0.0538	[197]	[237]
X_POSIX	2	write	0	2621440	1048576	0.0070	0.0440	[237]	[26]
X_POSIX	3	write	0	3670016	1048576	0.0067	0.0485	[26]	[213]

Aligned: transferred 1000.0 MiB at 380.28 MiB/s

# Module	Rank	Wt/Rd	Segment	Offset	Length	Start(s)	End(s)	[OST]
X_POSIX	0	write	0	0	1048576	0.0054	0.0066	[197]
X_POSIX	1	write	0	1048576	1048576	0.0053	0.0064	[102]
X_POSIX	2	write	0	2097152	1048576	0.0061	0.0072	[106]
X_POSIX	3	write	0	3145728	1048576	0.0053	0.0064	[120]

Hands on exercises:
https://xgitlab.cels.anl.gov/ATPESC-IO/hands-on

Tuning low-level (POSIX) file I/O

Making efficient use of a no-frills I/O API

Even in this small workload, we pay a nearly 20% performance penalty when I/O accesses are not aligned to file stripes (1 MB)

Unaligned: transferred 1000.0 MiB at 310.14 MiB/s

# Module	Rank	Wt/Rd	Segment	Offset	Length	Start(s)	End(s)	[OST]	
X_POSIX	0	write	0	524288	1048576	0.0065	0.0594	[32]	[197]
X_POSIX	1	write	0	1572864	1048576	0.0065	0.0538	[197]	[237]
X_POSIX	2	write	0	2621440	1048576	0.0070	0.0440	[237]	[26]
X_POSIX	3	write	0	3670016	1048576	0.0067	0.0485	[26]	[213]

Aligned: transferred 1000.0 MiB at 380.28 MiB/s

# Module	Rank	Wt/Rd	Segment	Offset	Length	Start(s)	End(s)	[OST]
X_POSIX	0	write	0	0	1048576	0.0054	0.0066	[197]
X_POSIX	1	write	0	1048576	1048576	0.0053	0.0064	[102]
X_POSIX	2	write	0	2097152	1048576	0.0061	0.0072	[106]
X_POSIX	3	write	0	3145728	1048576	0.0053	0.0064	[120]

Hands on exercises:
https://xgitlab.cels.anl.gov/ATPESC-IO/hands-on

Tuning high-level (HDF5) data access

Optimizing application interactions with the I/O stack

Recall that HDF5 provides a chunking mechanism to partition user datasets into contiguous chunks in the underlying file

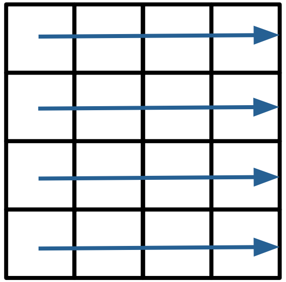
- Users can greatly improve performance of partial dataset I/O operations by choosing chunking parameters that match expected access patterns

Tuning high-level (HDF5) data access

Optimizing application interactions with the I/O stack

Recall that HDF5 provides a chunking mechanism to partition user datasets into contiguous chunks in the underlying file

- Users can greatly improve performance of partial dataset I/O operations by choosing chunking parameters that match expected access patterns



By default, HDF5 will store the dataset contiguously row-by-row (i.e., row-major format) in the file

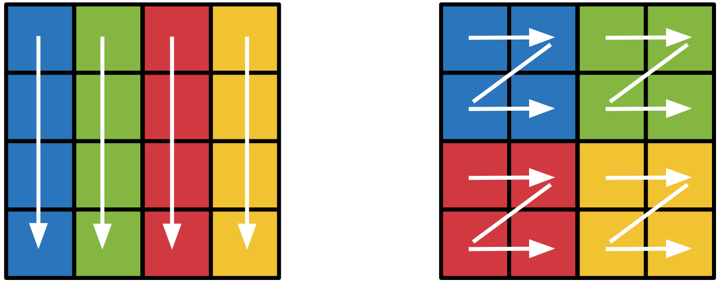
Hands on exercises:
https://xgitlab.cels.anl.gov/ATPESC-IO/hands-on

Tuning high-level (HDF5) data access

Optimizing application interactions with the I/O stack

Recall that HDF5 provides a chunking mechanism to partition user datasets into contiguous chunks in the underlying file

- Users can greatly improve performance of partial dataset I/O operations by choosing chunking parameters that match expected access patterns



column-based block-based

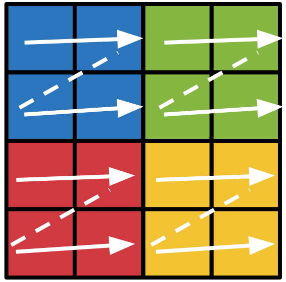
If dataset access patterns do not suit a simple row-major storage scheme, chunking can be applied to map chunks of dataset data to contiguous regions in the file

Tuning high-level (HDF5) data access

Optimizing application interactions with the I/O stack

Consider a 256-process (16-node) example where each process exclusively accesses a block of the dataset

- Each process writes a 2048x2048 block of the dataset (32 MB per-process, 8 GB total)



With no chunking, each process issues many smaller non-contiguous I/O requests to write their block, yielding low I/O performance

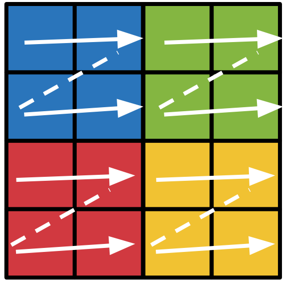
Hands on exercises:
https://xgitlab.cels.anl.gov/ATPESC-IO/hands-on

Tuning high-level (HDF5) data access

Optimizing application interactions with the I/O stack

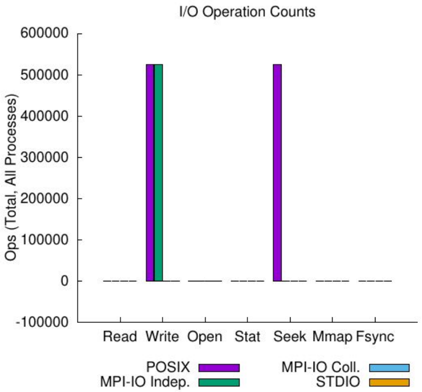
Consider a 256-process (16-node) example where each process exclusively accesses a block of the dataset

- Each process writes a 2048x2048 block of the dataset (32 MB per-process, 8 GB total)



jobid: 32972116	uid: 69628	nprocs: 256	runtime: 143 seconds

I/O performance *estimate* (at the MPI-IO layer): transferred 8192.0 MiB at 57.97 MiB/s



Most Common Access Sizes
(POSIX or MPI-IO)

	access size	count
POSIX	16384	524288
	96	2
	544	1
	328	1

256 individual HDF5 writes (1-per-process) yields 500K+ POSIX writes

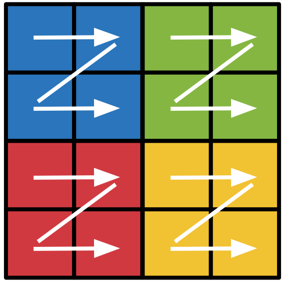
Hands on exercises:
https://xgitlab.cels.anl.gov/ATPESC-IO/hands-on

Tuning high-level (HDF5) data access

Optimizing application interactions with the I/O stack

Consider a 256-process (16-node) example where each process exclusively accesses a block of the dataset

- Each process writes a 2048x2048 block of the dataset (32 MB per-process, 8 GB total)



With chunking applied, each process can read their entire data block using one large, contiguous access in the file

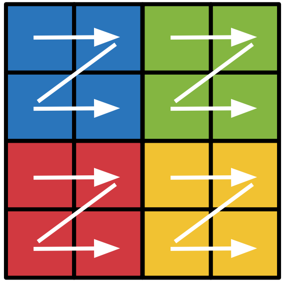
Hands on exercises:
https://xgitlab.cels.anl.gov/ATPESC-IO/hands-on

Tuning high-level (HDF5) data access

Optimizing application interactions with the I/O stack

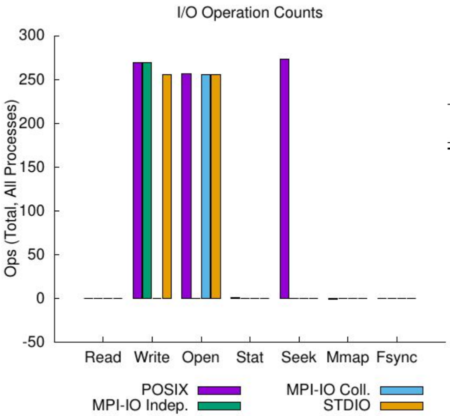
Consider a 256-process (16-node) example where each process exclusively accesses a block of the dataset

- Each process writes a 2048x2048 block of the dataset (32 MB per-process, 8 GB total)



jobid: 32972116	uid: 69628	nprocs: 256	runtime: 52 seconds

I/O performance *estimate* (at the MPI-IO layer): transferred 8192.0 MiB at 164.73 MiB/s



Most Common Access Sizes
(POSIX or MPI-IO)

	access size	count
POSIX	33554432	256
	2616	6
	96	2
	544	1

Appropriate chunking selection yields 2.8x performance increase

Argonne NATIONAL LABORATORY

ECP EXASCALE COMPUTING PROJECT

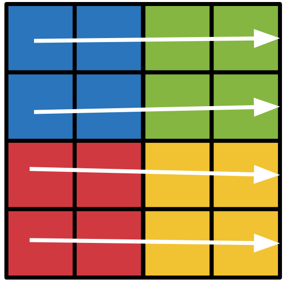
Hands on exercises:
https://xgitlab.cels.anl.gov/ATPESC-IO/hands-on

Tuning high-level (HDF5) data access

Optimizing application interactions with the I/O stack

An alternative optimization relies on collective I/O to improve the efficiency of this block-style data access

- Rely on MPI-IO layer collective buffering algorithm to generate contiguous storage accesses and to limit number of clients interacting with storage system



With collective I/O enabled, designated aggregator processes perform I/O on behalf of their peers, and communicate their data using MPI calls

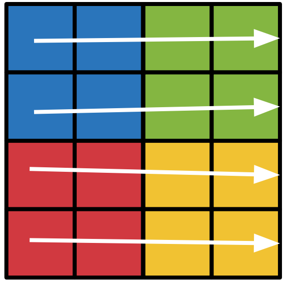
Hands on exercises:
https://xgitlab.cels.anl.gov/ATPESC-IO/hands-on

Tuning high-level (HDF5) data access

Optimizing application interactions with the I/O stack

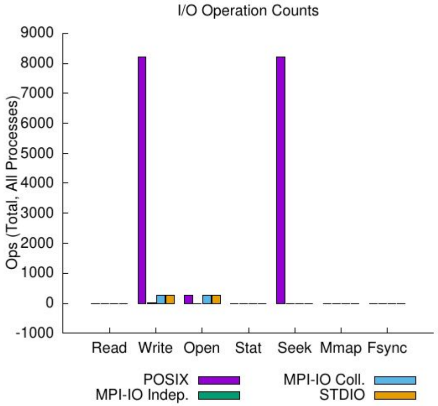
Consider a 256-process (16-node) example where each process exclusively accesses a block of the dataset

- Each process writes a 2048x2048 block of the dataset (32 MB per-process, 8 GB total)



jobid: 32972116	uid: 69628	nprocs: 256	runtime: 32 seconds

I/O performance *estimate* (at the MPI-IO layer): transferred 8192.0 MiB at 268.28 MiB/s



Most Common Access Sizes (POSIX or MPI-IO)

	access size	count
POSIX	1048576	8191
	96	2
	1046528	1
	2048	1

Collective I/O yields 4.6x improvement over no chunking, and 1.6x improvement over chunking

Hands on exercises:
https://xgitlab.cels.anl.gov/ATPESC-IO/hands-on

Summarizing I/O tuning options

As a user of I/O interface X, what tuning vectors do I have?

I/O Interface	Striping	Alignment	Collective I/O	Chunking
HDF5	✓	✓	✓	✓
PnetCDF	✓	✓	✓	X
MPI-IO	✓	✓	✓	X
POSIX	✓	✓-	X	X

Hands on exercises:
https://xgitlab.cels.anl.gov/ATPESC-IO/hands-on

Summarizing I/O tuning options

As a user of I/O interface X, what tuning vectors do I have?

I/O Interface	Striping	Alignment	Collective I/O	Chunking
HDF5	✓	✓	✓	✓
PnetCDF	✓	✓	✓	X
MPI-IO	✓	✓	✓	X
POSIX	✓	✓ -	X	X

Automatically align application data and library metadata, if user requests so

Collective I/O can be automatically aligned

POSIX I/O requires manually aligning every access

Argonne NATIONAL LABORATORY
ECP EXASCALE COMPUTING PROJECT

Hands on exercises:
https://xgitlab.cels.anl.gov/ATPESC-IO/hands-on

Summarizing I/O tuning options

As a user of I/O interface X, what tuning vectors do I have?

I/O Interface	Striping	Alignment	Collective I/O	Chunking
HDF5	✓	✓	✓	✓
PnetCDF	✓	✓	✓	X
MPI-IO	✓	✓	✓	X
POSIX	✓	✓-	X	X

In general, users should try to take advantage of high-level I/O libraries:

- I/O optimization strategies like collective I/O & chunking can net large performance gains, especially when combined with striping and alignment optimizations

Argonne NATIONAL LABORATORY
ECP EXASCALE COMPUTING PROJECT

Accounting for a changing HPC landscape

Adapting to technological shifts

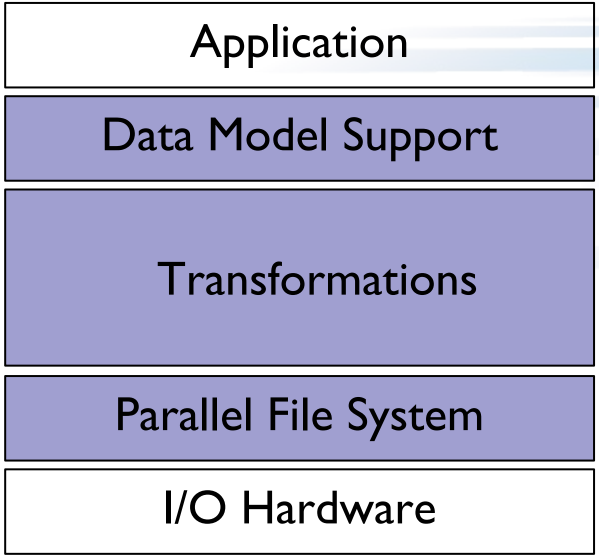
The various technologies covered today form much of the foundation of the traditional HPC data management stack

- Variations on this stack have been deployed at HPC facilities and leveraged by users for high-performance parallel I/O for decades

But, the HPC computing landscape is changing, even if slowly

Changes driven at both ends of the stack

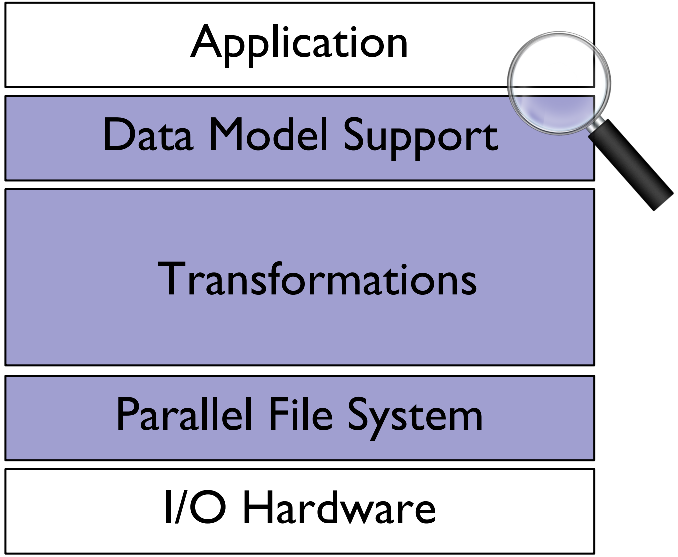
- Newly embraced compute paradigms
- Emerging storage technologies



Hands on exercises:
https://xgitlab.cels.anl.gov/ATPESC-IO/hands-on

Accounting for a changing HPC landscape

Adapting to technological shifts



Large-scale MPI applications are still the norm at most most HPC centers, but other non-MPI compute frameworks are gaining traction:

- Deep learning (TensorFlow, Keras, PyTorch)
- Data analytics frameworks (Spark, Dask)
- Other non-MPI distributed computing frameworks (Legion, UPC)

Many of these frameworks define their own data models and have their own mechanisms for managing distributed tasks

Hands on exercises:
https://xgitlab.cels.anl.gov/ATPESC-IO/hands-on

Instrumenting non-MPI applications with Darshan

Starting with Darshan version 3.2.0, Darshan supports instrumentation of non-MPI applications*

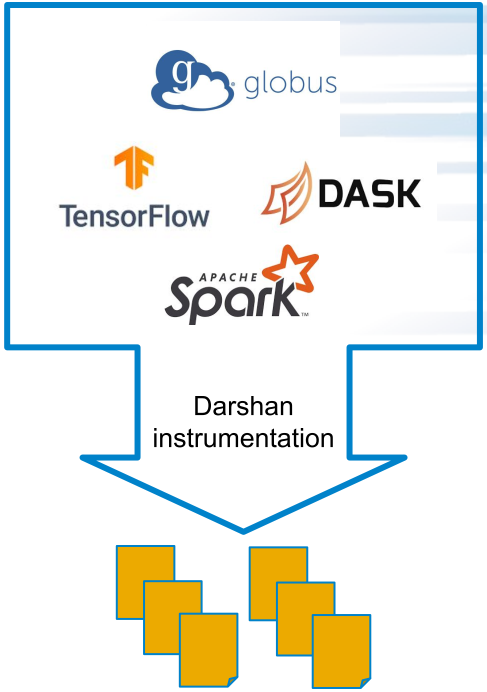
- Just set DARSHAN_ENABLE_NONMPI environment variable before running

Generates unique Darshan log for every process invoked

Extend Darshan instrumentation from traditional MPI applications to any type of executable

- Python frameworks
- File transfer utilities
- Data service daemons
- Other serial applications

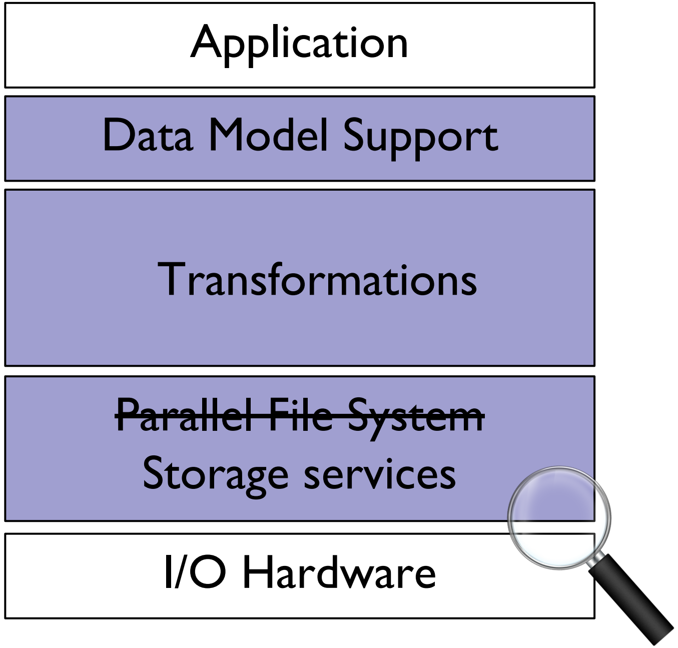
*1 caveat: applications must be dynamically-linked



Hands on exercises:
https://xgitlab.cels.anl.gov/ATPESC-IO/hands-on

Accounting for a changing HPC landscape

Adapting to technological shifts



HPC storage technology is changing to meet needs of diverse application workloads

- Users typically have more options than a traditional parallel file system over HDDs

Hardware trends enabling low-latency, high-bandwidth I/O to applications

- Burst buffers, NVM

Novel storage services offer compelling alternatives to traditional file systems

- Unify, DAOS

Hands on exercises:
https://xgitlab.cels.anl.gov/ATPESC-IO/hands-on

Understanding I/O beyond the application

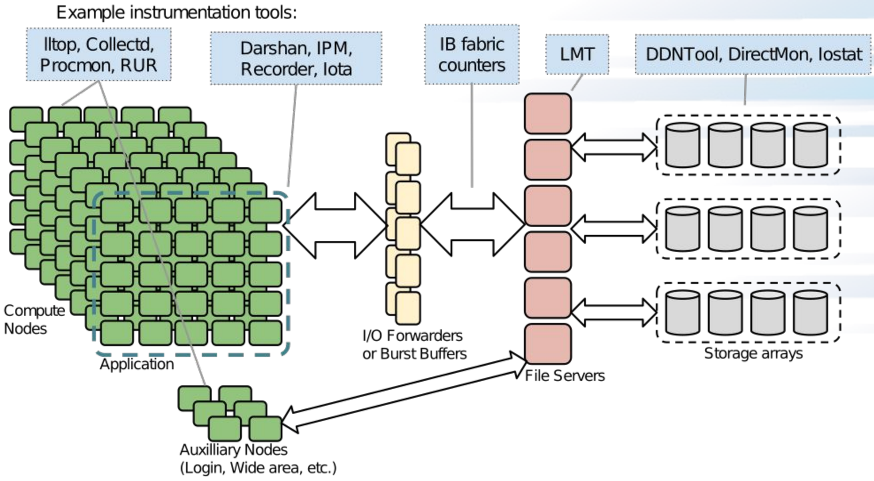
Into the wild...

Many storage resources at HPC facilities are shared between users

- Application-centric analysis can only tell us so much about HPC I/O behavior -- systems-level perspective is needed for complete picture

A more complete understanding of system I/O behavior is critical to reasoning about I/O performance

- How is my performance compared to others?
- What are the performance bottlenecks?
- How much is my I/O affected by contention?



Many existing tools can be used to help compile an accurate system-level view of I/O

Understanding I/O beyond the application

Forming a holistic view

The TOKIO (Total Knowledge of I/O) project aims to provide a framework for holistic characterization and analysis of HPC I/O workloads:

- Collect, integrate, and analyze disparate I/O data
- Define platform-independent blueprint for deploying and utilizing I/O characterization tools, data collection/storage services, and analysis methods
- Provide a trove of relevant data characterizing HPC I/O workloads

Stakeholders:

- Application scientists (productivity)
- Facility operators (efficiency)
- Researchers (optimization)

For more info: https://www.anl.gov/mcs/tokio-total-knowledge-of-io

Hands on exercises:
https://xgitlab.cels.anl.gov/ATPESC-IO/hands-on

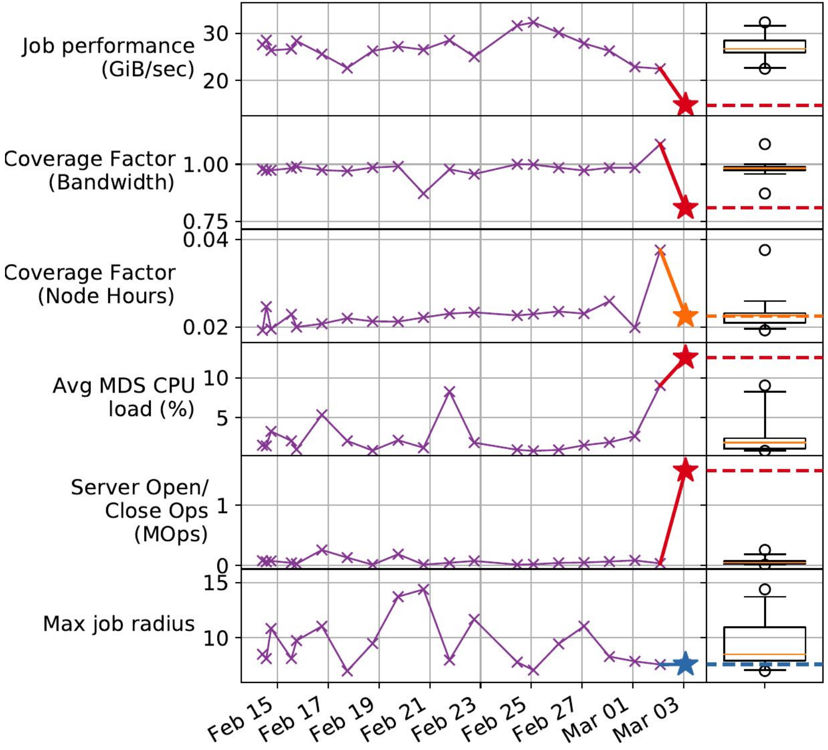
Understanding I/O beyond the application

A TOKIO example

TOKIO utility called UMAMI (Unified metrics and measurements interface) contextualizes application performance measurements with other system measurements

How does my performance compare to previous runs?

Do any metrics stand out that positively/negatively impacted my performance?



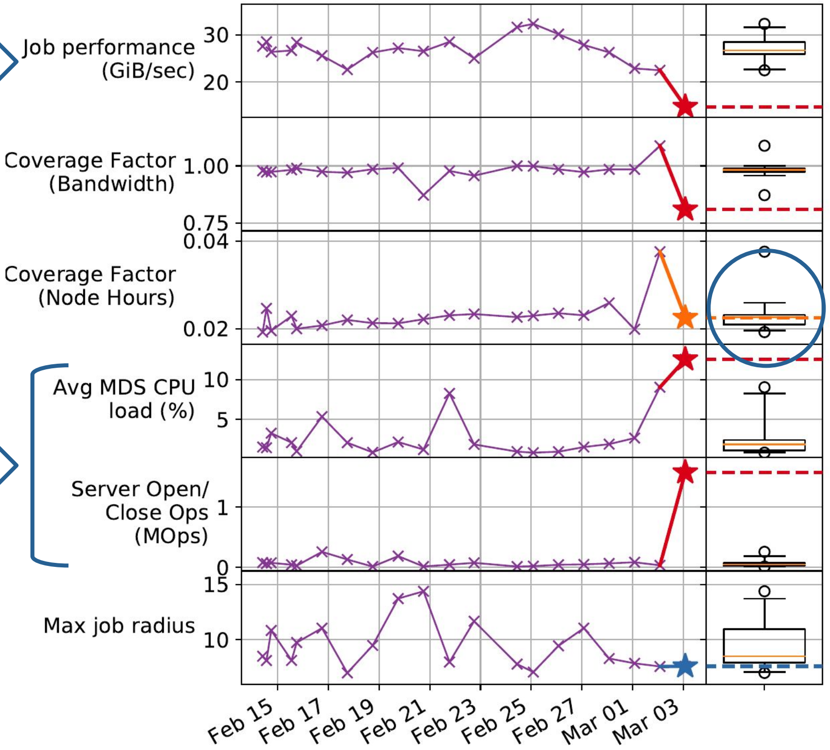
Understanding I/O beyond the application

A TOKIO example

Raw performance

Coverage factor represents contention

Metadata loads



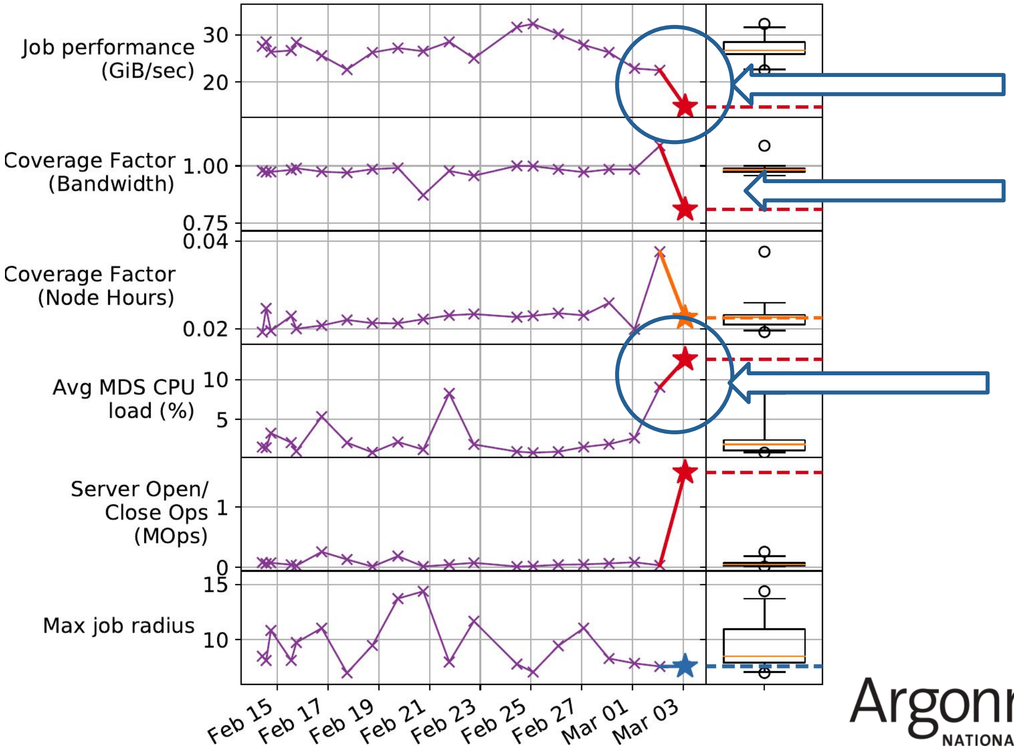
Historical measurements

Box plots to compare against historical measurements

Hands on exercises:
https://xgitlab.cels.anl.gov/ATPESC-IO/hands-on

Understanding I/O beyond the application

A TOKIO example



Low performance relative to recent runs

Low coverage factor, meaning other jobs were performing I/O while we were

High metadata server load also likely impacting performance

Thank you!