Replicating HPC I/O Workloads With Proxy Applications

James Dickson, Steven Wright, Stephen Jarvis - University of Warwick

Satheesh Maheswaran, Andy Herdman - UK Atomic Weapons Establishment

Marc C. Miller - Lawrence Livermore National Laboratory

Motivation

- I/O investigation goals?
  - Benchmarking systems
  - Tuning application behaviour
  - Tuning software stack
  - Changing paradigm
  - Changing hardware technology

Motivation

- Working with a mini application or proxy is less cumbersome and more streamlined, not to mention more portable
- Developing and maintaining a representative proxy for every application is time consuming and probably redundant
- Ideally we would like to experiment while minimising time spent making code changes and writing new implementations

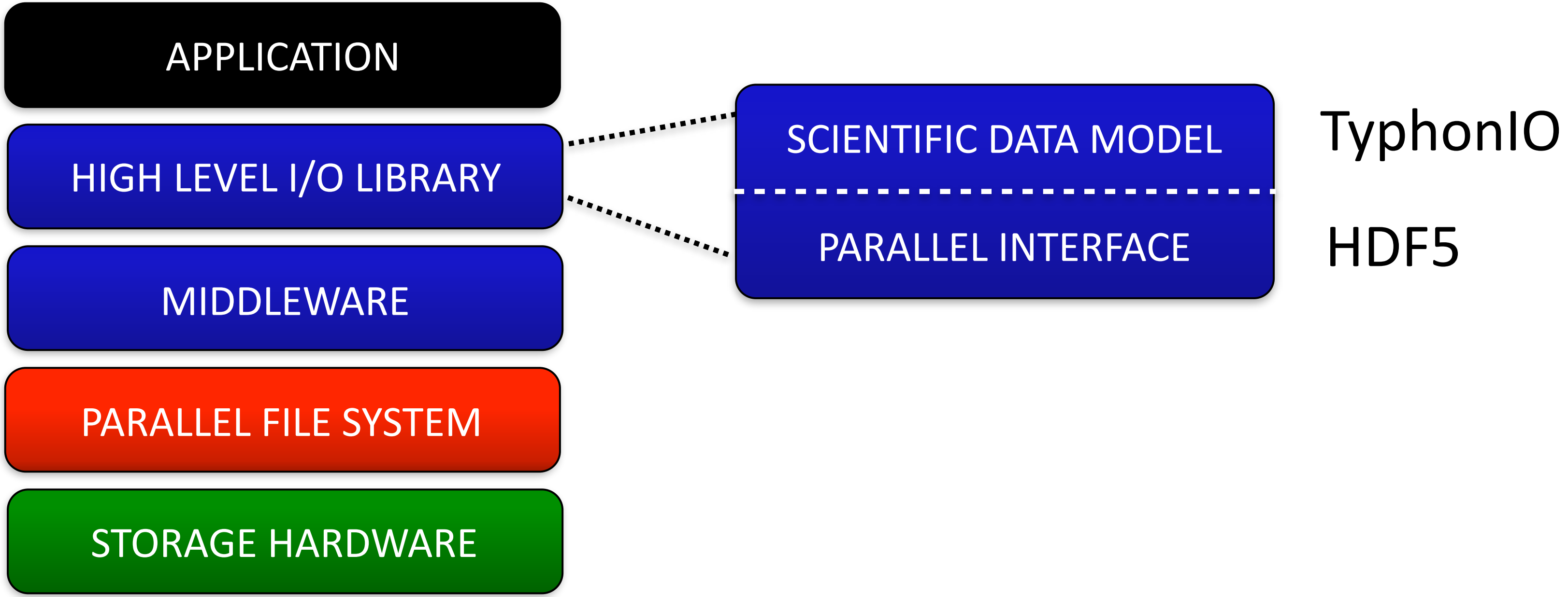
Outline

- Background: Proxy app and I/O library
- Replication Components
- Case Study
- Conclusion

Background: MACSio

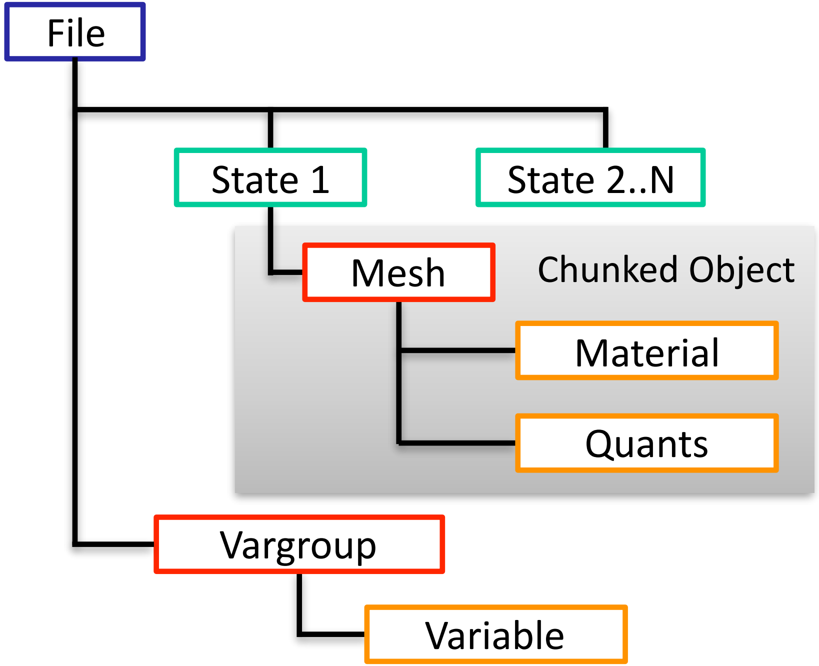
- "Multi-purpose Application-Centric, Scalable I/O Proxy Application"
- Two key characteristics:
  - Level of Abstraction: POSIX, MPI-IO, SILO, HDF5 and beyond…
  - Degree of Flexibility: dump type, dataset composition, user defined data objects
- Multi-purpose achieved through plugin based design, if you have a library or interface to work with, write a plugin!

Background: TyphonIO



Background: TyphonIO

- Overlays a hierarchical data model on the parallel I/O interface
- Designed to use HDF5 in a consistent way that can be optimised for the data model, e.g. efficient use of chunking in the mesh structure

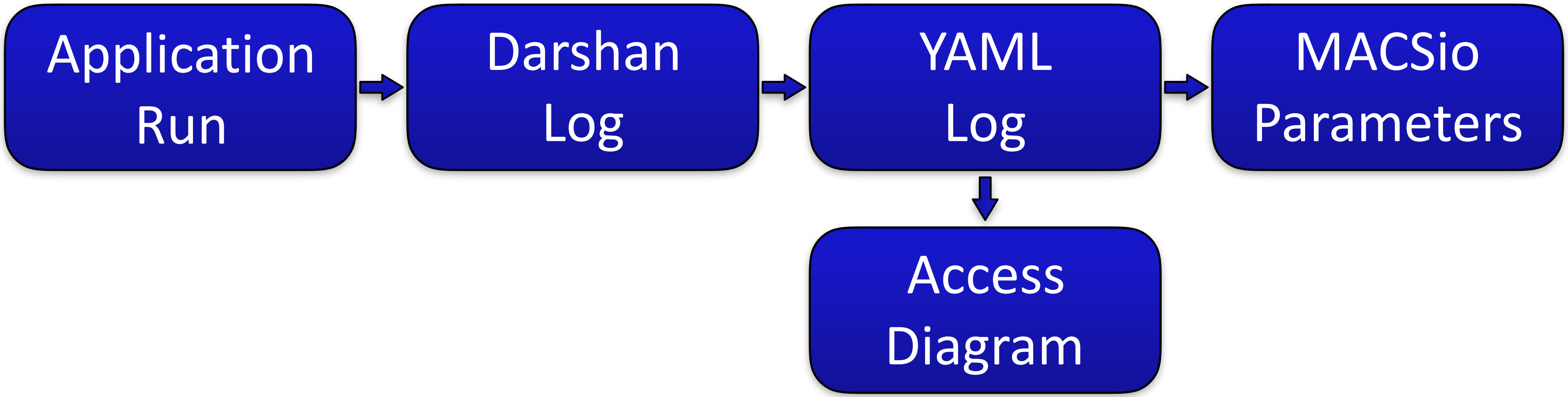


Replication: Profiling

- Darshan I/O characterisation chosen for lightweight profiling
- Instrumentation overhead indistinguishable from machine noise in our experiments
- Profiling produces counters for POSIX, MPI-IO, HDF5

Runtime (seconds)	1 Node	64 Nodes
Uninstrumented	309.25	352.33
Instrumented	307.43	352.29

Replication: Parameter Generation



Replication: Parameter Generation

Filesize =

$$\text{Processors} \left(\text{PartSize} \left(\alpha \text{ Variables} + \beta \right) + \gamma \text{ Variables} + \delta \right) + \psi \text{ Variables} + \eta$$

- MACSio currently weak scales, so increasing processor count increases the file size linearly
- Similarly, part size and dataset variable count give a linear increase in total bytes written
- Combining the linear equations gives the equation above to calculate a good estimate for the resultant file size based on dataset composition
- Constants α, β, γ, δ, ψ, η are derived experimentally from a dataset composition scaling study

Replication: Parameter Generation

bookleaf_1536_3880532.darshan.gz.yaml ×


```yaml
- CHARACTERISTICS:
    ACCESS_MODE: w
    MPIIO_OPEN_TIME: 133.37006399999999
    MPIIO_READ_TIME: 0.0
    MPIIO_WRITE_TIME: 58.72207
    POSIX_OPEN_TIME: 133.36487499999998
    POSIX_READ_TIME: 0.0
    POSIX_WRITE_TIME: 133.302945
  FILE: initial_dump.h5
  MPI_COUNTERS:
    MPIIO_ACCESS1_ACCESS: '10672'
    MPIIO_ACCESS1_COUNT: '1280'
    MPIIO_ACCESS2_ACCESS: '10080'
    MPIIO_ACCESS2_COUNT: '1280'
    MPIIO_ACCESS3_ACCESS: '5336'
    MPIIO_ACCESS3_COUNT: '2302'
    MPIIO_ACCESS4_ACCESS: '5152'
    MPIIO_ACCESS4_COUNT: '612'
    MPIIO_BYTES_READ: '0'
    MPIIO_BYTES_WRITTEN: '133534106'
    MPIIO_COLL_OPENS: '1536'
    MPIIO_COLL_READS: '0'
    MPIIO_COLL_WRITES: '0'
    MPIIO_FASTEST_RANK: '1450'
    MPIIO_FASTEST_RANK_BYTES: '63720'
    MPIIO_F_CLOSE_TIMESTAMP: '186.644951'
```


- Extracting counters such as BYTES_WRITTEN, NUM_PROCS, COLL_WRITES, [OPEN/CLOSE]_TIMESTAMP is enough to generate an input to MACSio for a similar dataset composition and I/O pattern
- In particular, using timestamps to distribute load across the simulation runtime is important to give an accurate representation of typical ‘bursty’ I/O hotspots spread out across runtime

Case Study: Bookleaf

- 2D unstructured Lagrangian hydrodynamics application
- Fixed checkpoint scheme: two per simulation
- The input deck used solves the Noh verification problem for ideal gases
- I/O is handled by TyphonIO

Experimental Setup

- ARCHER
  - 4920 node CRAY XC30
  - Two 12-core Ivy Bridge processors per node (118,080 cores total)
  - Three Lustre filesystems:
    - 12 OSSs
    - 4 OSTs/OSS
    - 10 4TB Discs/OST (RAID6)
    - 1 MDS + 1 MDT with 14 600GB discs (RAID1+0)
    - 10 LNet Router nodes with overlapping routing paths

archer

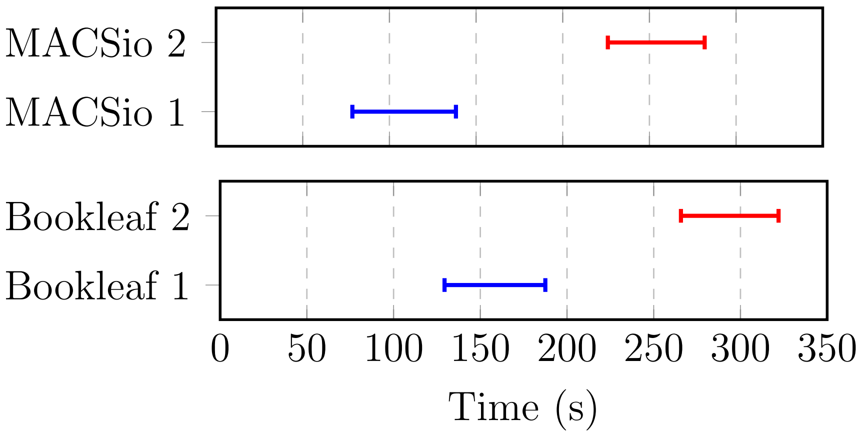
Experimental Setup: Input Parameters

- Part size represents the volume of data written from each rank
- Wait time is a basic time buffer between consecutive file accesses

Nodes	Part Size (Bytes)	Wait Time (s)
1	404 320	266
2	202 205	120
4	101 148	53
8	50 619	22
16	25 355	11
32	12 723	7
64	6407	5

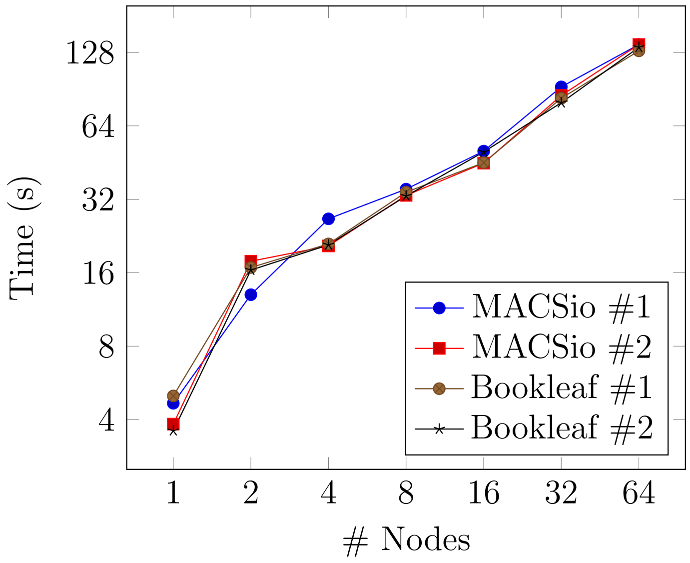
File Access Pattern

- File access times are offset by the initial setup in Bookleaf
- Accounting for this overhead is not necessary to accurately represent the I/O pattern so we don't factor it in, but this could easily be introduced

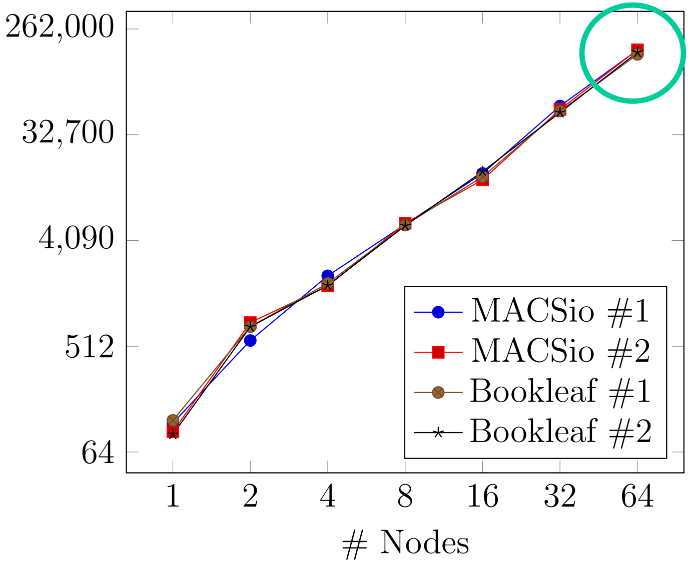


Results: I/O Time

Absolute I/O Time



Cumulative I/O Time across all ranks



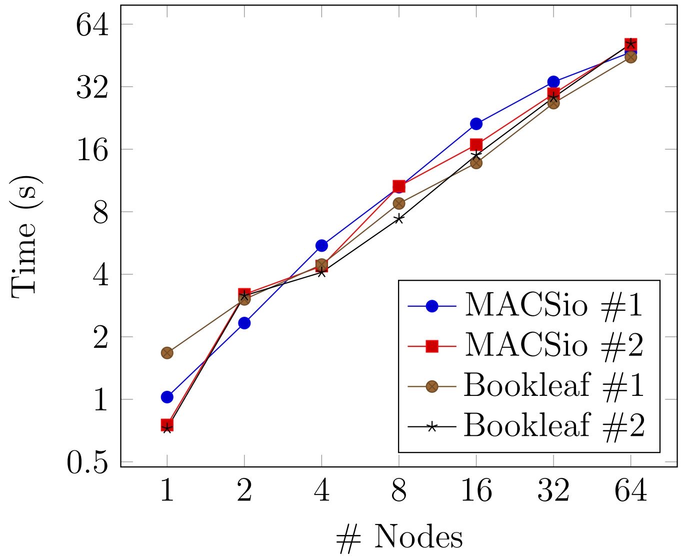
17,000s
1,536 ranks

≈ 110s writing per rank

Results: I/O Time

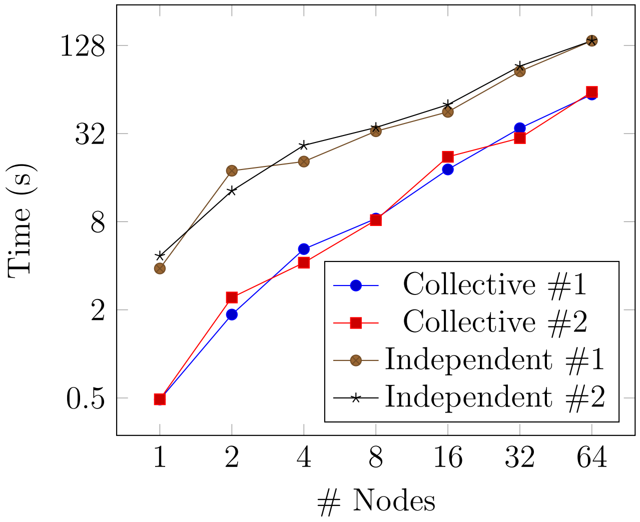
- Total, cumulative and slowest individual I/O time remain consistent for the original and replicated runs
- Looking at a wider range of Darshan counters, access sizes and frequencies are also consistent

Slowest Individual MPIIO Operation



Results: Testing Independent vs Collective I/O with MACSio

- Using the MACSio replication, a parameter tweak can be used to manipulate I/O library behaviour
- The switch to use collective buffering has a very predictable effect, reducing the number of small write operations and lowering the overall I/O time



Conclusion

- We use a proxy application and high level library to mimic an I/O pattern based off as lightweight profiling as possible
- I/O characterisation and a small amount of application familiarity is enough to produce a proxy that is workable
- Once a parameter set has been identified, we can chop and change strategy, library and platform with a reasonable amount of simplicity

Next Steps

- More irregular I/O patterns from range of applications
- Exercise different parallel interfaces
- Multiple concurrent workloads

Acknowledgements

- UK Atomic Weapons Establishment Technical Outreach Programme

- UK Engineering and Physical Sciences Research Council

Thank You
Any Questions?

J.Dickson@warwick.ac.uk