ARGONNE ATPESC2025 EXTREME-SCALE COMPUTING

UNDERSTANDING AND TUNING HPC I/O PERFORMANCE

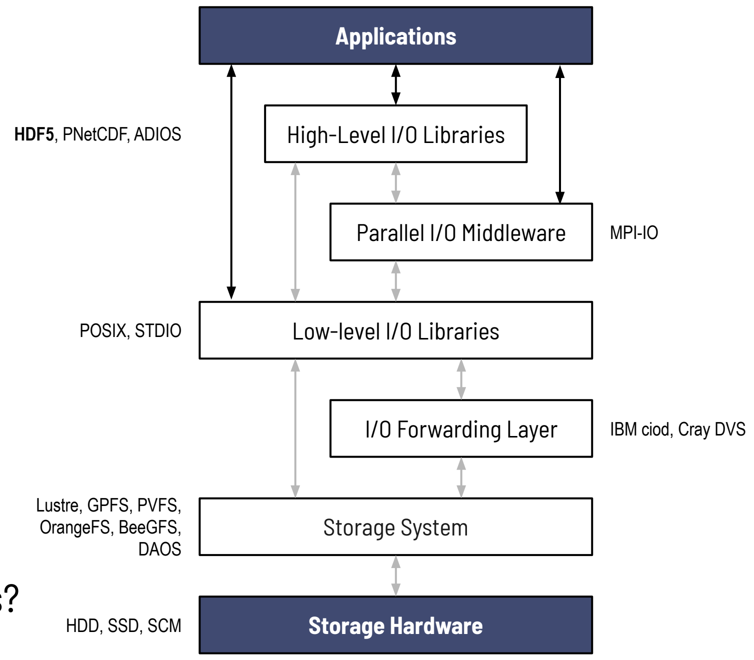
Jean Luca Bez
jlbez@lbl.gov
Lawrence Berkeley National Laboratory

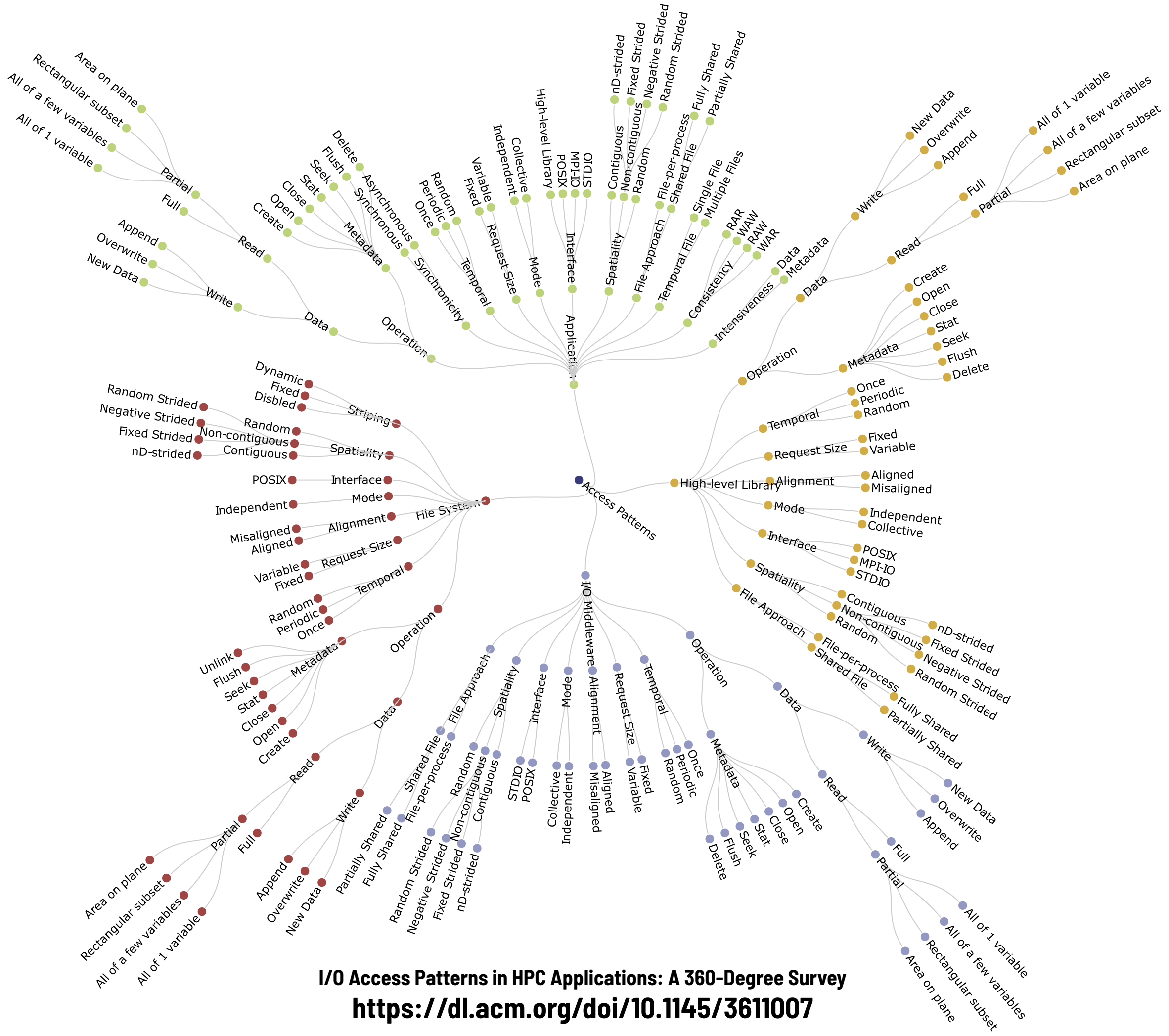
BERKELEY LAB

Scientific Data Division

SURVEYING THE HPC I/O LANDSCAPE

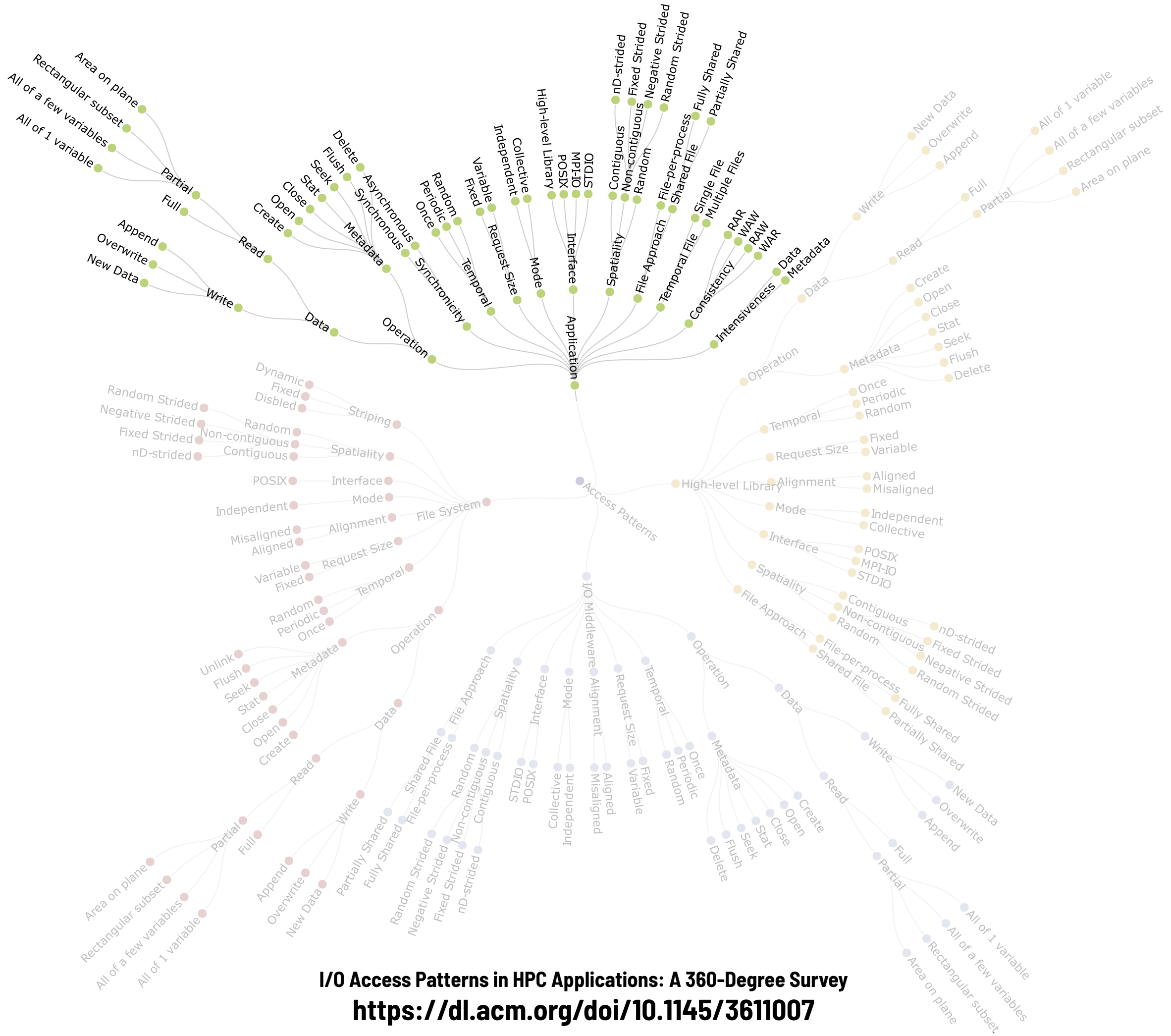
- We have a complex data management ecosystem!
- Using the HPC I/O stack efficiently is a **tricky problem**
- **Interplay of factors** can affect I/O performance
- Various **optimizations** techniques available
- Tons of tunable **parameters** across the stack
- HPC applications are evolving and facing new challenges
- Understanding I/O behavior is difficult
    - How to turn observations into actionable tuning decisions?





I/O Access Patterns in HPC Applications: A 360-Degree Survey

https://dl.acm.org/doi/10.1145/3611007



I/O Access Patterns in HPC Applications: A 360-Degree Survey

https://dl.acm.org/doi/10.1145/3611007

METRICS TO THE RESCUE?

- **Darshan** is a popular tool to collect **I/O profiling**
- Extended tracing mode (**DXT**) for a fine grain view
- There are other profiling tools that capture I/O
  - Recorder, TAU, IOPin, Score-P, etc...
- How to optimize the I/O of my application?

HTML (pyDarshan) based Darshan Report

h5bench_write (2025-08-04)

Darshan Summary Report

Job Summary

Job ID	7047514
User ID	35504
# Processes	64
Run time (s)	21.3769
Start Time	2025-08-04 17:31:28
End Time	2025-08-04 17:31:49
Command Line	/home/jlbez/h5bench/install/bin/h5bench_write /flare/ATPESC2025/usr/jlbez/h5bench-storage/0bff01a5-7047514.aurora-pbs-0001.hostmgmt.cm.aurora.alcf.anl.gov/h5bench.cfg /flare/ATPESC2025/usr/jlbez/h5bench-storage/h5bench.h5

Darshan Log Information

Log Filename	jlbez_h5bench_write_id7047514-100059_8-4-63088-1207275197476997 2726_37.darshan
Runtime Library Version	3.4.7
Log Format Version	3.41
POSIX (ver=4) Module Data	0.23 KiB
MPI-IO (ver=3) Module Data	0.17 KiB
LUSTRE (ver=2) Module Data	0.36 KiB
STDIO (ver=2) Module Data	0.24 KiB
DXT_POSIX (ver=1) Module Data	0.83 KiB
DXT_MPIIO (ver=2) Module Data	34.26 KiB
APMPI (ver=1) Module Data	13.58 KiB
HEATMAP (ver=1) Module Data	7.42 KiB

I/O Summary

Heat Map: HEATMAP MPIIO

Heat map of I/O (in bytes) over time broken down by MPI rank. Bins are populated based on the number of bytes read/written in the given time interval. The top edge bar graph sums each time slice across ranks to show aggregate I/O volume over time, while the right edge bar graph sums each rank across time slices to show I/O distribution across ranks.

Heat Map: DXT_MPIIO

DXT_MPIIO trace data available, but DXT-based heatmaps are not currently generated by default by the job summary tool. To force generation of heatmaps based on DXT trace data, users can specify the '--enable_dxt_heatmap' option.

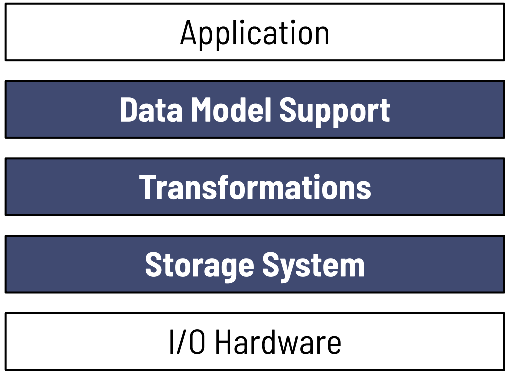
Heat Map: HEATMAP POSIX

Heat Map: DXT_POSIX

DXT_POSIX trace data available, but DXT-based heatmaps are not currently generated by default by the job summary tool. To force generation of heatmaps based on DXT trace data, users can specify the '--enable_dxt_heatmap' option.

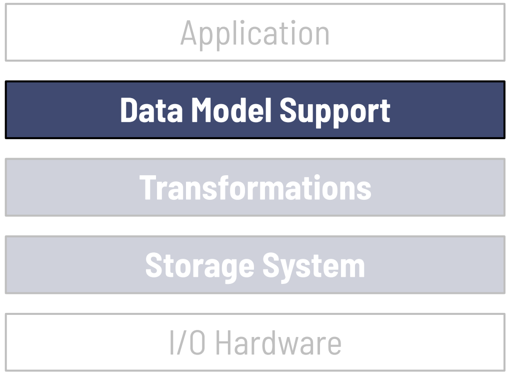
A LOOK UNDER THE HOOD OF AN HPC APPLICATION

- 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 and performance.



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 and performance.



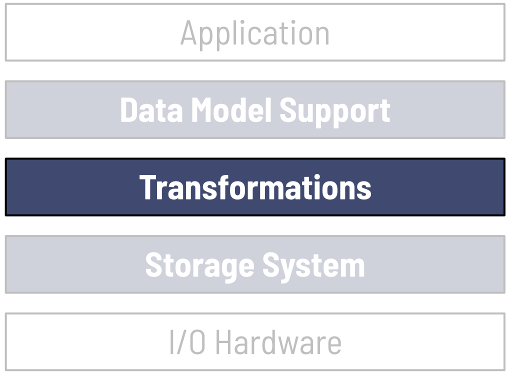
e.g.: **HDF5 stats***

- Accessed files/datasets
- Operation counts
- Total read/write volumes
- Common access info (including details of hyperslab accesses)
- Chunking parameters
- Dataset dimensionality and size
- MPI-IO usage
- I/O timing

*Note: HDF5 instrumentation is not typically enabled for facility Darshan installs – you will need to install this version yourself.

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 and performance.

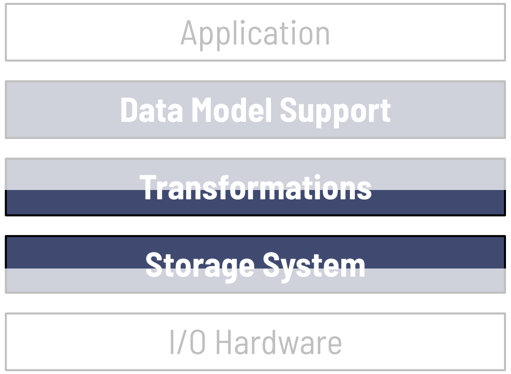


e.g.: **MPI-IO stats**

- Operation counts (open, read, write, sync, etc)
- Collective and independent I/O
- Total read/write volumes
- Access size info
  - Common values
  - Histograms
- I/O timing

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 and performance.

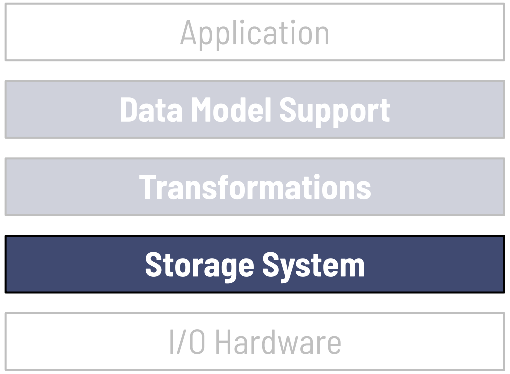


e.g.: **POSIX stats**

- Operation counts (open, read, write, seek, stat)
- Total read/write volumes
- File alignment
- Access size/stride info
  - Common values
  - Histograms
- I/O timing

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 and performance.



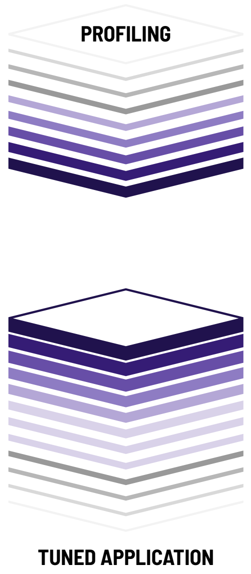
e.g.: **Lustre stats***

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

*Note: Lustre instrumentation is typically enabled for facility Darshan installs that have a Lustre-based parallel file system.

WHAT IS THE PROBLEM?

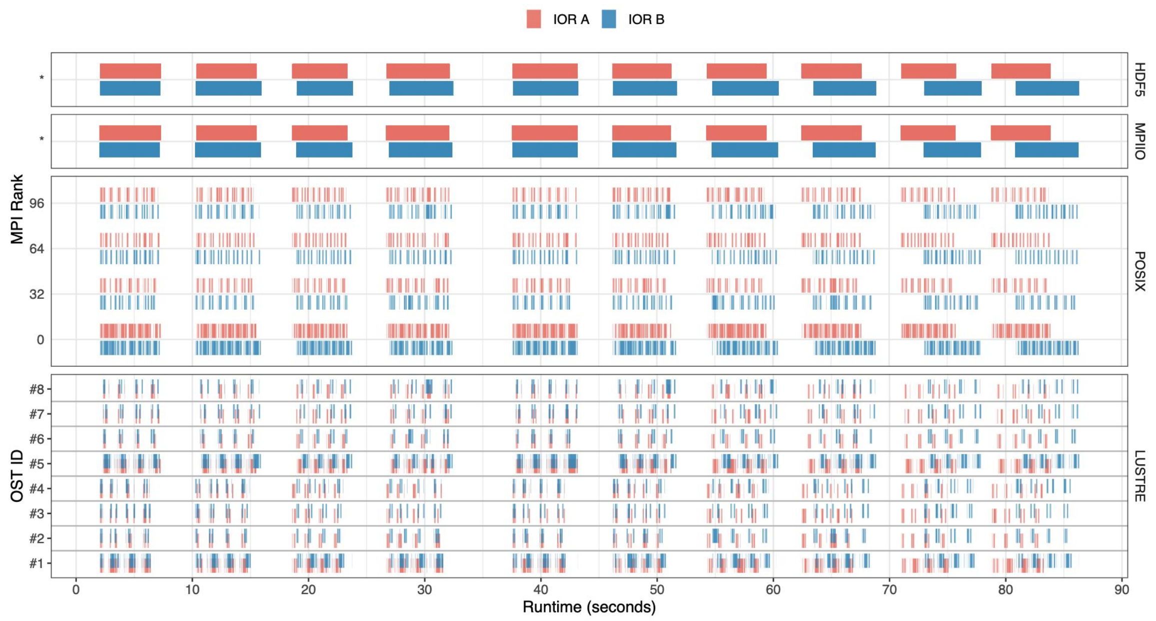
- There is still a **gap** between profiling and tuning
- How to convert I/O metrics to meaningful information?
    - **Visualize** characteristics, behavior, and bottlenecks
    - **Detect** root causes of I/O bottlenecks
    - **Map** I/O bottlenecks into actionable items
    - **Guide** end-user to tune I/O performance



TUNING THE STORAGE SYSTEM

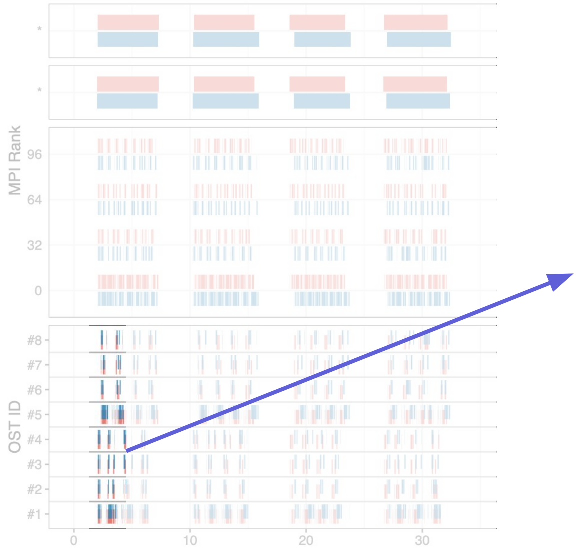
- How to ensure **storage resources** match **application I/O needs**?
  - For some parallel file systems like Lustre, users have direct control over file striping parameters
- **BAD NEWS!** ☹
  - Users may have to have some knowledge of the file system to get good I/O performance.
  - Your choices could have an impact on others using the system
  - Shared file systems aren't perfect (nor perfectly tuned) for every workload!
- **GOOD NEWS!** ☺
  - Users can often get higher I/O performance than system defaults with thoughtful tuning

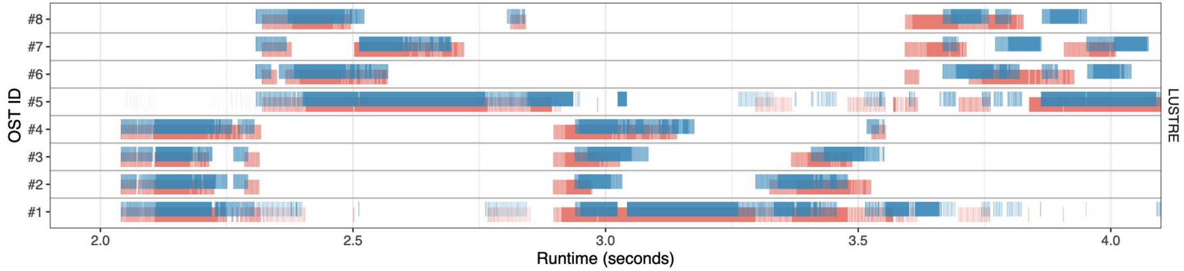
UNDERSTANDING TRANSFORMATIONS



- 2 **IOR** instances
  - Started simultaneously
- Disjoint sets of 16 nodes
  - 8 ranks per node
  - 128 ranks per application
- Access to a **single shared-file**
  - 32MB block size (per rank)
  - 4MB transfer sizes
  - HDF5 + MPI interface

UNDERSTANDING TRANSFORMATIONS

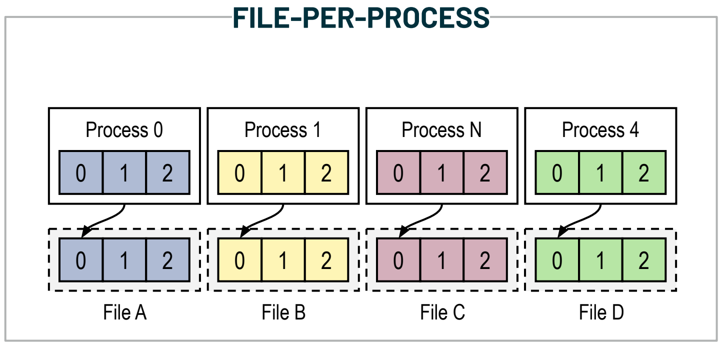


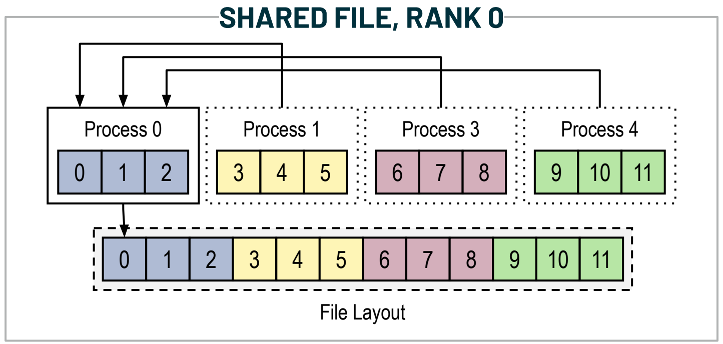


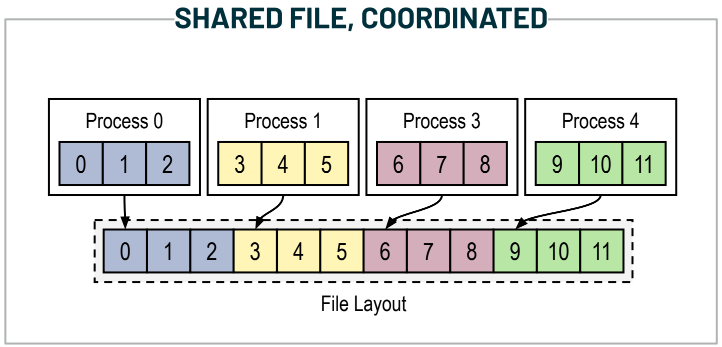
TUNING THE STORAGE SYSTEM

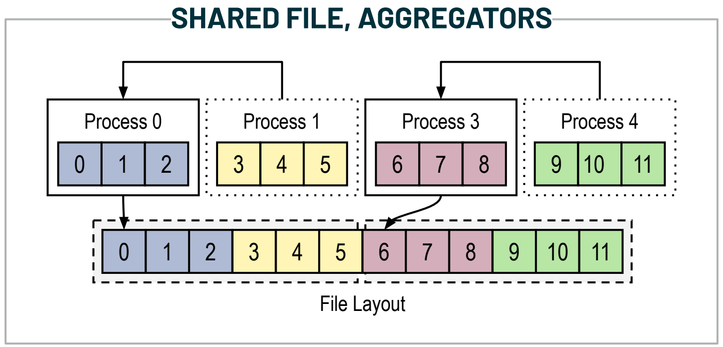
- Tuning decisions can and should be made independently for different **file approaches**

FILE APPROACH



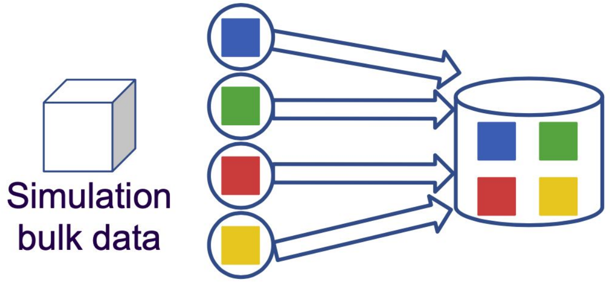






TUNING THE STORAGE SYSTEM

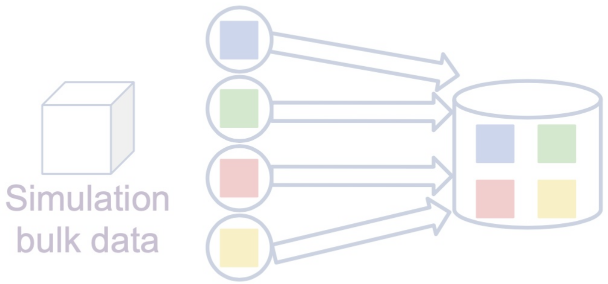
- Tuning decisions can and should be made independently for different **file approaches**



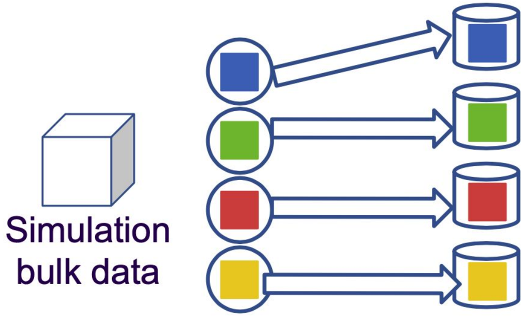
Simulation clients write data to 1 storage server.

TUNING THE STORAGE SYSTEM

- Tuning decisions can and should be made independently for different **file approaches**



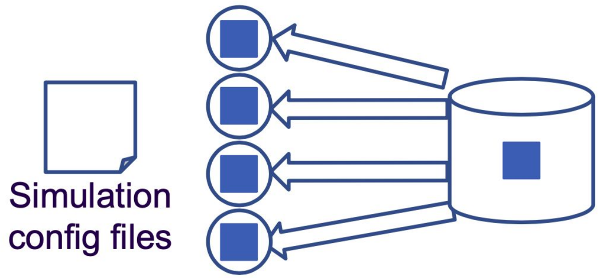
Simulation clients write data to 1 storage server.



Simulation clients load balance writes across multiple servers.

TUNING THE STORAGE SYSTEM

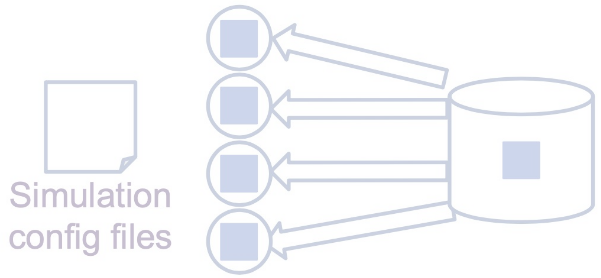
- Tuning decisions can and should be made independently for different **file approaches**
  - On the other hand, smaller files often benefit from being stored on a single server



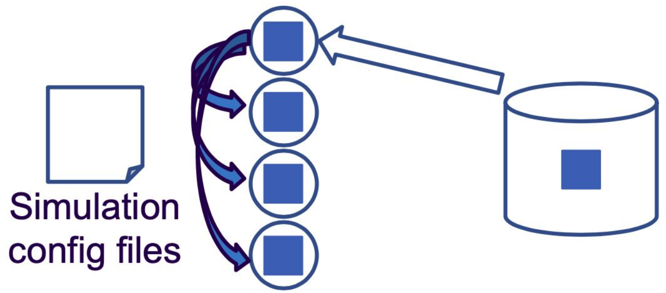
Simulation clients read config data from 1 storage server.

TUNING THE STORAGE SYSTEM

- Tuning decisions can and should be made independently for different **file approaches**
    - On the other hand, smaller files often benefit from being stored on a single server



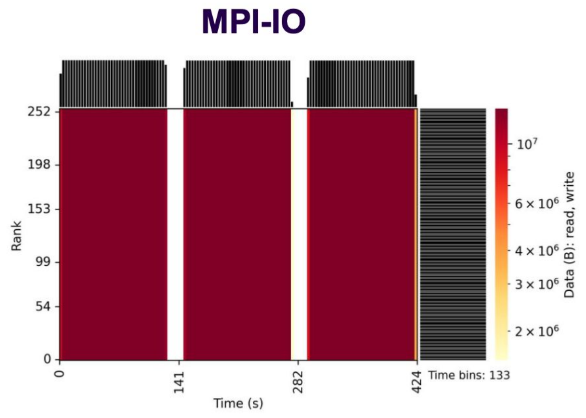
Simulation clients read config data from 1 storage server.

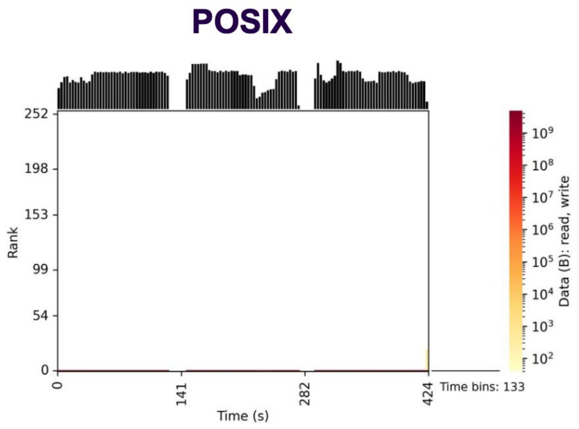


Better yet, limit storage contention by having 1 client read data and distribute using communication (e.g., MPI).

TUNING THE STORAGE SYSTEM

- Be aware of what file system settings are available to you
- Do **not** assume system **defaults** are always the best for your case… you might be surprised
  - ALCF Polaris and NERSC Perlmutter Lustre scratch file systems both have a **default stripe width** of 1 (i.e., files are stored on **one** server)





256 process (4 node) h5bench runs on NERSC Perlmutter

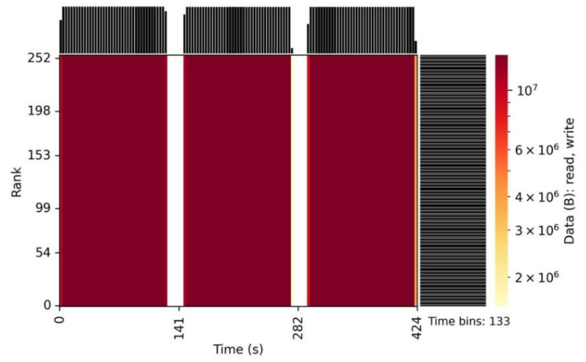
h5bench contains lots of parameters for controlling characteristics of generated HDF5 workloads

https://github.com/hpc-io/h5bench

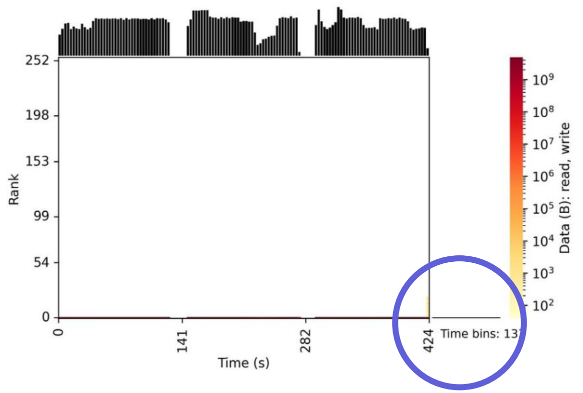
TUNING THE STORAGE SYSTEM

- Be aware of what file system settings are available to you
- Do **not** assume system **defaults** are always the best for your case… you might be surprised
  - ALCF Polaris and NERSC Perlmutter Lustre scratch file systems both have a **default stripe width** of 1 (i.e., files are stored on **one** server)

MPI-IO



POSIX



All the I/O is funneled through rank 0!

MPI-IO collective I/O driver for Lustre assigns dedicated aggregator processes for each stripe, yielding a single aggregator for files of 1 stripe.

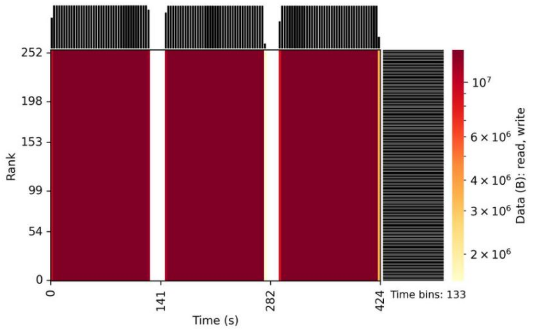
TUNING THE STORAGE SYSTEM

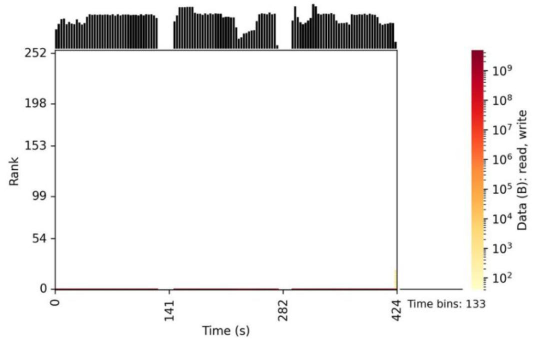
- Ensuring storage resources match application I/O needs

MPI-IO

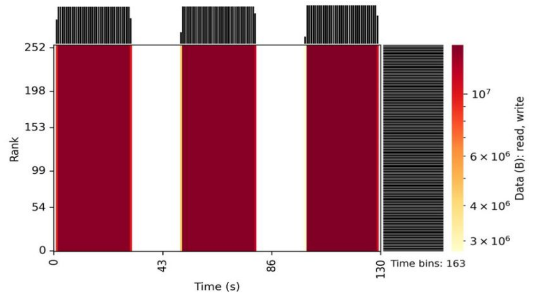
POSIX

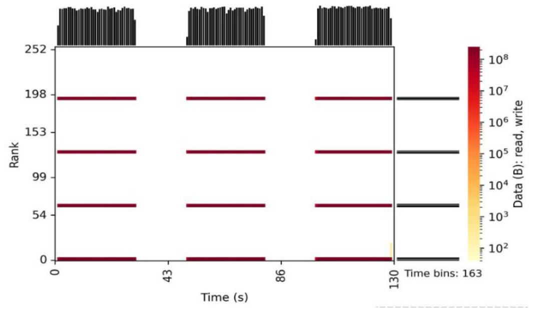
1 stripe





16 stripes





Manually setting the stripe width to 16 yields more I/O aggregators and better performance:

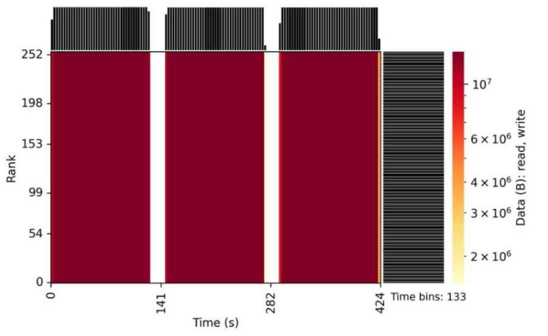
```
> lfs setstripe -c 16 testFile
```

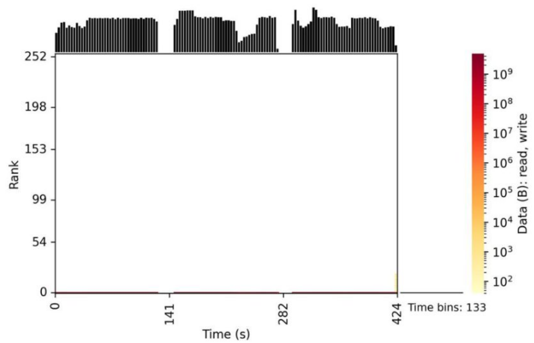
TUNING THE STORAGE SYSTEM

- Ensuring storage resources match application I/O needs

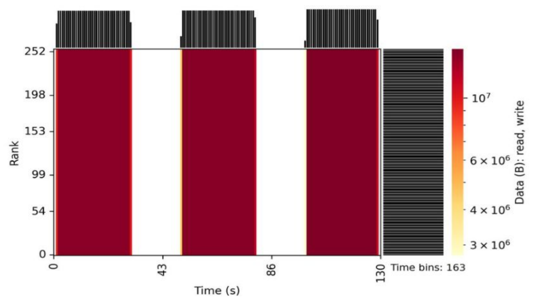
MPI-IO | **POSIX**

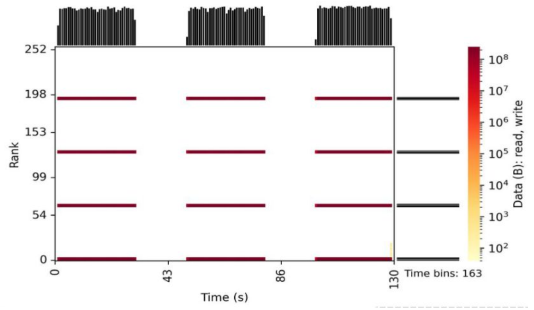
1 stripe





16 stripes





1341.13 MiB/s

Manually setting the stripe width to 16 yields more I/O aggregators and better performance:

```
> lfs setstripe -c 16 testFile
```

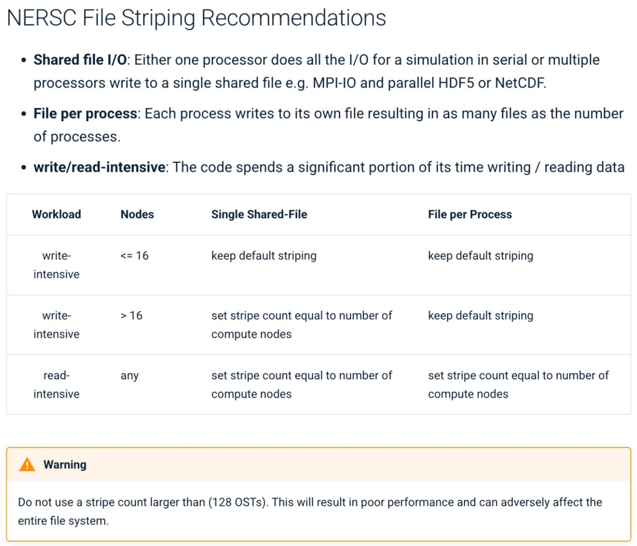
4X!

5571.27 MiB/s

TUNING THE STORAGE SYSTEM

- Consult **facilities documentation** for established best practice!
  - **Suggestions** and commands for properly striping different types of files and workloads

NERSC



NERSC File Striping Recommendations

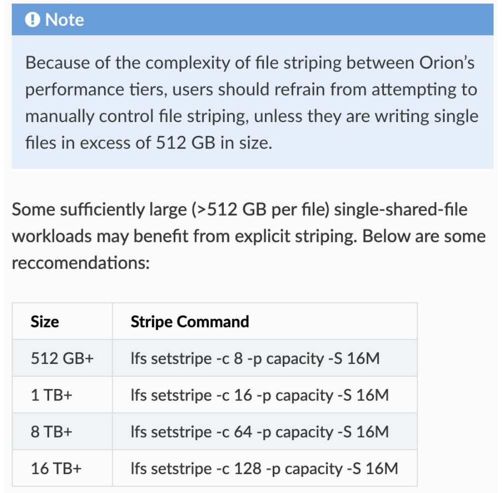
- **Shared file I/O**: Either one processor does all the I/O for a simulation in serial or multiple processors write to a single shared file e.g. MPI-IO and parallel HDF5 or NetCDF.
- **File per process**: Each process writes to its own file resulting in as many files as the number of processes.
- **write/read-intensive**: The code spends a significant portion of its time writing / reading data

Workload	Nodes	Single Shared-File	File per Process
write-intensive	<= 16	keep default striping	keep default striping
write-intensive	> 16	set stripe count equal to number of compute nodes	keep default striping
read-intensive	any	set stripe count equal to number of compute nodes	set stripe count equal to number of compute nodes

⚠ **Warning**

Do not use a stripe count larger than (128 OSTs). This will result in poor performance and can adversely affect the entire file system.

OAK RIDGE National Laboratory | LEADERSHIP COMPUTING FACILITY



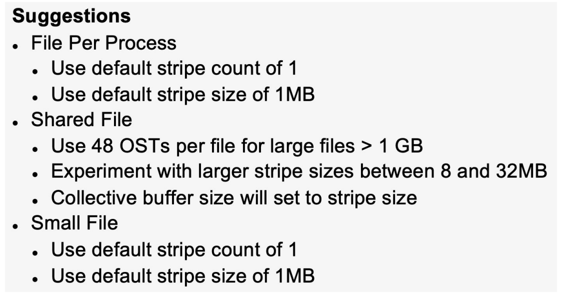
Note

Because of the complexity of file striping between Orion's performance tiers, users should refrain from attempting to manually control file striping, unless they are writing single files in excess of 512 GB in size.

Some sufficiently large (>512 GB per file) single-shared-file workloads may benefit from explicit striping. Below are some reccomendations:

Size	Stripe Command
512 GB+	lfs setstripe -c 8 -p capacity -S 16M
1 TB+	lfs setstripe -c 16 -p capacity -S 16M
8 TB+	lfs setstripe -c 64 -p capacity -S 16M
16 TB+	lfs setstripe -c 128 -p capacity -S 16M

Argonne NATIONAL LABORATORY

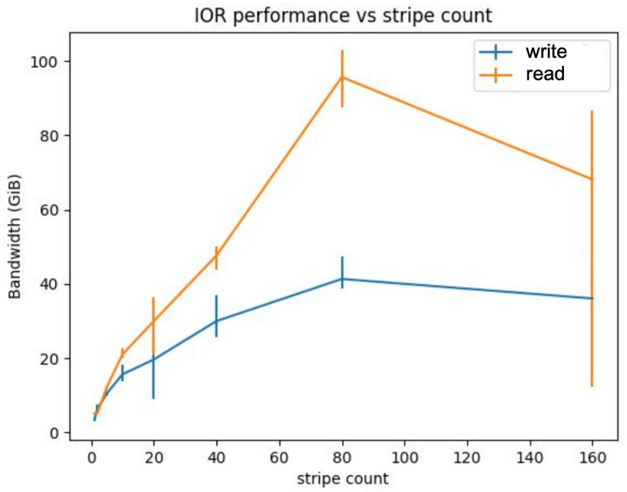


Suggestions

- 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

TUNING THE STORAGE SYSTEM

- Consult **facilities documentation** for established best practice!
    - Sometimes (quite often!) you may even **need to experiment yourself**!



128-node example of the IOR benchmark using various stripe counts on ALCF Polaris:
https://github.com/radix-io/io-sleuthing/tree/main/examples/striping

For more I/O intensive programs, it's typically better to err on the side of **more** storage servers.

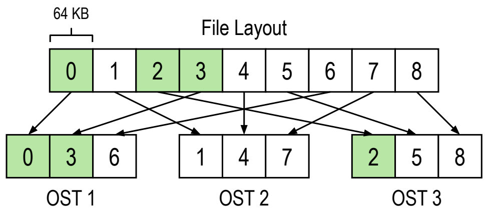
The following command stripes across all servers:

```
> lfs setstripe -c -1 testFile
```

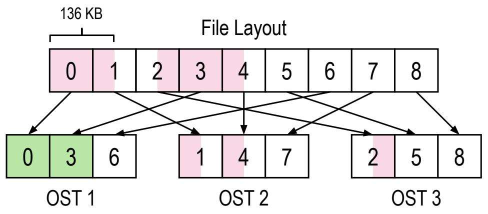
TUNING THE STORAGE SYSTEM

- Files are broken down into stripes and distributed for parallelism
  - A single I/O operation might need to reach **multiple servers** (OSTs in Lustre)
  - Requests **not aligned** to the PFS stripe boundaries might perform **poorly**

REQUEST ALIGNED TO STRIPE

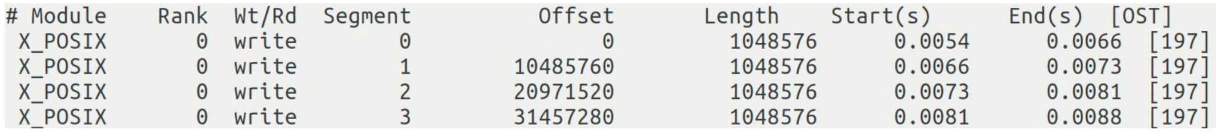


REQUEST NOT ALIGNED TO STRIPE



TUNING THE STORAGE SYSTEM

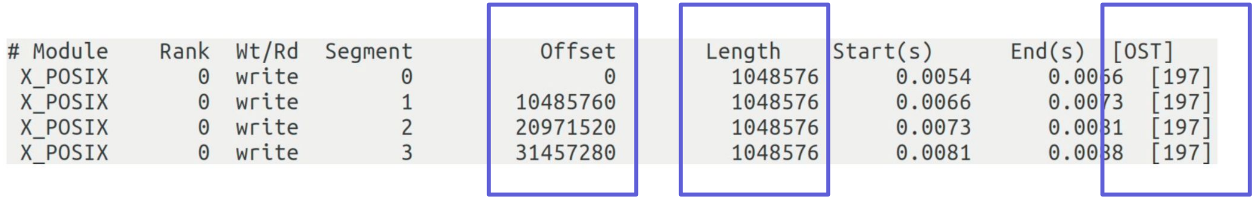
- Simple 10-process (10-node) run where processes write in an interleaved fashion to a **single shared file**:
  - Use a tracing tool (e.g. Darshan DXT or Recorder) to get details about individual accesses



```
# 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       0  write        1        10485760      1048576      0.0066      0.0073  [197]
 X_POSIX       0  write        2        20971520      1048576      0.0073      0.0081  [197]
 X_POSIX       0  write        3        31457280      1048576      0.0081      0.0088  [197]
```

TUNING THE STORAGE SYSTEM

- Simple 10-process (10-node) run where processes write in an interleaved fashion to a single shared file:
  - Use a tracing tool (e.g. Darshan DXT or Recorder) to get details about individual accesses

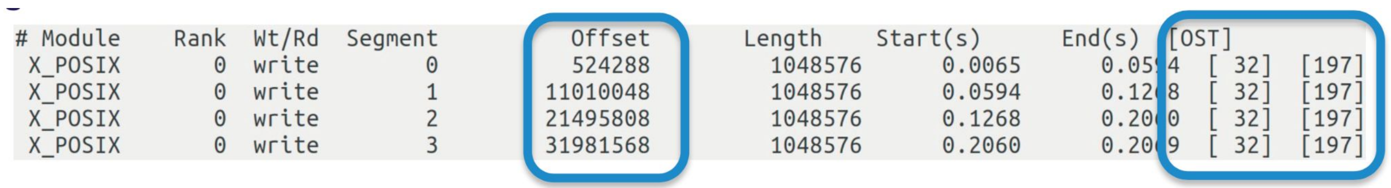


```
# Module     Rank  Wt/Rd  Segment          Offset       Length  Start(s)      End(s)  [OST]
 X_POSIX        0  write        0               0      1048576    0.0054      0.0066   [197]
 X_POSIX        0  write        1        10485760      1048576    0.0066      0.0073   [197]
 X_POSIX        0  write        2        20971520      1048576    0.0073      0.0081   [197]
 X_POSIX        0  write        3        31457280      1048576    0.0081      0.0088   [197]
```

- Each access is **aligned** to the Lustre stripe size (**1 MiB**)
- Each process interacts with a **single** Lustre **server** (OST)

TUNING THE STORAGE SYSTEM

- Simple 10-process (10-node) run where processes write in an interleaved fashion to a single shared file:
  - Use a tracing tool (e.g. Darshan DXT or Recorder) to get details about individual accesses



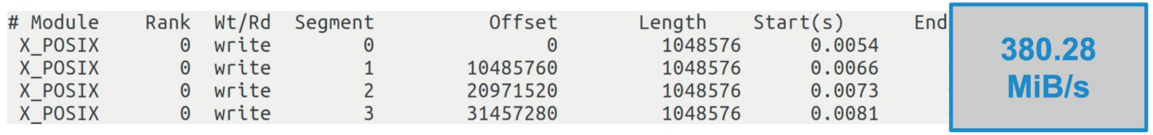
```
# Module    Rank  Wt/Rd  Segment       Offset       Length    Start(s)      End(s)  [OST]
 X_POSIX       0  write        0       524288      1048576      0.0065      0.0594  [ 32]  [197]
 X_POSIX       0  write        1     11010048      1048576      0.0594      0.1268  [ 32]  [197]
 X_POSIX       0  write        2     21495808      1048576      0.1268      0.2060  [ 32]  [197]
 X_POSIX       0  write        3     31981568      1048576      0.2060      0.2069  [ 32]  [197]
```

- Each access spans two Lustre stripes due to **unaligned** offsets
- Each process interacts with **two** Lustre **servers** (OSTs)

TUNING THE STORAGE SYSTEM

- Even in this small workload
  - We pay a **~20% performance penalty** when I/O accesses are not aligned to file stripes (1 MB)

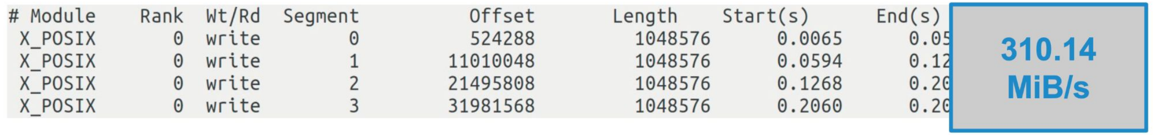
aligned



```
# Module      Rank  Wt/Rd  Segment           Offset        Length    Start(s)      End
  X_POSIX        0  write        0                0       1048576      0.0054
  X_POSIX        0  write        1         10485760       1048576      0.0066
  X_POSIX        0  write        2         20971520       1048576      0.0073
  X_POSIX        0  write        3         31457280       1048576      0.0081
```

380.28 MiB/s

unaligned



```
# Module      Rank  Wt/Rd  Segment          Offset       Length    Start(s)     End(s)
  X_POSIX        0  write        0          524288      1048576      0.0065       0.05
  X_POSIX        0  write        1        11010048      1048576      0.0594       0.12
  X_POSIX        0  write        2        21495808      1048576      0.1268       0.20
  X_POSIX        0  write        3        31981568      1048576      0.2060       0.20
```

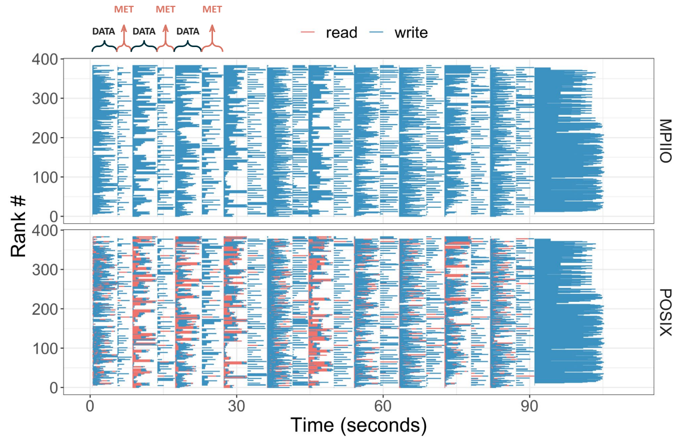
310.14 MiB/s

NO NEED TO REINVENT THE WHEEL

- Accounting for **subtle I/O performance factors** like file alignment can be a painstaking process…
- As highlighted by other presentations:
  - High-level I/O libraries like HDF5 and PnetCDF can help **mask** much of the complexity needed for transforming scientific computing I/O workloads into performant POSIX-level file system accesses
- Use high-level I/O libraries wherever you can!
  - But also **read** the **documentation** and **follow best practices**!

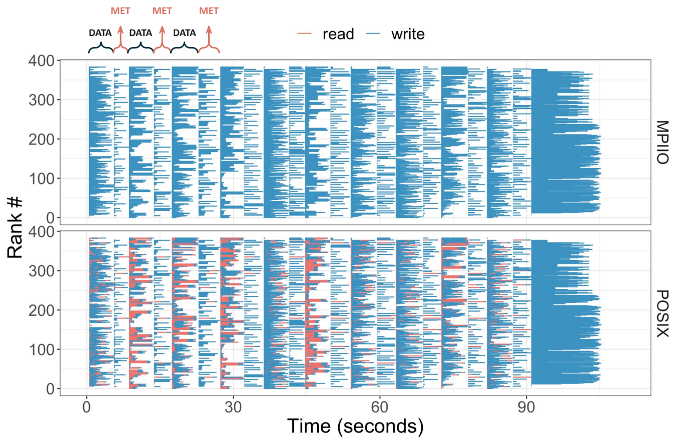
TUNING HIGH-LEVEL LIBRARIES

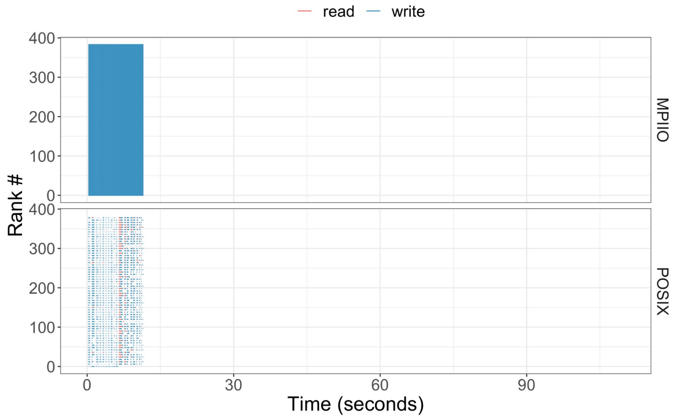
- e.g. OpenPMD / WarpX
  - 64 compute nodes, 6 ranks per node, and a total of 384 MPI ranks
  - Mesh size is $[65536 \times 256 \times 256]$, 10 iterations, total file size is ≈121GB



TUNING HIGH-LEVEL LIBRARIES

- e.g. OpenPMD / WarpX
  - **Collective** I/O using ROMIO hints with 1 agg/node and 16 MB collective buffer size
  - GPFS **large block** I/O with HDF5 **collective metadata**
  - Collective operations used for data and metadata





BASELINE

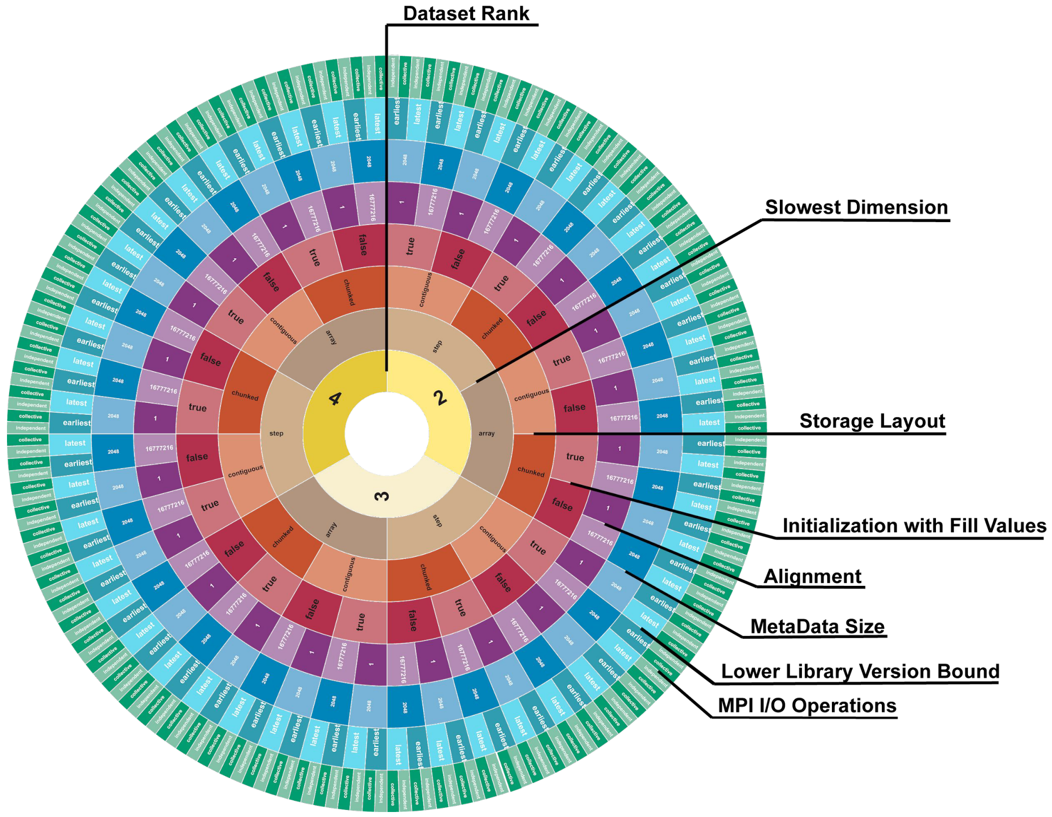
110.6s

6.8x

OPTIMIZED

16.1s

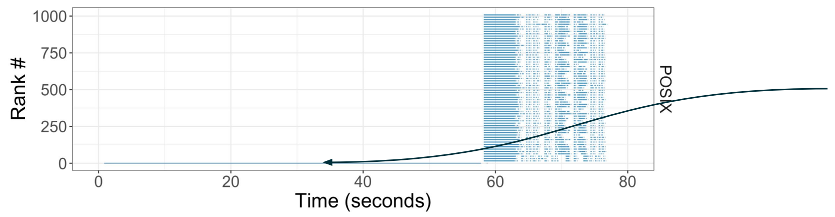
HDF5 TUNING PARAMETER SPACE



Courtesy image from Suren Byna

TUNING HIGH-LEVEL LIBRARIES

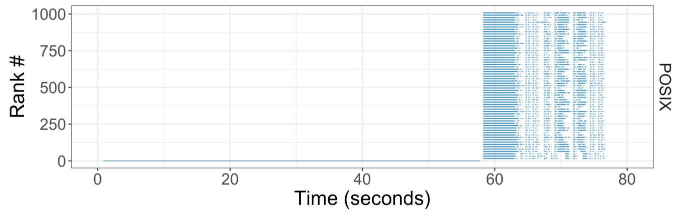
- e.g. E2E Benchmarks
  - 4 compute nodes, 6 ranks per node, and a total of 1024 MPI ranks
  - 1024 processes arranged in a 32 x 32 x 16 distribution, total file size is ≈41GB
  - 44% of the time is taken by rank 0!

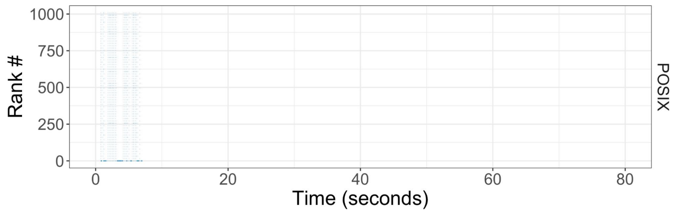


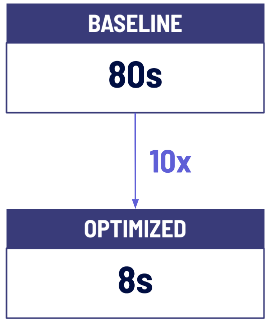
Rank 0 is **sequentially writing fill values** to all of the defined variables (10 in this workload), issuing over 40 thousand write requests with of ≈1MB

TUNING HIGH-LEVEL LIBRARIES

- e.g. E2E Benchmarks
  - 4 compute nodes, 6 ranks per node, and a total of 1024 MPI ranks
  - 1024 processes arranged in a 32 x 32 x 16 distribution, total file size is ≈41GB
  - 44% of the time is taken by rank 0!
  - Disabling the data filling (`NC_NOFILL` in NetCDF) translates to **7.3x speedup**

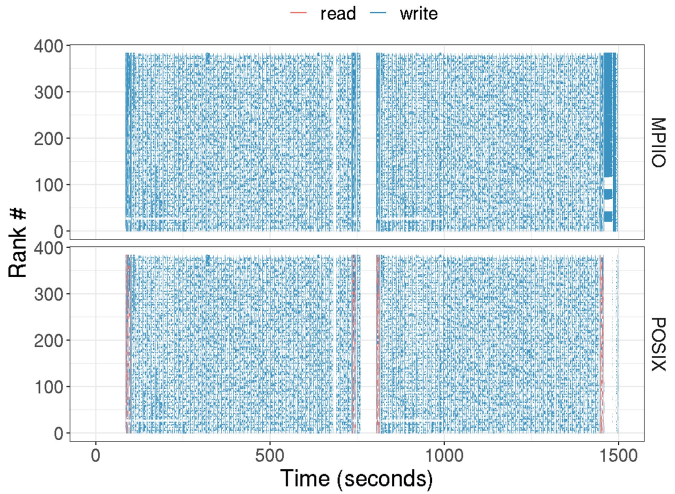






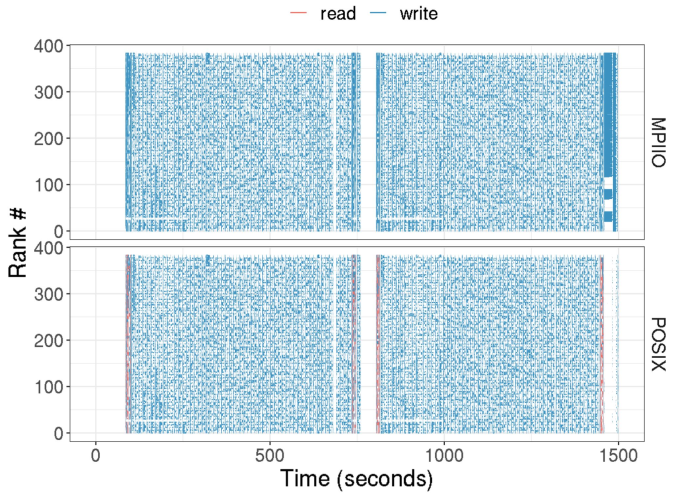
TUNING HIGH-LEVEL LIBRARIES

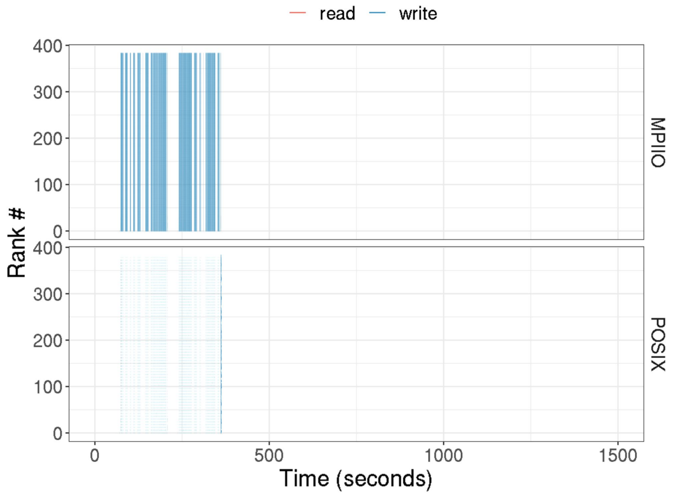
- e.g. FLASH
  - 64 compute nodes, 6 ranks per node, and a total of 384 MPI ranks
  - 2 checkpoint files (≈2.3TB each) and 2 plot file (≈14GB each) both using HDF5 backend

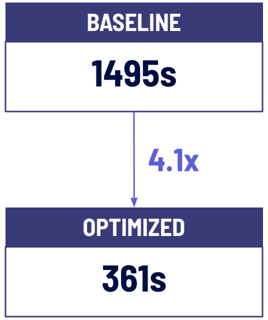


TUNING HIGH-LEVEL LIBRARIES

- e.g. FLASH
  - Collective I/O using ROMIO hints with 1 agg/node and 16 MB collective buffer size provides 3.2x speedup
  - Setting the HDF5 alignment size to 16 MB provides an additional 1.18x speedup
  - Deferring the HDF5 metadata flush provides another 1.1x speedup







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	🙂	🙂	🙂	🙁
MPI-IO	🙂	🙂	🙂	🙁
POSIX	🙂	😐	🙁	🙁

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	☺	☺	☺	☹
MPI-IO	☺	☺	☺	☹
POSIX	☺	😐	☹	☹

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

POSIX I/O requires manually aligning every access

Collective I/O can be automatically aligned

SOMETIMES WE NEED MORE!

- By default, Darshan captures a (large!) fixed set of counters for each file
- With DXT (**Darshan Extended Tracing**):
  - Darshan **traces** every **read/write operation** (for POSIX and MPI-IO interfaces)
- Enabled by setting `DXT_ENABLE_IO_TRACE` env variable

```
export DXT_ENABLE_IO_TRACE=1
```

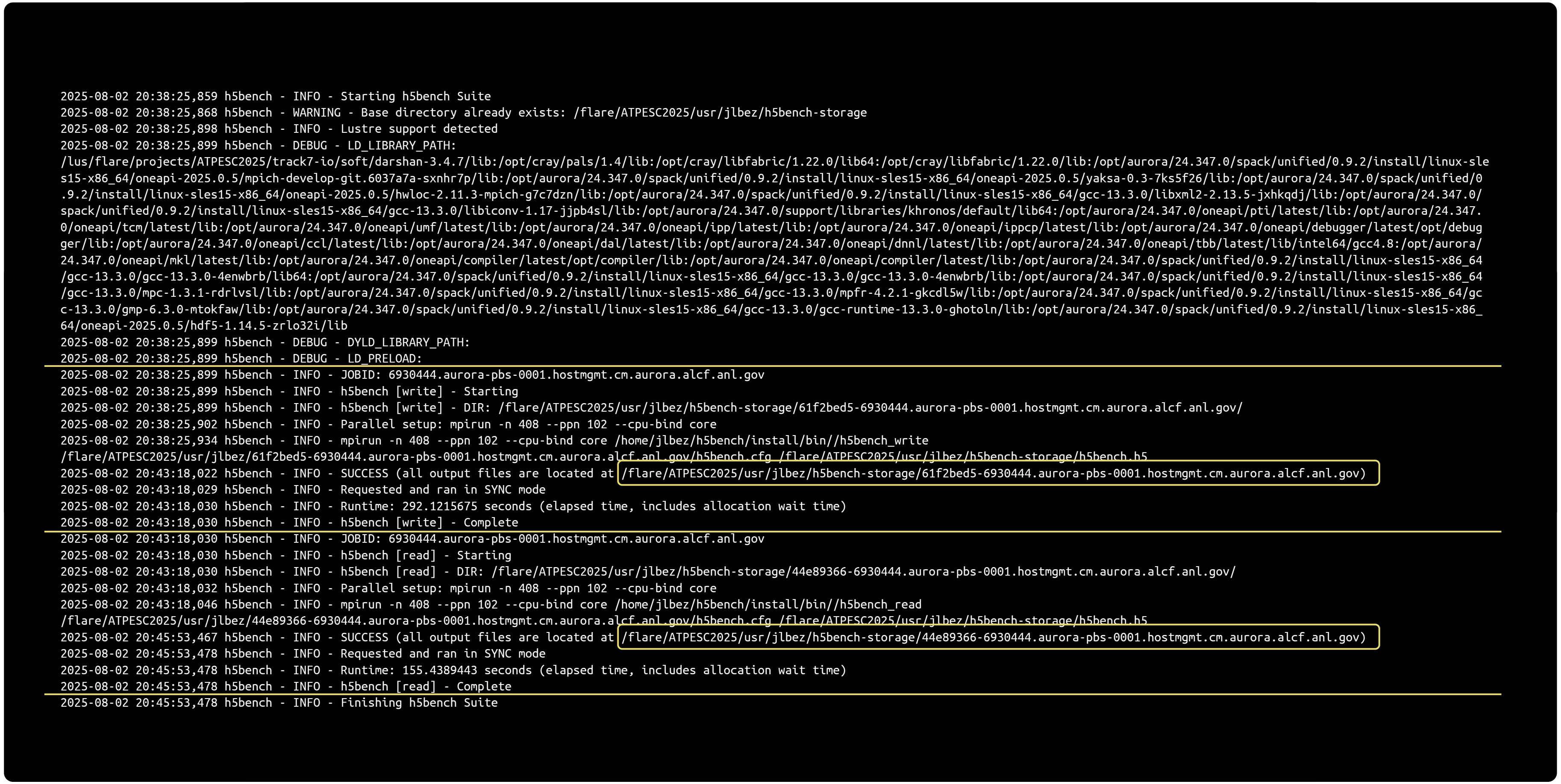
- **Finer grained** instrumentation data comes at a cost of additional overhead and larger logs!
  - Hence, this option is **not** enabled by default in facilities!

```
# ***************************************************
# DXT_POSIX module data
# ***************************************************

# DXT, file_id: 13771918696892050919, file_name:
/gpfs/alpine/csc300/scratch/anonymous/Flash-X-apr8.gcc/FLASH_IO_hdf5_1.10.6/2366525/flash.par
# DXT, rank: 0, hostname: d11n01
# DXT, write_count: 0, read_count: 3
# DXT, mnt_pt: /gpfs/alpine, fs_type: gpfs
# Module    Rank  Wt/Rd  Segment          Offset       Length    Start(s)      End(s)
 X_POSIX       0   read        0               0             783      0.0110      0.0110
 X_POSIX       0   read        1             783               0      0.0111      0.0111
 X_POSIX       0   read        2             783               0      0.0111      0.0111

# DXT, file_id: 17855743881390289785, file_name:
/gpfs/alpine/csc300/scratch/anonymous/Flash-X-apr8.gcc/FLASH_IO_hdf5_1.10.6/2366525/flash.log
# DXT, rank: 0, hostname: d11n01
# DXT, write_count: 62, read_count: 0
# DXT, mnt_pt: /gpfs/alpine, fs_type: gpfs
# Module    Rank  Wt/Rd  Segment          Offset       Length    Start(s)      End(s)
 X_POSIX       0  write        0               0            4105      0.0518      0.0527
 X_POSIX       0  write        1            4105            4141      0.0530      0.0530
 X_POSIX       0  write        2            8246            4127      0.0532      0.0532
 X_POSIX       0  write        3           12373            4097      0.0534      0.0547
...
```

Trace includes the **timestamp**, file **offset**, and **size** of every I/O **operation** on every **rank**. `darshan-dxt-parser` utility can provide a raw text dump of the trace.

- Build **h5bench** and try to run it with the **atpsec.json** configuration
  - [https://github.com/hpdc-io/**h5bench**](https://github.com/hpdc-io/h5bench)
  - [https://github.com/raxid-io/hands-on/**h5bench**](https://github.com/raxid-io/hands-on/h5bench)
- Remember to collect **Darshan logs** and **traces**!
- What should I **look at**?
  - What can you infer about the application I/O **behavior** from Darshan's report?
  - What is the **I/O bandwidth** and **time**?
  - Do you see any opportunities to **tune** the I/O?
- Try making changes to the I/O patterns with exposed configuration

```json
{
    "mpi": {
        "command": "srun",
        "configuration": "-N 4 -n 512"
    },
    "vol": {

    },
    "file-system": {
        "lustre": {
            "stripe-size": "1M",
            "stripe-count": "1"
        }
    },
    "directory": "SCRATCHDIR",
    "benchmarks": [
        {
            "benchmark": "write",
            "file": "h5bench.h5",
            "configuration": {
                "MEM_PATTERN": "CONTIG",
                "FILE_PATTERN": "CONTIG",
                "TIMESTEPS": "5",
                "DELAYED_CLOSE_TIMESTEPS": "2",
                "COLLECTIVE_DATA": "YES",
                "COLLECTIVE_METADATA": "NO",
                "EMULATED_COMPUTE_TIME_PER_TIMESTEP": "5 s",
                "NUM_DIMS": "1",
                "DIM_1": "4194304",
                "DIM_2": "1",
                "DIM_3": "1",
                "CSV_FILE": "output.csv",
                "MODE": "SYNC"
            }
        },
        {
            "benchmark": "read",
            "file": "h5bench.h5",
            "configuration": {
                "MEM_PATTERN": "CONTIG",
                "FILE_PATTERN": "CONTIG",
                "READ_OPTION": "FULL",
                "TIMESTEPS": "5",
                "DELAYED_CLOSE_TIMESTEPS": "2",
                "COLLECTIVE_DATA": "YES",
                "COLLECTIVE_METADATA": "NO",
                "EMULATED_COMPUTE_TIME_PER_TIMESTEP": "5 s",
                "NUM_DIMS": "1",
                "DIM_1": "4194304",
                "DIM_2": "1",
                "DIM_3": "1",
                "CSV_FILE": "output.csv",
                "MODE": "SYNC"
            }
        }
    ]
}
```

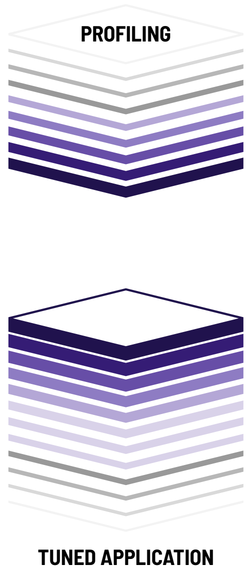


```
2025-08-02 20:38:25,859 h5bench - INFO - Starting h5bench Suite
2025-08-02 20:38:25,868 h5bench - WARNING - Base directory already exists: /flare/ATPESC2025/usr/jlbez/h5bench-storage
2025-08-02 20:38:25,898 h5bench - INFO - Lustre support detected
2025-08-02 20:38:25,899 h5bench - DEBUG - LD_LIBRARY_PATH:
/lus/flare/projects/ATPESC2025/track7-io/soft/darshan-3.4.7/lib:/opt/cray/pals/1.4/lib:/opt/cray/libfabric/1.22.0/lib64:/opt/cray/libfabric/1.22.0/lib:/opt/aurora/24.347.0/spack/unified/0.9.2/install/linux-sle
s15-x86_64/oneapi-2025.0.5/mpich-develop-git.6037a7a-sxnhr7p/lib:/opt/aurora/24.347.0/spack/unified/0.9.2/install/linux-sles15-x86_64/oneapi-2025.0.5/yaksa-0.3-7ks5f26/lib:/opt/aurora/24.347.0/spack/unified/0
.9.2/install/linux-sles15-x86_64/oneapi-2025.0.5/hwloc-2.11.3-mpich-g7c7dzn/lib:/opt/aurora/24.347.0/spack/unified/0.9.2/install/linux-sles15-x86_64/gcc-13.3.0/libxml2-2.13.5-jxhkqdj/lib:/opt/aurora/24.347.0/
spack/unified/0.9.2/install/linux-sles15-x86_64/gcc-13.3.0/libiconv-1.17-jjpb4sl/lib:/opt/aurora/24.347.0/support/libraries/khronos/default/lib64:/opt/aurora/24.347.0/oneapi/pti/latest/lib:/opt/aurora/24.347.
0/oneapi/tcm/latest/lib:/opt/aurora/24.347.0/oneapi/umf/latest/lib:/opt/aurora/24.347.0/oneapi/ipp/latest/lib:/opt/aurora/24.347.0/oneapi/ippcp/latest/lib:/opt/aurora/24.347.0/oneapi/debugger/latest/opt/debug
ger/lib:/opt/aurora/24.347.0/oneapi/ccl/latest/lib:/opt/aurora/24.347.0/oneapi/dal/latest/lib:/opt/aurora/24.347.0/oneapi/dnnl/latest/lib:/opt/aurora/24.347.0/oneapi/tbb/latest/lib/intel64/gcc4.8:/opt/aurora/
24.347.0/oneapi/mkl/latest/lib:/opt/aurora/24.347.0/oneapi/compiler/latest/opt/compiler/lib:/opt/aurora/24.347.0/oneapi/compiler/latest/lib:/opt/aurora/24.347.0/spack/unified/0.9.2/install/linux-sles15-x86_64
/gcc-13.3.0/gcc-13.3.0-4enwbrb/lib64:/opt/aurora/24.347.0/spack/unified/0.9.2/install/linux-sles15-x86_64/gcc-13.3.0/gcc-13.3.0-4enwbrb/lib:/opt/aurora/24.347.0/spack/unified/0.9.2/install/linux-sles15-x86_64
/gcc-13.3.0/mpc-1.3.1-rdrlvsl/lib:/opt/aurora/24.347.0/spack/unified/0.9.2/install/linux-sles15-x86_64/gcc-13.3.0/mpfr-4.2.1-gkcdl5w/lib:/opt/aurora/24.347.0/spack/unified/0.9.2/install/linux-sles15-x86_64/gc
c-13.3.0/gmp-6.3.0-mtokfaw/lib:/opt/aurora/24.347.0/spack/unified/0.9.2/install/linux-sles15-x86_64/gcc-13.3.0/gcc-runtime-13.3.0-ghotoln/lib:/opt/aurora/24.347.0/spack/unified/0.9.2/install/linux-sles15-x86_
64/oneapi-2025.0.5/hdf5-1.14.5-zrlo32i/lib
2025-08-02 20:38:25,899 h5bench - DEBUG - DYLD_LIBRARY_PATH:
2025-08-02 20:38:25,899 h5bench - DEBUG - LD_PRELOAD:
2025-08-02 20:38:25,899 h5bench - INFO - JOBID: 6930444.aurora-pbs-0001.hostmgmt.cm.aurora.alcf.anl.gov
2025-08-02 20:38:25,899 h5bench - INFO - h5bench [write] - Starting
2025-08-02 20:38:25,899 h5bench - INFO - h5bench [write] - DIR: /flare/ATPESC2025/usr/jlbez/h5bench-storage/61f2bed5-6930444.aurora-pbs-0001.hostmgmt.cm.aurora.alcf.anl.gov/
2025-08-02 20:38:25,902 h5bench - INFO - Parallel setup: mpirun -n 408 --ppn 102 --cpu-bind core
2025-08-02 20:38:25,934 h5bench - INFO - mpirun -n 408 --ppn 102 --cpu-bind core /home/jlbez/h5bench/install/bin//h5bench_write
/flare/ATPESC2025/usr/jlbez/61f2bed5-6930444.aurora-pbs-0001.hostmgmt.cm.aurora.alcf.anl.gov/h5bench.cfg /flare/ATPESC2025/usr/jlbez/h5bench-storage/h5bench.h5
2025-08-02 20:43:18,022 h5bench - INFO - SUCCESS (all output files are located at /flare/ATPESC2025/usr/jlbez/h5bench-storage/61f2bed5-6930444.aurora-pbs-0001.hostmgmt.cm.aurora.alcf.anl.gov)
2025-08-02 20:43:18,029 h5bench - INFO - Requested and ran in SYNC mode
2025-08-02 20:43:18,030 h5bench - INFO - Runtime: 292.1215675 seconds (elapsed time, includes allocation wait time)
2025-08-02 20:43:18,030 h5bench - INFO - h5bench [write] - Complete
2025-08-02 20:43:18,030 h5bench - INFO - JOBID: 6930444.aurora-pbs-0001.hostmgmt.cm.aurora.alcf.anl.gov
2025-08-02 20:43:18,030 h5bench - INFO - h5bench [read] - Starting
2025-08-02 20:43:18,030 h5bench - INFO - h5bench [read] - DIR: /flare/ATPESC2025/usr/jlbez/h5bench-storage/44e89366-6930444.aurora-pbs-0001.hostmgmt.cm.aurora.alcf.anl.gov/
2025-08-02 20:43:18,032 h5bench - INFO - Parallel setup: mpirun -n 408 --ppn 102 --cpu-bind core
2025-08-02 20:43:18,046 h5bench - INFO - mpirun -n 408 --ppn 102 --cpu-bind core /home/jlbez/h5bench/install/bin//h5bench_read
/flare/ATPESC2025/usr/jlbez/44e89366-6930444.aurora-pbs-0001.hostmgmt.cm.aurora.alcf.anl.gov/h5bench.cfg /flare/ATPESC2025/usr/jlbez/h5bench-storage/h5bench.h5
2025-08-02 20:45:53,467 h5bench - INFO - SUCCESS (all output files are located at /flare/ATPESC2025/usr/jlbez/h5bench-storage/44e89366-6930444.aurora-pbs-0001.hostmgmt.cm.aurora.alcf.anl.gov)
2025-08-02 20:45:53,478 h5bench - INFO - Requested and ran in SYNC mode
2025-08-02 20:45:53,478 h5bench - INFO - Runtime: 155.4389443 seconds (elapsed time, includes allocation wait time)
2025-08-02 20:45:53,478 h5bench - INFO - h5bench [read] - Complete
2025-08-02 20:45:53,478 h5bench - INFO - Finishing h5bench Suite
```

```
Configuration file: /flare/ATPESC2025/usr/jlbez/61f2bed5-6930444.aurora-pbs-0001.hostmgmt.cm.aurora.alcf.anl.gov/h5bench.cfg
Output data file: /flare/ATPESC2025/usr/jlbez/h5bench.h5

================ Benchmark Configuration ==================
File: /flare/ATPESC2025/usr/jlbez/61f2bed5-6930444.aurora-pbs-0001.hostmgmt.cm.aurora.alcf.anl.gov/h5bench.cfg
Number of particles per rank: 1024
Number of time steps: 5
Emulated compute time per timestep: 5
Mode: SYNC
Collective metadata operations: YES
Collective buffering for data operations: YES
Number of dimensions: 1
    Dim_1: 1024
Standard deviation for varying particle size in normal distribution = 1
===========================================================

Start benchmark: h5bench_write
Number of particles per rank: 0 M
Total number of particles: 0M
Collective Metadata operations: ON
Opened HDF5 file...
Writing Timestep_0 ...
    data_write_contig_contig_MD_array: Finished writing time step
Computing...
Writing Timestep_1 ...
    data_write_contig_contig_MD_array: Finished writing time step

...

Computing...
Writing Timestep_4 ...
    data_write_contig_contig_MD_array: Finished writing time step

================== Performance Results ==================
Total number of ranks: 408
Total emulated compute time: 20.000 s
Total write size: 63.750 MB
Raw write time: 1.030 s
Metadata time: 0.001 s
H5Fcreate() time: 0.522 s
H5Fflush() time: 0.032 s
H5Fclose() time: 0.003 s
Observed completion time: 21.680 s
SYNC Raw write rate: 61.880 MB/s
SYNC Observed write rate: 37.941 MB/s
===========================================================
```

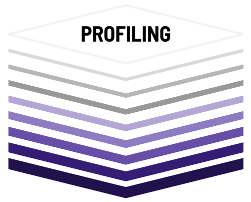
GOING BACK TO THE PROBLEM...

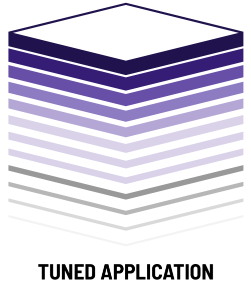
- There is still a **gap** between profiling and tuning
- How to convert I/O metrics to meaningful information?
  - **Visualize** characteristics, behavior, and bottlenecks
  - **Detect** root causes of I/O bottlenecks
  - **Map** I/O bottlenecks into actionable items
  - **Guide** end-user to tune I/O performance



GOING BACK TO THE PROBLEM...

- There is still a **gap** between profiling and tuning
- How to convert I/O metrics to meaningful information?
  - **Visualize** characteristics, behavior, and bottlenecks
  - **Detect** root causes of I/O bottlenecks
  - **Map** I/O bottlenecks into actionable items
  - **Guide** end-user to tune I/O performance





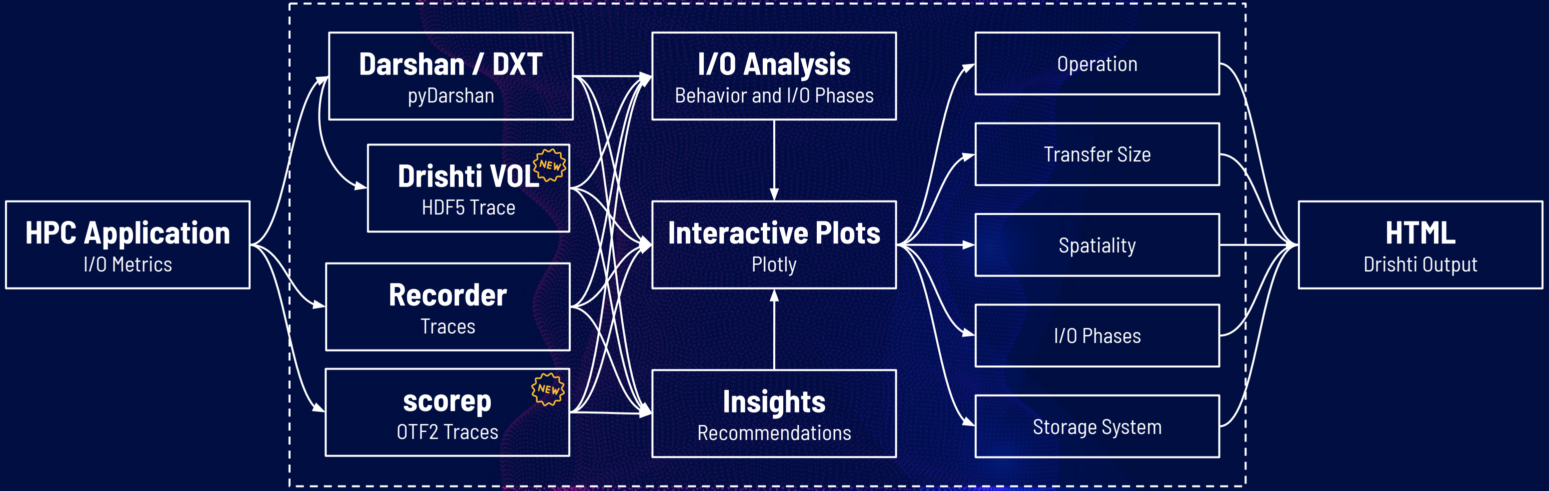
TOWARDS A SOLUTION...

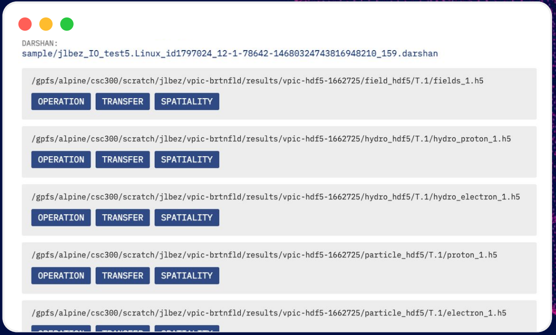
- Sanskrit word meaning "**point of focus**"
  - **Interactive** web based analysis framework
  - Pinpoint root **causes** of I/O performance problems
  - **Detects** typical I/O performance pitfalls
  - Provide a set of **actionable recommendations**
- Working to support multiple sources of I/O metrics

TOWARDS A SOLUTION...

- Sanskrit word meaning “**point of focus**”
  - **Interactive** web based analysis framework
  - Pinpoint root **causes** of I/O performance problems
  - **Detects** typical I/O performance pitfalls
  - Provide a set of **actionable recommendations**
- Working to support multiple sources of I/O metrics

Drishti


HPC Application
I/O Metrics
Darshan / DXT
pyDarshan
Drishti VOL
NEW
HDF5 Trace
Recorder
Traces
scorep
NEW
OTF2 Traces
I/O Analysis
Behavior and I/O Phases
Interactive Plots
Plotly
Insights
Recommendations
Operation
Transfer Size
Spatiality
I/O Phases
Storage System
HTML
Drishti Output



DARSHAN:
sample/jlbez_IO_test5.Linux_id1797024_12-1-78642-14680324743816948210_159.darshan

/gpfs/alpine/csc300/scratch/jlbez/vpic-brtnfld/results/vpic-hdf5-1662725/field_hdf5/T.1/fields_1.h5

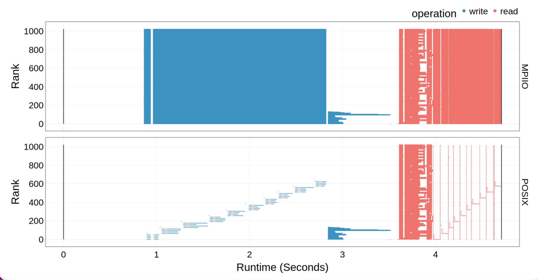
/gpfs/alpine/csc300/scratch/jlbez/vpic-brtnfld/results/vpic-hdf5-1662725/hydro_hdf5/T.1/hydro_proton_1.h5

/gpfs/alpine/csc300/scratch/jlbez/vpic-brtnfld/results/vpic-hdf5-1662725/hydro_hdf5/T.1/hydro_electron_1.h5

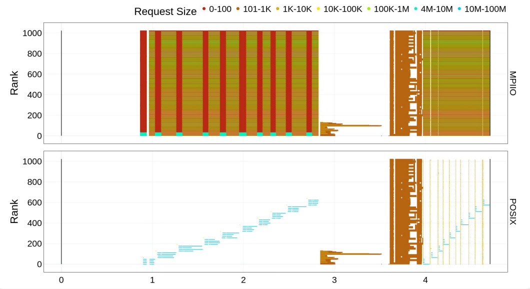
/gpfs/alpine/csc300/scratch/jlbez/vpic-brtnfld/results/vpic-hdf5-1662725/particle_hdf5/T.1/proton_1.h5

/gpfs/alpine/csc300/scratch/jlbez/vpic-brtnfld/results/vpic-hdf5-1662725/particle_hdf5/T.1/electron_1.h5

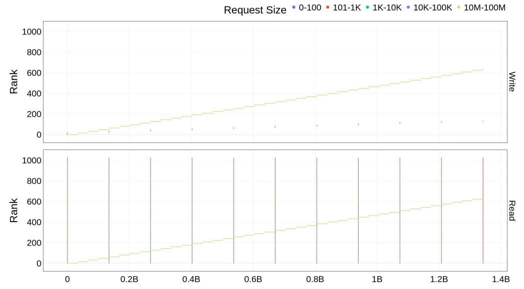
Operation



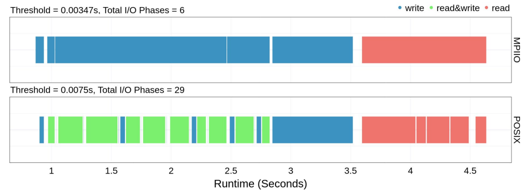
Transfer Size



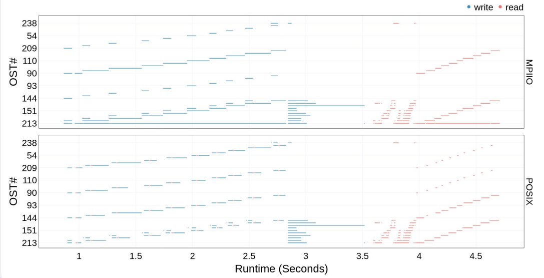
Spatial Locality

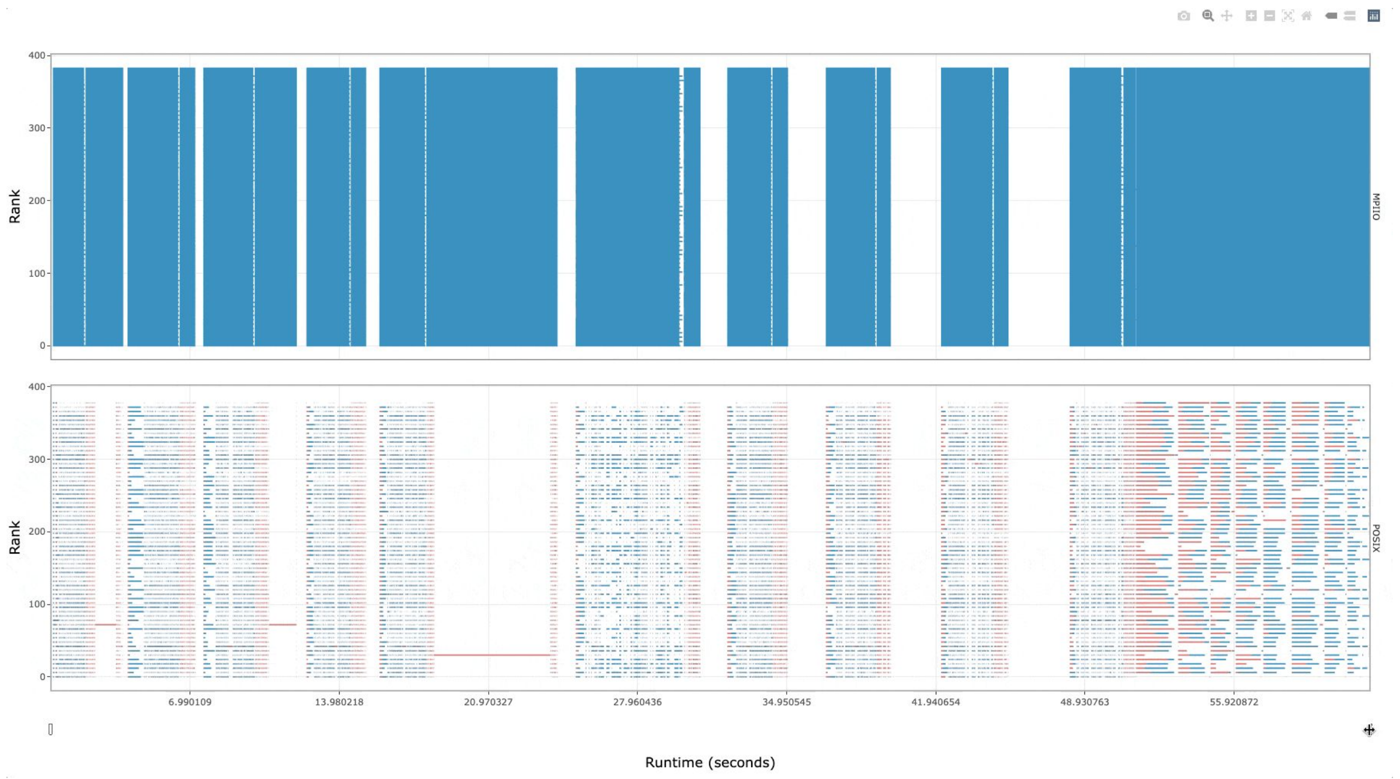


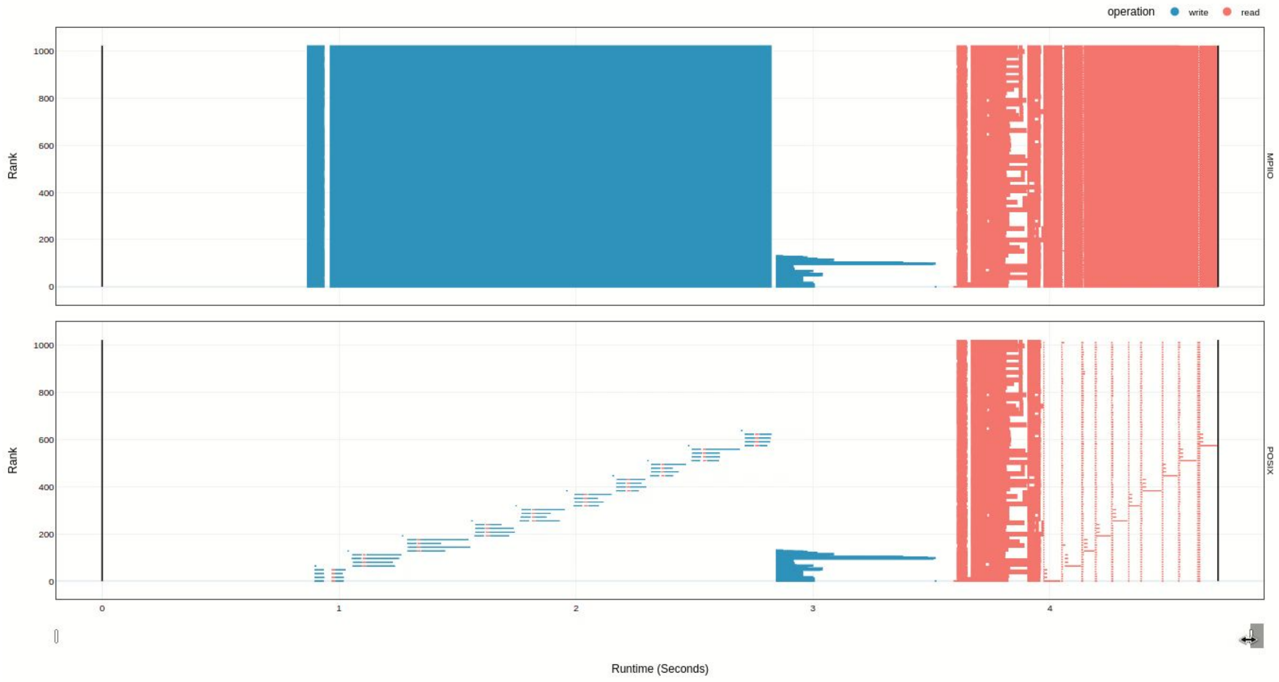
I/O Phases

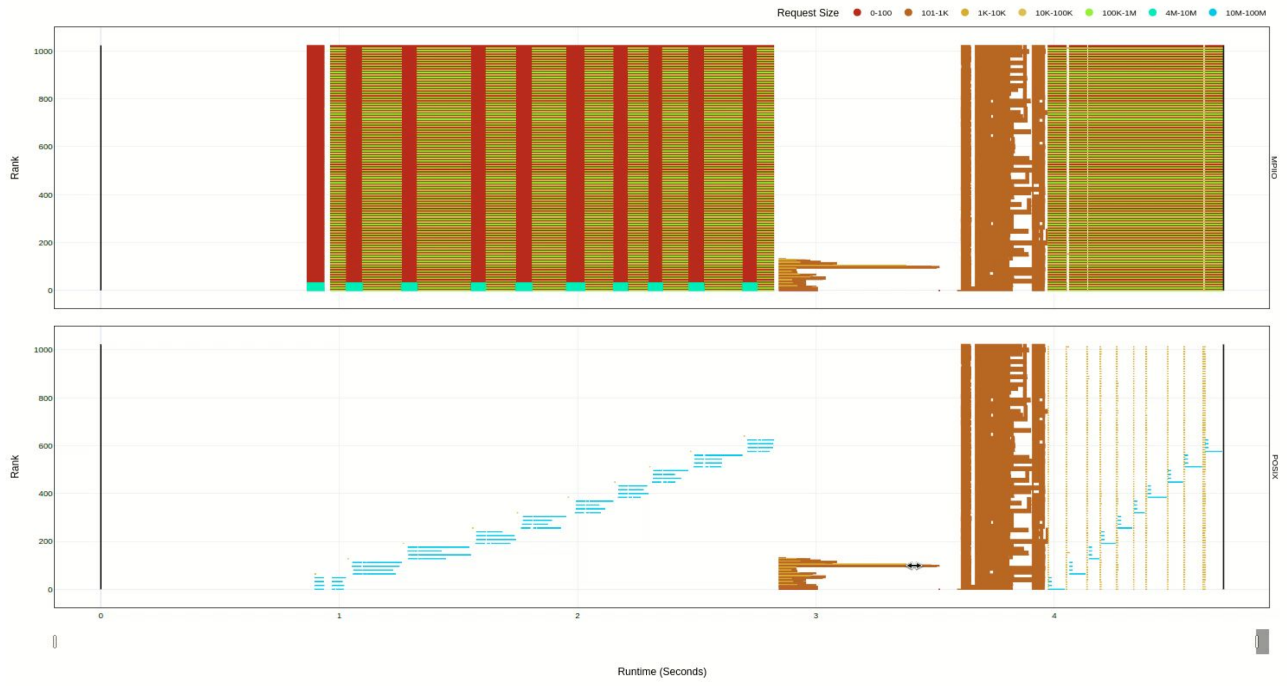


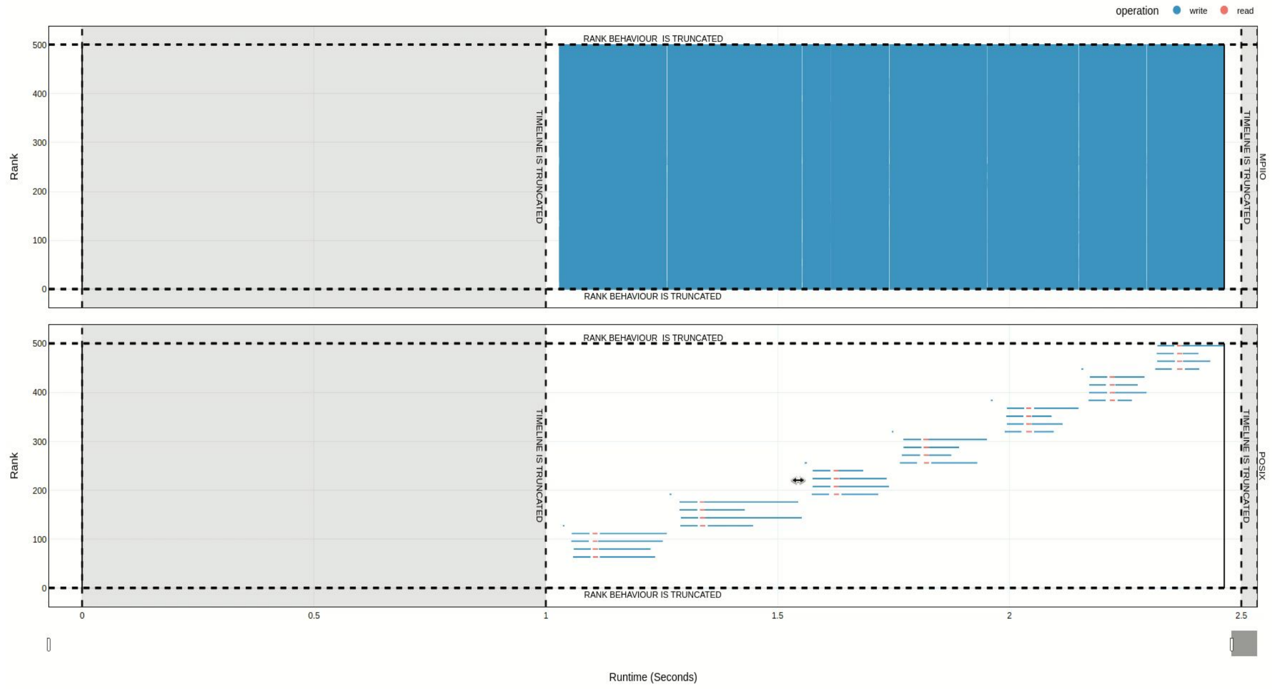
OST Usage




Rank
400
300
200
100
0
MPIIO
Rank
400
300
200
100
0
POSIX
6.990109
13.980218
20.970327
27.960436
34.950545
41.940654
48.930763
55.920872
Runtime (seconds)


operation
write
read
Rank
1000
800
600
400
200
0
MPIIO
POSIX
Runtime (Seconds)
0
1
2
3
4


Request Size
● 0-100
● 101-1K
● 1K-10K
● 10K-100K
● 100K-1M
● 4M-10M
● 10M-100M
Rank
Rank
MPIIO
POSIX
0
200
400
600
800
1000
0
1
2
3
4
Runtime (Seconds)


operation
write
read
RANK BEHAVIOUR IS TRUNCATED
TIMELINE IS TRUNCATED
MPIIO
POSIX
Rank
500
400
300
200
100
0
0
0.5
1
1.5
2
2.5
Runtime (Seconds)

HANDS-ON

DOCKER OR NERSC

```
# Download some files for the hands-on exercise
$ wget https://github.com/jeanbez/dxt-sample-logs/raw/main/samples-openpmd.tar.gz
$ tar zxvf samples-openpmd.tar.gz

# On NERSC systems you can also use the container version with Shifter
$ shifter --image=docker:hpcio/dxt-explorer:pre-release

# Download the files for local interactive exploration on your browser!
```

DRISHTI TRIGGERS

HEURISTIC-BASED

Level	Description
HIGH	High probability of harming I/O performance.
WARN	Detected issues that could cause a significant negative impact on the I/O performance. The confidence of these recommendations is low as available metrics might not be sufficient to detect application design, configuration, or execution choices.
OK	Best practices have been followed.
INFO	Relevant information regarding application configuration.

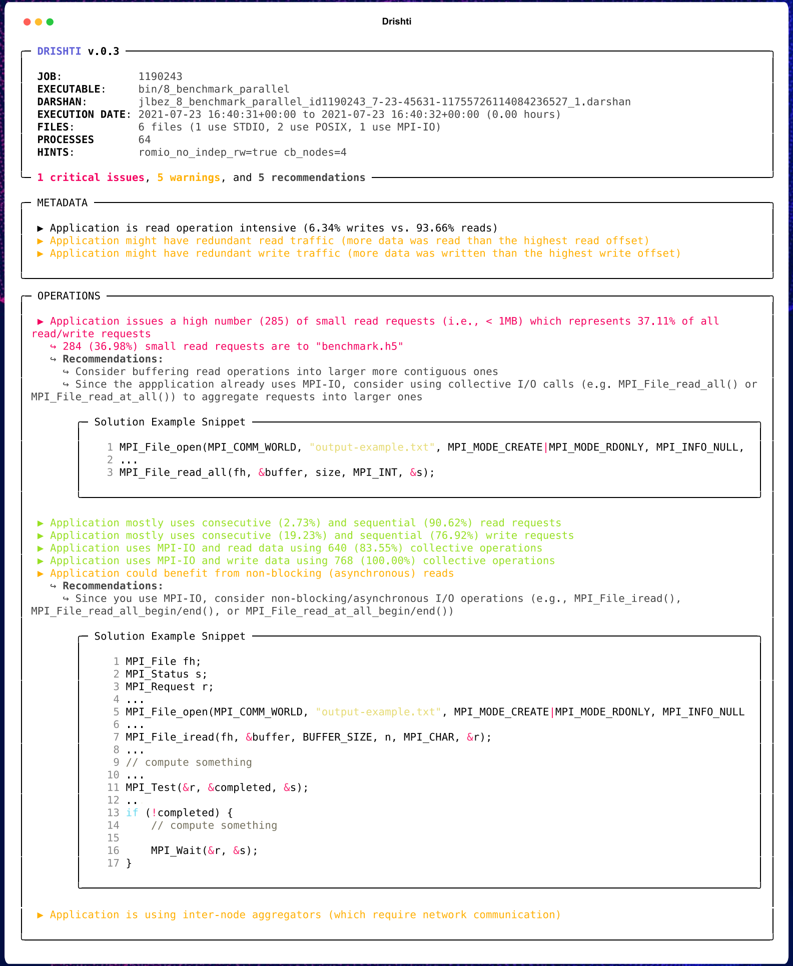
Overall **information** about the Darshan log and execution

Number of **critical issues**, warning, and **recommendations**

Drishti checks metrics for **over 30 triggers**

Highlight the **file** that triggered the issue

Sample **code solutions** are provided



Drishti

```
┌ DRISHTI v.0.3 ─────────────────────────────────────────────────────────────────────────────────
│
│ JOB:            1190243
│ EXECUTABLE:     bin/8_benchmark_parallel
│ DARSHAN:        jlbez_8_benchmark_parallel_id1190243_7-23-45631-11755726114084236527_1.darshan
│ EXECUTION DATE: 2021-07-23 16:40:31+00:00 to 2021-07-23 16:40:32+00:00 (0.00 hours)
│ FILES:          6 files (1 use STDIO, 2 use POSIX, 1 use MPI-IO)
│ PROCESSES       64
│ HINTS:          romio_no_indep_rw=true cb_nodes=4
│
└ 1 critical issues, 5 warnings, and 5 recommendations ──────────────────────────────────────────

┌ METADATA ──────────────────────────────────────────────────────────────────────────────────────
│
│ ▶ Application is read operation intensive (6.34% writes vs. 93.66% reads)
│ ▶ Application might have redundant read traffic (more data was read than the highest read offset)
│ ▶ Application might have redundant write traffic (more data was written than the highest write offset)
│
└────────────────────────────────────────────────────────────────────────────────────────────────

┌ OPERATIONS ────────────────────────────────────────────────────────────────────────────────────
│
│ ▶ Application issues a high number (285) of small read requests (i.e., < 1MB) which represents 37.11% of all
│read/write requests
│   ↪ 284 (36.98%) small read requests are to "benchmark.h5"
│   ↪ Recommendations:
│     ↪ Consider buffering read operations into larger more contiguous ones
│     ↪ Since the appplication already uses MPI-IO, consider using collective I/O calls (e.g. MPI_File_read_all() or
│MPI_File_read_at_all()) to aggregate requests into larger ones
│
│         ┌ Solution Example Snippet ───────────────────────────────────────────────────────────────
│         │
│         │   MPI_File_open(MPI_COMM_WORLD, "output-example.txt", MPI_MODE_CREATE|MPI_MODE_RDONLY, MPI_INFO_NULL,
│         │   ...
│         │   MPI_File_read_all(fh, &buffer, size, MPI_INT, &s);
│         │
│         └─────────────────────────────────────────────────────────────────────────────────────────
│
│ ▶ Application mostly uses consecutive (2.73%) and sequential (90.62%) read requests
│ ▶ Application mostly uses consecutive (19.23%) and sequential (76.92%) write requests
│ ▶ Application uses MPI-IO and read data using 640 (83.55%) collective operations
│ ▶ Application uses MPI-IO and write data using 768 (100.00%) collective operations
│ ▶ Application could benefit from non-blocking (asynchronous) reads
│   ↪ Recommendations:
│     ↪ Since you use MPI-IO, consider non-blocking/asynchronous I/O operations (e.g., MPI_File_iread(),
│MPI_File_read_all_begin/end(), or MPI_File_read_at_all_begin/end())
│
│         ┌ Solution Example Snippet ───────────────────────────────────────────────────────────────
│         │
│         │   MPI_File fh;
│         │   MPI_Status s;
│         │   MPI_Request r;
│         │   ...
│         │   MPI_File_open(MPI_COMM_WORLD, "output-example.txt", MPI_MODE_CREATE|MPI_MODE_RDONLY, MPI_INFO_NULL
│         │   ...
│         │   MPI_File_iread(fh, &buffer, BUFFER_SIZE, n, MPI_CHAR, &r);
│         │   ...
│         │   // compute something
│         │   ...
│         │   MPI_Test(&r, &completed, &s);
│         │   ..
│         │   if (!completed) {
│         │       // compute something
│         │
│         │       MPI_Wait(&r, &s);
│         │   }
│         │
│         └─────────────────────────────────────────────────────────────────────────────────────────
│
│ ▶ Application is using inter-node aggregators (which require network communication)
│
└────────────────────────────────────────────────────────────────────────────────────────────────
```

Current version only checks **profiling** metrics

Severity based on certainty and impact: **high**, **medium**, **low**, info

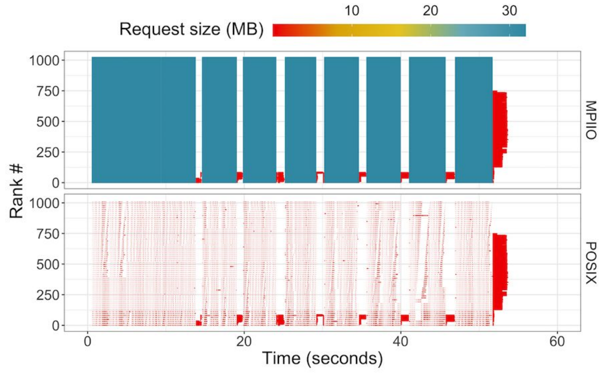
Provides **actionable feedback** for users

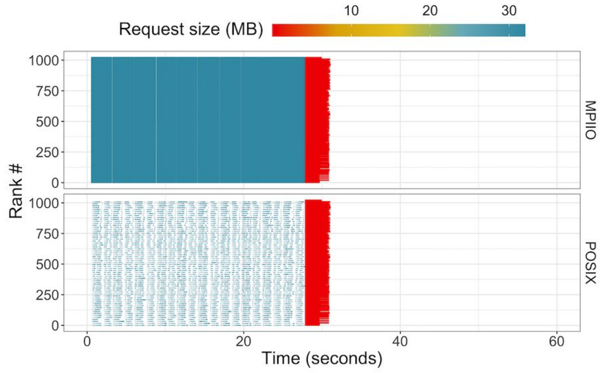
Drishti can check for HDF5 usage to fine **tune the recommendations**

Multiple output **formats**: textual, SVG, HTML

WARPX / OPENPMD

USE CASE



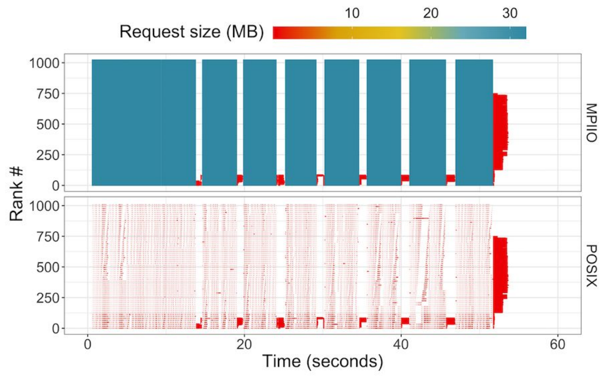


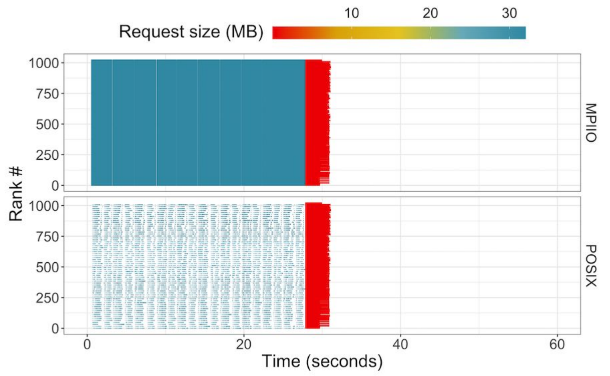
```
╭─ METADATA ─────────────────────────────────────────────────────────────────────

 ▶ Application is write operation intensive (60.83% writes vs. 39.17% reads)
 ▶ Application is write size intensive (64 15% write vs  35 85% read)
 ▶ Application issues a high number (100.00%) of misaligned file requests

╰────────────────────────────────────────────────────────────────────────────────
╭─ OPERATIONS ───────────────────────────────────────────────────────────────────

 ▶ Application issues a high number (275840) of small read requests (i.e., < 1MB) which
represents 100.00% of all read/write requests
   ↪ 275840 (100.00%) small read requests are to "8a_parallel_3Db_0000001.h5"
 ▶ Application issues a high number (427386) of small write requests (i.e., < 1MB) which
represents 99.75% of all read/write requests
   ↪ 275840 (64.38%) small write requests are to "8a_parallel_3Db_0000001.h5"
 ▶ Application mostly uses consecutive (97.67%) and sequential (2.16%) read requests
 ▶ Application mostly uses consecutive (97.85%) and sequential (1.17%) write requests
 ▶ Application uses MPI-IO and write data using 7680 (92.50%) collective operations
 ▶ Application could benefit from non-blocking (asynchronous) reads
 ▶ Application could benefit from non blocking (asynchronous) writes

╰────────────────────────────────────────────────────────────────────────────────
```

```
╭─ METADATA ─────────────────────────────────────────────────────────────────────

 ▶ Application is write operation intensive (90.85% writes vs. 9.15% reads)
 ▶ Application is write size intensive (91 14% write vs  8 86% read)
 ▶ Application might have redundant read traffic (more data read than the highest offset)

╰────────────────────────────────────────────────────────────────────────────────
╭─ OPERATIONS ───────────────────────────────────────────────────────────────────

 ▶ Application is issuing a high number (565) of random read operations (35.25%)
 ▶ Application mostly uses consecutive (88.56%) and sequential (7.02%) write requests
 ▶ Application uses MPI-IO and write data using 8448 (100.00%) collective operations
 ▶ Application could benefit from non-blocking (asynchronous) reads
 ▶ Application could benefit from non-blocking (asynchronous) writes

╰────────────────────────────────────────────────────────────────────────────────
```

WARPX / OPENPMD

USE CASE





```
METADATA

▶ Application is write operation intensive (60.83% writes vs. 39.17% reads)
▶ Application is write size intensive (64 15% write vs  35 85% read)
▶ Application issues a high number (100.00%) of misaligned file requests
```

```
OPERATIONS

▶ Application issues a high number (275840) of small read requests (i.e., < 1MB) which
represents 100.00% of all read/write requests
   ↪ 275840 (100.00%) small read requests are to "8a_parallel_3Db_0000001.h5"
▶ Application issues a high number (427386) of small write requests (i.e., < 1MB) which
represents 99.75% of all read/write requests
   ↪ 275840 (64.38%) small write requests are to "8a_parallel_3Db_0000001.h5"
▶ Application mostly uses consecutive (97.67%) and sequential (2.16%) read requests
▶ Application mostly uses consecutive (97.85%) and sequential (1.17%) write requests
▶ Application uses MPI-IO and write data using 7680 (92.50%) collective operations
▶ Application could benefit from non-blocking (asynchronous) reads
▶ Application could benefit from non blocking (asynchronous) writes
```

```
METADATA

▶ Application is write operation intensive (90.85% writes vs. 9.15% reads)
▶ Application is write size intensive (91 14% write vs  8 86% read)
▶ Application might have redundant read traffic (more data read than the highest offset)
```

```
OPERATIONS

▶ Application is issuing a high number (565) of random read operations (35.25%)
▶ Application mostly uses consecutive (88.56%) and sequential (7.02%) write requests
▶ Application uses MPI-IO and write data using 8448 (100.00%) collective operations
▶ Application could benefit from non-blocking (asynchronous) reads
▶ Application could benefit from non-blocking (asynchronous) writes
```

AMREX

USE CASE

2.1× SPEEDUP

from 211 to 100 seconds

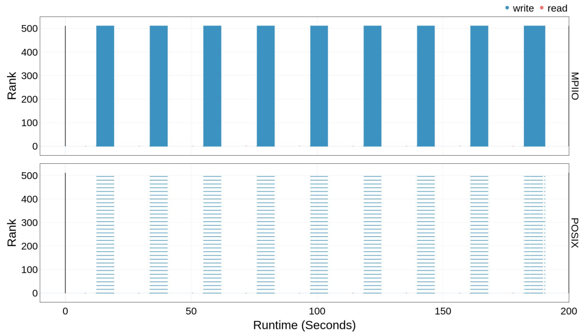
SETUP

512 ranks (32 nodes)

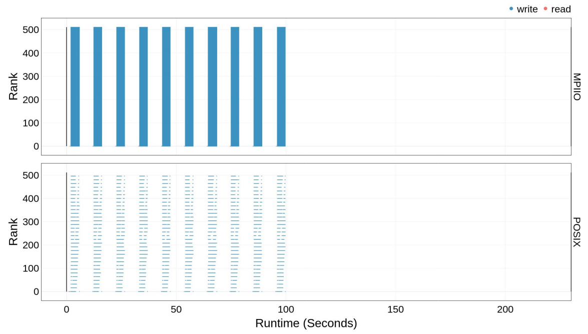
1024 domain size

1 level, 6 components, 2 particles per cell

10 output plot files



```
METADATA ───────────────────────────────────────────────────────────

 ▸ Application is write operation intensive (99.98% writes vs. 0.02% reads)
 ▸ Application is write size intensive (100.00% write vs. 0.00% read)

OPERATIONS ─────────────────────────────────────────────────────────

 ▸ Application issues a high number (491640) of small write requests (i.e., < 1MB) which
represents 99.99% of all read/write requests
   ↪ 98328 (20.00%) small write requests are to "plt00001.h5"
   ↪ 98328 (20.00%) small write requests are to "plt00002.h5"
   ↪ 98328 (20.00%) small write requests are to "plt00005.h5"
   ↪ 98328 (20.00%) small write requests are to "plt00009.h5"
   ↪ 98328 (20.00%) small write requests are to "plt00000.h5"
   ↪ 98328 (20.00%) small write requests are to "plt00004.h5"
   ↪ 98328 (20.00%) small write requests are to "plt00003.h5"
   ↪ 98328 (20.00%) small write requests are to "plt00006.h5"
   ↪ 98328 (20.00%) small write requests are to "plt00007.h5"
   ↪ 98328 (20.00%) small write requests are to "plt00008.h5"
   ↪ Recommendations:
  ↪ Consider buffering write operations into larger more contiguous ones
  ↪ Since the application already uses MPI-IO, consider using collective I/O calls (e.g.
MPI_File_write_all() or MPI_File_write_at_all()) to aggregate requests into larger ones
 ▸ Application mostly uses consecutive (25.41%) and sequential (32.79%) read requests
 ▸ Application mostly uses consecutive (0.01%) and sequential (99.98%) write requests
 ▸ Application issues a high number (491640) of small write requests to a shared file (i.e., <
1MB) which represents 99.99% of all shared file write requests
   ↪ 49164 (10.00%) small writes requests are to "plt00001.h5"
   ↪ 49164 (10.00%) small writes requests are to "plt00002.h5"
   ↪ 49164 (10.00%) small writes requests are to "plt00005.h5"
   ↪ 49164 (10.00%) small writes requests are to "plt00009.h5"
   ↪ 49164 (10.00%) small writes requests are to "plt00000.h5"
   ↪ 49164 (10.00%) small writes requests are to "plt00004.h5"
   ↪ 49164 (10.00%) small writes requests are to "plt00003.h5"
   ↪ 49164 (10.00%) small writes requests are to "plt00006.h5"
   ↪ 49164 (10.00%) small writes requests are to "plt00007.h5"
   ↪ 49164 (10.00%) small writes requests are to "plt00008.h5"
   ↪ Recommendations:
  ↪ Consider coalescing write requests into larger more contiguous ones using MPI-IO
collective operations
 ▸ Application uses MPI-IO and write data using 15360 (99.81%) collective operations
 ▸ Application could benefit from non-blocking (asynchronous) reads
   ↪ Recommendations:
  ↪ Since you use HDF5, consider using the ASYNC I/O VOL connector
(https://github.com/hpc-io/vol-async)
  ↪ Since you use MPI-IO, consider non-blocking/asynchronous I/O operations
 ▸ Application could benefit from non-blocking (asynchronous) writes
   ↪ Recommendations:
  ↪ Since you use HDF5, consider using the ASYNC I/O VOL connector
(https://github.com/hpc-io/vol-async)
  ↪ Since you use MPI-IO, consider non-blocking/asynchronous I/O operations
```



```
METADATA ───────────────────────────────────────────────────────────

▶Application is write operation intensive (99.61% writes vs. 0.39% reads)
▶Application is write size intensive (100.00% write vs. 0.00% read)

OPERATIONS ─────────────────────────────────────────────────────────

▶Application mostly uses consecutive (24.79%) and sequential (33.06%) read requests
▶Application mostly uses consecutive (0.16%) and sequential (99.64%) write requests
▶Application uses MPI-IO and write data using 15360 (99.81%) collective operations
```

SYSTEM REPORT

I/O ISSUES OVERVIEW

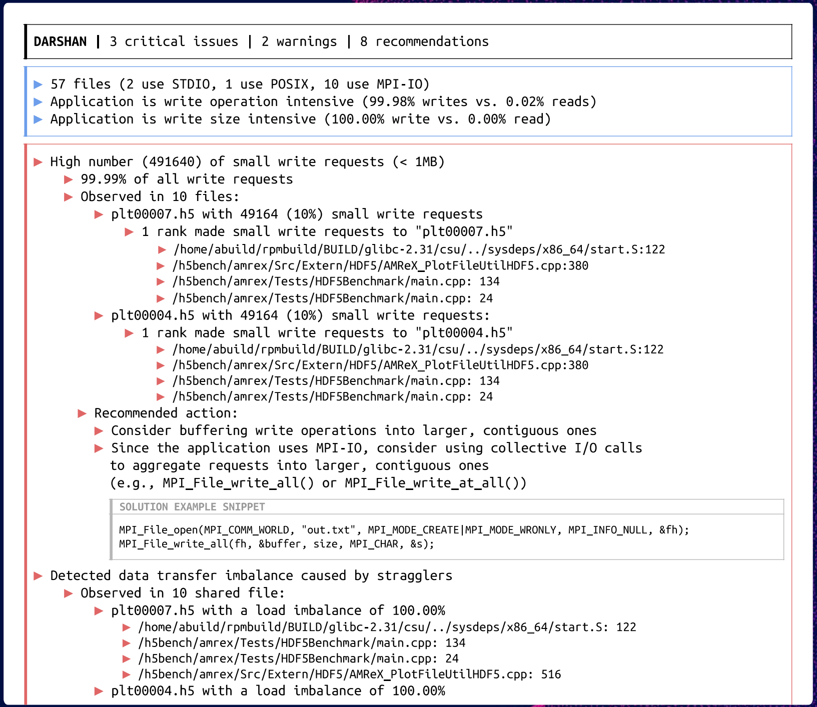
Level	Interface	Detected Behavior	Jobs	Total (%)	Relative* (%)
HIGH	STDIO	High STDIO usage (>10% of total transfer size uses STDIO)	43,120	38.29	52.1
OK	POSIX	High number of sequential read operations (≥ 80%)	38,104	33.84	58.14
OK	POSIX	High number of sequential write operations (≥ 80%)	64,486	57.26	98.39
INFO	POSIX	Write operation count intensive (>10% more writes than reads)	26,114	23.19	39.84
INFO	POSIX	Read operation count intensive (>10% more reads than writes)	23,168	20.57	35.35
INFO	POSIX	Write size intensive (>10% more bytes written then read)	23,568	20.93	35.96
INFO	POSIX	Read size intensive (>10% more bytes read then written)	40,950	36.36	62.48
WARN	POSIX	Redundant reads	14,518	12.89	22.15
WARN	POSIX	Redundant writes	59	0.05	0.09
HIGH	POSIX	High number of small (<1MB) read requests (>10% of total read requests)	64,858	57.59	98.96
HIGH	POSIX	High number of small (<1MB) write requests (>10% of total write requests)	64,552	57.32	98.49
HIGH	POSIX	High number of misaligned memory requests (>10%)	36,337	32.27	55.44
HIGH	POSIX	High number of misaligned file requests (>10%)	65,075	57.79	99.29
HIGH	POSIX	High number of random read requests (>20%)	26,574	23.6	40.54
HIGH	POSIX	High number of random write requests (>20%)	559	0.5	0.85
HIGH	POSIX	High number of small (<1MB) reads to shared-files (>10% of total reads)	60,121	53.39	91.73
HIGH	POSIX	High number of small (<1MB) writes to shared-files (>10% of total writes)	55,414	49.21	84.55
HIGH	POSIX	High metadata time (at least one rank spends >30 seconds)	9,410	8.36	14.35
HIGH	POSIX	Data transfer imbalance between ranks causing stragglers (>15% difference)	40,601	36.05	61.95
HIGH	POSIX	Time imbalance between ranks causing stragglers (>15% difference)	40,533	35.99	61.84

CROSS LAYER EXPLORATION

SOURCE CODE

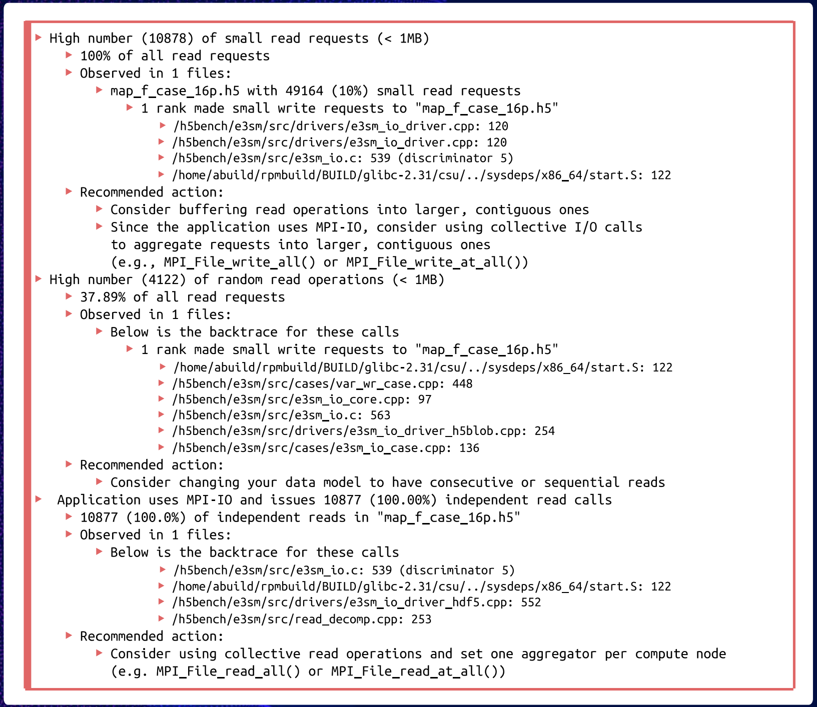
NEW

AMREX



```
DARSHAN | 3 critical issues | 2 warnings | 8 recommendations

▶ 57 files (2 use STDIO, 1 use POSIX, 10 use MPI-IO)
▶ Application is write operation intensive (99.98% writes vs. 0.02% reads)
▶ Application is write size intensive (100.00% write vs. 0.00% read)

▶ High number (491640) of small write requests (< 1MB)
    ▶ 99.99% of all write requests
    ▶ Observed in 10 files:
        ▶ plt00007.h5 with 49164 (10%) small write requests
            ▶ 1 rank made small write requests to "plt00007.h5"
                ▶ /home/abuild/rpmbuild/BUILD/glibc-2.31/csu/../sysdeps/x86_64/start.S:122
                ▶ /h5bench/amrex/Src/Extern/HDF5/AMReX_PlotFileUtilHDF5.cpp:380
                ▶ /h5bench/amrex/Tests/HDF5Benchmark/main.cpp: 134
                ▶ /h5bench/amrex/Tests/HDF5Benchmark/main.cpp: 24
        ▶ plt00004.h5 with 49164 (10%) small write requests:
            ▶ 1 rank made small write requests to "plt00004.h5"
                ▶ /home/abuild/rpmbuild/BUILD/glibc-2.31/csu/../sysdeps/x86_64/start.S:122
                ▶ /h5bench/amrex/Src/Extern/HDF5/AMReX_PlotFileUtilHDF5.cpp:380
                ▶ /h5bench/amrex/Tests/HDF5Benchmark/main.cpp: 134
                ▶ /h5bench/amrex/Tests/HDF5Benchmark/main.cpp: 24
      ▶ Recommended action:
        ▶ Consider buffering write operations into larger, contiguous ones
        ▶ Since the application uses MPI-IO, consider using collective I/O calls
          to aggregate requests into larger, contiguous ones
          (e.g., MPI_File_write_all() or MPI_File_write_at_all())

          SOLUTION EXAMPLE SNIPPET

          MPI_File_open(MPI_COMM_WORLD, "out.txt", MPI_MODE_CREATE|MPI_MODE_WRONLY, MPI_INFO_NULL, &fh);
          MPI_File_write_all(fh, &buffer, size, MPI_CHAR, &s);

▶ Detected data transfer imbalance caused by stragglers
    ▶ Observed in 10 shared file:
        ▶ plt00007.h5 with a load imbalance of 100.00%
            ▶ /home/abuild/rpmbuild/BUILD/glibc-2.31/csu/../sysdeps/x86_64/start.S: 122
            ▶ /h5bench/amrex/Tests/HDF5Benchmark/main.cpp: 134
            ▶ /h5bench/amrex/Tests/HDF5Benchmark/main.cpp: 24
            ▶ /h5bench/amrex/Src/Extern/HDF5/AMReX_PlotFileUtilHDF5.cpp: 516
        ▶ plt00004.h5 with a load imbalance of 100.00%
```

E3SM

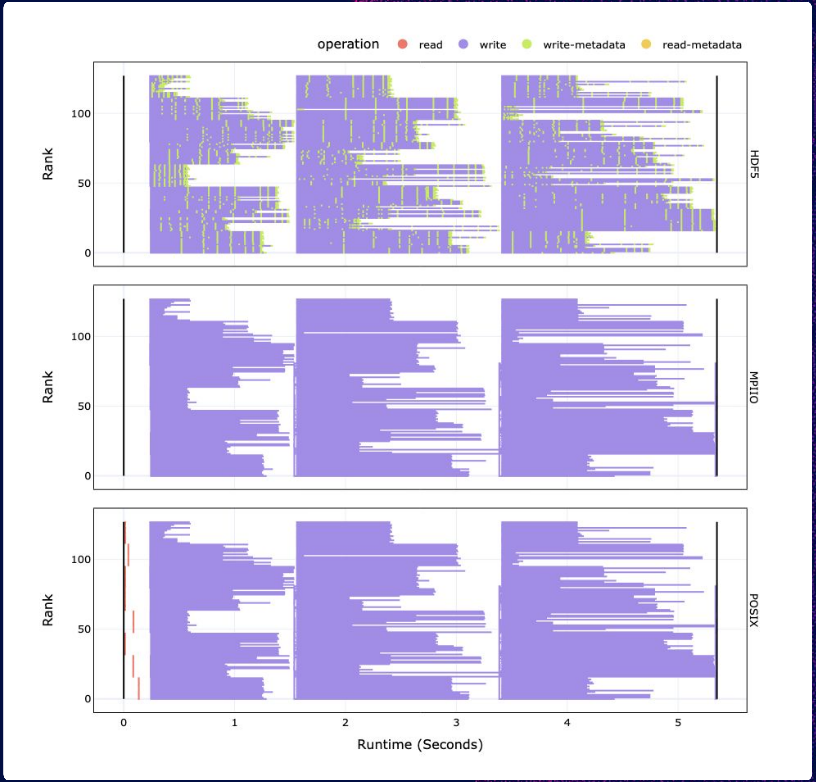


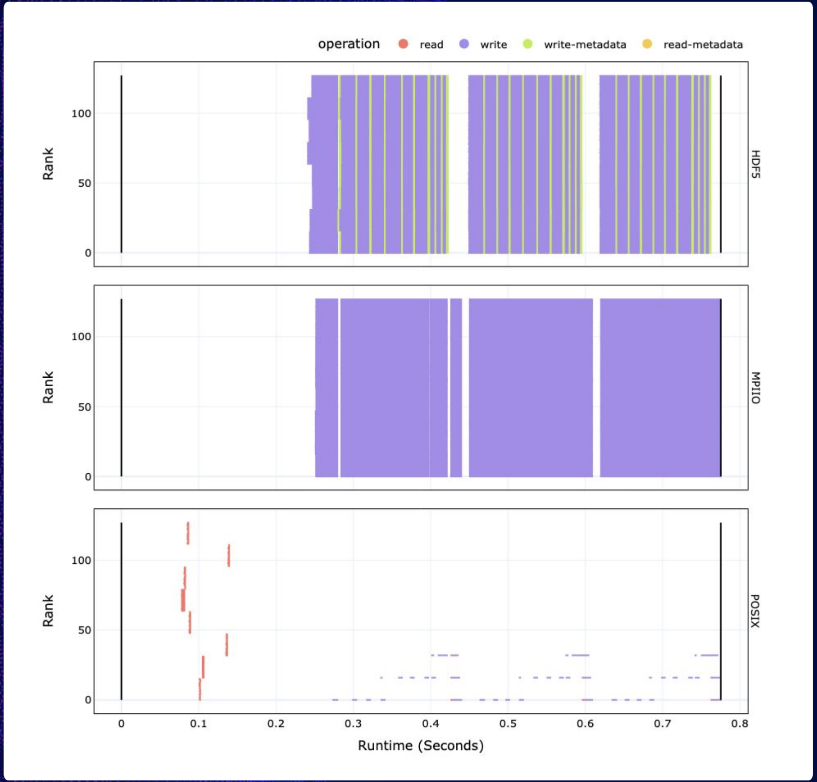
```
▶ High number (10878) of small read requests (< 1MB)
    ▶ 100% of all read requests
    ▶ Observed in 1 files:
        ▶ map_f_case_16p.h5 with 49164 (10%) small read requests
            ▶ 1 rank made small write requests to "map_f_case_16p.h5"
                ▶ /h5bench/e3sm/src/drivers/e3sm_io_driver.cpp: 120
                ▶ /h5bench/e3sm/src/drivers/e3sm_io_driver.cpp: 120
                ▶ /h5bench/e3sm/src/e3sm_io.c: 539 (discriminator 5)
                ▶ /home/abuild/rpmbuild/BUILD/glibc-2.31/csu/../sysdeps/x86_64/start.S: 122
    ▶ Recommended action:
        ▶ Consider buffering read operations into larger, contiguous ones
        ▶ Since the application uses MPI-IO, consider using collective I/O calls
          to aggregate requests into larger, contiguous ones
          (e.g., MPI_File_write_all() or MPI_File_write_at_all())
▶ High number (4122) of random read operations (< 1MB)
    ▶ 37.89% of all read requests
    ▶ Observed in 1 files:
        ▶ Below is the backtrace for these calls
            ▶ 1 rank made small write requests to "map_f_case_16p.h5"
                ▶ /home/abuild/rpmbuild/BUILD/glibc-2.31/csu/../sysdeps/x86_64/start.S: 122
                ▶ /h5bench/e3sm/src/cases/var_wr_case.cpp: 448
                ▶ /h5bench/e3sm/src/e3sm_io_core.cpp: 97
                ▶ /h5bench/e3sm/src/e3sm_io.c: 563
                ▶ /h5bench/e3sm/src/drivers/e3sm_io_driver_h5blob.cpp: 254
                ▶ /h5bench/e3sm/src/cases/e3sm_io_case.cpp: 136
    ▶ Recommended action:
        ▶ Consider changing your data model to have consecutive or sequential reads
▶  Application uses MPI-IO and issues 10877 (100.00%) independent read calls
    ▶ 10877 (100.0%) of independent reads in "map_f_case_16p.h5"
    ▶ Observed in 1 files:
        ▶ Below is the backtrace for these calls
                ▶ /h5bench/e3sm/src/e3sm_io.c: 539 (discriminator 5)
                ▶ /home/abuild/rpmbuild/BUILD/glibc-2.31/csu/../sysdeps/x86_64/start.S: 122
                ▶ /h5bench/e3sm/src/drivers/e3sm_io_driver_hdf5.cpp: 552
                ▶ /h5bench/e3sm/src/read_decomp.cpp: 253
    ▶ Recommended action:
        ▶ Consider using collective read operations and set one aggregator per compute node
          (e.g. MPI_File_read_all() or MPI_File_read_at_all())
```

Hammad Ather (UO), Jean Luca Bez (LBNL), Yankun Xia (OSU), Suren Byna (OSU)

DRISHTI
HDF5 VOL CONNECTOR

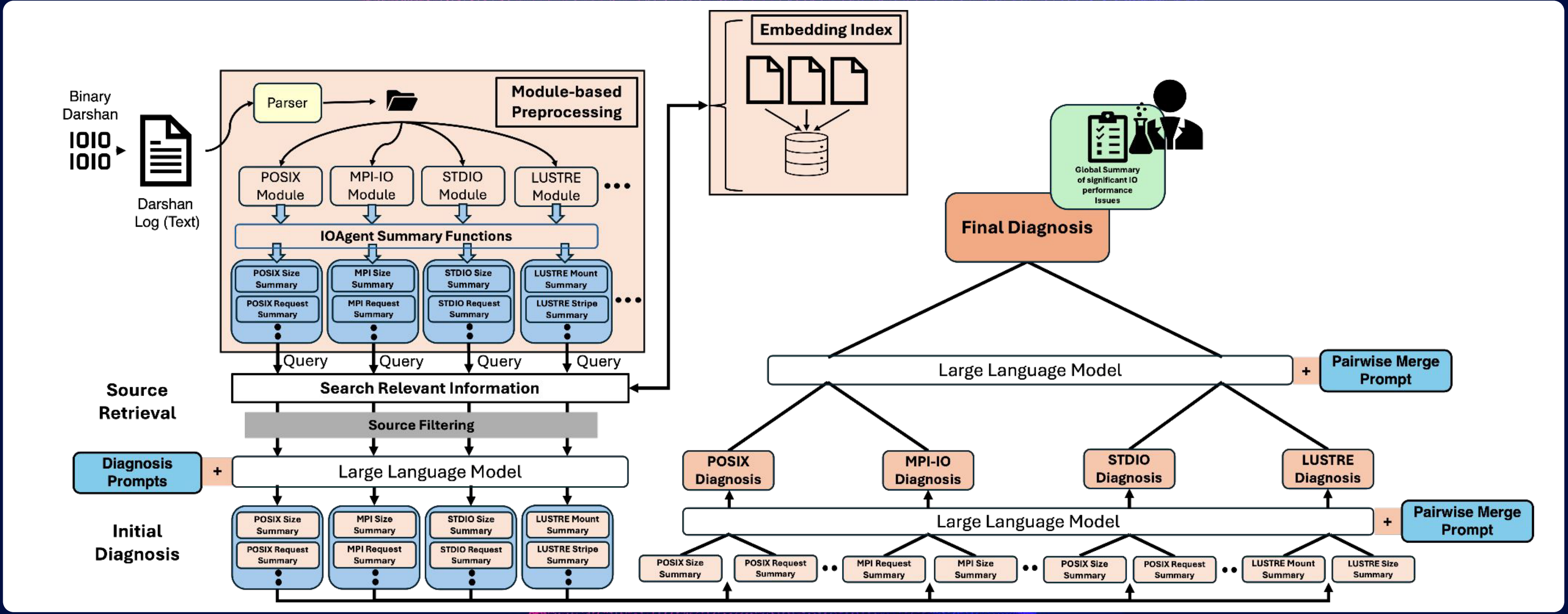
NEW





I/O NAVIGATOR

EXPLORING LLM-DRIVEN I/O PERFORMANCE DIAGNOSIS



Chris Egersdoerfer (UD), Arnav Sareen (UNCC), Jean Luca Bez (LBNL), Dongkuan Xu (NCSU), Suren Byna (OSU), Dong Dai (UD)

ARGONNE
ATPESC2025
EXTREME-SCALE COMPUTING

UNDERSTANDING AND TUNING HPC I/O PERFORMANCE

Jean Luca Bez
jlbez@lbl.gov
Lawrence Berkeley National Laboratory

BERKELEY LAB

Scientific Data Division