Argonne NATIONAL LABORATORY

Techniques for Modeling Large-scale HPC I/O Workloads

Shane Snyder, Philip Carns, Robert Latham, Misbah Mubarak, Robert Ross

Argonne National Laboratory

Christopher Carothers

Rensselaer Polytechnic Institute

Babak Behzad, Huong Vu Thanh Luu

University of Illinois at Urbana-Champaign

Surendra Byna, Prabhat

Lawrence Berkeley National Laboratory

U.S. DEPARTMENT OF ENERGY

Background & Motivation

- Meaningful storage system analysis is contingent on the use of representative I/O workloads
  - Conclusive I/O analysis needs to be done in context of workloads expected in production

- Storage system designs/algorithms have typically been evaluated using workloads from the following sources:
  - **I/O traces:** capture detailed info about each I/O function of interest from some target application
  - **I/O kernels:** manually developed representations of application I/O workloads
  - **I/O characterizations**: condensed representations of the salient characteristics of application I/O workloads

- Each method has inherent tradeoffs & no one method works best in all scenarios
  - There is great benefit in giving researchers flexibility in types of workloads they can use to drive their analyses

Modeling HPC I/O workloads

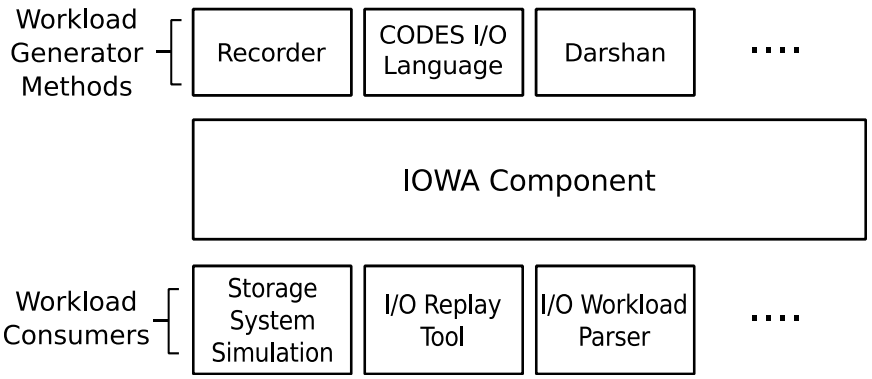
- HPC I/O workloads are particularly difficult to model
  - Large-scale (10,000-100,000 application processes, & growing)
  - Many distinct I/O strategies
  - Coordinated access
  - Proliferation of I/O libraries and data interfaces

- Further, drawing meaningful conclusions from HPC I/O analysis is nontrivial
  - HPC storage systems are shared among many competing I/O workloads
    - The performance of some workload of interest is dependent on the imposed I/O workloads of other jobs in the system
  - HPC systems are complex and the I/O stack is deep
    - Performance issues may be difficult to isolate

A solution: *IOWA* (I/O workload abstraction)

- IOWA is an interface allowing arbitrary *workload consumers* to ingest representative I/O workloads from a number of distinct *workload generator methods*

- IOWA allows I/O researchers to generate workloads from a range of sources:
  - I/O traces
  - I/O kernels
  - I/O characterizations
  - Mathematical models
  - …



- Researchers now have the option to choose workload sources most suitable for their study:
  - What sources are amenable to the evaluation they are performing?
  - What sources are actually available or attainable?

IOWA workload model

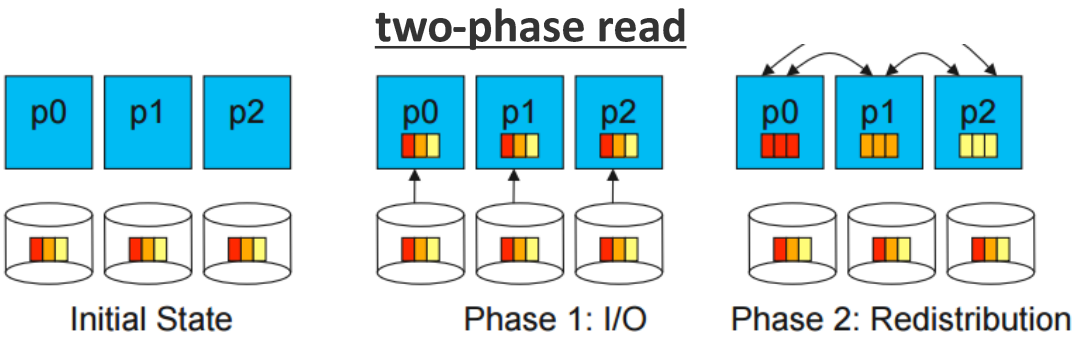
- IOWA embodies the following design criteria:
  - ☑ Workloads composed of an ordered, identifiable set of application processes (e.g. MPI ranks)
  - ☑ Independent streams of workload operations generated for each process
  - ☑ Workload operations include not only I/O primitives, but operations for modeling application computation and synchronization points
  - ☑ Ability to "undo" generation of an operation, for compatibility with optimistic DES systems

- Workloads modeled at the POSIX layer
  - Primarily for portability of workloads
  - Is this the best idea? More on this later…

- Currently supported operations:
  - **open, read, write, close** => I/O primitives
  - **delay** => models application computation
  - **barrier** => models collective communication

IOWA collective I/O model

- HPC I/O workloads often include coordinated, collective operations that enable optimizations such as two-phase I/O:
  - *Phase 1*: processes read large, contiguous regions of file
  - *Phase 2*: processes redistribute data amongst themselves



- Key terms:
  - Aggregators – workload processes that perform I/O on behalf of other processes
    - Typically, $num\_aggs << num\_procs$
  - File domain – the file extent that a given aggregator is responsible for

- IOWA workload generator methods must emulate the two-phase algorithm to accurately reproduce collective I/O workloads

IOWA workload sources: Recorder I/O traces

- Recorder is a multi-level (HDF5, MPI-IO, POSIX) I/O tracing tool
- For each I/O function, Recorder traces:
  - Functional parameters
  - Timestamp call began & duration of the call
  - Return code
- MPI-IO calls are also traced to give hints to the workload generator about which POSIX calls are issued as part of a collective I/O operation

```
1395246918.82860 MPI_Barrier (MPI_COMM_WORLD) 0 2.74235
1395246921.58050 open64 (/etc/romio-hints, 0, 0) -1 0.00812
1395246921.66498 open64 (/projects/SSSPPg/hluu//sample_dataset1k.h5part, 2, 0) 5 0.00045
1395246926.45888 MPI_Barrier () 0 0.00004
1395246926.46044 MPI_Barrier (MPI_COMM_WORLD) 0 0.00004
1395246926.53713 write (null, buf=0x1fa2b590b8, 16777216) 16777216 0.12372
1395246926.80371 write (null, buf=0x1fa2b590b8, 16777216) 16777216 0.11414
1395246926.98298 write (null, buf=0x1fa2b590b8, 16777216) 16777216 0.11806
1395246927.15213 write (null, buf=0x1fa2b590b8, 16777216) 16777216 0.11534
1395246927.36947 write (null, buf=0x1fa2b590b8, 16777216) 16777216 0.11358
```

[1] H. Luu et al. A multi-level approach for understanding I/O activity in HPC applications. In IEEE International Conference on Cluster Computing (CLUSTER), pages 1–5, 2013.

IOWA workload sources: CODES I/O kernels

- CODES is a highly parallel simulation toolkit for modeling exascale storage systems
  - Built on top of the ROSS optimistic DES
- CODES includes a domain-specific language for describing I/O workloads
  - Originally only used in CODES storage models
  - Includes I/O primitives, delay & synchronization mechanisms
  - Variable assignment, conditional, & loop constructs

```
sync g;
open f;

o = 0;
sync g;
if ((r % p) == 0)
{
    if ((r / p) < 512)
    {
        o = (0 * 68719476736) + (((r / p) * 8) * 16777216);
        writeat f, 16777216, o;
    };
};
```

[2] J. Cope et al. CODES: Enabling Co-design of Multilayer Exascale Storage Architectures. In Proceedings of the Workshop on Emerging Supercomputing Technologies 2011, ACM.

IOWA workload sources: Darshan I/O characterizations

- Darshan is a lightweight, I/O characterization tool for HPC applications
- For each accessed file, Darshan captures:
  - Counts of I/O operations at different layers (POSIX, MPI-IO, HDF5, PnetCDF)
  - I/O access information (histograms, common access sizes & strides)
  - Cumulative I/O timers and timestamps
- Per-file statistics recorded at each process, shared file records aggregated at shutdown
- Darshan enabled by default on production systems at the ALCF, NERSC, and NCSA
  - Breadth of HPC application I/O workloads in Darshan logs

```
#<rank> <file>                  <counter>          <value>          <name suffix>
-1      14818971734818452778    CP_POSIX_READS     0                .../vpicio_test.h5
-1      14818971734818452778    CP_POSIX_WRITES    133138           .../vpicio_test.h5
-1      14818971734818452778    CP_POSIX_OPENS     8193             .../vpicio_test.h5
-1      14818971734818452778    CP_POSIX_SEEKS     4180             .../vpicio_test.h5
-1      14818971734818452778    CP_BYTES_READ      0                .../vpicio_test.h5
-1      14818971734818452778    CP_BYTES_WRITTEN   2199023259968    .../vpicio_test.h5
-1      14818971734818452778    CP_MAX_BYTE_READ   0                .../vpicio_test.h5
-1      14818971734818452778    CP_MAX_BYTE_WRITTEN 2199023261831   .../vpicio_test.h5
```

[3] P. Carns et al. Understanding and improving computational science storage access through continuous characterization. Trans. Storage, 7:1–26, October 2011.

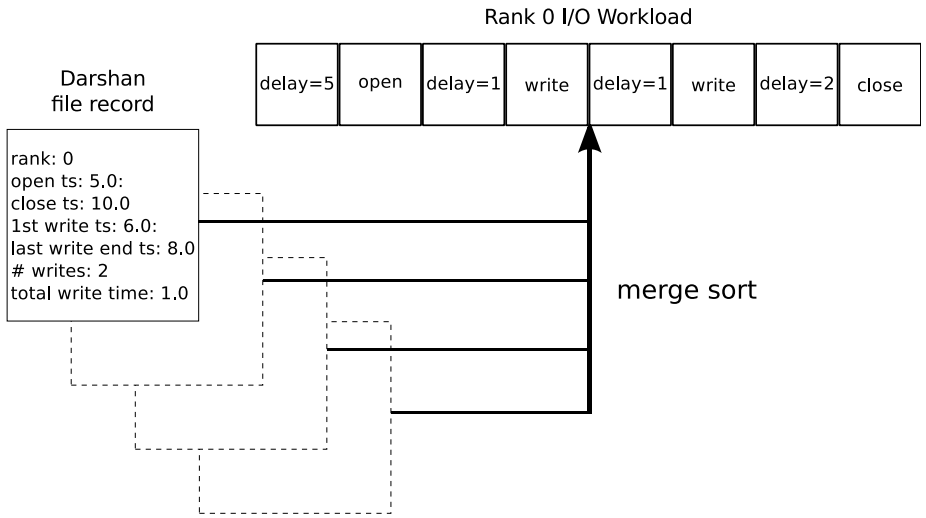
Generating workloads from Darshan logs

- Challenges:
  - The timespan in which I/O occurred is known, but not the complete timeline of I/O operations
  - Info on I/O access parameters limited to per-file histogram of access sizes and the most commonly occurring I/O sizes and strides
  - Shared file records are further collapsed into a single aggregate file record
    - Obscures individual ranks' roles in the shared file workload

- Approach:
  - Apply heuristics to classify the I/O strategy for each given file record:
    - Independent I/O to independent file
    - Independent I/O to shared file
    - Collective I/O to shared file
  - Formulate assumptions based on this classification to simplify regenerating the workload
    - E.g., mimicking a collective I/O algorithm when regenerating collective I/O workloads

Generating workloads from Darshan logs

- For each workload process, we:
  - Iterate Darshan's per-file records, generating workload operations which belong to this process
  - Operations from each file record are merged into an aggregate process workload



- Simplifying assumptions:
  - Constant sized delay between I/O operations, determined using observed idle time
  - Access sizes are assigned from common access sizes and default histogram bin sizes
  - Offsets assigned sequentially through a given file

- Shared file records are classified into 2 distinct cases: independent & collective I/O
  - round-robin strategy used in each case to evenly distribute I/O among processes performing the I/O
    - Independent I/O => all processes
    - Collective I/O => "aggregator" processes

Example use case I: storage system simulation

- One potential application of IOWA is to inject I/O workloads into storage system simulations
  - Could be used to analyze storage architecture/algorithm models with I/O workloads of interest

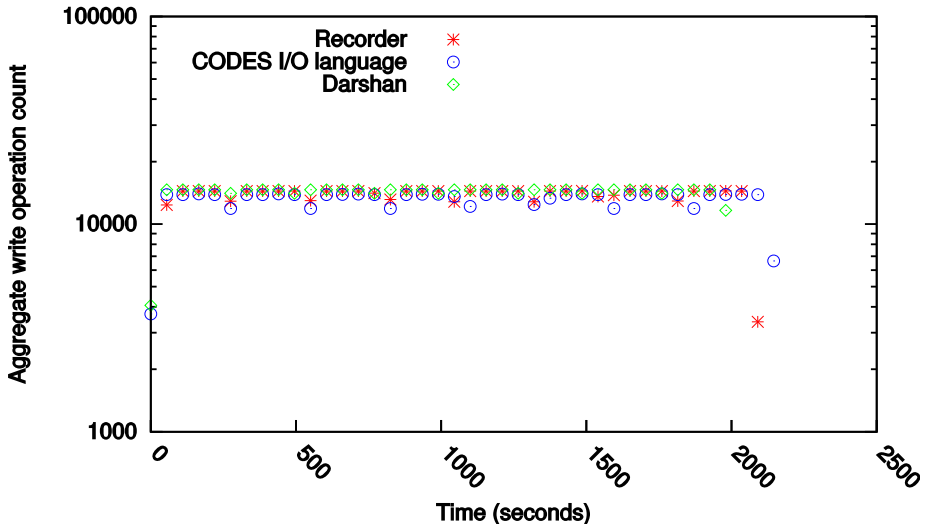
- We integrated IOWA into an existing CODES model of Intrepid, a decommissioned IBM BG/P system at the ALCF
  - Model includes major components of the BG/P architecture: compute nodes, I/O nodes, file servers, storage devices, and interconnects
    - Compute node components modified to interface with IOWA for obtaining I/O workloads

- We used this model to compare the execution of each of the IOWA workload generators' representations of the VPIC-IO workload
  - VPIC-IO is an I/O kernel of the VPIC plasma physics simulation code
  - VPIC-IO leverages HDF5 collectives to write time-varying datasets to file

Example use case I: storage system simulation

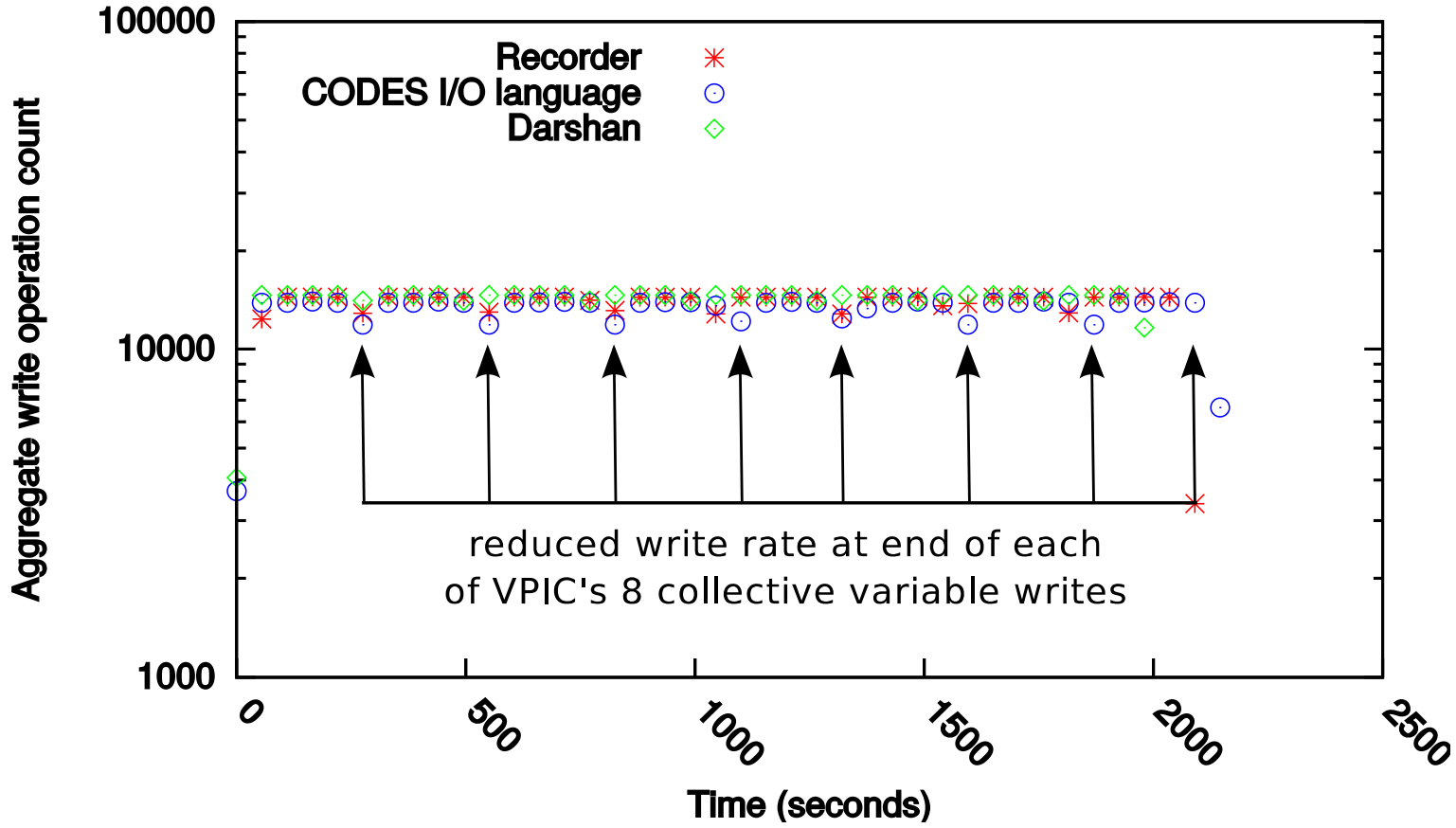
- IOWA workload sources for VPIC-IO workload obtained as follows:
  - Recorder traces and Darshan logs using link-time instrumentation on Mira (BG/Q system @ the ALCF)
  - CODES I/O kernels crafted manually

- Figure shows aggregate write operation counts over 40 distinct intervals using each IOWA generator's model of the VPIC-IO workload (workload size = 8K ranks)
  - Why do Recorder and CODES I/O language workloads experience reduced write rate?



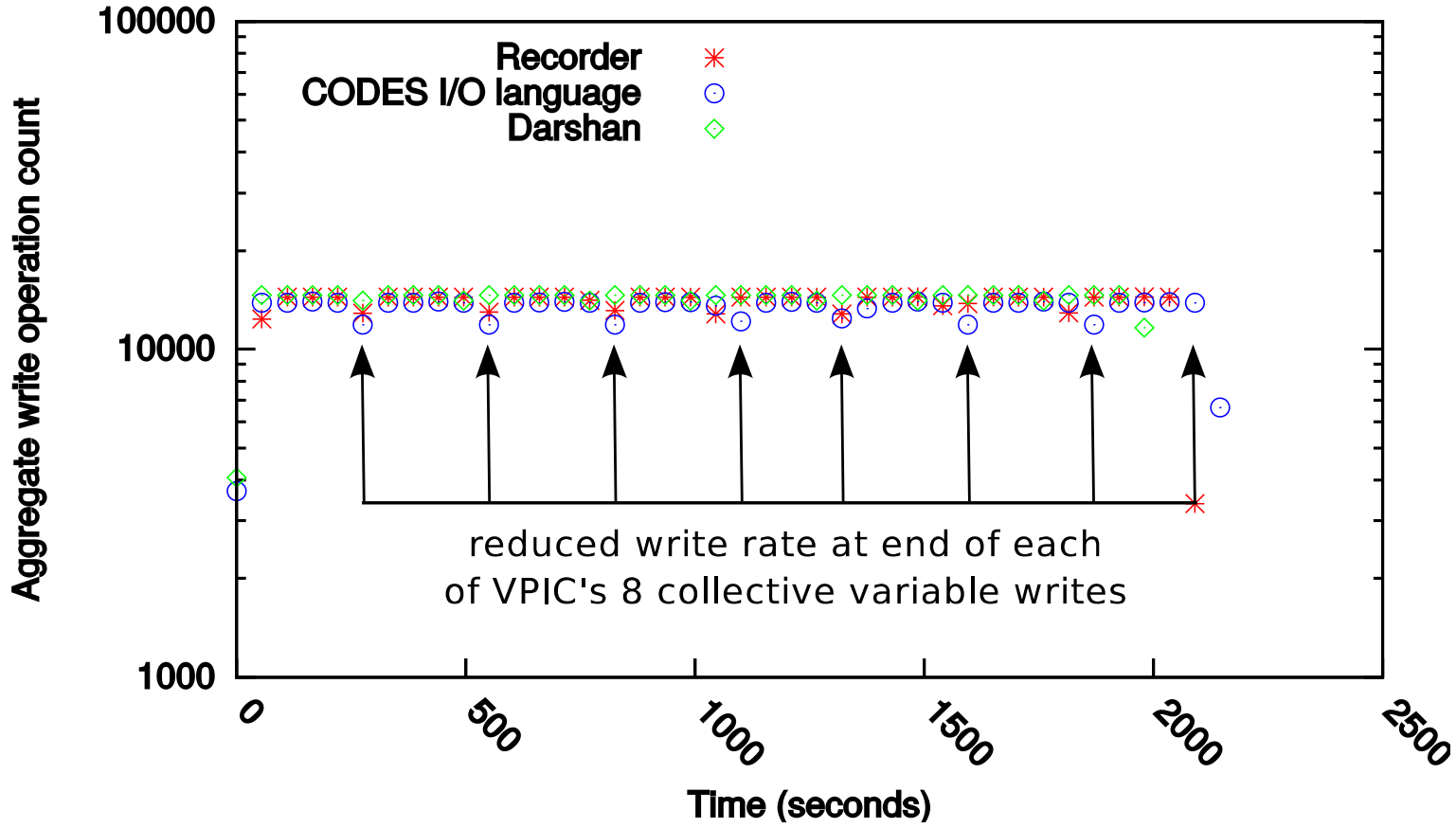
Snyder et al. @ PMBS'15

Example use case I: storage system simulation



- Recorder generates delays exactly as described in trace files, possibly reproducing runtime anomalies (e.g., a straggling process)

Example use case I: storage system simulation



- CODES I/O language modeling of two-phase collective I/O results in idle "aggregator" processes in the final round of each collective

Example use case II: storage system I/O replay

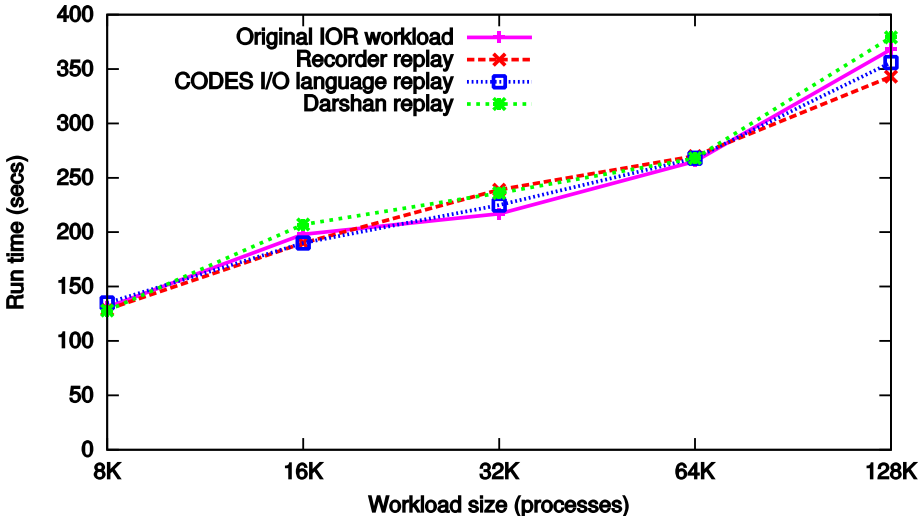
- Another useful application of IOWA is for replaying I/O workloads on real HPC systems
  - Could be used to analyze a workload's performance on a new platform (without need for compiling/configuring/executing the application on this system)

- We developed an MPI-based I/O replay tool that interfaces with IOWA to replay arbitrary workloads on a real system
  - POSIX calls used to replay IOWA I/O operations
  - MPI_Barrier() used to replay IOWA synchronization operations
  - High-resolution sleep function used to replay delays

- We then used this replay tool to compare the performance of a real I/O workload to each IOWA generator's model of the workload on Mira
  - Again, Recorder traces and Darshan logs obtained using link-time instrumentation on Mira, CODES I/O kernels crafted manually

Example use case II: storage system I/O replay

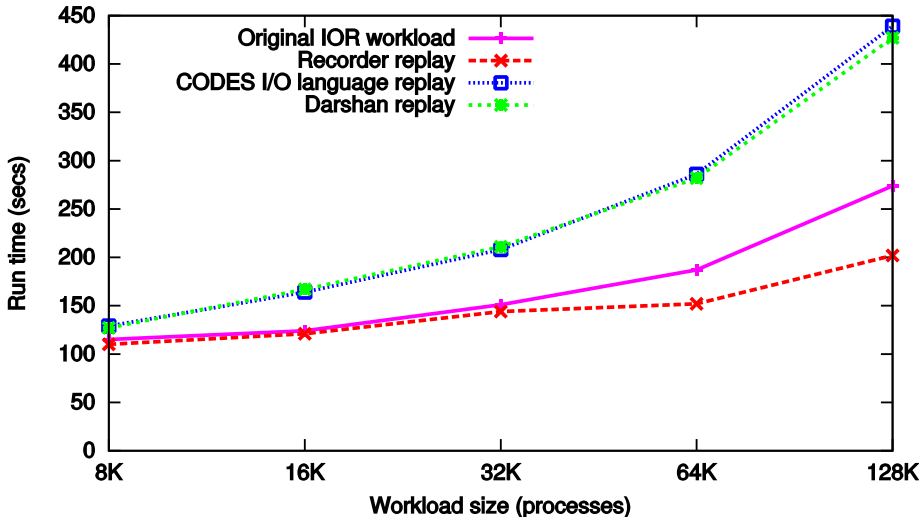
- We use a simple independent checkpointing I/O workload as a proof of concept of the IOWA design
  - This type of workload is common in HPC applications that use the checkpoint-restart model for resilience
- Figure shows the run time of the original workload and each generator's representation using our MPI replay tool on Mira
  - Workload scaled from 8K-128K application processes



- Each example obtains comparable performance on this workload, with no more than 10% error in any case
  - This workload is straightforward enough to be reproducible using any of the IOWA workload generators

Workload modeling challenges: collective I/O

- Figure shows the performance of each IOWA generator at reproducing a shared-file checkpointing workload rather than independent
  - Uses MPI-IO two-phase collective I/O algorithm

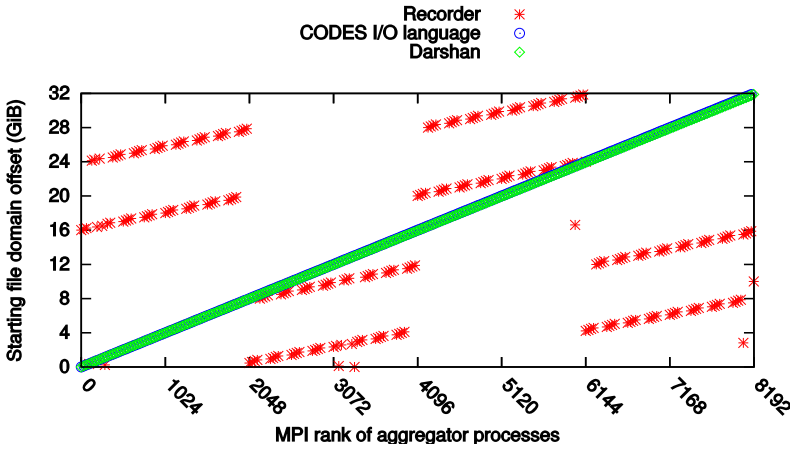


- Recorder workload generator exhibits up to a 25% decrease in runtime compared to the target IOR workload at larger scales?
  - We believe this is an artifact of modeling collective communication with a barrier
    - A barrier is likely inadequate at modeling the cost of large-scale collective communication in the two-phase I/O algorithm

- Why do the Darshan & CODES I/O language generators exhibit such poor performance (up to 55% increase in runtime)?

Workload modeling challenges: collective I/O

- Figure shows the distribution of “file domains” among “aggregators” for the IOR collective I/O workload
  - Gives us indication of which parts of the file were accessed by which aggregators



- Darshan & CODES I/O language generators simply assign file domains to aggregators sequentially through the file
  - Mira’s MPI-IO driver prefers special “bridge” nodes as aggregators
  - File domains aligned to GPFS lock boundaries
  - Recorder traces embed this data, but is difficult to construct for CODES I/O kernels and Darshan I/O characterizations

- **Best practice:** accurately modeling collective I/O typically will require accounting for platform-specific optimizations and topology details
  - Capturing and replaying workloads at a higher layer likely preferable on real systems

Snyder et al. @ PMBS’15

Conclusions

- We have designed IOWA, an I/O workload abstraction offering I/O researchers flexibility in choosing different workload generation methods
  - Allows researchers to choose appropriate workloads based on the study being performed and the resources available

- We also evaluated the relative merits of 3 distinct IOWA workload sources:
  - **Recorder I/O traces** are most accurate, but at cost of size and ease of modifying workload characteristics
  - **CODES I/O kernels** are most flexible, but can be cumbersome to develop
  - **Darshan I/O characterizations** are small and offer access to a breadth of HPC workloads, but at a cost of workload accuracy

- Accurately modeling HPC I/O workloads is a difficult problem, providing many non-obvious challenges to I/O researchers
  - E.g., modeling I/O workloads at POSIX layer is enticing for workload portability, but complicates modeling high-level collective I/O workloads

Acknowledgements

- Thank you for your time!
- IOWA software is available as part of the CODES project:
  - http://www.mcs.anl.gov/research/projects/codes/
- Questions??

This material is based on work supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computer Research Program under contract DE-AC02-06CH11357. The research used resources of the Argonne Leadership Computing Facility at Argonne National Laboratory, which is a DOE Office of Science User Facility.