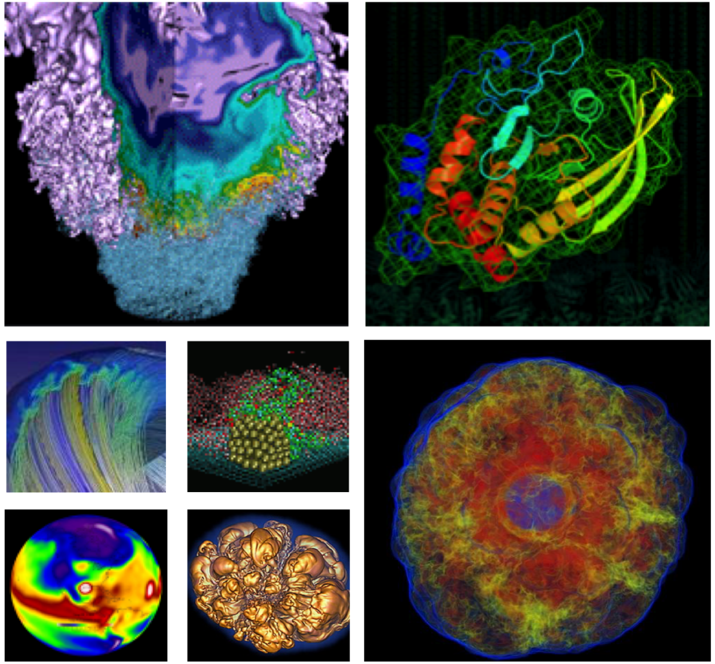
H5Prov: I/O Performance Analysis of Science Applications Using HDF5 File-level Provenance



Tonglin Li, Quincey Koziol, Houjun Tang, Jialin Liu, Suren Byna

Cray User Group 2019, Montréal, Canada

U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

Outline

- Motivation
- HDF5 in DOE labs
- Virtual Object Layer (VOL)
- Provenance VOL: H5Prov
- Design and implementation
- Experiment setup
- Provenance overhead
- Provenance trace analysis

Complex systems are hard to understand!

- More layers introduced to the HPC storage system hierarchy
- Much more complex center-wide I/O behaviors and performance issues
- Current I/O profiling tools:
  - Darshan (ANL): application I/O (from outside)
  - TOKIO (LBL): combining multiple infrastructure levels
    - Component-level monitoring logs
    - Topology related info (Slurm logs, Cray SDB)
    - Application I/O (Darshan)
    - Filesystem load (LMT)

The missing piece: I/O tracing inside the applications

- Current provenance and profiling/logging systems skips **intra-application** level I/O behaviors for a reason (or two):
  - Hard to find a generic way to collect from different applications
  - Need to insert code to applications: changing code is painful!

Our approach:
Tracking at the I/O middle-ware level

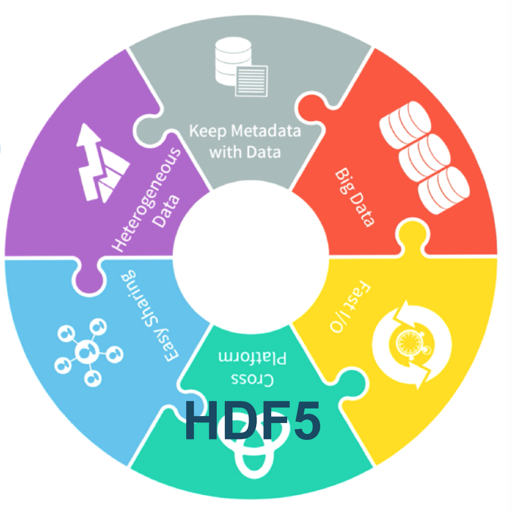
- H5Prov: A provenance logging system within HDF5
  - Application data semantic is visible
  - Non-invasive: no code change
  - Generic to all HDF5 applications (they are a lot!)

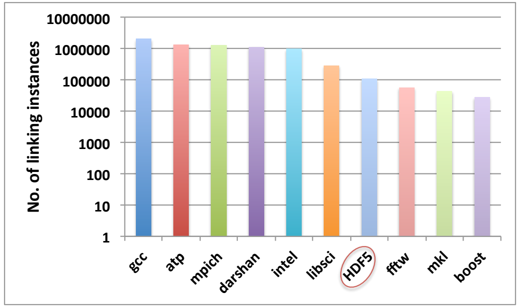
HDF5

- HDF5 is designed to organize, store, discover, access, analyze, share, and preserve diverse, complex data in continuously evolving heterogeneous computing and storage environments.
- For every size and type of system: several KB ~ TB
- Imagine a filesystem in a file:
  - A H5 file as a root directory, contains
    - Groups (subdirectories)
    - Datasets (data files)

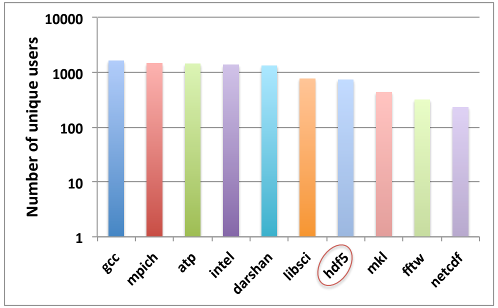
Usage of HDF5 at DOE Labs

- NASA/NOAA satellite data (Aura, JPSS-1, etc.)
  - Highest Technology Readiness Level (TRL 9) - “Flight proven” through successful mission operations
- Heavily used on DOE supercomputing systems

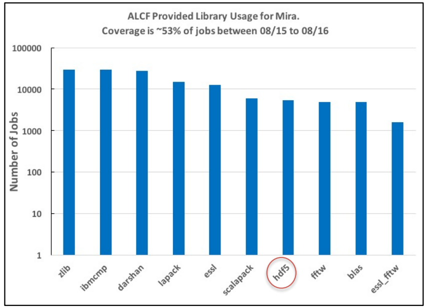




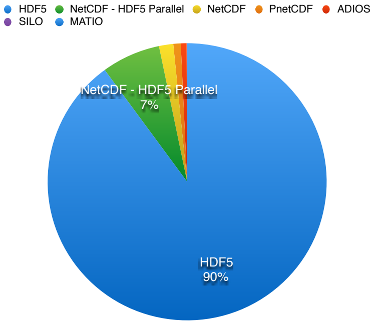
a. Number of linking instances on Edison (NERSC)



b. Number of unique users on Edison (NERSC)



c. Number of linking instances on Mira (ALCF)



OLCF I/O libraries

Virtual Object Layer (VOL)

- VOL: an abstract layer intercepts object level operations, customized implementation of VOL interface.
- Provide the same HDF5 data model and API, but allow different storage solution.
- Examples:
  - Openstack Swift
  - Ceph Rados
  - and Intel DAOS

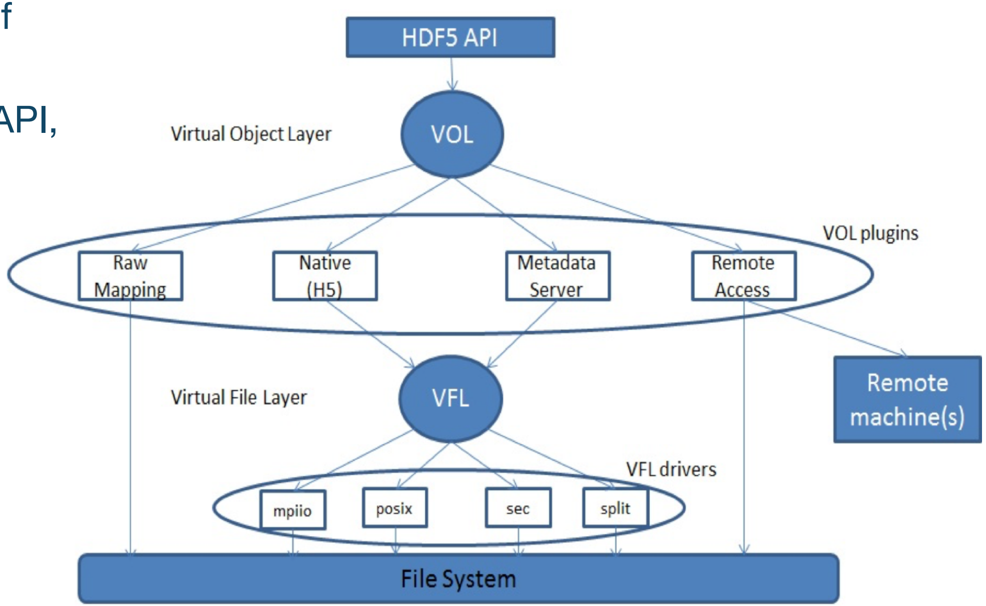
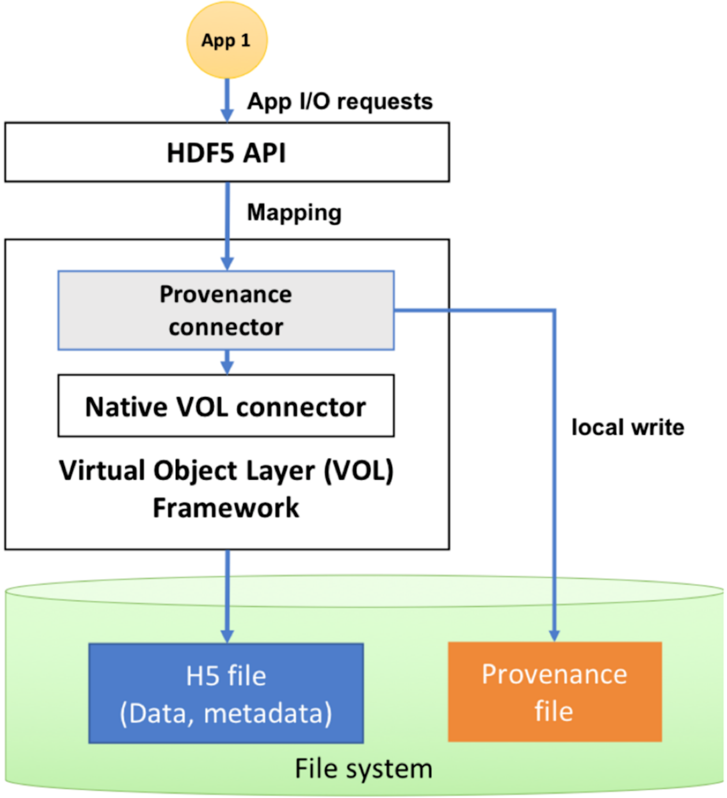


Figure courtesy of HDF Group

Provenance VOL: H5Prov

- Timer on
  - Unwrap request (and object) and get context
  - Pass to native calls
    - Native call execution
  - Update metadata and prepare new context
  - Wrap native return/result with context
  - Write provenance
- Timer off
- Return wrapped object



Features

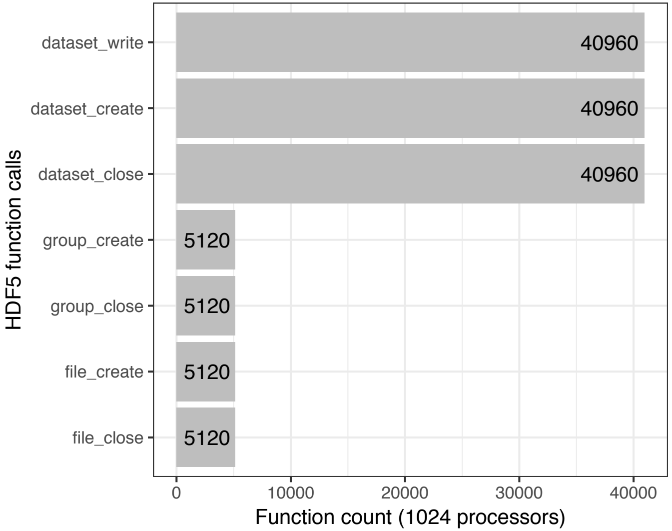
- What to capture
  - Runtime info: user name, process ID, thread ID, MPI rank, etc.
  - HDF5 function name, start time, lasting time
  - Statistics:
    - Accessed/created dataset, group count, etc.
    - Read/write size, etc.
- Non-invasive deployment
  - Setup a ENV variable to enable/disable
  - No change of application code

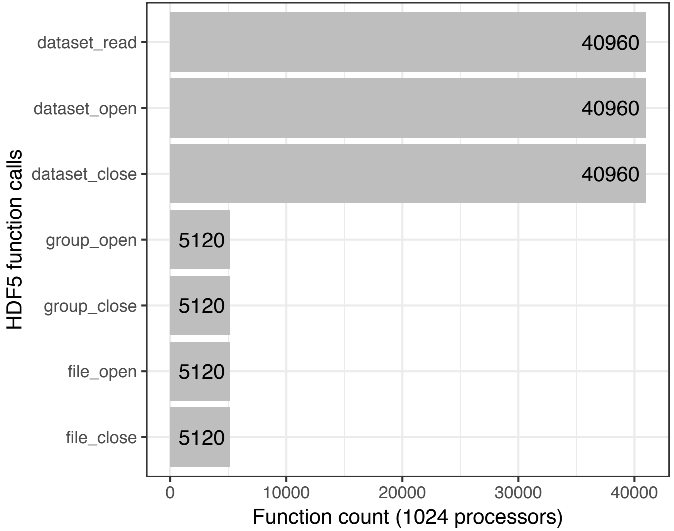
Experiment setup

- Test-bed: Cori@NERSC
  - 2-128 Haswell nodes
  - 32 physical cores and 128GB RAM per node
  - 64-4096 MPI ranks
- Storage system configuration
  - Lustre stripe count 64 and 128, stripe size of 16MB
  - Burst-Buffer
- Benchmark
  - VPIC-IO
  - BDCATS-IO

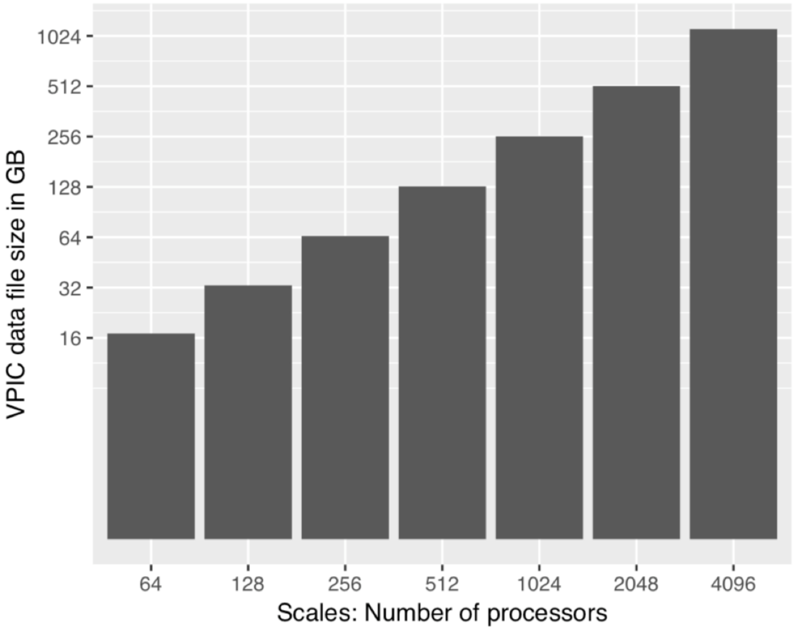
Invested HDF5 functions

- `file_create`
- `file_open`
- `file_close`
- `group_create`
- `group_open`
- `group_close`
- `dataset_create`
- `dataset_open`
- `dataset_close`
- `dataset_read`
- `dataset_write`

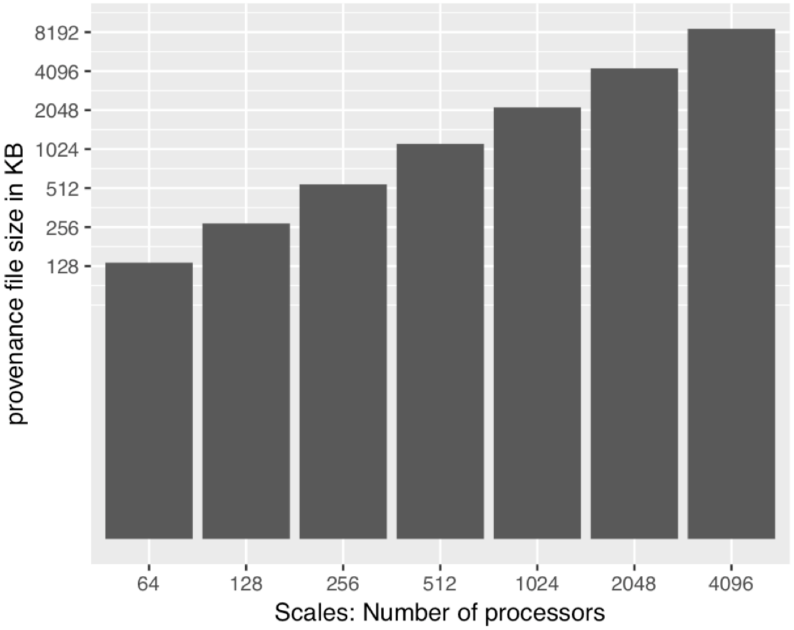




H5Prov overhead: trace file sizes

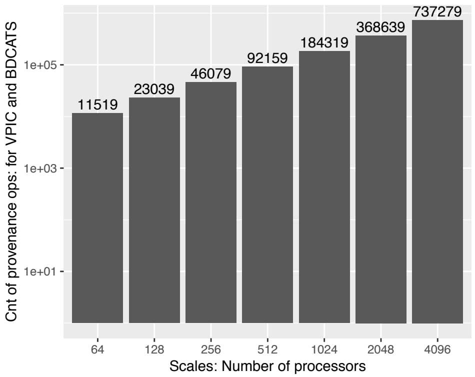


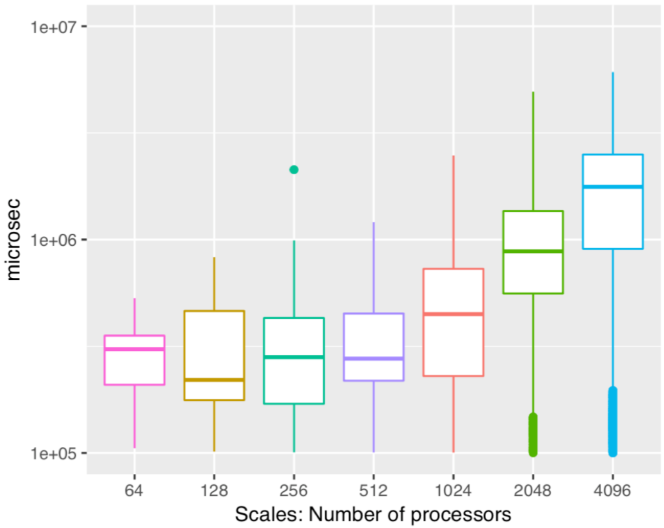
(a) Accessed data size



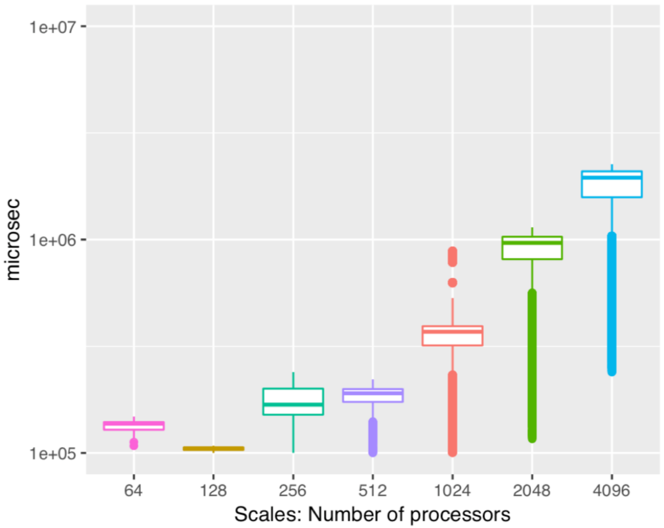
(b) H5Prov trace file sizes

H5Prov overhead: time consumption





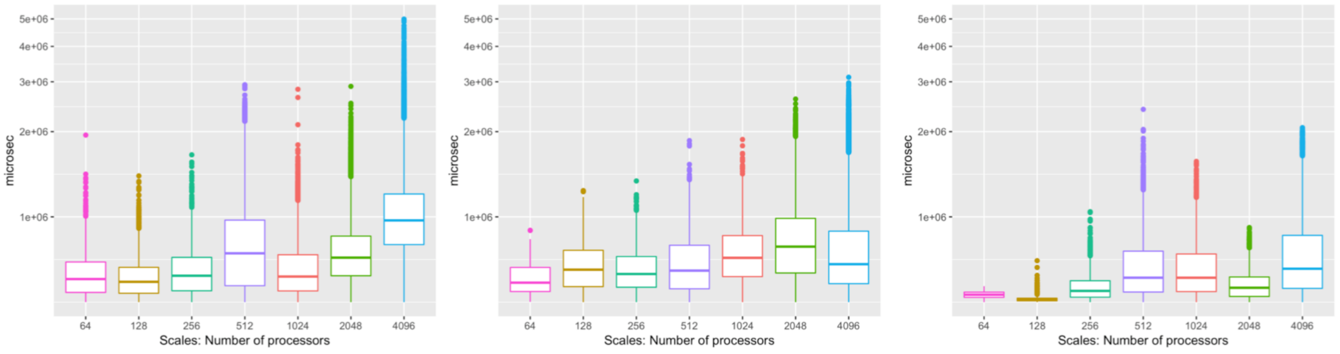
(a) Lustre: stripe count = 128



(b) Burst-buffer

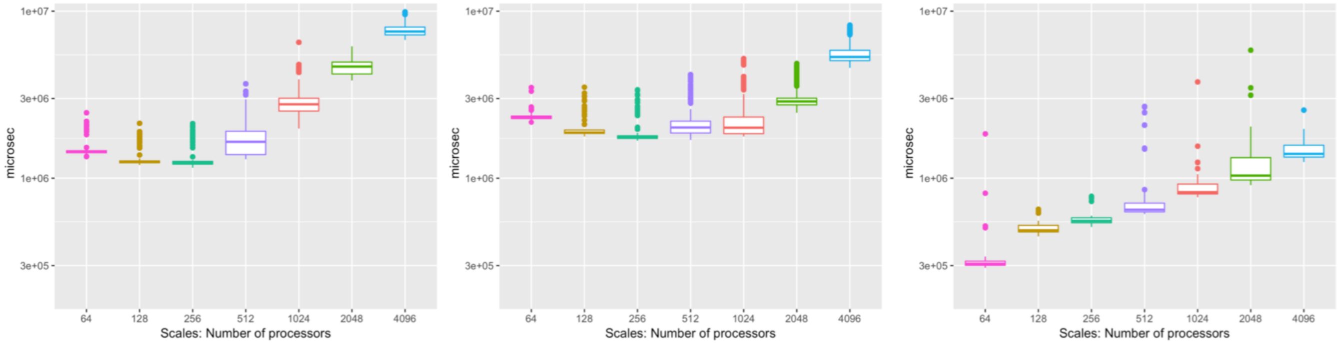
Trace sample: Dataset read and write

- Burst-buffer helps read and write differently
  - Read/write
  - Latency numbers
  - Variance
  - Scalability



(a) Lustre: stripe count = 64 (b) Lustre: stripe count = 128 (c) Burst-buffer

Figure 5: Dataset read scaling: BDCATS

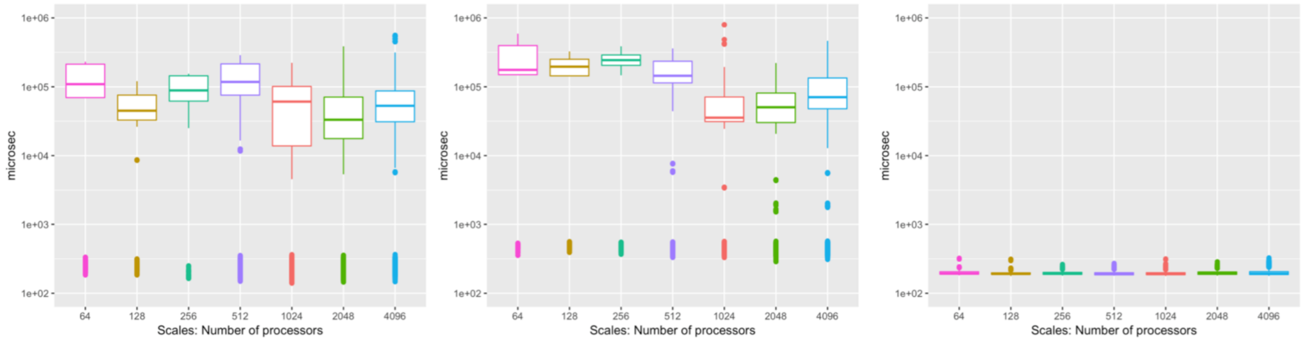


(a) Lustre: stripe count = 64 (b) Lustre: stripe count = 128 (c) Burst-buffer

Figure 6: Dataset write scaling: VPIC

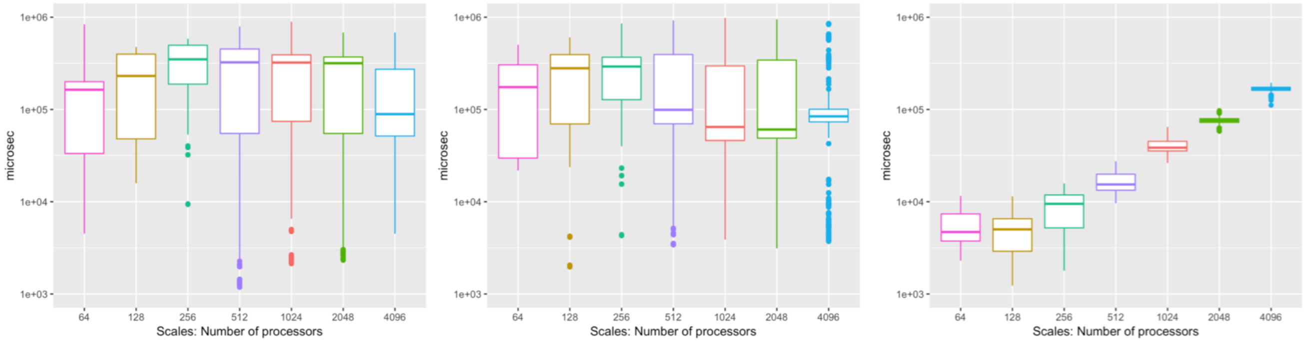
Trace sample: Group create and open

Slower open than to create?



(a) Lustre: stripe count = 64 (b) Lustre: stripe count = 128 (c) Burst-buffer

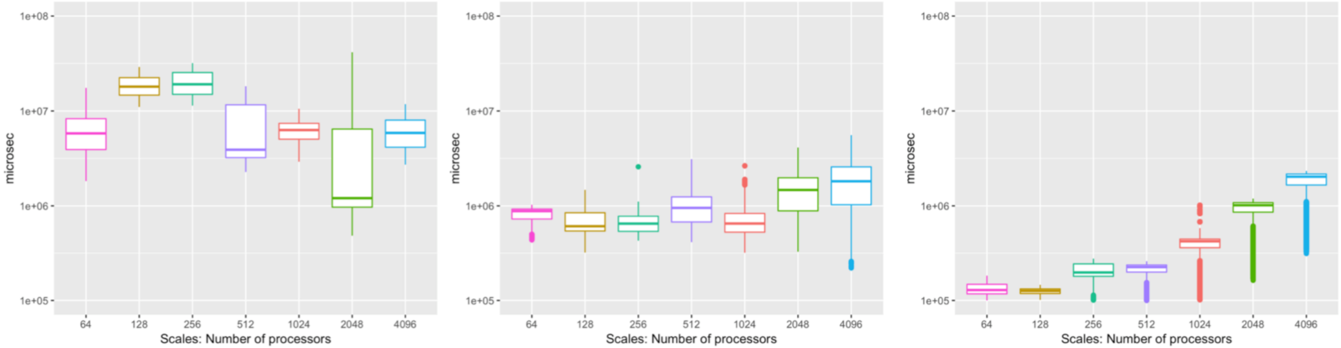
Figure 11: Group create scaling: VPIC



(a) Lustre: stripe count = 64 (b) Lustre: stripe count = 128 (c) Burst-buffer

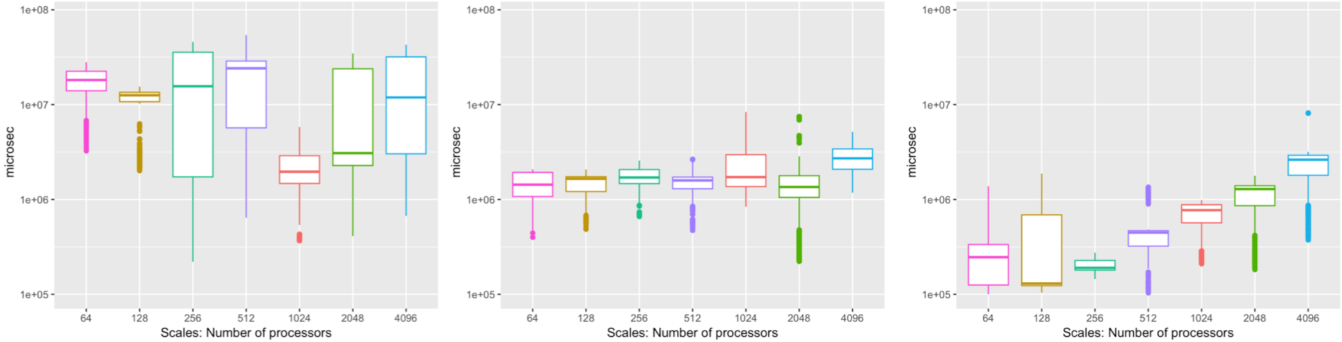
Figure 12: Group open scaling: BDCATS

Trace sample: File create and open



(a) Lustre: stripe count = 64 (b) Lustre: stripe count = 128 (c) Burst-buffer

Figure 15: File open scaling: BDCATS

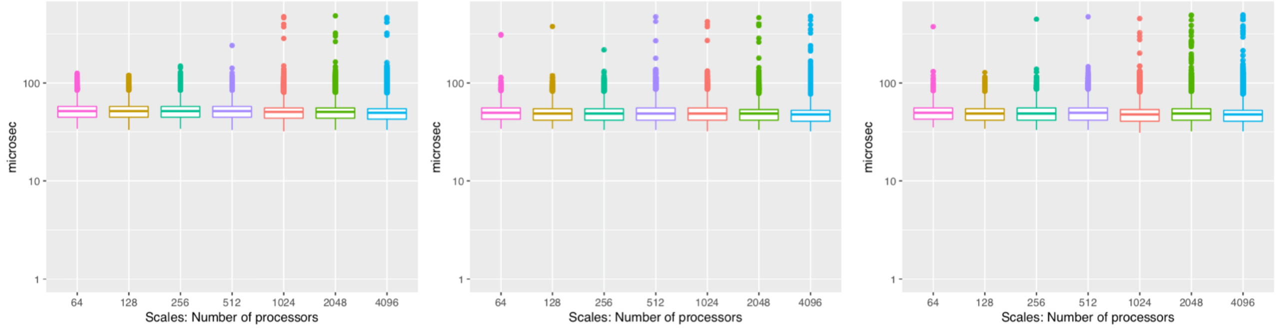


(a) Lustre: stripe count = 64 (b) Lustre: stripe count = 128 (c) Burst-buffer

Figure 16: File create scaling: VPIC

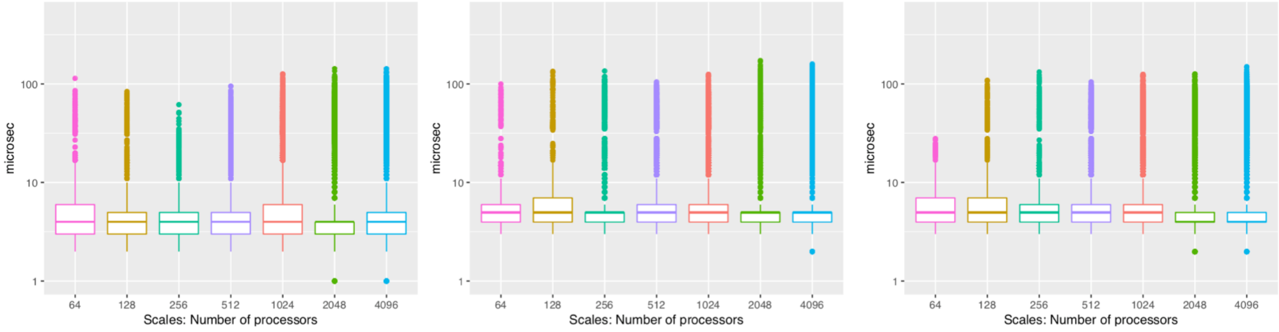
Something odd…

- VPIC: read/write
- BDCATS: read-only



(a) Luster: stripe count = 64 (b) Luster: stripe count = 128 (c) Burst-buffer

Figure 9: Dataset close scaling: BDCATS



(a) Luster: stripe count = 64 (b) Luster: stripe count = 128 (c) Burst-buffer

Figure 8: Dataset close scaling: VPIC

Future work

- Optimization on staging provenance trace files
- Extend capturing coverage
- Add parallel support
- Data mining/machine learning for I/O pattern discovery

Conclusion

- H5Prov provides provenance with application in mind
- Low overhead
- High scalability
- Non-invasive deployment
- Fine granularity

Thank You