I/O CHARACTERISTICS OF SCIENTIFIC APPLICATIONS

CHEN WANG[1], KATHRYN MOHROR[2], MARC SNIR[1]
University of Illinois at Urbana-Champaign[1], Lawrence Livermore National Laboratory[2]

MOTIVATION

Most widely deployed parallel file systems (PFS) support POSIX semantics. The fundamental problem behind the performance issues stemming from POSIX semantics is that the PFS is ignorant of application synchronization logic and the happens-before order of concurrent I/O operations; the PFS must make worse-case assumptions and serialize all potentially conflicting I/O operations. It is a common believe that most HPC applications do not require strict POSIX semantics. However, there has been no experimental result that confirms this claim.

KEY QUESTIONS

- What are the common I/O characteristics of HPC applications?
- How are files accessed? Sequentially or in a strided manner? Do applications perform write-after-write or read-after-write?
- What metadata operations are used? Is it possible to relax metadata operations?
- Do HPC applications require POSIX semantics? Given an application, can we decide if it will run correctly on a file system with weaker consistency models?

WEAK CONSISTENCY SEMANTICS

Commit Consistency Semantics. We define *commit consistency semantics* as a less strict consistency model, where "commit" operations are explicitly executed by processes, and I/O updates performed by a process to a file before a commit become globally visible upon return of the commit operation. Many user-level and Burst Buffer (BB) PFSs (e.g., BSCFS and UnifyFS) provide commit consistency semantics. Note that the "commit" operation is system-specific. For example, in UnifyFS, a commit can be performed with an `fsync` operation which makes writes performed by an individual process globally visible. A `close()` call usually also has the effect of a commit.

Session Consistency Semantics. We define *session consistency semantics* as semantics that guarantee writes by a process are visible to another process when the modified file is closed by the writing process and subsequently opened by the reading process, with the `close` happening before the `open`. Commonly known as close-to-open semantics, several PFSs implement this model including NFS and Gfarm/BB. The major difference between session semantics and commit semantics is when the writes become visible to other processes. In commit semantics, updates become globally visible after a commit operation by the writer. In session semantics one needs a pair of operations, one executed by the writer and the other by the reader.

DETECTING CONFLICTS IN WEAK CONSISTENCY SEMANTICS

Conflicting accesses can occur when two I/O operations access the same location of a file. We call this situation an *overlap*. Overlaps can cause conflicts in four cases.

- RAW-[S|D]: read-after-write by the same process (S) or by different processes (D).
- WAW-[S|D]: write-after-write by the same process (S) or by different processes (D).

These four cases are *potential conflicts*. Whether they are actual conflicts depends on the PFS semantics. In the majority of PFSs, conflicting accesses by the same process will take effect in the right order so that only RAW-D and WAW-D are potentially problematic. Note that a write-after-read pair cannot cause a conflict, as we assume conflicting operations are properly synchronized so the read will complete before the write starts.

We denote each I/O operation as a tuple $(t, r, os, oe, type)$, where t is the entry timestamp, r is the rank of the process who made the call, os and oe are the starting and ending offsets of this I/O operation, and $type$ indicates a read or write operation. Two tuples $(t_1, r_1, os_1, oe_1, type_1)$ and $(t_2, r_2, os_2, oe_2, type_2)$, where $t_1 < t_2$, are a conflict pair if the following conditions are satisfied:

1. The pair overlaps: either $os_1 \leq os_2 \leq oe_1$ or $os_2 \leq os_1 \leq oe_2$.
2. The first operation is a write: $type_1 = write$.
3. For commit semantics: process r_1 does not execute any commit operation after t_1 and before t_2.
4. For session semantics: there is no close operation on process r_1 with t_c and open operation on process r_2 with t_o so that $t_1 < t_c < t_o < t_2$

RESULTS

We collected I/O traces from 17 HPC applications using Recorder[1]. These applications perform I/O using the POSIX API and a variety of I/O libraries.

Out result shows that all but one of the applications we studied can execute correctly with session semantics, provided that conflicts on the same process are properly handled.

The one exception can be handled with a single line change to an I/O library. Under commit semantics, the results are similar since applications do not make much use of `fsync` or other commit operations.

Applications	MPI	HDF5
ENZO, NWChem, GAMESS, Nek5000, LAMMPS, GTC, QMCPACK, MILC-QCD, HACC-IO, VPIC-IO	Intel MPI 2018	HDF5 1.12.0
pF3D-IO, VASP	MVAPICH 2.2	
LBANN	MVAPICH 2.3	HDF5 1.10.5
ParaDiS, Chombo, FLASH, MACSio	Intel MPI 2018	HDF5 1.8.20

Table 1: Applications and I/O libraries used

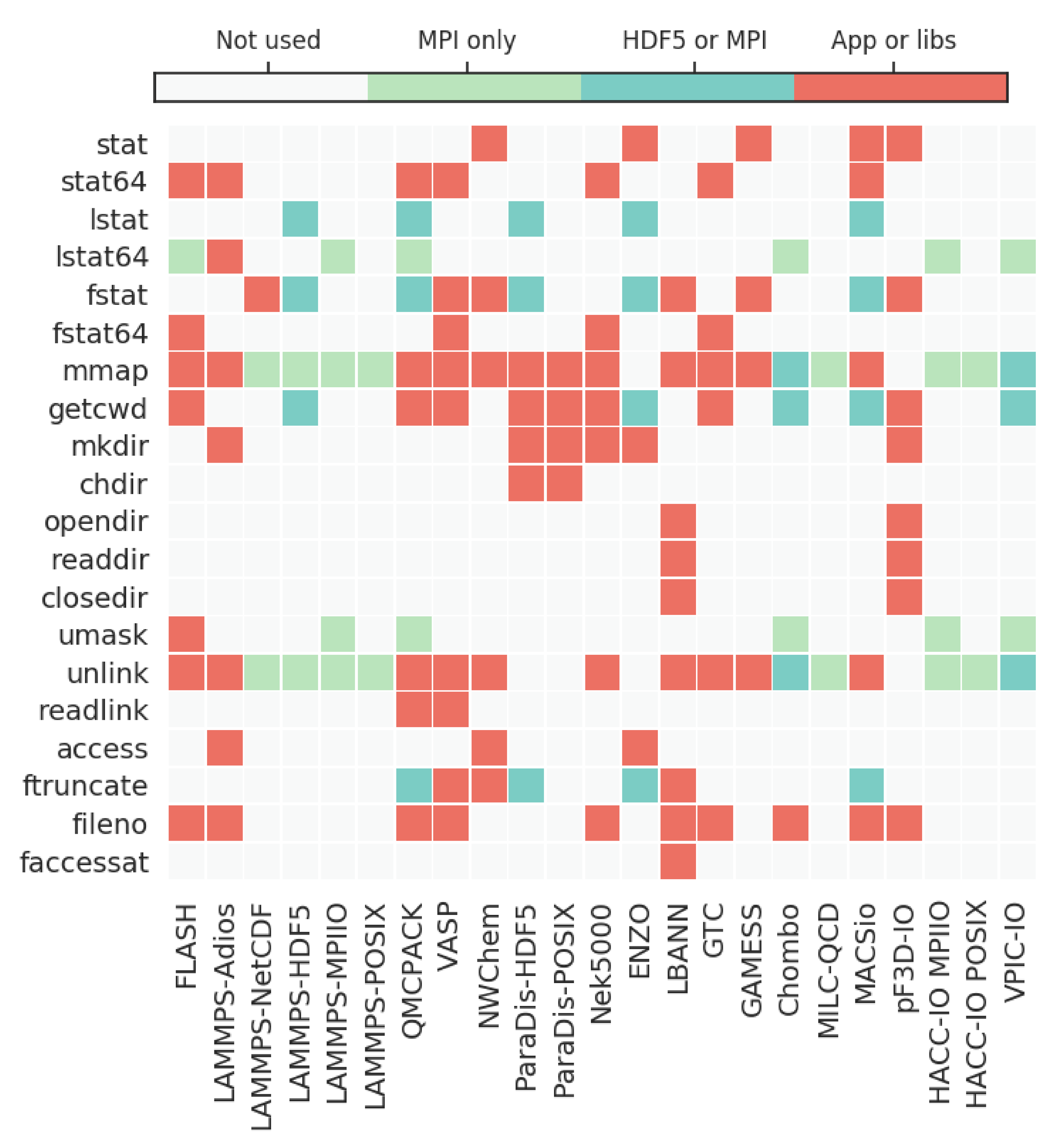

Figure 1: Metadata operations used by applications and I/O libraries

Application	I/O Library	WAW S	WAW D	RAW S	RAW D
FLASH	HDF5	✓	✓		
ENZO	HDF5			✓	
NWChem	POSIX	✓		✓	
pF3D-IO	POSIX			✓	
MACSio	Silo	✓			
GAMESS	POSIX	✓			
LAMMPS	ADIOS	✓			
LAMMPS	NetCDF	✓			
LAMMPS	HDF5				
LAMMPS	MPI-IO				
LAMMPS	POSIX				
MILC-QCD	POSIX				
ParaDiS	HDF5				
ParaDiS	POSIX				
VASP	POSIX				
LBANN	POSIX				
QMCPACK	HDF5				
Nek5000	POSIX				
GTC	POSIX				
Chombo	HDF5				
HACC-IO	MPI-IO				
HACC-IO	POSIX				
VPIC-IO	HDF5				

Table 2: Conflicts with session semantics.

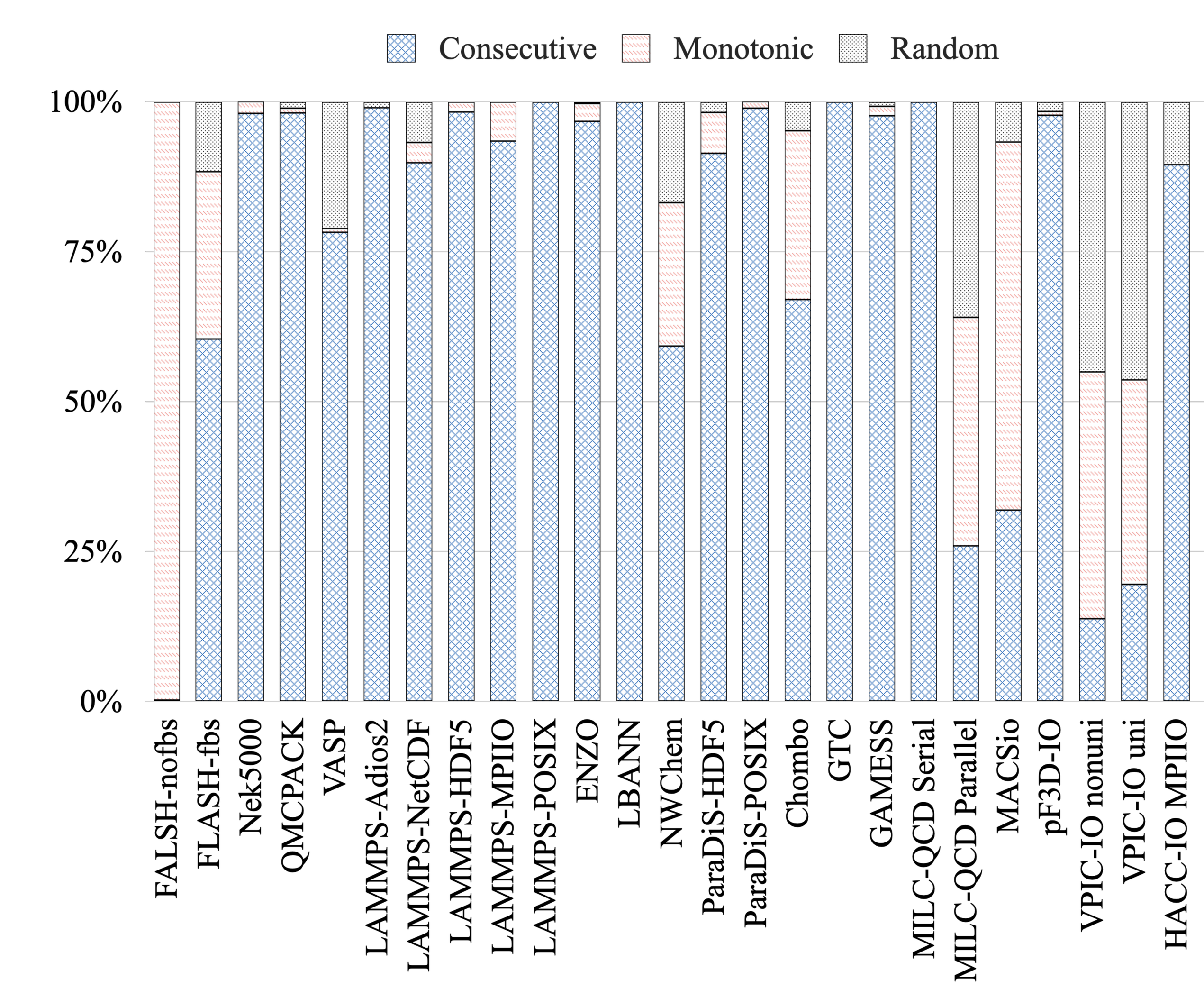

Figure 2: Access patterns overview

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

[1] https://github.com/uiuc-hpc/Recorder