Drishti and HDF5: What is actually happening in my application?

Jean Luca Bez — Scientific Data Division, LBNL

Suren Byna (OSU and LBNL) and Hammad Ather (Univ. of Oregon and LBNL)

HDF5 User Group (HUG) Meeting 2023
Complex I/O stack!

- Using the HPC I/O stack efficiently is a **tricky problem**
- **Interplay of factors** can affect I/O performance
- Various **optimizations** techniques available
- Plethora of tunable **parameters**
- Each layer brings a new set of parameters
Metrics to the rescue?

- **Darshan** is a popular tool to collect **I/O profiling**
- Extended tracing mode (**DXT**) for a fine grain view
- **Recorder** and **TAU** are other I/O profiling tools
- How to optimize the I/O of my application?
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
Drishti

- 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
HPC Application
- I/O Metrics

- Recorder
  - Traces

- Darshan / DXT
  - pyDarshan

I/O Analysis
- Behavior and I/O Phases

Insights
- Recommendations

Interactive Plots
- Plotly

- Transfer Size
- Spatiality
- I/O Phases
- Storage System

Operation

HTML
- Drishti Output

Drishti and HDF5 | Jean Luca Bez | HUG’23
Interactive Analysis

**Operation**

![Operation Diagram](image)

**Transfer Size**

![Transfer Size Diagram](image)

**Spatial Locality**

![Spatial Locality Diagram](image)

**I/O Phases**

![I/O Phases Diagram](image)

**OST Usage**

![OST Usage Diagram](image)
# Drishti Triggers

<table>
<thead>
<tr>
<th>Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HIGH</td>
<td>High probability of harming I/O performance.</td>
</tr>
<tr>
<td>WARN</td>
<td>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.</td>
</tr>
<tr>
<td>OK</td>
<td>Best practices have been followed.</td>
</tr>
<tr>
<td>INFO</td>
<td>Relevant information regarding application configuration.</td>
</tr>
</tbody>
</table>
**USE CASE**

**METADATA**
- Application is write operation intensive (90.0% writes vs. 9.17% reads)
- Application issues a high number (100.00%) of misaligned file requests

**OPERATIONS**
- Application issues a high number (275848) of small read requests (i.e., < 1MB) which represents 100.00% of all read/write requests
- Application issues a high number (427308) of small write requests (i.e., < 1MB) which represents 99.75% of all read/write requests

**METADATA**
- Application is write operation intensive (90.05% writes vs. 9.15% reads)
- 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 (7.82%) write requests
- Application uses MPI-I0 and write data using 848 (92.50%) 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 (60.85% writes vs. 39.15% 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 (275848) of small read requests (i.e., < 1MB) which represents 108.89% of all read/write requests
  - 275848 (108.89%) small read requests are to “/a_parallel_3Dh_00000001.h5”
- Application issues a high number (427300) of small write requests (i.e., < 1MB) which represents 99.75% of all read/write requests
  - 427300 (64.38%) small write requests are to “/a_parallel_3Dh_00000001.h5”
- Application mostly uses consecutive (97.04%) and sequential (1.17%) read requests
- Application mostly uses consecutive (97.85%) and sequential (7.82%) 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 (91.34% write vs. 8.66% 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 (15.25%)
- Application mostly uses consecutive (86.95%) and sequential (7.82%) write requests
- Application uses MPI-IO and write data using 8448 (108.68%) collective operations
- Application could benefit from non-blocking (asynchronous) reads
- Application could benefit from non-blocking (asynchronous) writes
AMReX
USE CASE

• Uses highly parallel Adaptive Mesh Refinement (AMR) algorithms
  – Solve partial differential equations
  – Block-structured meshes

• Astrophysics, atmospheric modeling, combustion, cosmology, and particle accelerators

• Experimental setup:
  – 512 ranks (32 nodes) in Cori
  – 1024 domain size
  – 1 level, 6 components, 2 particles per cell
  – 10 output plot files
AMReX
USE CASE

2.1× speedup from 211 to 100 seconds

METAATA
- Application is write operation intensive (99.9% writes vs. 0.01% reads)
- Application is write size intensive (100% write vs. 0.0% read)

OPERATIONS
- Application issues a high number (491680) of small write requests (i.e., 4MB) which represents 99.9% of all read/write requests
  - 9312 (20%) small writes are to /v0/80000.15
  - 9312 (20%) small writes are to /v0/80002.15
  - 9312 (20%) small writes are to /v0/80004.15
  - 9312 (20%) small writes are to /v0/80006.15
  - 9312 (20%) small writes are to /v0/80008.15
  - 9312 (20%) small writes are to /v0/80010.15
  - 9312 (20%) small writes are to /v0/80012.15
  - 9312 (20%) small writes are to /v0/80014.15
  - 9312 (20%) small writes are to /v0/80016.15
- Consider buffering write operations into larger more contiguous areas
  - Since the application already uses MPI-IO, consider using collective I/O calls (e.g., MPI_FILE_WRITE_ALL() or MPI_FILE_WRITE_ALL() to aggregate requests into larger areas
- Application mostly uses concurrent (20%) and sequential (79.9%) read requests
- Application issues a high number (491680) of small write requests to a shared file (i.e., 4MB) which represents 99.9% of all shared file write requests
  - 45164 (9%) small writes are requests to /v0/80000.15
  - 45164 (9%) small writes are requests to /v0/80002.15
  - 45164 (9%) small writes are requests to /v0/80004.15
  - 45164 (9%) small writes are requests to /v0/80006.15
  - 45164 (9%) small writes are requests to /v0/80008.15
  - 45164 (9%) small writes are requests to /v0/80010.15
  - 45164 (9%) small writes are requests to /v0/80012.15
  - 45164 (9%) small writes are requests to /v0/80014.15
  - 45164 (9%) small writes are requests to /v0/80016.15
- Consider coalescing write requests into larger more contiguous areas using MPI-IO collective operations
- Application would benefit from non-blocking (asynchronous) reads

Recommendations:
- Since you use HDF5, consider using the H5RECF 1/0 VDI LWC converter (https://github.com/ls4/h5vdi-lwc)
- Since you use MPI-IO, consider rowblocking/asynchronous I/O operations
- Application would benefit from non-blocking (asynchronous) writes

Recommendations:
- Since you use HDF5, consider using the H5RECF 1/0 VDI LWC converter (https://github.com/ls4/h5vdi-lwc)
- Since you use MPI-IO, consider rowblocking/asynchronous I/O operations
Coming Soon!

- Improved recommendations
  - Issues and suggestions pointing to the source-code location
  - Enhanced code snippets for tuning
- API to handle other sources of metrics
  - Full support for Recorder 2.0 (github.com/uiuc-hpc/Recorder)
  - File system metrics
- Support for interdependent and complex triggers
Drishti and HDF5: What is actually happening in my application?

- Pinpointing root causes of I/O inefficiencies **requires**:
  - Detailed metric analysis
  - Understanding of the HPC I/O stack
- **Drishti** is an interactive I/O analysis framework
  - Seeks to close the gap between *trace collection, analysis, and tuning*
  - Detects common root causes of I/O performance inefficiencies
  - Provides actionable recommendations to the users