Revisiting MPI-IO Consistency: Potential Revisions and Their Impact on HDF5

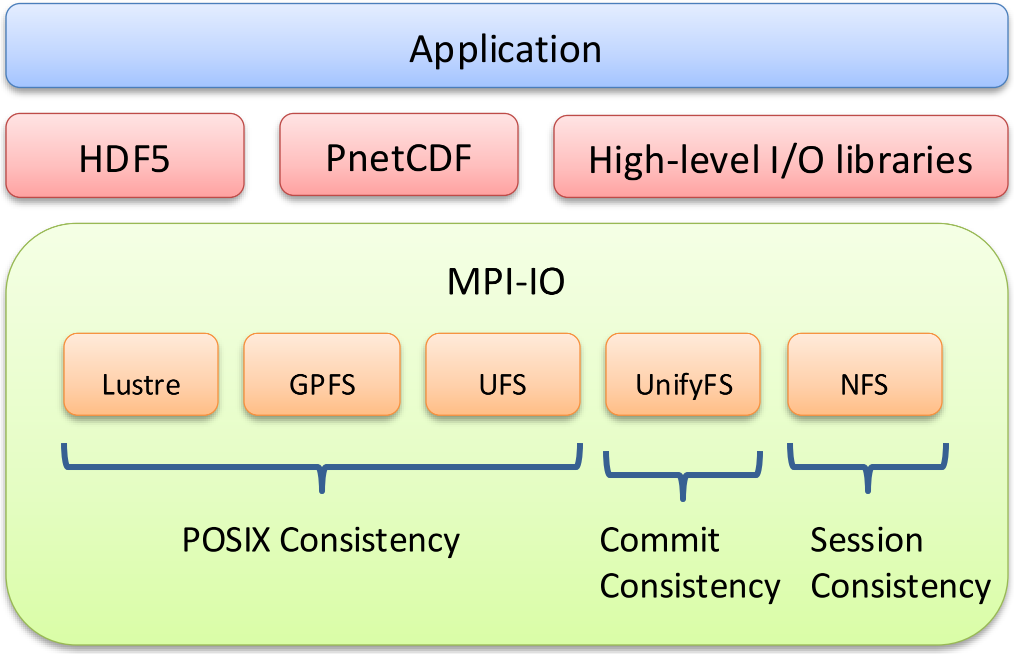
2024 HDF5 User Group Meeting

Chen Wang[*], Kathryn Mohror[*], and Marc Snir[+]

[*]Lawrence Livermore National Laboratory, [+]University of Illinois Urbana-Champaign

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."*

```
Process 0:
write();
MPI_Barrier();
```

```
Process 1:
MPI_Barrier();
read();
```

P1's read() is guaranteed to see the update made by P0's write()

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();
```

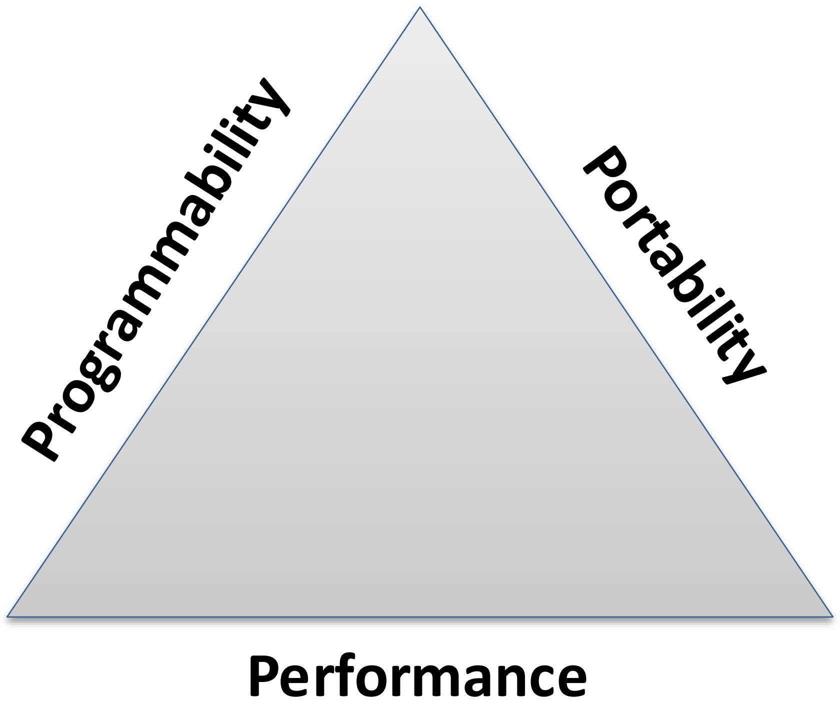
MPI-IO Consistency: HDF5 Example

```
Process 0:

H5Dwrite();
H5Fflush();
MPI_Barrier();
H5Fflush();
```

```
Process 1:

H5Fflush();
MPI_Barrier();
H5Fflush();
H5Dread();
```

H5Fflush() is implemented using MPI_File_sync()

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.*

persistency

consistency

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 call:
   - 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: Ignore MPI_File_sync() on POSIX Systems

- Portability
- Performance

```
Process 0:

H5Dwrite();
H5Fflush();
MPI_Barrier();
H5Fflush();
```

```
Process 1:

H5Fflush();
MPI_Barrier();
H5Fflush();
H5Dread();
```

Correct MPI Semantics

- Portability
- Performance

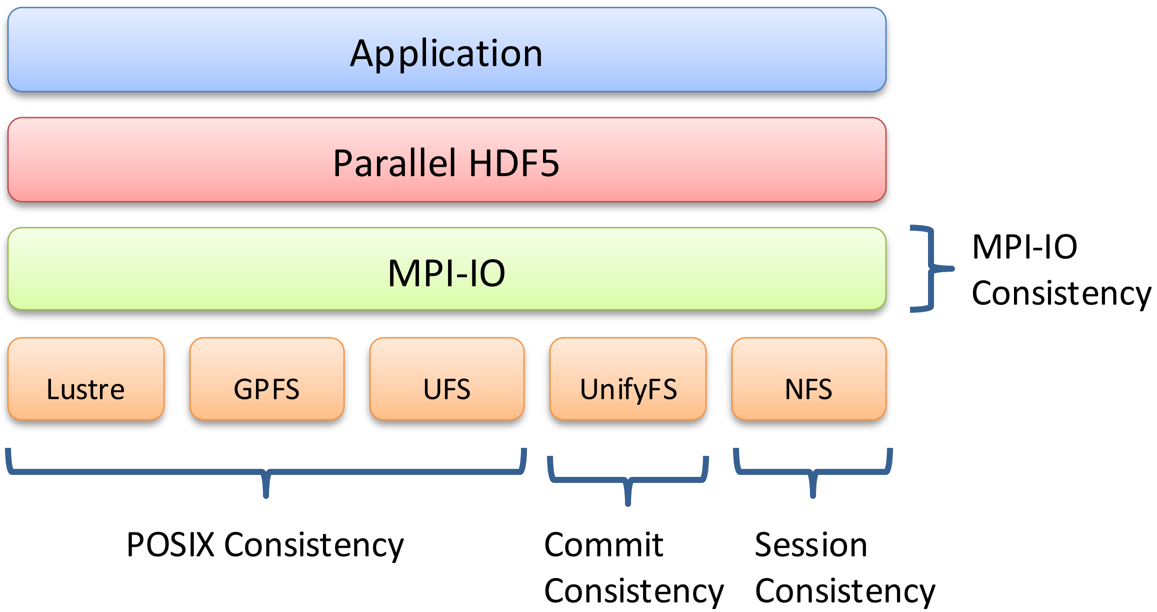
```
Process 0:

H5Dwrite();
MPI_Barrier();
```

```
Process 1:

MPI_Barrier();
H5Dread();
```

Incorrect MPI Semantics

Temporary Reliever: Ignore MPI_File_sync() on POSIX Systems

- We designed an algorithm to check whether a given execution comforts to the specified consistency semantics.
- HDF5 1.14.2
  - testpar/*: **7 tests violate MPI semantics**



Portable HDF5 code needs to conform to the MPI standard, without making any assumption on the underlying file system semantics.

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

Balance between *performance* and *programmability*!

1. Consistency and persistency: Do not enforce implementations to transfer data to disk when only consistency is needed.

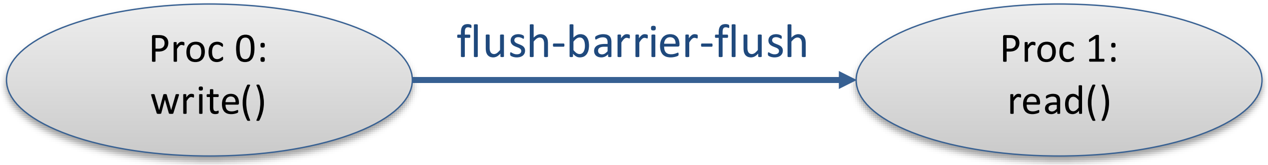
2. Granularity?
   — Should the new API synchronize the entire file or a file range?

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.**

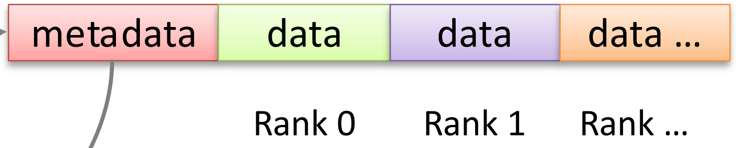
```
Rank 0:

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);
```

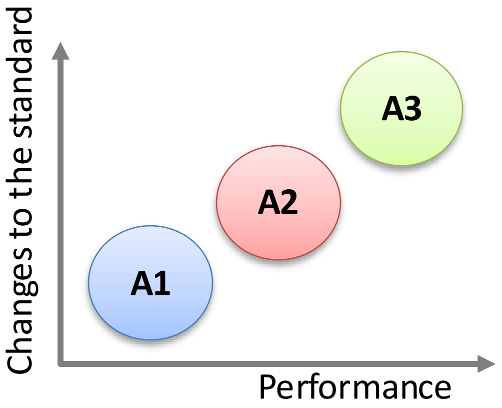



```
Rank 1:

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 and Range
Distinguish producer and consumer	No Producer: flush Consumer: flush	Yes Producer: flush Consumer: refresh	Yes Producer: flush Consumer: refresh



Current Status

- Have been discussing these alternative designs with the MPI-IO working group. Preparing for the revision draft.

- ROMIO Implementation for MPICH and MVAPICH:
  - POSIX Systems: Lustre, GPFS and UFS driver
    - **13 files and 247 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.**

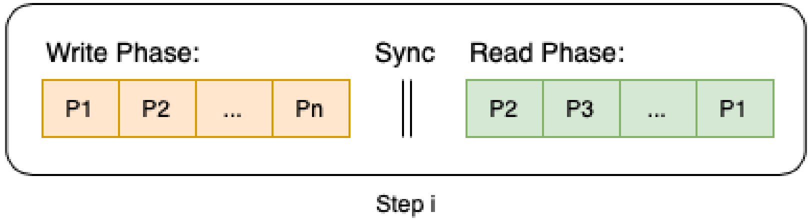
For HDF5 Developers and Users

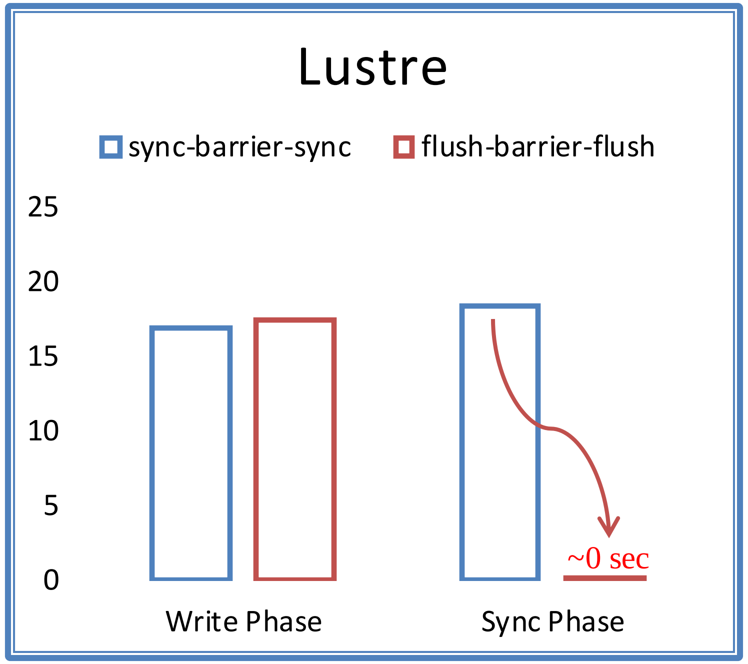
- Developers:
  - Current sync-barrier-sync still works. Existing code will not break.
  - When available, use *flush-barrier-refresh* for consistency-only scenarios. Do not ignore flush()/refresh(). They are no-ops on POSIX systems, and they can make sure the HDF5 code will run correctly on non-POSIX systems.
  - Other opportunity:
    - Introduce H5Ffence() to simplify programming, so users do not need to make explicit MPI calls, i.e., H5Dwrite() -> H5Ffench() -> H5Dread().

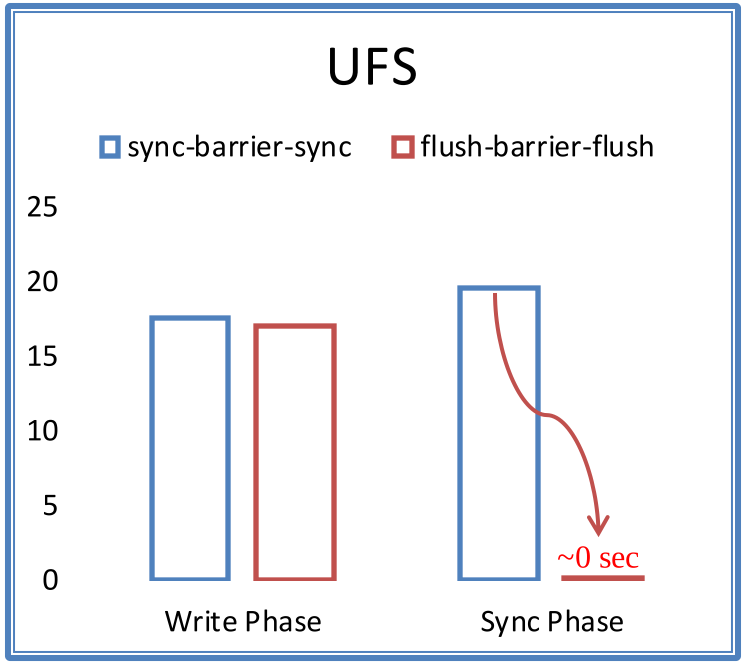
- Users:
  - Nothing.
  - If available, use the new API (H5Ffence()) for better performance.

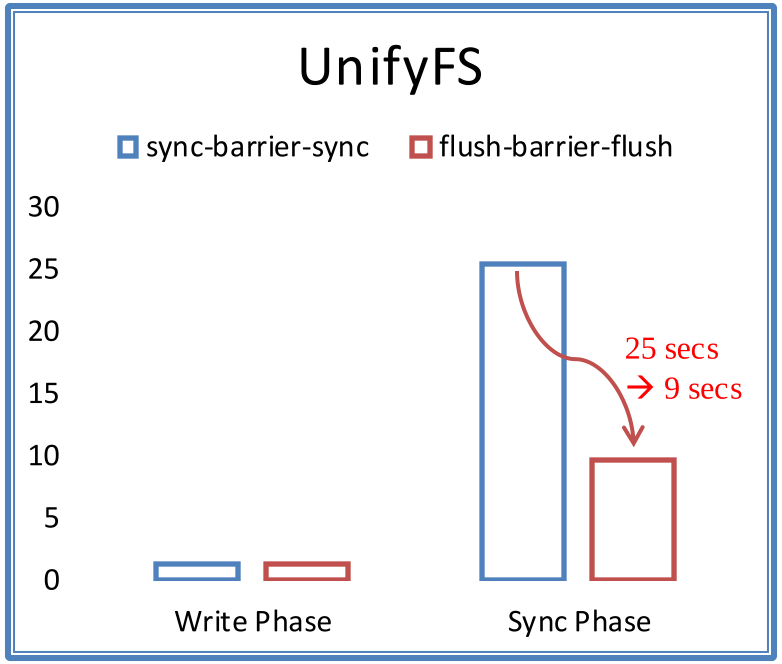
MVAPICH Strided I/O 4MB Chunk

- 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

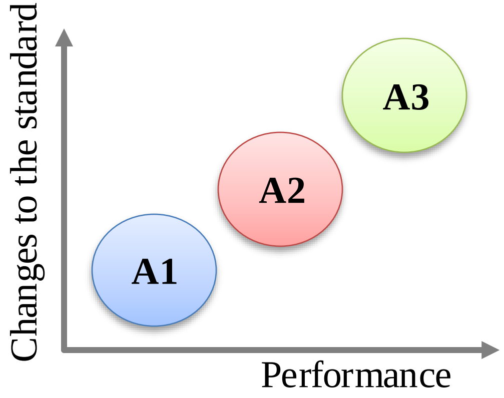
- Bring it to vote!
- Evaluate performance improvement for more systems (especially non-POSIX systems) and real applications.
- Design an algorithm to check for unnecessary synchronization calls.
- Thank you! Questions?
  - Contact: Chen Wang (wang116@llnl.gov)

References:

- 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
- Sushma Yellapragada, Chen Wang, and Marc Snir. "Verifying IO Synchronization from MPI Traces", PDSW, 2021.
  - https://doi.org/10.1109/PDSW54622.2021.00012

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 and Range
Distinguish producer and consumer	No Producer: flush Consumer: flush	Yes Producer: flush Consumer: refresh	Yes Producer: flush Consumer: refresh



Lawrence Livermore National Laboratory

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

MVAPICH Strided I/O 4MB Chunk