ISC High Performance

IMAGINE TOMORROW

MAY 21 – 25, 2023 | HAMBURG, GERMANY

Illuminating the I/O Optimization Path of Scientific Applications

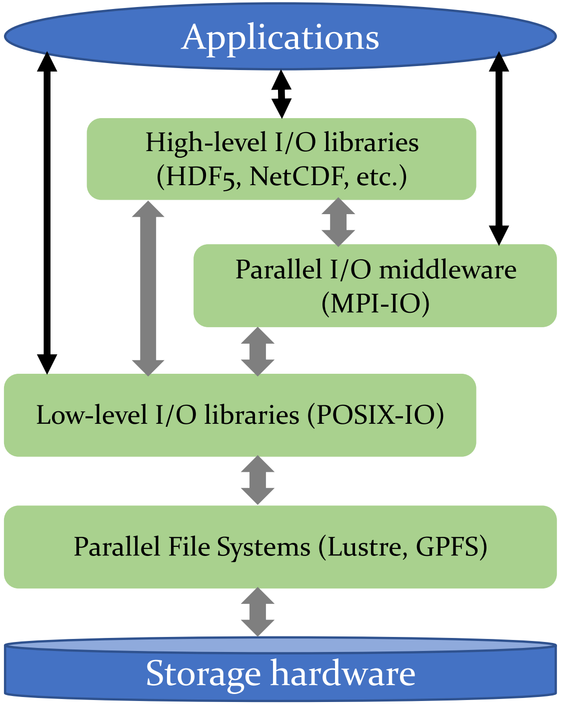
Hammad Ather, Jean Luca Bez, Boyana Norris, Suren Byna

ISC HPC 2023

BERKELEY LAB
THE OHIO STATE UNIVERSITY
UNIVERSITY OF OREGON
EXASCALE COMPUTING PROJECT

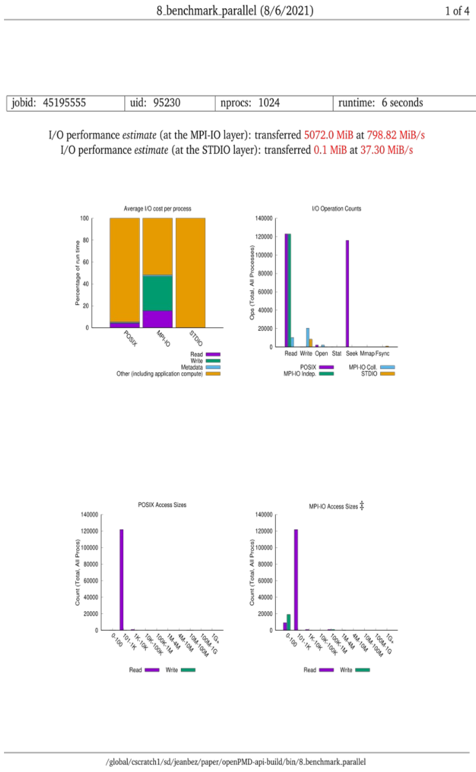
HPC I/O stack – Complex interdependencies among layers

- HPC I/O stack → **complex**
- Large **tuning parameter space**
- **I/O profiling tools** for understanding I/O performance



Understanding I/O performance – Darshan and DXT

- **Darshan** is a lightweight HPC I/O profiling tool
- **Darshan Extended Trace (DXT)**
  - **Fine grain** view of the application behavior
  - **Interface** (POSIX or MPI-IO), **operation** (read/write)
  - MPI Rank, segment, offset, request size
  - Start and end timestamp
- **Challenge:** How to visualize and extract insights from DXT data?
  - Identify I/O bottlenecks
  - Optimize the application

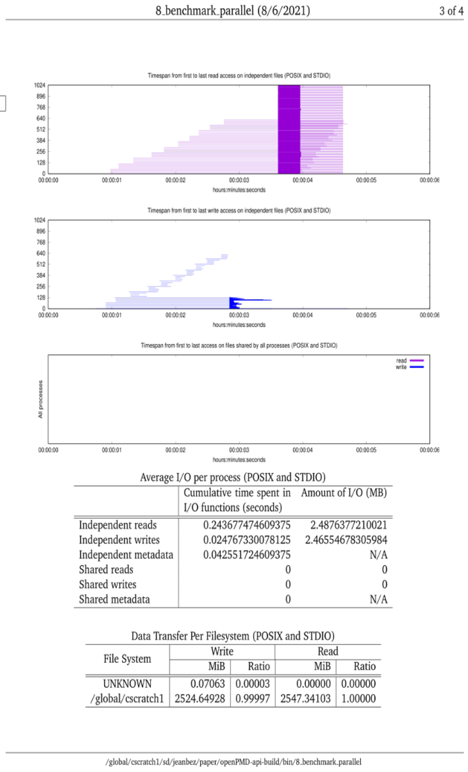


8.benchmark_parallel (8/6/2021) 1 of 4

jobid: 45195555	uid: 95230	nprocs: 1024	runtime: 6 seconds

I/O performance *estimate* (at the MPI-IO layer): transferred 5072.0 MiB at 798.82 MiB/s
I/O performance *estimate* (at the STDIO layer): transferred 0.1 MiB at 37.30 MiB/s

/global/cscratch1/sd/jeanbez/paper/openPMD-api-build/bin/8.benchmark_parallel



8.benchmark_parallel (8/6/2021) 3 of 4

Average I/O per process (POSIX and STDIO)

	Cumulative time spent in I/O functions (seconds)	Amount of I/O (MB)
Independent reads	0.243677474609375	2.4876377210021
Independent writes	0.0247767330078125	2.46554678305984
Independent metadata	0.042551724609375	N/A
Shared reads	0	0
Shared writes	0	0
Shared metadata	0	N/A

Data Transfer Per Filesystem (POSIX and STDIO)

File System	Write		Read	
	MiB	Ratio	MiB	Ratio
UNKNOWN	0.07063	0.00003	0.00000	0.00000
/global/cscratch1	2524.64928	0.99997	2547.34103	1.00000

/global/cscratch1/sd/jeanbez/paper/openPMD-api-build/bin/8.benchmark_parallel

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

# DXT, file_id: 5076057741753365924, file_name: /global/cscratch1/sd/tonglin/data_e2e/3d_28_16_16_32_32_32-36745115-1-nodes.nc4
# DXT, rank: 0, hostname: nid00604
# DXT, write_count: 10249, read_count: 0
# DXT, mnt_pt: /global/cscratch1, fs_type: lustre
# DXT, Lustre stripe_size: 16777216, Lustre stripe_count: 244
# DXT, Lustre OST obdidx: 132 52 146 214 86 200 176 24 16 6 224 76 90 198 190 112 114 58 78 102 74 32 68 36 48 208 30 194 238 182 126 96
28 142 188 34 44 22 164 54 140 92 110 20 156 62 72 150 84 144 94 128 38 202 8 148 134 158 186 98 46 138 154 168 108 82 106 80 0 136 210
118 4 10 40 14 184 196 172 18 12 174 116 162 64 120 50 166 56 26 192 180 178 104 170 124 42 122 152 130 70 100 160 88 247 243 227 219 215
177 233 221 223 207 89 229 91 213 237 199 205 245 209 193 155 189 123 149 211 169 235 145 201 81 157 21 97 165 175 179 143 161 31 53 41
181 231 225 183 67 129 119 85 71 77 5 29 107 61 9 113 11 147 103 13 111 133 33 63 121 127 141 35 93 101 109 75 23 99 117 167 49 185 115 7
135 3 57 95 43 27 191 1 163 51 15 153 187 55 151 239 79 25 137 47 217 17 39 59 171 69 173 37 203 125 131 87 19 195 65 45 139 105 241 83
159 73 197 2 216 234 218 222 246 220 226 232 244 206 212 236 228 240 242
# Module    Rank  Wt/Rd  Segment          Offset       Length    Start(s)      End(s)  [OST]
 X_POSIX       0  write        0               0         1955      0.1331      0.1359  [132]
 X_POSIX       0  write        1      3758106112         2038      0.1360      0.1372  [ 83]
 X_POSIX       0  write        2     11274300928         1978      0.1372      0.1384  [  7]
 X_POSIX       0  write        3     18790497792         4096      0.1384      0.1391  [ 41]
 X_POSIX       0  write        4     18790501888          366      0.1391      0.1391  [ 41]
 X_POSIX       0  write        5     18790502254          366      0.1391      0.1392  [ 41]
 X_POSIX       0  write        6     18790502922          150      0.1392      0.1392  [ 41]
 X_POSIX       0  write        7     18790503072          366      0.1392      0.1392  [ 41]
 X_POSIX       0  write        8            9728          256      0.6889      0.6894  [132]
 X_POSIX       0  write        9           13824          256      0.6894      0.6922  [132]
 X_POSIX       0  write       10           17920          256      0.6922      0.6926  [132]
 X_POSIX       0  write       11           22016          256      0.6926      0.6930  [132]
 X_POSIX       0  write       12           26112          256      0.6930      0.6937  [132]
 X_POSIX       0  write       13           30208          256      0.6937      0.6942  [132]
 X_POSIX       0  write       14           34304          256      0.6943      0.6946  [132]
 X_POSIX       0  write       15           38400          256      0.6946      0.6951  [132]
 X_POSIX       0  write       16           42496          256      0.6951      0.6956  [132]
 X_POSIX       0  write       17           46592          256      0.6956      0.6961  [132]
 X_POSIX       0  write       18           50688          256      0.6961      0.6966  [132]
 X_POSIX       0  write       19           54784          256      0.6966      0.6970  [132]
 X_POSIX       0  write       20           58880          256      0.6970      0.6974  [132]
```

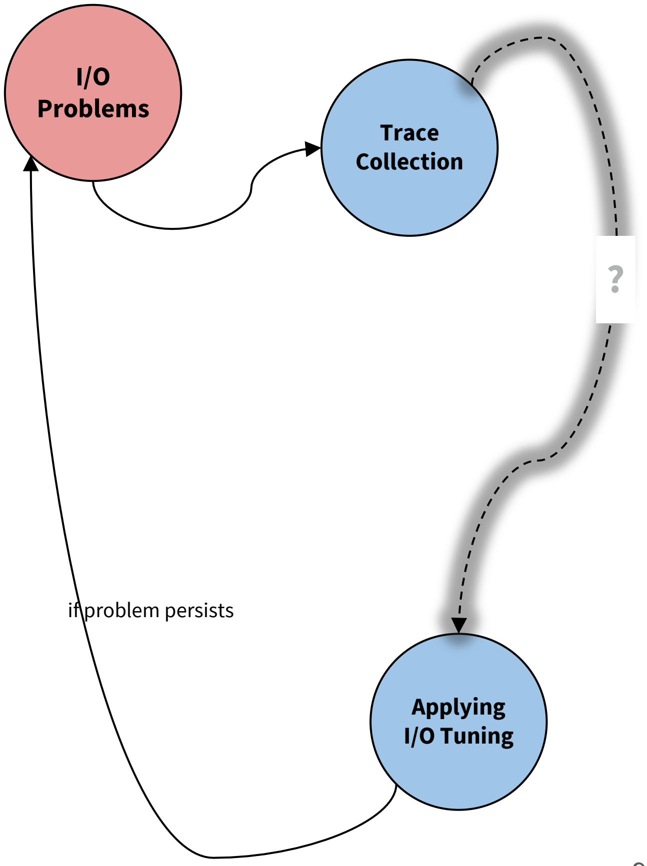
```
# *****************************************************
# DXT_MPIIO module data
# *****************************************************

# DXT, file_id: 5076057741753365924, file_name: /global/cscratch1/sd/tonglin/data_e2e/3d_28_16_16_32_32_32-36745115-1-nodes.nc4
# DXT, rank: 0, hostname: nid00604
# DXT, write_count: 12, read_count: 0
# DXT, mnt_pt: /global/cscratch1, fs_type: lustre
# Module    Rank  Wt/Rd  Segment       Length    Start(s)      End(s)
 X_MPIIO       0  write        0          331      0.1315      0.1392
 X_MPIIO       0  write        1       262144      0.6802      1.1448
 X_MPIIO       0  write        2       262144      1.1451      1.7255
 X_MPIIO       0  write        3       262144      1.7257      2.3791
 X_MPIIO       0  write        4       262144      2.3794      3.0459
 X_MPIIO       0  write        5       262144      3.0462      4.2975
 X_MPIIO       0  write        6       262144      4.2978      5.4152
 X_MPIIO       0  write        7       262144      5.4154      6.3356
 X_MPIIO       0  write        8       262144      6.3358      7.0600
 X_MPIIO       0  write        9       262144      7.0602      7.8717
 X_MPIIO       0  write       10       262144      7.8719      8.7132
 X_MPIIO       0  write       11           96      9.5743      9.5746
```

More details on Darshan Utilities: https://www.mcs.anl.gov/research/projects/darshan/docs/darshan-util.html

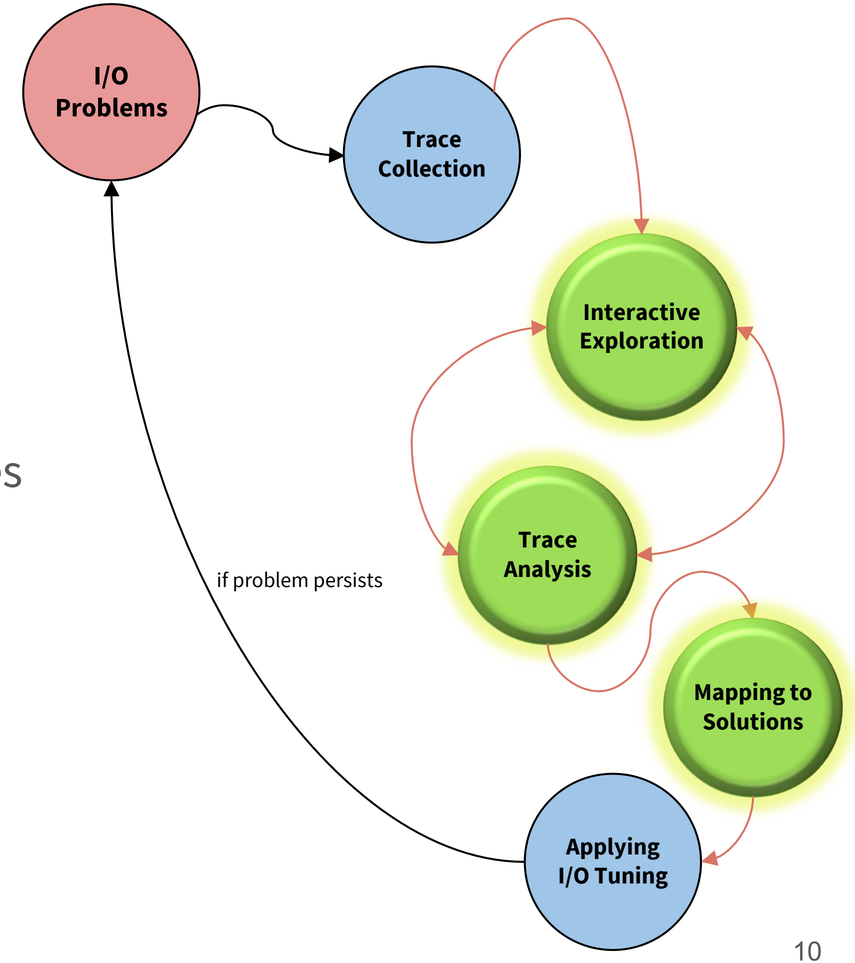
Missing dots in tuning I/O

- Despite the availability of fine grain traces
  - Gaps between **trace collection, analysis, and tuning**



Closing the translation gap

- A solution to close this gap requires
  - **Analysis** of collected metrics and traces
  - **Diagnosis of root causes** of poor performance
  - **Recommend** performance improving solutions

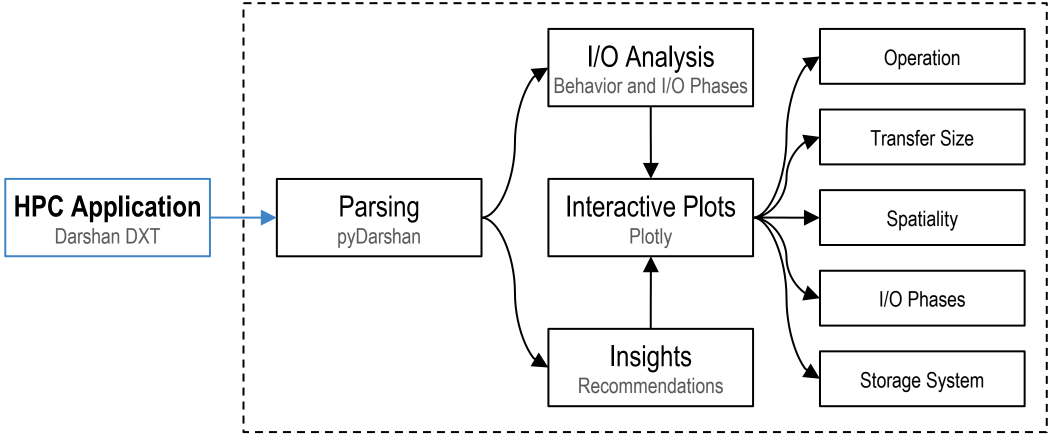


Envisioning a solution – Based on visualization and guidance

- Provide **interactive visualization** based on I/O trace files
- Display **contextual information** about I/O calls
- Understand how the application issues its **I/O requests over time**
- **Observe transformations** as the requests traverse the I/O software stack
- Detect and characterize the **distinct I/O phases** of an application
- Understand how the **ranks access the file system** in I/O operations
- Provide an extensible **community-driven framework**
- **Identify** and **highlight** common root causes of **I/O performance problems**
- Provide a **set of actionable items** based on the detected I/O bottlenecks

Drishti: Guiding end-users in the I/O optimization journey

- Sanskrit word; meaning **‘focused gaze’**
- **Interactive web based log** analysis framework to visualize **Darshan DXT** logs
  - Pinpoint root causes of I/O performance problems
  - Provide a set of actionable recommendations



Drishti

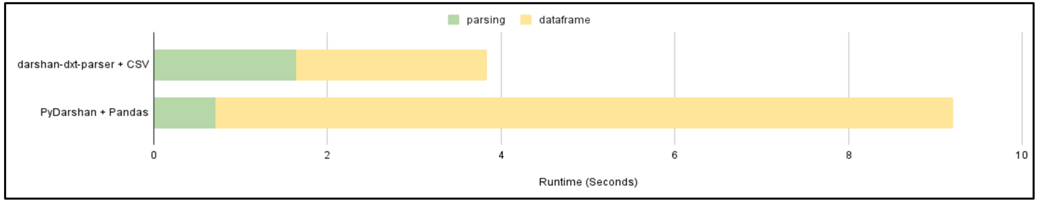
github.com/hpc-io/drishti-io

docker pull hpcio/drishti

https://jeanbez.gitlab.io/isc23

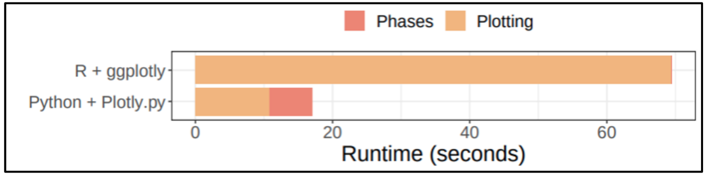
Extracting I/O Behavior from Traces

- Command line solution named **darshan-dxt-parser**
  - Parsed data -> Textual format -> CSV
- As an alternative, we explore the novel **PyDarshan**
  - Provides interface to binary Darshan log files -> Pandas dataframe
  - PyDarshan has shortcomings too
    - Loop over all the ranks to get a rectangular structure

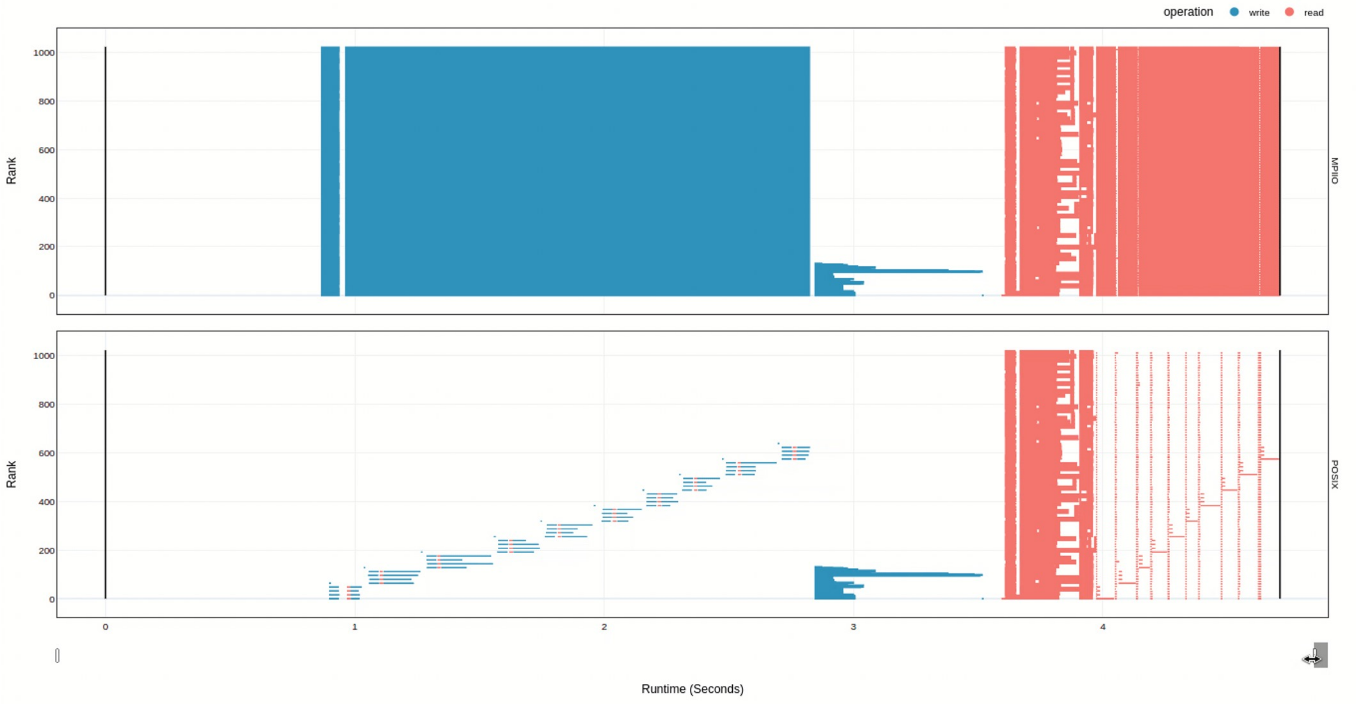


Interactively Exploring I/O Behavior

- **Static plots** limited in the information they represent
  - Space constraints
  - Pixel resolution
- We consider two solutions
  - **R** using **ggplot2**
  - **Python** using **plotly**
- Opted to rely on PyDarshan and plotly
  - Performance speedup
  - Modularity

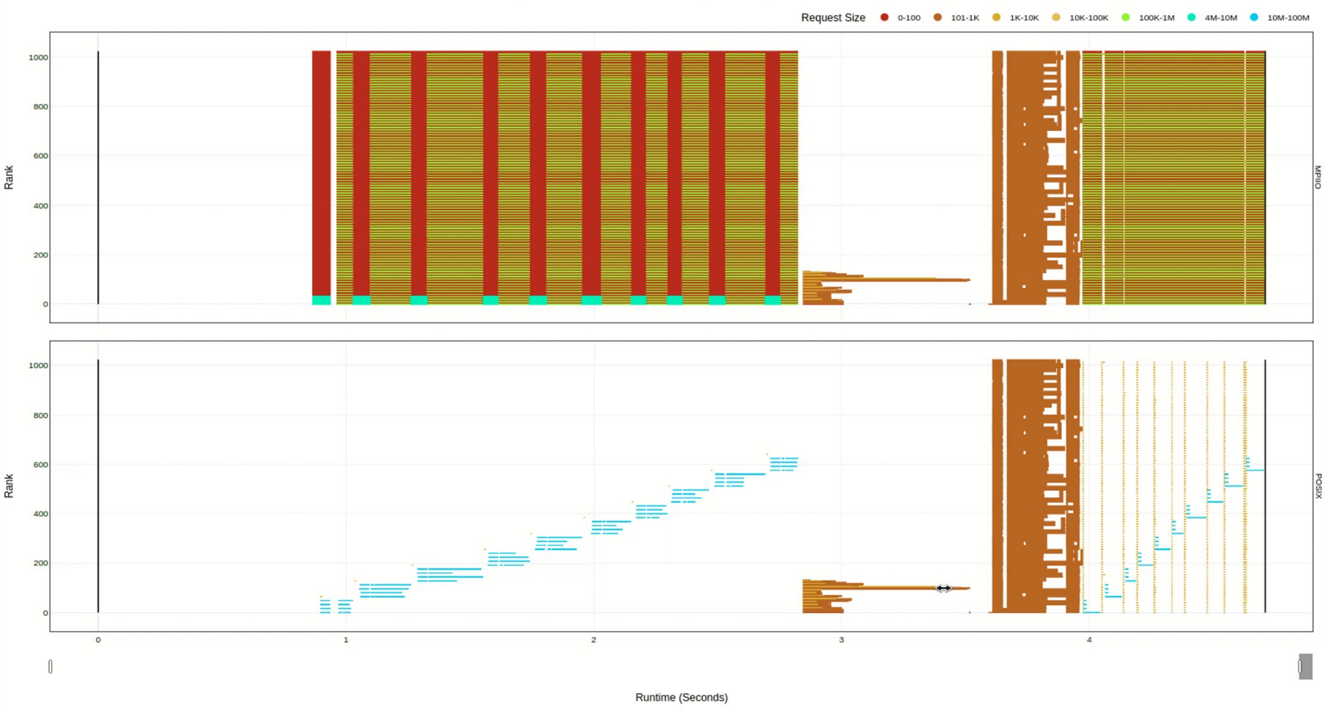


I/O Operations



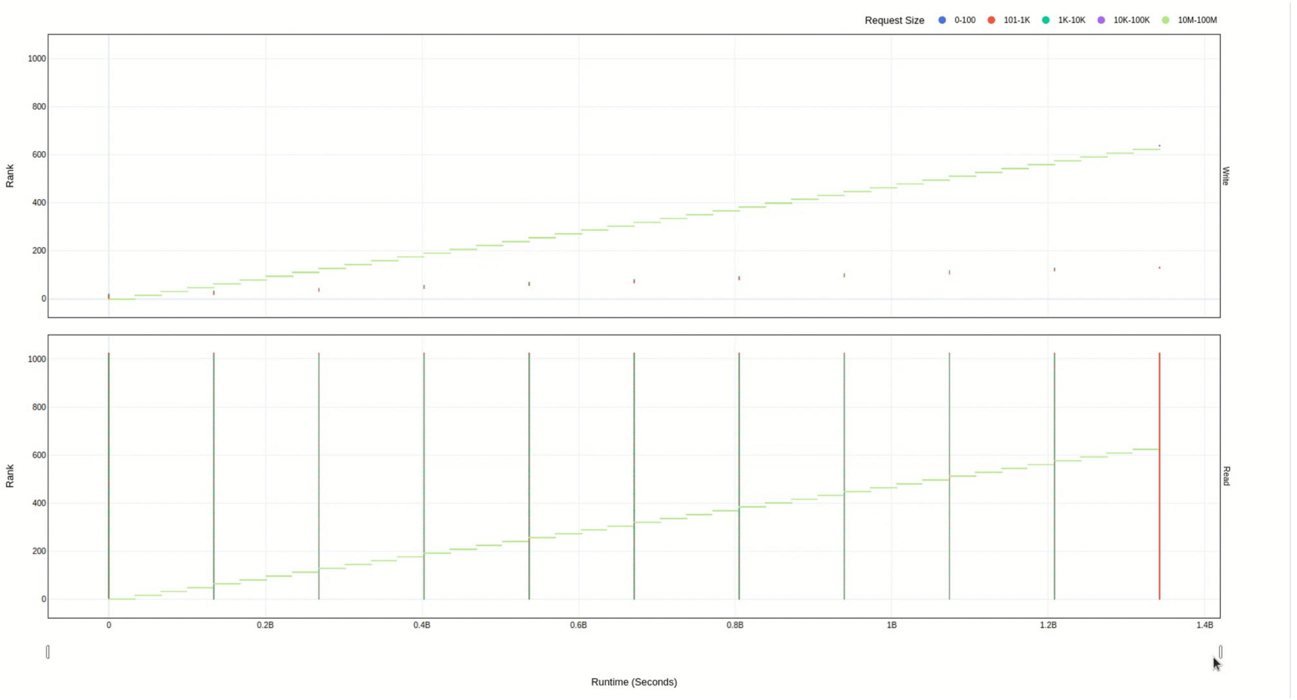
Explore the timeline by **zooming in and out** and observing how the **MPI-IO** calls are translated to the **POSIX** layer. Visualize relevant information in the context of each I/O call (rank, operation, duration, request size, and OSTs if Lustre) by hovering over a given operation.

Data Transfers



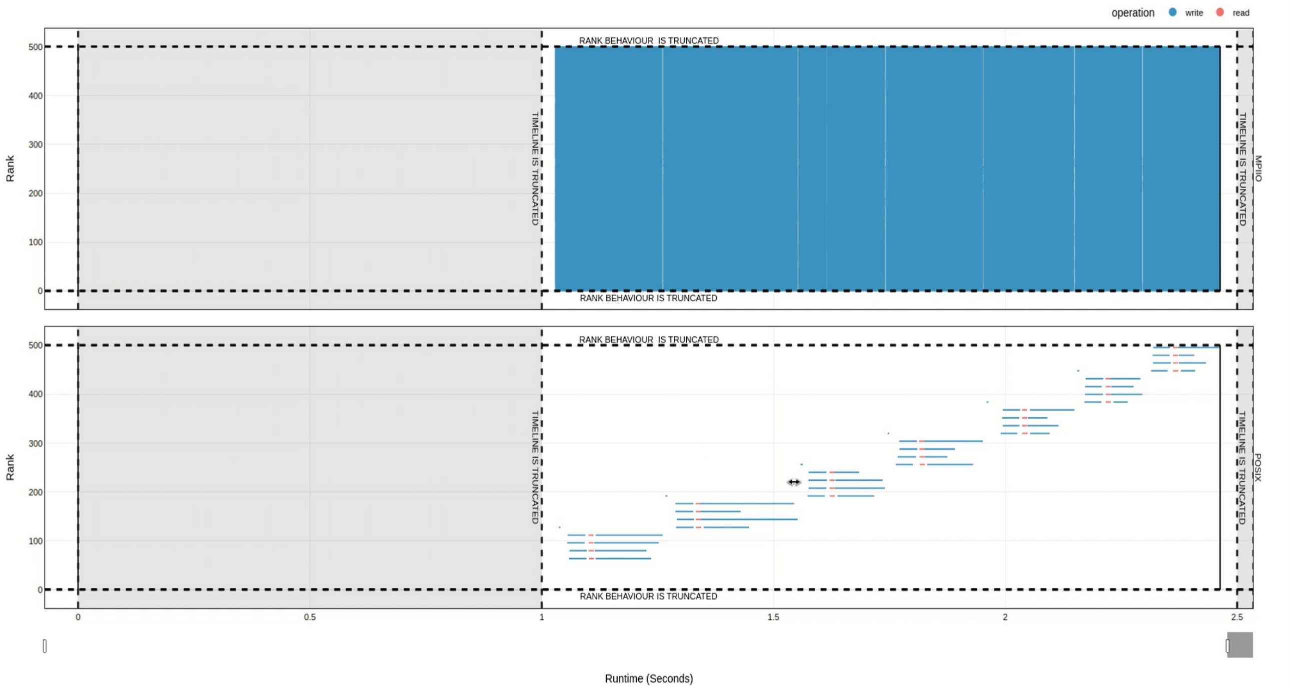
Explore the operations by size in **POSIX** and **MPI-IO**. You can, for instance, identify small or metadata operations from this visualization.

Spatiality



Explore the **spatiality** of accesses in file by each rank with contextual information.

Focused Operation View



Explore the timeline by **zooming in and out** and observing how the **MPI-IO** calls are translated to the **POSIX** layer. Visualize relevant information in the context of each I/O call (rank, operation, duration, request size, and OSTs if Lustre) by hovering over a given operation.

Automatic Detection of I/O Bottlenecks

- Variety of tools that seek to analyze performance
  - Neither provide support for auto detection of I/O bottlenecks
- Drishti provides framework which provides
  - Actionable set of recommendations in form of a **report**
  - **Multi layered plots** to diagnose and highlight bottlenecks

ROOT CAUSES OF I/O PERFORMANCE BOTTLENECKS

Root Causes	Darshan	DXT	System	Drishti
Too many I/O phases	✓	✓	✗	✓
Stragglers in each I/O phase	✓	✓	✗	✓
Bandwidth limited by a single OST I/O bandwidth	✗	✗	✓	✗
Limited by the small data size	✓	✓	✗	✓
Rank 0 heavy-workload	✓	✓	✗	✓
Unbalanced I/O workload among MPI ranks	✓	✓	✗	✓
Large number of small I/O requests	✓	✓	✗	✓
Unbalanced I/O workload on OSTs	✓	✓	✓	✓
Bad file system weather	✗	✗	✓	✗
Redundant/overlapping I/O accesses	✓	✓	✗	✓
I/O resource contention at OSTs	✗	✗	✓	✗
Heavy metadata load	✓	✗	✗	✓

Drishti Reports

- Relies on **counters** available in Darshan profiling logs
- Detects typical I/O performance pitfalls
    - Based on **32 checks** covering common I/O performance pitfalls
- Insights classified into **four** categories
    - **HIGH**, **WARN**, **OK**, **INFO**
- Provides recommendations in form of a **report**

Level	Interface	Detected Behavior
HIGH	STDIO	High STDIO usage* (> 10% of total transfer size uses STDIO)
OK	POSIX	High number* of sequential read operations (≥ 80%)
OK	POSIX	High number* of sequential write operations (≥ 80%)
INFO	POSIX	Write operation count intensive* (> 10% more writes than reads)
INFO	POSIX	Read operation count intensive* (> 10% more reads than writes)
INFO	POSIX	Write size intensive* (> 10% more bytes written then read)
INFO	POSIX	Read size intensive* (> 10% more bytes read then written)
WARN	POSIX	Redundant reads
WARN	POSIX	Redundant writes
HIGH	POSIX	High number* of small† reads (> 10% of total reads)
HIGH	POSIX	High number* of small† writes (> 10% of total writes)
HIGH	POSIX	High number* of misaligned memory requests (> 10%)
HIGH	POSIX	High number* of misaligned file requests (> 10%)
HIGH	POSIX	High number* of random read requests (> 20%)
HIGH	POSIX	High number* of random write requests (> 20%)
HIGH	POSIX	High number* of small† reads to shared-files (> 10% of reads)
HIGH	POSIX	High number* of small† writes to shared-files (> 10% of writes)
HIGH	POSIX	High metadata time* (one or more ranks spend > 30 seconds)
HIGH	POSIX	Rank o heavy workload
HIGH	POSIX	Data transfer imbalance between ranks (> 15% difference)
HIGH	POSIX	Stragglers detected among the MPI ranks
HIGH	POSIX	Time imbalance* between ranks (> 15% difference)
WARN	MPI-IO	No MPI-IO calls detected from Darshan logs
HIGH	MPI-IO	Detected MPI-IO but no collective read operation
HIGH	MPI-IO	Detected MPI-IO but no collective write operation
WARN	MPI-IO	Detected MPI-IO but no non-blocking read operations
WARN	MPI-IO	Detected MPI-IO but no non-blocking write operations
OK	MPI-IO	Detected MPI-IO and collective read operations
OK	MPI-IO	Detected MPI-IO and collective write operations
HIGH	MPI-IO	Detected MPI-IO and inter-node aggregators
WARN	MPI-IO	Detected MPI-IO and intra-node aggregators
OK	MPI-IO	Detected MPI-IO and one aggregator per node

* Trigger has a threshold that could be further tunned. Default value in parameters.
† Small requests are consider to be < 1MB.

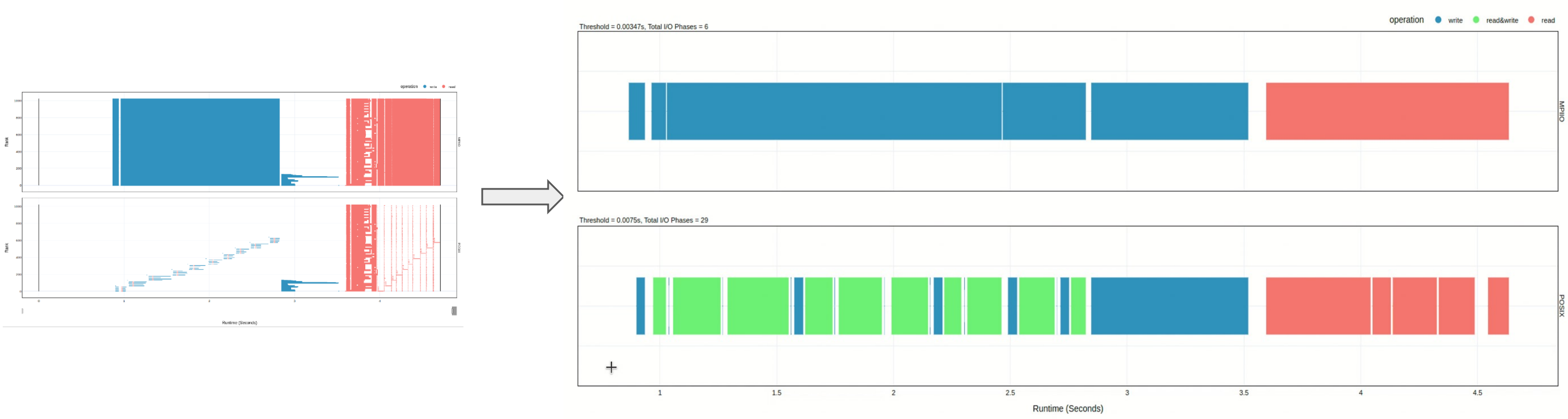
Exploring I/O Phases and Bottlenecks

- I/O phase is a **time period** where an application is accessing its data
- Factors outside an application's scope could cause an I/O phase to take longer
  - Network Interference
  - Storage system congestion
- Drishti adds an **interactive visualization** to detect I/O phases based on DXT trace data
  - Multi layered plots to detect workload imbalance and rank 0 heavy workload
- Gives a detailed picture of I/O phases and I/O patterns in the data
  - Helpful in extracting information related **bottlenecks** such as stragglers

Exploring I/O Phases and Bottlenecks (contd …)

- Finding I/O phases is **not a trivial task** due the sheer amount of data
    - Millions of operations in order of milliseconds
- We use **PyRanges** to find similar and overlapping behavior between ranks
    - PyRanges is a genomics library used for handling genomic intervals
    - Use a threshold value to merge I/O phases closer to each other
- While computing I/O phases, we keep track of the **duration** between each I/O phase
    - **Threshold** to merge I/O phases close to each other

Exploring I/O Phases and Bottlenecks (contd …)

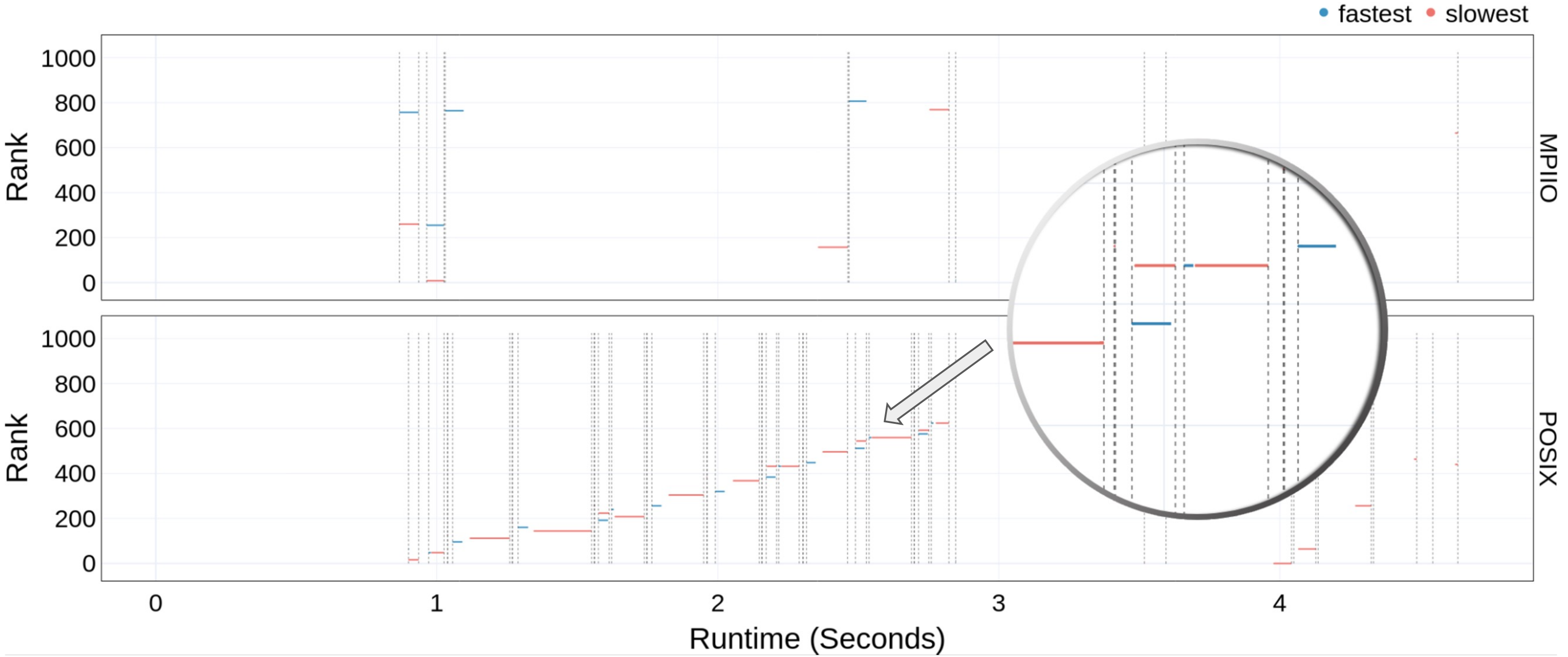


Explore the **I/O phases** of the application. Contextual information like the **fastest rank, fastest rank duration, slowest rank**, and **slowest rank duration** are available when hovering over an I/O phase.

Exploring I/O Phases and Bottlenecks (contd …)

- I/O stragglers in each phase could define a **critical path** impairing performance
- **Exclusive plot** to highlight the I/O phases and fast and straggler in each phase
  - Handle each interface separately
- Detect **slow ranks** across the entire execution or storage servers

Stragglers



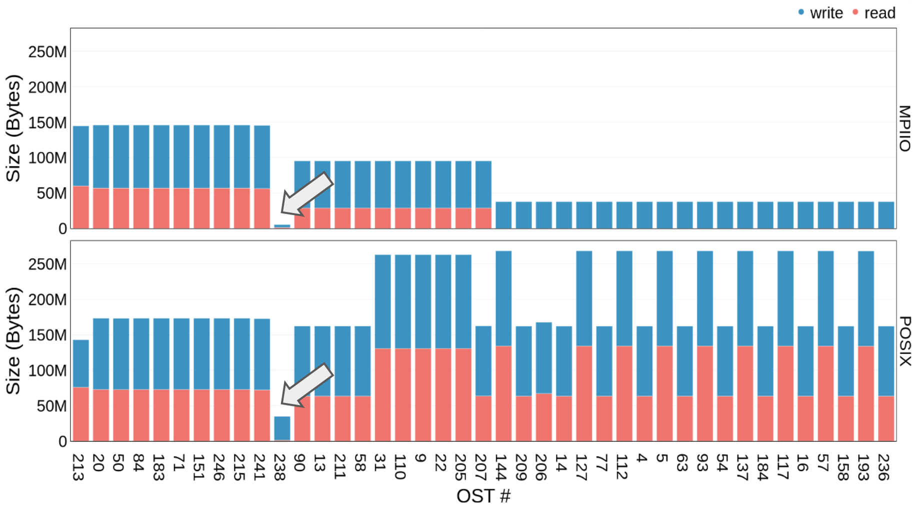
Explore the **stragglers** in the entire execution. Upon hovering over a phase, all the information related to the **fastest** and **slowest** rank is shown. The dotted lines represent the start and the end of a phase

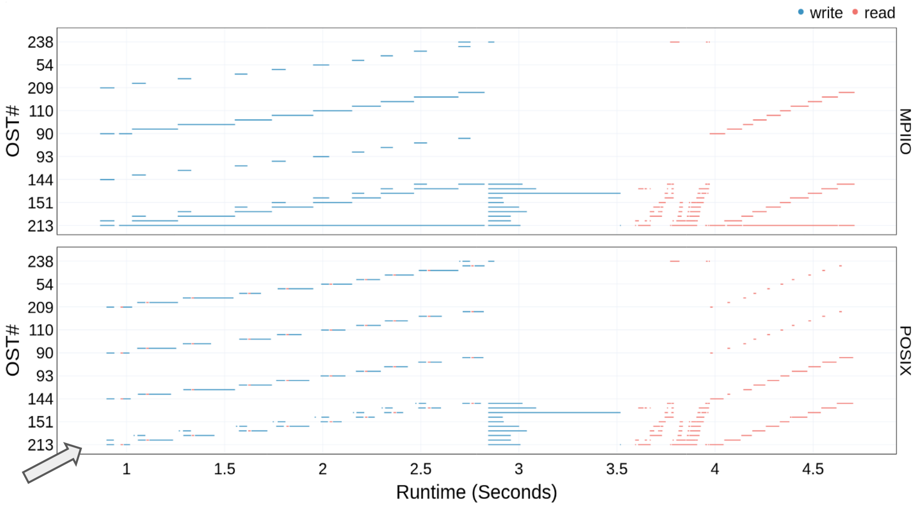
File System Usage

- DXT captures information related to the **storage servers**
- Drishti provides a visualization to explore the **OST usage** of the I/O requests
- Also provides a visualization to depict **data transfer sizes** for each OST

Towards Exploring File System Usage (contd ...)

- The plots show
  - Very **small data transfer** for OST# 238 for MPI IO
    - Size increases at POSIX due to transformations as the request goes through the stack
  - OST# 213 is **most used** for MPI IO
  - OST# 238 is the **least used** across both MPI IO and POSIX





Explore the **file system usage** for the entire execution. The plot on the **left** shows **OST usage data transfer** sizes and the plot on the **right** shows **OST usage of I/O requests** over time

Putting Drishti into practice

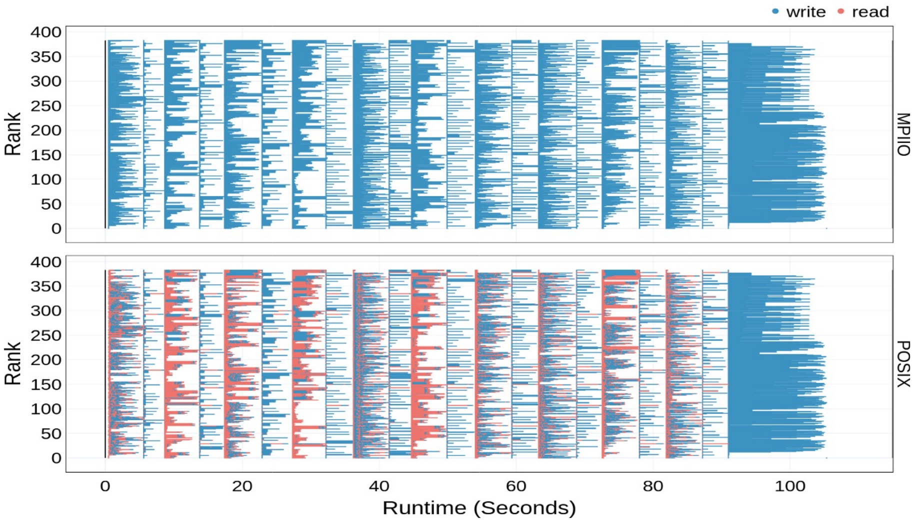
- Demonstrate Drishti to identify I/O performance bottlenecks
- Experiments conducted at:
    - **Cori** at NERSC
    - **Summit** at OLCF
- **Four** use cases:
    - OpenPMD
    - AMReX
    - E2E (available in companion repo)
    - H5Bench Write (available in companion repo)
- Used **h5bench** to generate the benchmarks

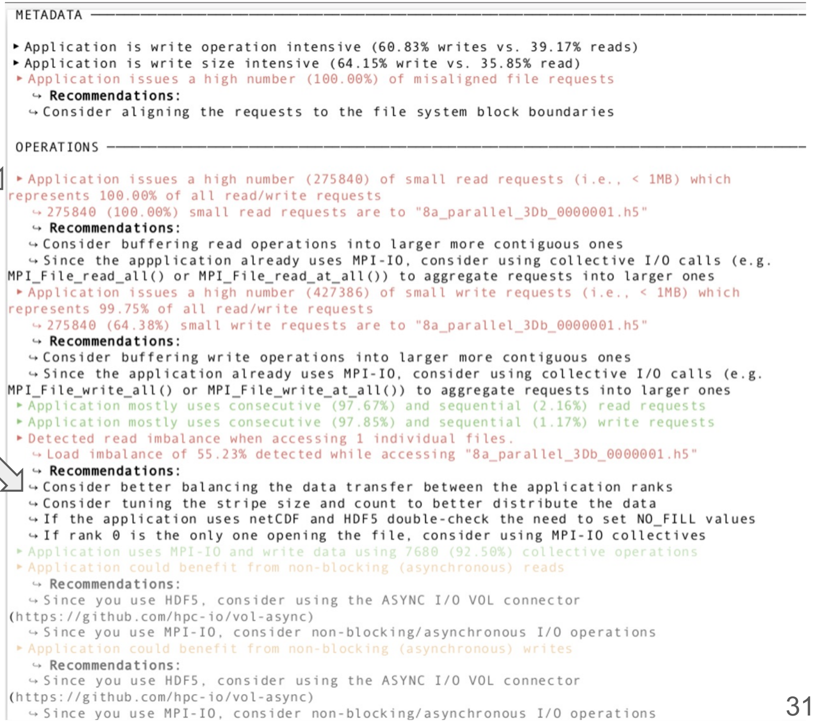
I/O Bottlenecks in OpenPMD

- Open Standard for Particle Mesh Data Files (OpenPMD)
  - Particle and mesh data in scientific simulations and experiments
- **Summit** with 64 compute nodes, 6 ranks per node, and a total of 384 MPI ranks
  - Mesh size is [65536 $\times$ 256 $\times$ 256], 10 iterations
- For this scenario, OpenPMD takes an average **110.6 seconds**

I/O Bottlenecks in OpenPMD

- Majority of the read and write requests are **small**
  - I/O calls are not using the **MPI-IO's collective** option



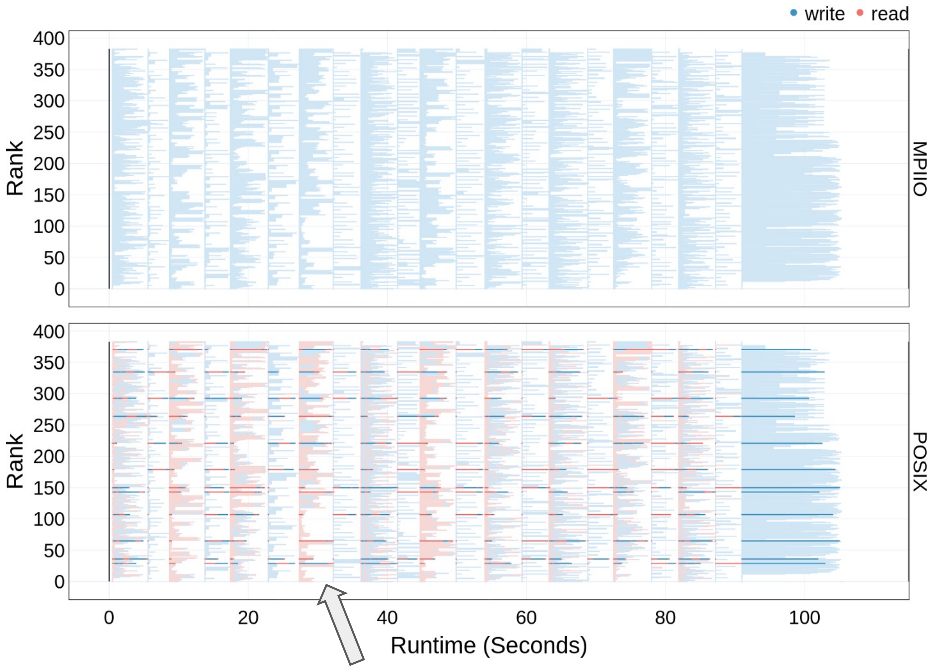


```
METADATA
▸ Application is write operation intensive (60.83% writes vs. 39.17% reads)
▸ Application is write size intensive (64.15% write vs. 35.85% read)
▸ Application issues a high number (100.00%) of misaligned file requests
  ↪ Recommendations:
  ↪ Consider aligning the requests to the file system block boundaries

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"
  ↪ 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
▸ 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"
  ↪ 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 (97.67%) and sequential (2.16%) read requests
▸ Application mostly uses consecutive (97.85%) and sequential (1.17%) write requests
▸ Detected read imbalance when accessing 1 individual files.
  ↪ Load imbalance of 55.23% detected while accessing "8a_parallel_3Db_0000001.h5"
  ↪ Recommendations:
  ↪ Consider better balancing the data transfer between the application ranks
  ↪ Consider tuning the stripe size and count to better distribute the data
  ↪ If the application uses netCDF and HDF5 double-check the need to set NO_FILL values
  ↪ If rank 0 is the only one opening the file, consider using MPI-IO collectives
▸ Application uses MPI-IO and write data using 7680 (92.50%) 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
```

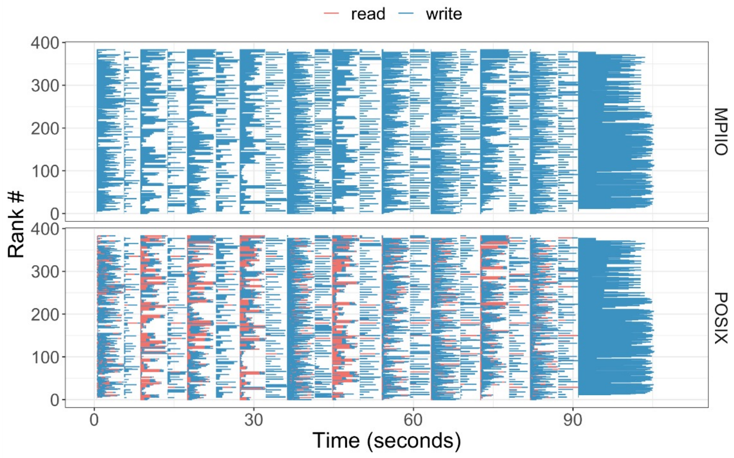
I/O Bottlenecks in OpenPMD

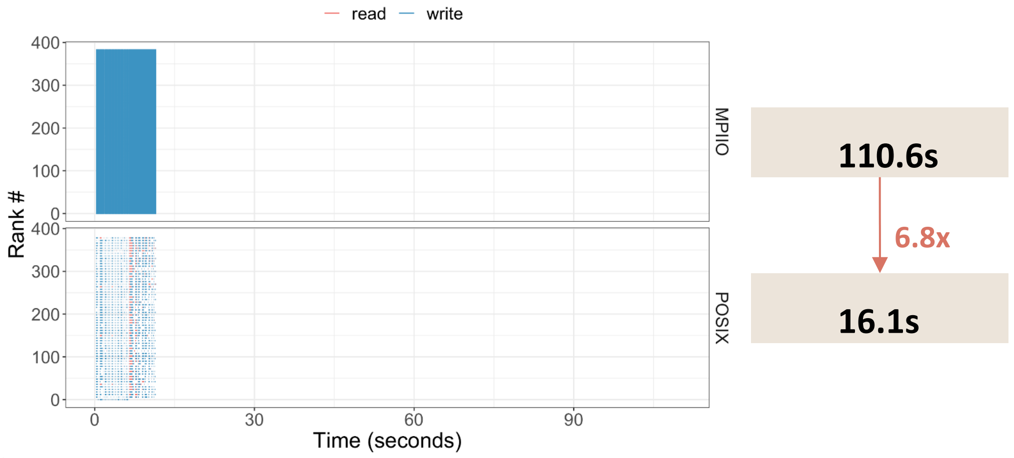
- Moreover, Drishti detected an **imbalance** when accessing the data



I/O Bottlenecks in OpenPMD - optimized

- Collective HDF5 metadata were not actually collective due to an issue introduced in HDF5 1.10.5
  - Fixed that issue by using **HDF5 1.10.4** (or using **1.10.6** or later) and enabling collective metadata I/O
- Drishti suggested larger buffer sizes
  - Used ROMIO hints to set the aggregators to **1 agg/node** and set the **cb_buffer_size** to 16 MB
  - Used GPFS **large block** I/O
- With HDF5 1.10.4 combined with other optimizations gives a total of **6.8x** speedup from baseline



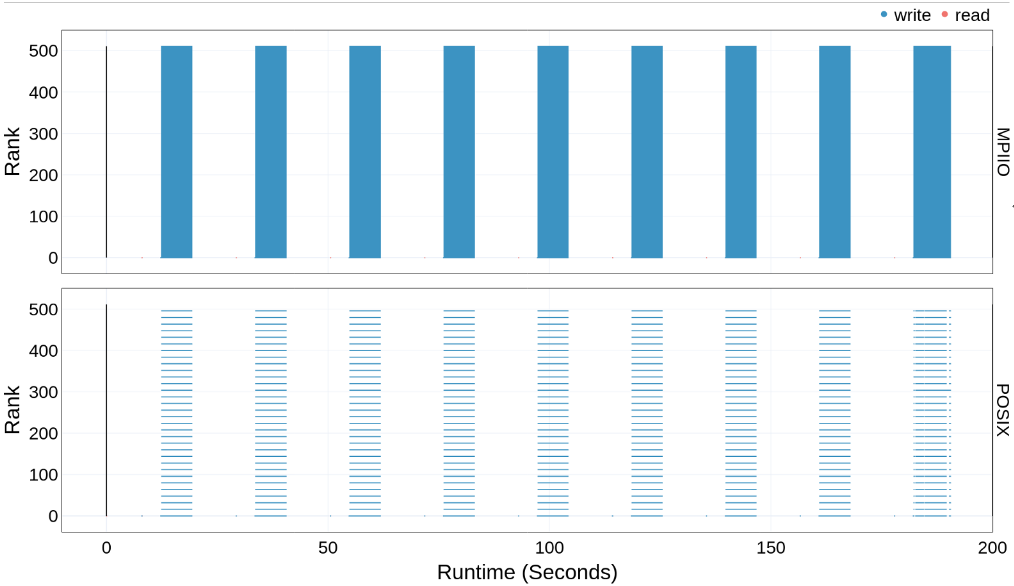


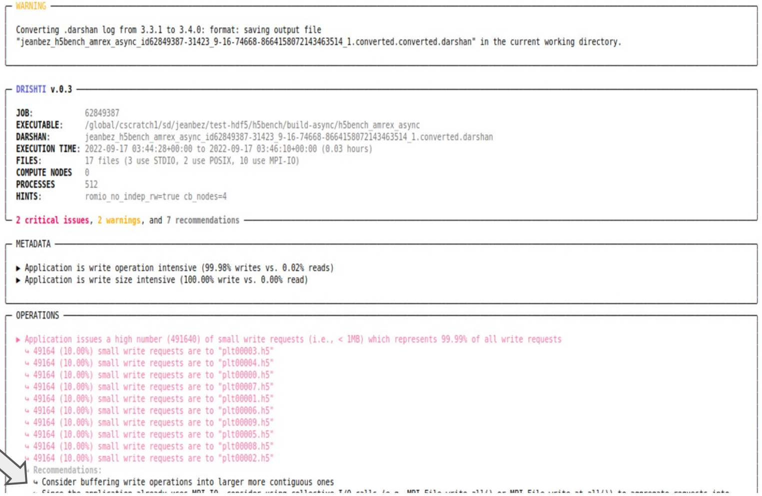
Improving AMReX with Asynchronous I/O

- AMReX is an adaptive mesh refinement (AMR) framework
  - Solves partial differential equations on block-structured meshes
  - Used by several applications in the Exascale Computing Project (ECP)
- I/O benchmark on **Cori** with 32 compute nodes, 512 ranks
  - 1024 domain size, 10 plot files, 10 seconds sleep time between writes

Improving AMReX with Asynchronous I/O (contd …)

- The report suggests using
  - Larger buffer sizes
  - Asynchronous I/O VOL connector



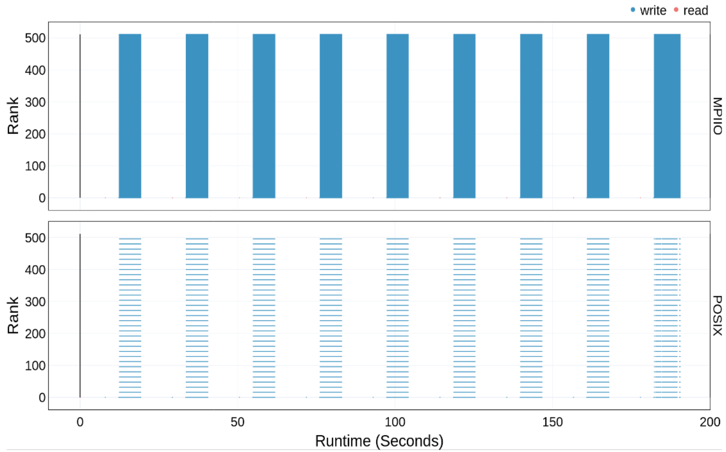


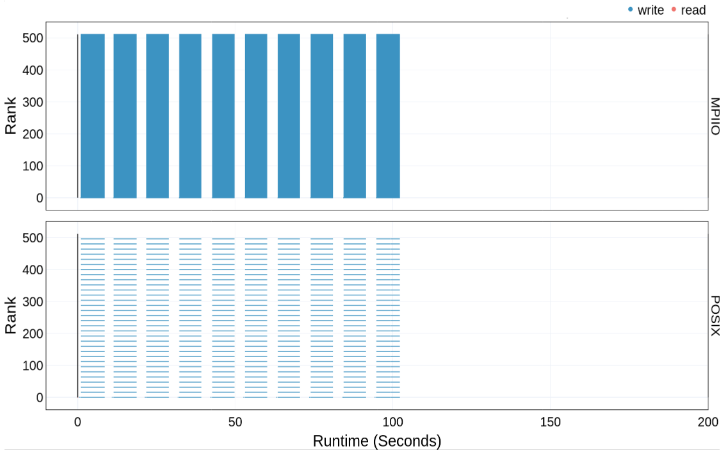
Consider buffering write operations into larger more contiguous ones

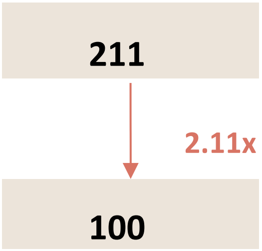
Since you use HDF5, consider using the ASYNC I/O VOL connector (https://github.com/hpc-io/vol-async)

Improving AMReX with Asynchronous I/O (contd ...)

- Added asynchronous I/O VOL Connector
  - Makes the operations **non blocking**
  - Hide time spent in time I/O while the application continues its computation
- Majority of request sizes are very small (< 1MB) for all the 10 plot files
  - Set **stripe size** to 16MB







Future Work

- Integrate **additional metrics** and system **logs** to broaden the spectrum of I/O issues
  - Global API to consume metrics from distinct sources e.g. Recorder
- Map performance optimization recommendations to the **exact source code** line numbers
  - Static code analysis
  - Modified code instead of generic snippets in Drishti reports
- **Community guidelines** on how to contribute to this tool
  - Aid in keeping up with the latest advancements in I/O libraries and systems
  - Reach out to novel systems for support

Conclusion

- Pinpointing root causes of I/O inefficiencies requires:
    - Detailed metric **analysis**
    - **Understanding** of the HPC I/O stack
- Drishti is an interactive web based **analysis framework** which
    - Seeks to **close the gap** between trace collection, analysis, and tuning
    - **Automatically** detects common root causes of I/O performance inefficiencies
    - Provides actionable **recommendations** to the user
- Applicability demonstrated with **optimization** of OpenPMD and AMReX applications
- Companion Repository: **https://jeanbez.gitlab.io/isc23**

Contact: Suren Byna
https://sbyna.github.io

Thanks to: ECP EXASCALE COMPUTING PROJECT

Illuminating the I/O Optimization Path of Scientific Applications

Hammad Ather, Jean Luca Bez, Boyana Norris, Suren Byna

BERKELEY LAB
The Ohio State University
University of Oregon
ECP Exascale Computing Project