Rethinking I/O Consistency in the MPI Standard

ESSA'25
June 4, 2025

Chen Wang and Kathryn Mohror
Lawrence Livermore National Laboratory

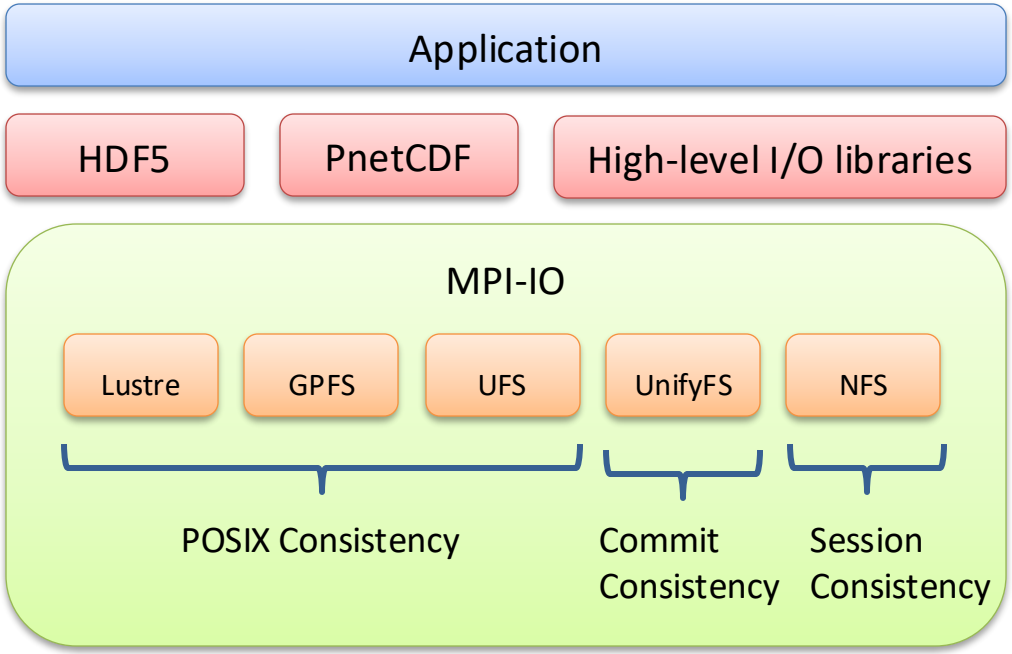
Special thanks to Quincey Koziol, Edgar Gabriel, and the MPI I/O Working Group for their valuable feedback and advice.

MPI and MPI-IO

- **MPI (Message Passing Interface)** is the dominant parallel programming model in high-performance computing (HPC) applications.
  - MPI-1.0 (May 1994): Initial Release
  - The most recent release is MPI 4.1 in November 2023.
- The initial standard did not include support for I/O.
- **Parallel I/O** capabilities were introduced later in MPI-2.0 (1997), and have since become an integral part of the MPI standard.
  - MPI's I/O consistency semantics are defined in ***Chapter 14.6.1, titled Consistency and Semantics***, which has remained largely unchanged across versions.

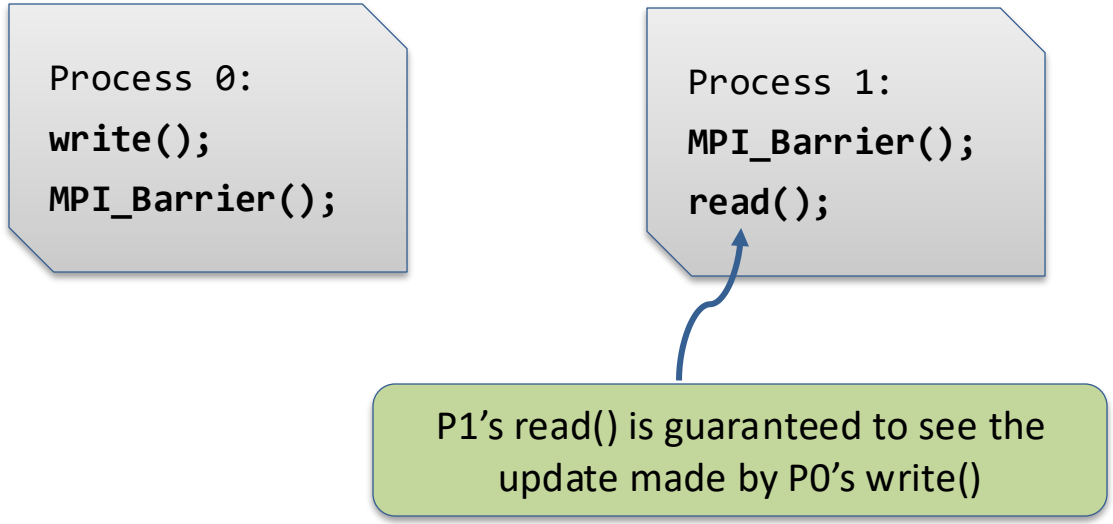
Consistency Model

A consistency model specifies a contract between the programmer and the system. If the programmer follows the rules, then the system guarantees the outcome of read/write operations is predictable.



POSIX Consistency (Sequential Consistency)

*"**After** a write() to a regular file has successfully returned, any successful read() from each byte position in the file that was modified by that write **shall return the data specified by the write()** for that position until such byte positions are again modified."*



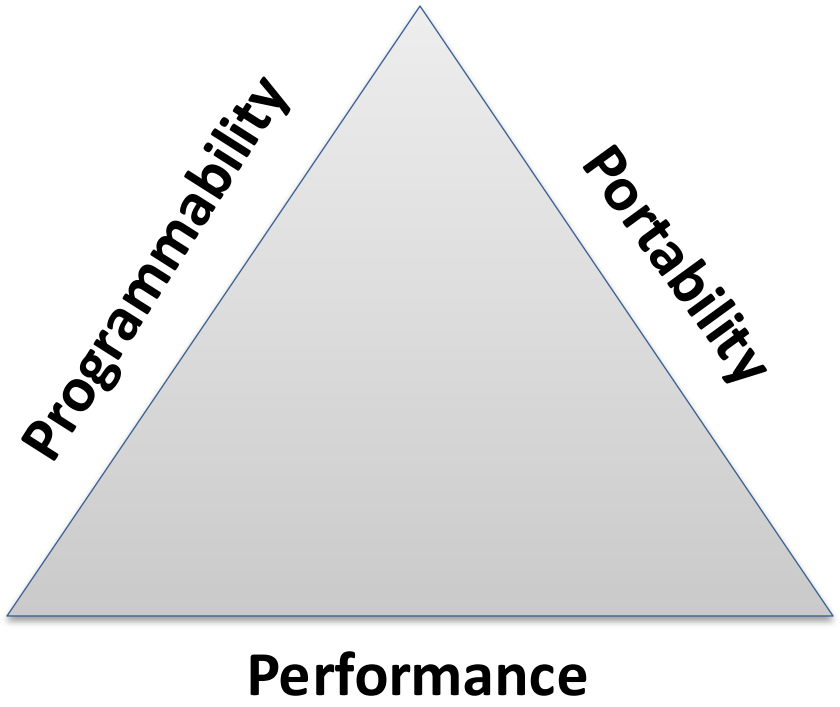
MPI-IO Consistency (Standard 4.1; page 702)

- For **conflicting accesses,** the default MPI semantics do **not** guarantee sequential consistency.
- Sequential consistency can be achieved by using **"sync-barrier-sync".**
- MPI 1.0 published in 1994. The support of parallel I/O was added in MPI 2.0 (1997). The I/O chapter has not changed much over the years.

```
Process 0:

MPI_File_write();
MPI_File_sync();
MPI_Barrier();
MPI_File_sync();
```

```
Process 1:

MPI_File_sync();
MPI_Barrier();
MPI_File_sync();
MPI_File_read();
```

3P Criteria: Programmability, Portability, and Performance



The current MPI consistency design has good programmability and portability, but has potential performance issues.

Adve, Sarita Vikram. "Designing memory consistency models for shared-memory multiprocessors". Diss. University of Wisconsin, Madison, 1993.

MPI_File_sync (Standard 4.1; page 704)

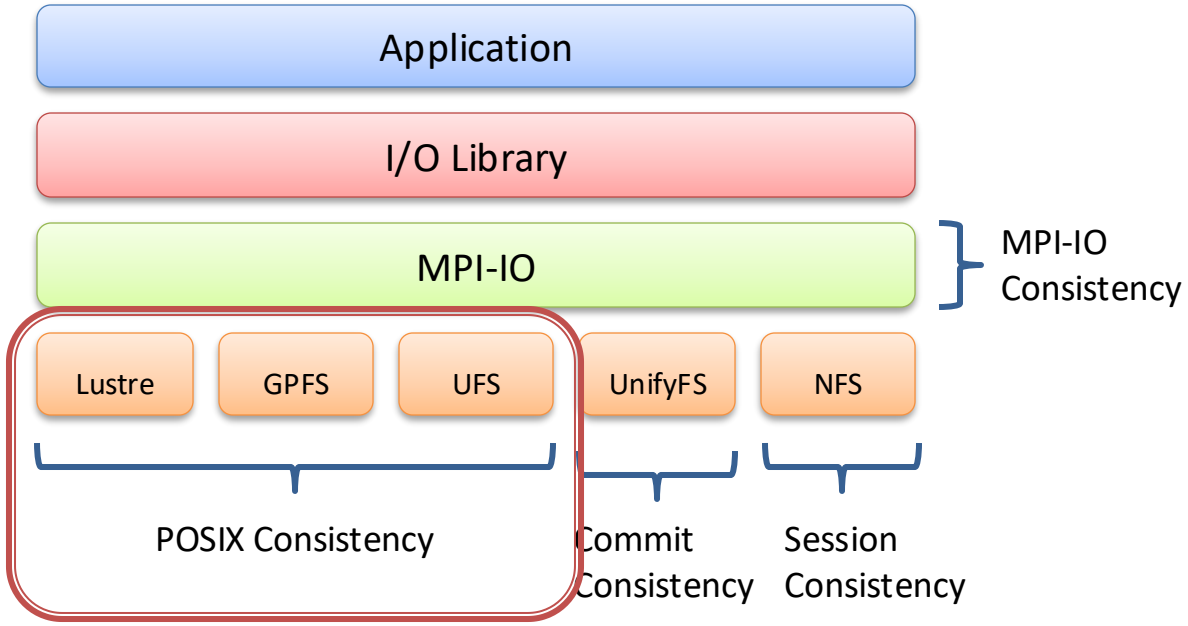
- *"… causes all previous writes by the calling process to be transferred to the **storage device**."*
- *"… all such updates become visible to subsequent reads by the calling process."*
- *MPI_File_sync is a collective operation.*

Performance Issues of the Current MPI-IO Consistency Design

1. Consistency and persistency:
   - Two properties provided in a single call. Consistency-only scenarios doesn't require a flush to the disks.
2. Granularity:
   - Synchronize the entire file but apps/libraires may only update a small region
3. Collective (global synchronization):
   - MPI_File_sync() involves more processes than necessary. **All** processes opened the file collectively need to be synchronized.
4. Cannot distinguish producer/consumer:
   - The current sync-barrier-sync construct requires both syncs to be MPI_File_sync(), which reduce flexibilities and hinders file system optimizations.

Temporary Reliever: Skip MPI_File_sync() on POSIX Systems

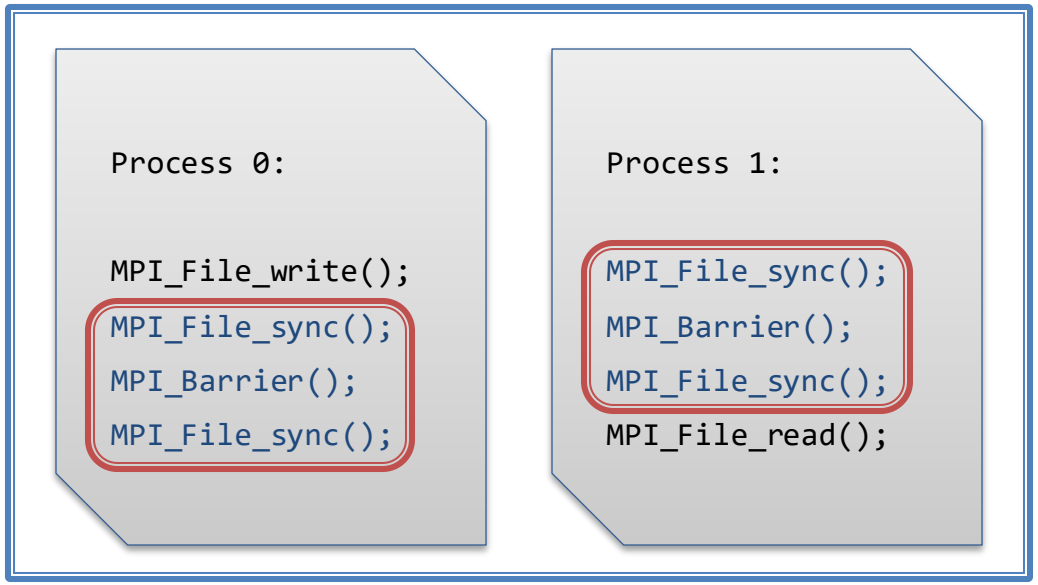
- Assuming, the underlying file system is a POSIX system (most likely), then we can skip MPI_File_sync and still get the same currentness.
  - *Because POSIX guarantees write is strongly consistent.*



Temporary Reliever: Skip MPI_File_sync() on POSIX Systems

😊 Portability

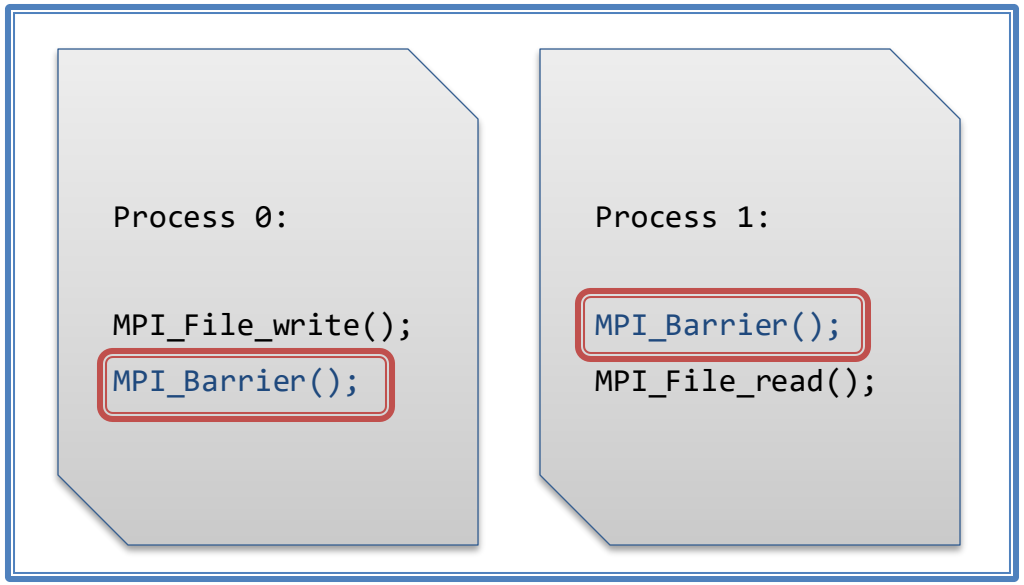
😡 Performance



Correct MPI Semantics (sync-barrier-sync)

😡 Portability

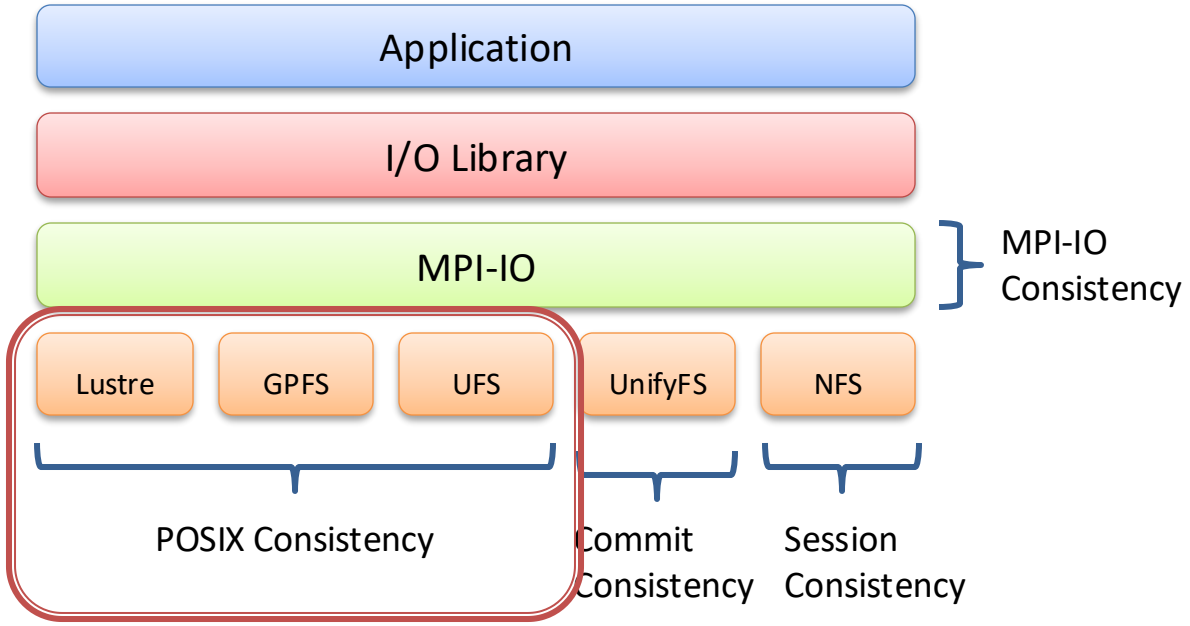
😊 Performance



Incorrect MPI Semantics (skip sync calls)

Temporary Reliever: Skip MPI_File_sync() on POSIX Systems

- In other words, we can violate MPI standard and get away with it.
- Some I/O libraries are actually doing this.



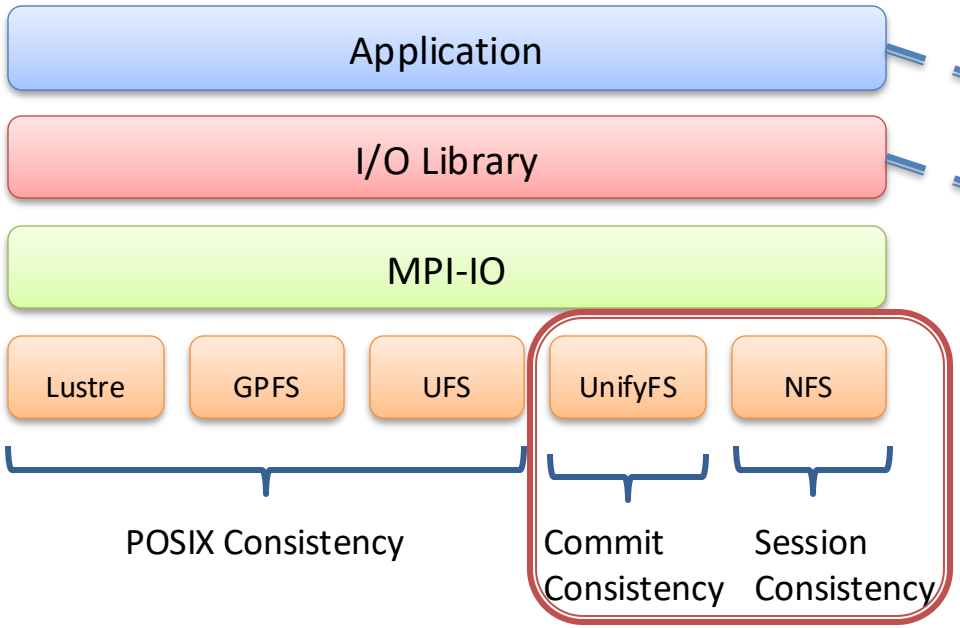
Temporary Reliever: Skip MPI_File_sync() on POSIX Systems

- We have developed a tool, **VerifyIO**, that can test consistency semantics adherence.
  - Paper was accepted by IPDPS. Will give a talk on Friday afternoon (19. Storage and I/O session)
- We selected 91 built-in test programs from the three libraries and ran our verification algorithm against MPI-IO semantics.

	HDF5	NetCDF	PnetCDF
Test Programs written correctly	x	x	x
Library Implemented Correctly	✓	x	x

NetCDF and PnetCDF implementations have consistency bugs!

Temporary Reliever: Skip MPI_File_sync() on POSIX Systems



Portable MPI code (e.g., HDF5 and PnetCDF) needs to conform to the MPI standard, without making any assumption on the underlying file system semantics.

Otherwise, may have currentness issues.

Long-Term Solution: Revisions to the MPI Consistency Design

- **The fundamental issue**: The current design imposes high overhead for consistency-only scenarios; It limits optimizations for I/O libraries and FS drivers.
- **Solution: Introduce new MPI-IO functions dedicated for consistency-only scenarios**.
  - The new APIs can be implemented with very low (nearly zero) cost on POSIX systems, delivering both portability and good performance for users!
  - Applications and I/O libraries can use them to write portable code without worrying about the extra overhead.
  - New APIs should allow more optimization opportunities for non-POSIX file systems.

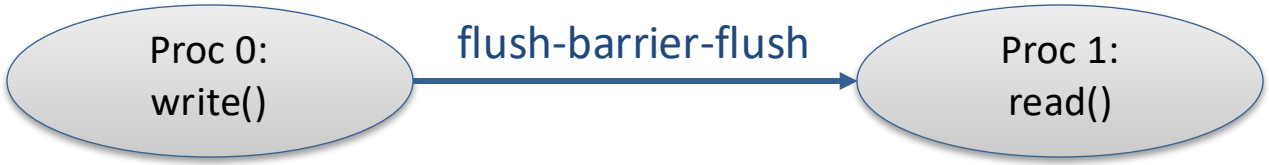
Decisions to Make

We want both *performance* and *portability*

1. Consistency and persistency: Do not enforce implementations to transfer data to disk when only consistency is needed.
2. Granularity?
   - Should the new API synchronize the entire file or a file range?
3. Collective vs. non-collective?
   - Should the new API be collective call or non-collective call?
4. Producer/Consumer?
   - Should we introduce two calls to distinguish producer and consumer?

Alternative Design: A1

- `MPI_File_flush(MPI_File fh)`
  - Guarantees all previously writes to fh by the calling processes become visible to subsequent reads of fh.
  - MPI_File_flush is a **collective** function.



Alternative Design: A2

- `MPI_File_flush(MPI_File fh)`
  - Guarantees all previously writes to fh by the calling processes become visible to subsequent reads of fh.
  - MPI_File_flush is **non-collective**.
- `MPI_File_refresh(MPI_File fh)`
  - Causes all previous flushes on the same file become visible to subsequent reads of fh by the calling process.
  - MPI_File_refresh is **non-collective.**

```
Rank 0:

MPI_File_write();
MPI_File_flush();
MPI_Send();
```

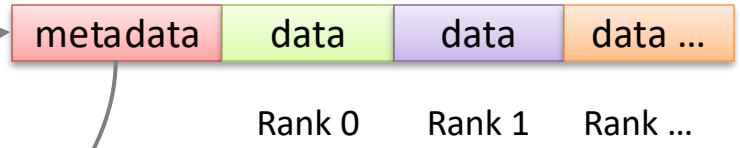
```
Rank 1:

MPI_Recv();
MPI_File_refresh();
MPI_File_read();
```

Alternative Design: A3 = A2 + finer granularity

- `MPI_File_flush_range(MPI_File fh, MPI_Offset offset, int count, …)`
- `MPI_File_refresh_range(MPI_File fh, MPI_Offset offset, int count, …)`

Rank 0:

```
MPI_File_write_at(fh, 0, buf, 1, metatype,);
MPI_File_write_at(fh, o[0], buf, 999, MPI_INT,);
MPI_File_flush_range(fh, 0, 1, metadata);
MPI_Send(rank 1);
```



Rank 1:

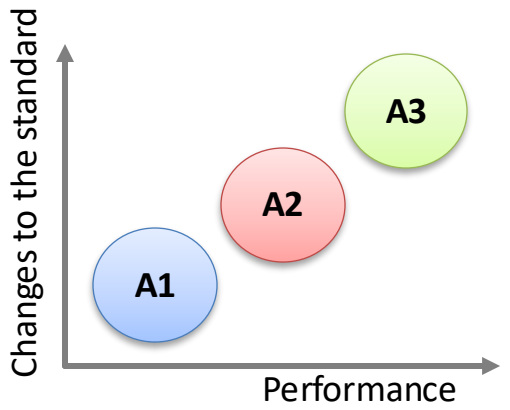
```
MPI_Recv(rank 0);
MPI_File_refresh_range(fh, 0, 1, metadata);
MPI_File_read_at(fh, 0, buf, 1, metatype,);
MPI_File_write_at(fh, o[1], buf, 999, MPI_INT,);
```

Rank 2,3,4…:

```
MPI_File_write_at(fh, o[rank],
999, MPI_INT)
...
MPI_File_fetch()
```

Alternative Designs

	A1 flush-barrier-flush	A2 flush-barrier-refresh	A3 flush-barrier-refresh
New APIs	**MPI_File_flush**	**MPI_File_flush** **MPI_File_refresh**	**MPI_File_flush** **MPI_File_fetch** **MPI_File_refresh_range** **MPI_File_refresh_range**
Collective	Yes	No	No
Granularity	File	File	File range
Distinguish producer and consumer	No Producer: flush Consumer: flush	Yes Producer: flush Consumer: refresh	Yes Producer: flush Consumer: refresh



Current Status

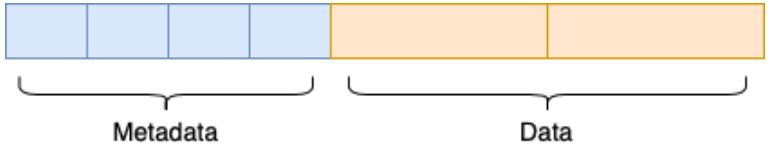
- The MPI-IO working agreed to go with Design A2:
  - MPI_File_flush() + MPI_File_refresh()
  - ***flush-barrier-refresh***
- Now waiting for reading and the forum voting.
  - Issue: https://github.com/mpi-forum/mpi-issues/issues/995
- ROMIO Implementation for MPICH and MVAPICH:
  - POSIX Systems: Lustre, GPFS and UFS driver
    - **13 files and ~250 LOC changes**
      - MPI_File_flush(): noop
      - MPI_File_refresh(): noop
  - Non-POSIX System: UnifyFS
    - The new design opens up more possibilities for implementation optimizations.
    - **21 Files and 778 additional LOC for a new ROMIO driver.**

Impact for I/O Library Developers and Application Users

- I/O Library Developers:
  - Current sync-barrier-sync still works. Existing code will not break.
  - When available, use ***flush-barrier-refresh*** for consistency-only scenarios.
  - Do not ignore flush()/refresh(). They are no-ops on POSIX systems.
- Application Users:
  - Nothing.

Synthetic Scenario 1: Small metadata I/O

- One shared file; The beginning region of the file is used to store “metadata”.
- At each timestep:
  - Rank 0 updates metadata area.
  - One rank of each node reads from metadata area.



Current:

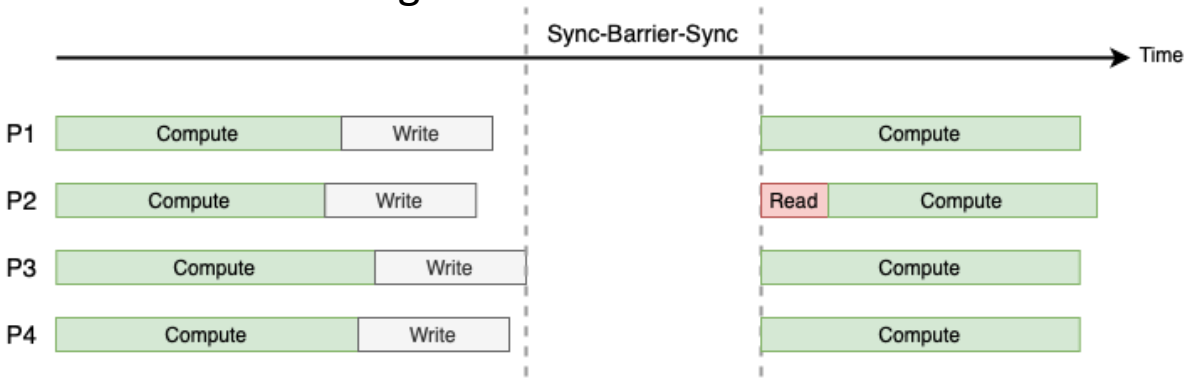
```
Rank 0:
MPI_File_write();
MPI_File_sync();
MPI_Barrier(world);
MPI_File_sync();

All ranks:
MPI_File_sync();
MPI_Barrier(world);
MPI_File_sync();
MPI_File_read();
```

Proposed:

```
Rank 0:
MPI_File_write();
MPI_File_flush();
MPI_Barrier(subcomm);

Rank % ppn == 0:
MPI_Barrier(subcomm);
MPI_File_refresh();
MPI_File_read();
```

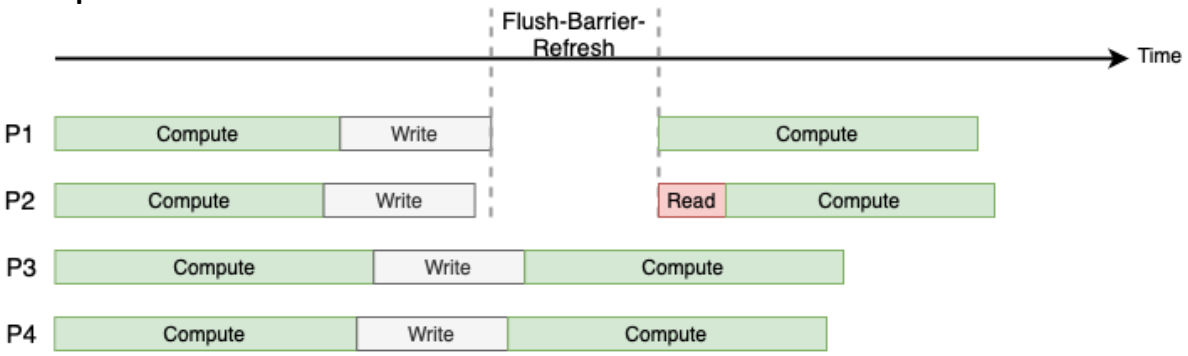
Synthetic Scenario 1: Small metadata I/O

Current MPI-IO Design:



Assume P1 updates metadata
And P2 reads metadata

Proposed:

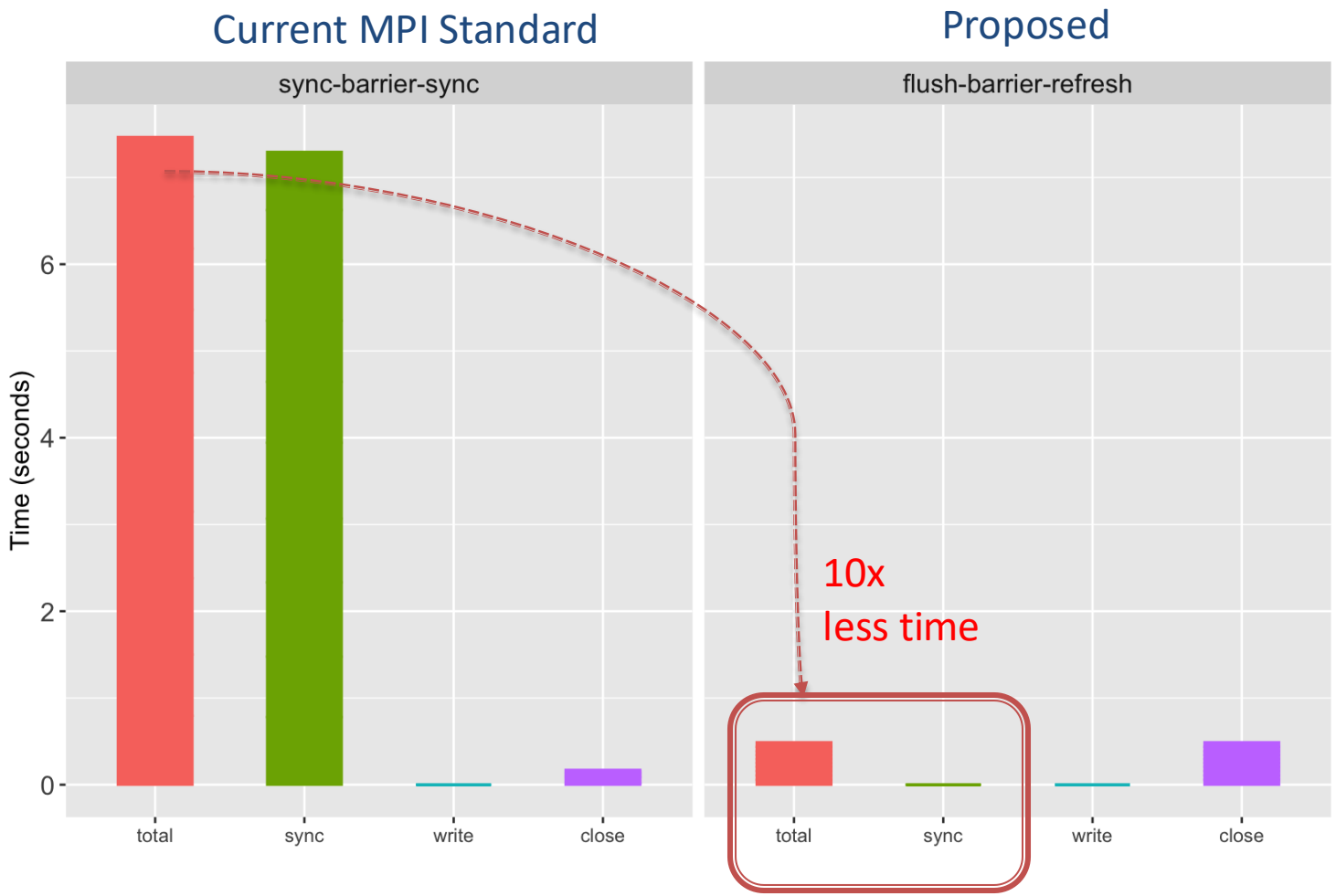


Sources of performance improvement:

1. Flush/Refresh do not need to write to disk: they are no-ops on POSIX systems, i.e., zero cost.
2. Barrier only involves subset of processors.
3. Only producer/consumer need to call flush and refresh, reducing the possibility of waiting for stragglers.

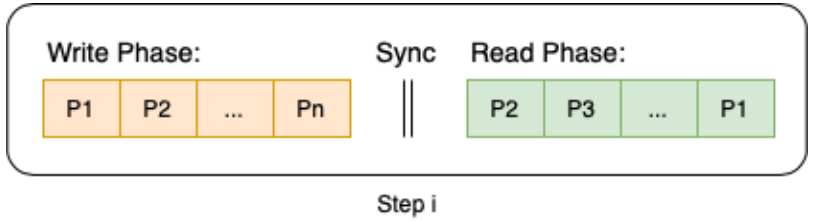
Synthetic Scenario 1: Small metadata I/O

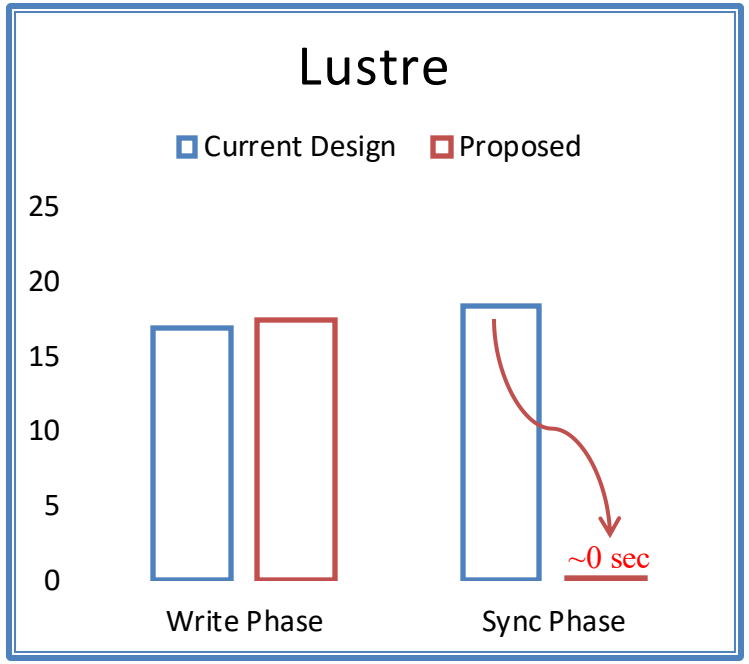
- Quartz at LLNL. 16 nodes.
  - 36 processes/node.
- MVAPICH 2.3.7
- Lustre no tuning.
- 10 timesteps
- 100x1K metadata
- Assuming non-involving ranks stay idle.

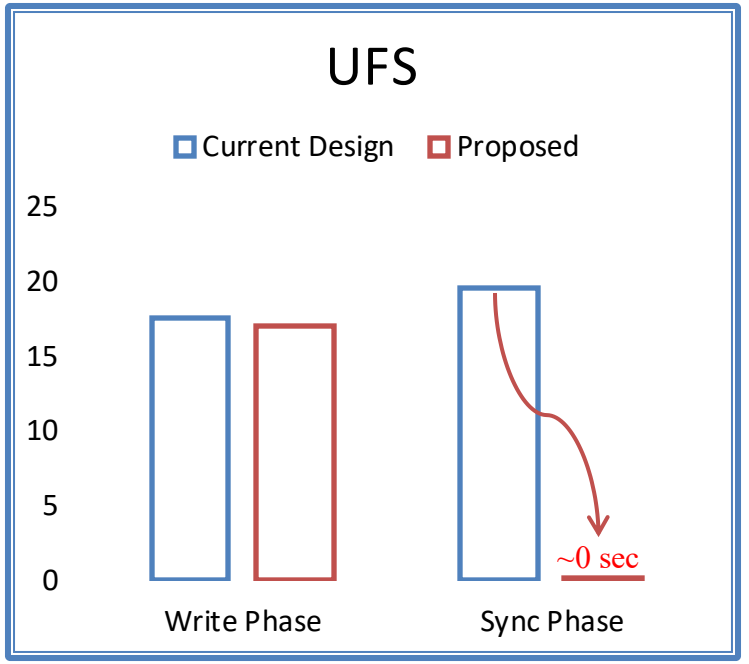


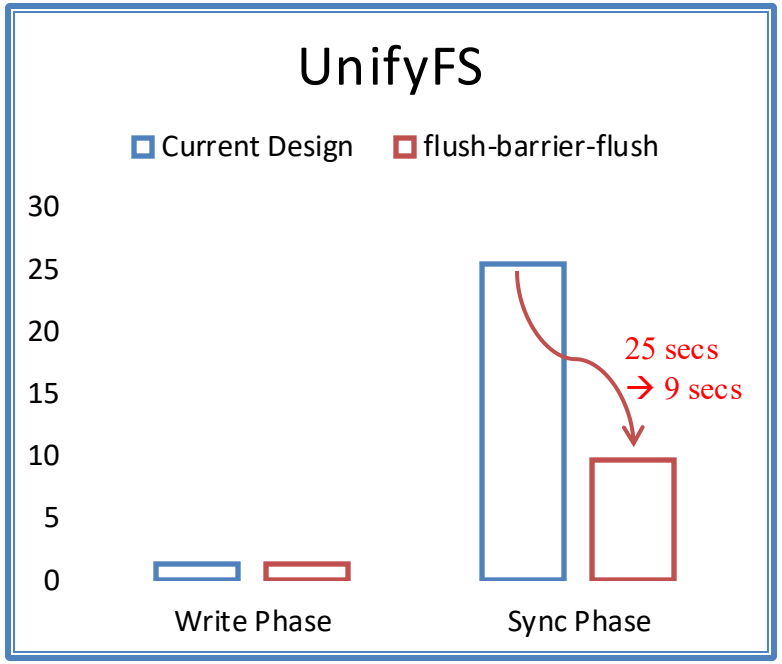
Synthetic Scenario 2: Big Strided I/O (4MB Chunk)

- Benchmark
  - 225GB single shared file
  - 16 nodes; 36 processes/node









In every file system, the use of the new APIs reduces the total time by half!
total time includes open, close, write, read, and sync time.

Future Work

- Bring it to vote!
- Evaluate performance improvement for more systems (especially non-POSIX systems) and applications.

References:

— Chen Wang, Zhaobin Zhu, Kathryn Mohror, Sarah Neuwirth, and Marc Snir. " VerifyIO: Verifying Adherence to Parallel I/O Consistency Semantics ”. IPDPS, 2025 (To Appear).

— Chen Wang, Kathryn Mohror, and Marc Snir. "Formal Definitions and Performance Comparison of Consistency Models for Parallel File Systems", IEEE TPDS, 2024.
  • https://doi.org/10.1109/tpds.2024.3391058

— Chen Wang, Kathryn Mohror, and Marc Snir. "File System Semantics Requirements of HPC Applications", HPDC, 2021.
  • https://doi.org/10.1145/3431379.3460637

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC