Improving MPI Independent Write Performance Using A Two-Stage Write-Behind Buffering Method

Alok Choudhary

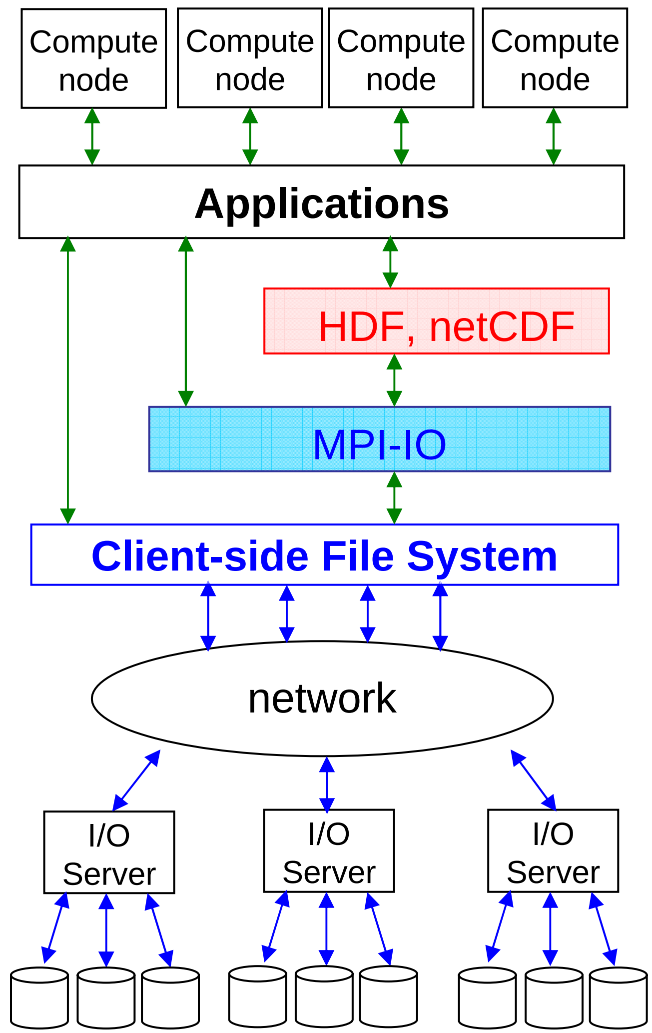
Wei-keng Liao*, Avery Ching*, Kenin Coloma*, Alok Choudhary*, and Mahmut Kandemir**

*EECS Department
Northwestern University

**CSE Department
Pennsylvania State University

The NSF Next Generation Software (NGS) Workshop
IEEE International Parallel & Distributed Processing Symposium, 2007

HPC I/O System Layers



MPI I/O

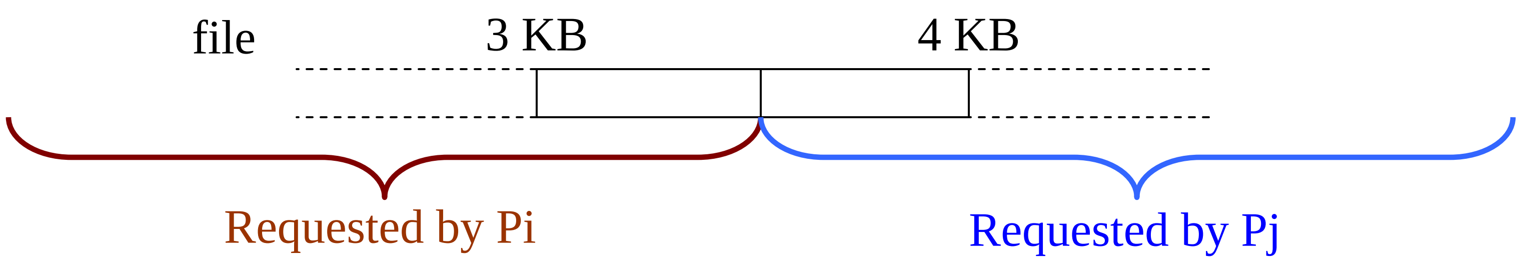
- Collective I/O functions
  - Synchronous: all processes must participate
  - Better performance
    - In many MPI I/O implementations, processes collaborate with each other for better I/O strategies, for example, two-phase I/O
  - Require defining file views
    - Constructing MPI derived data types for file views can be complicated
- Independent I/O functions
  - No synchronization requirement
  - Worse performance
    - Difficult to improve due to its arbitrary nature
  - No need of file view
    - Often read/write with explicit file offsets

Write-only I/O Patterns

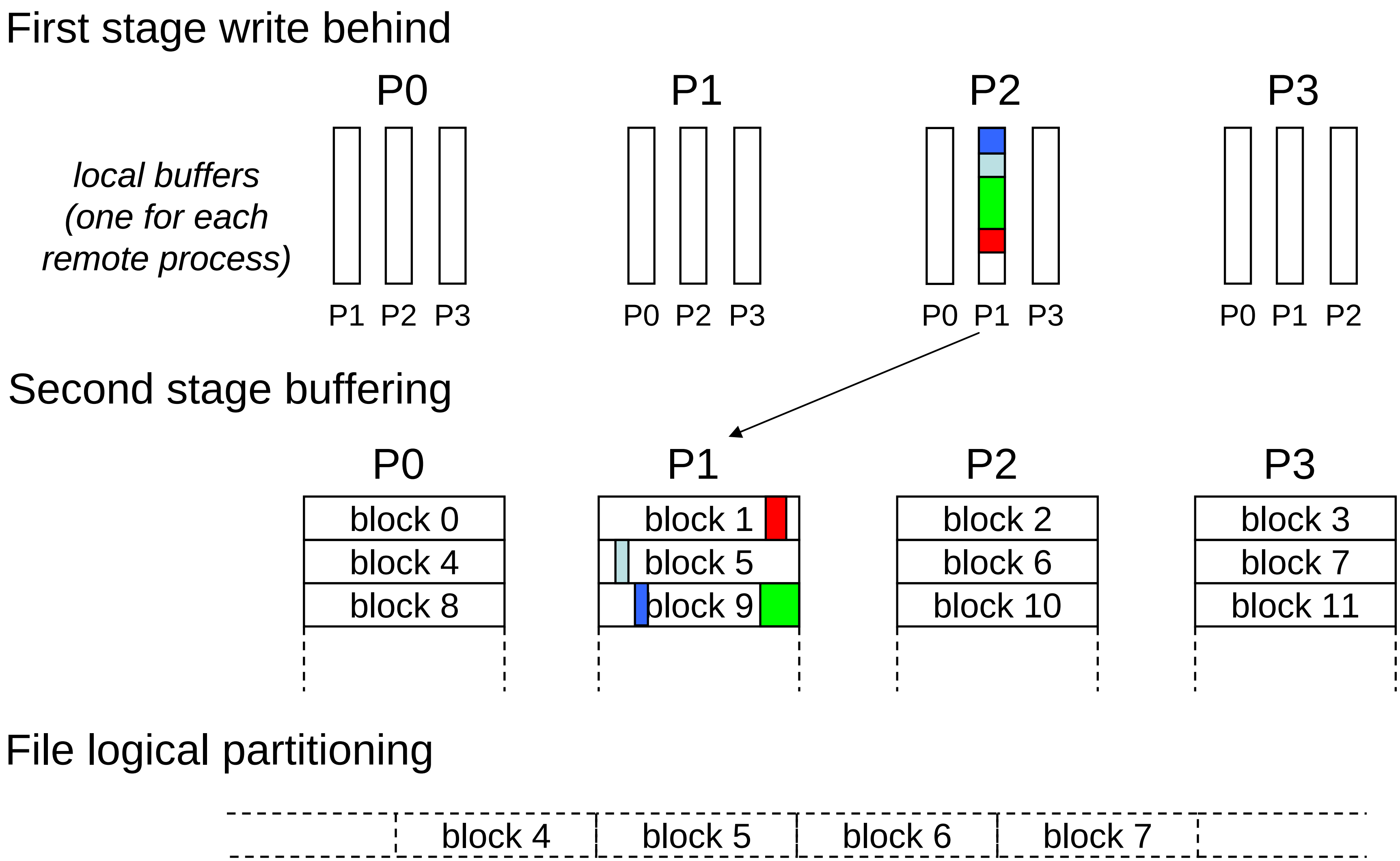
- Check-pointing for long-run applications
  - Data is periodically written to files
    - Data for restart in case of interruption
    - Data for post-simulation analysis
  - Once written, data will not be accessed for the rest of the run
- Write-behind strategy
  - Improve I/O by accumulating small requests to large requests for better network utilization
  - Part of the client-side file caching

Parallel File Systems

- Many adopt client-side file caching
- Cache coherence control
  - Distributed file locking protocol is used to avoid centralized management
  - Lock granularity is not in byte range
    - For example, file block size, disk sector size, file stripe size, etc.
  - Conflicting locks serialize I/O parallelism
    - In particular, conflicts at block level, not in byte range



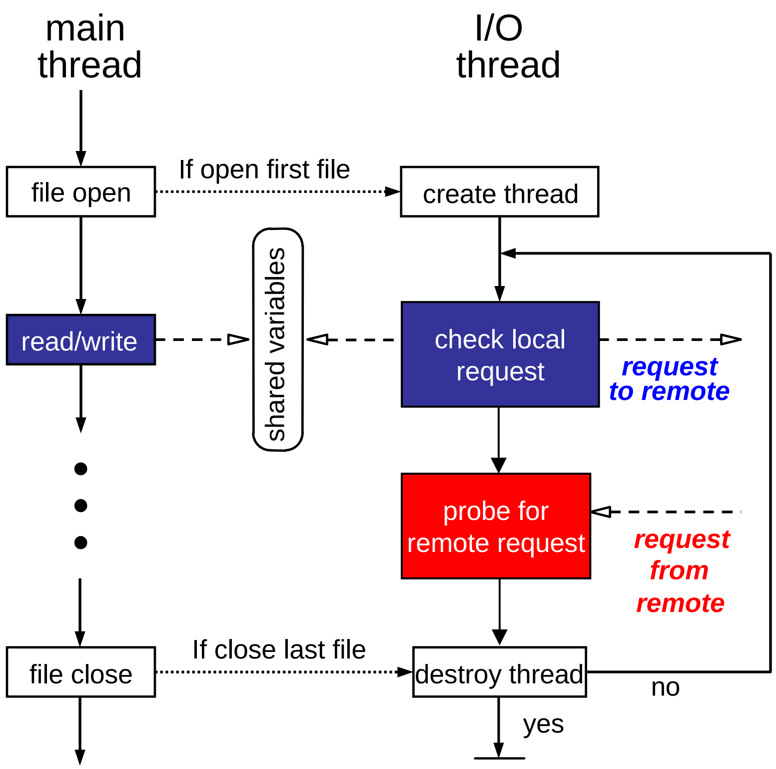
Two-stage Write Behind



Design

- Goal
  - Enable write behind locally
  - Avoid file system lock conflict globally
- 1^{st}-stage buffering
  - Data is accumulated locally
  - Flushed to 2^{nd}-stage buffers once full
  - Flushing is based on the file block assignment at the 2^{nd}-stage
- 2^{nd}-stage buffering
  - A file is logically divided into blocks
  - The ownership of blocks is statically assigned to all MPI processes in round robin
  - Evict blocks to file system if allocated buffers exceed a pre-defined upper bound, for example 64 MB

I/O Thread



- One thread per MPI process
  - Created at the first file open
  - Destroyed at the last file close
- Handle local write behind buffering
  - Write to first-stage buffers
  - Communicate with main thread through a mutex protected variable
  - Flush to remote 2^{nd}-stage buffers
- Receive data from remote
  - Receive flushed data and copy to the 2^{nd}-stage buffers
  - MPI_Iprobe( ) is used to probe remote requests
- I/O
  - Write to file system when 2^{nd}-stage buffers are full

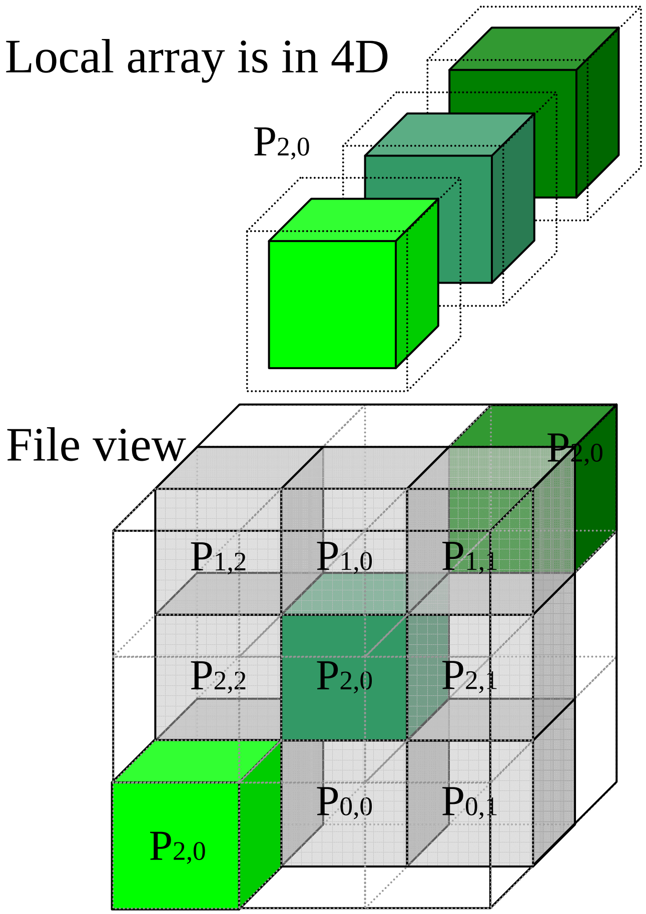
Experiments

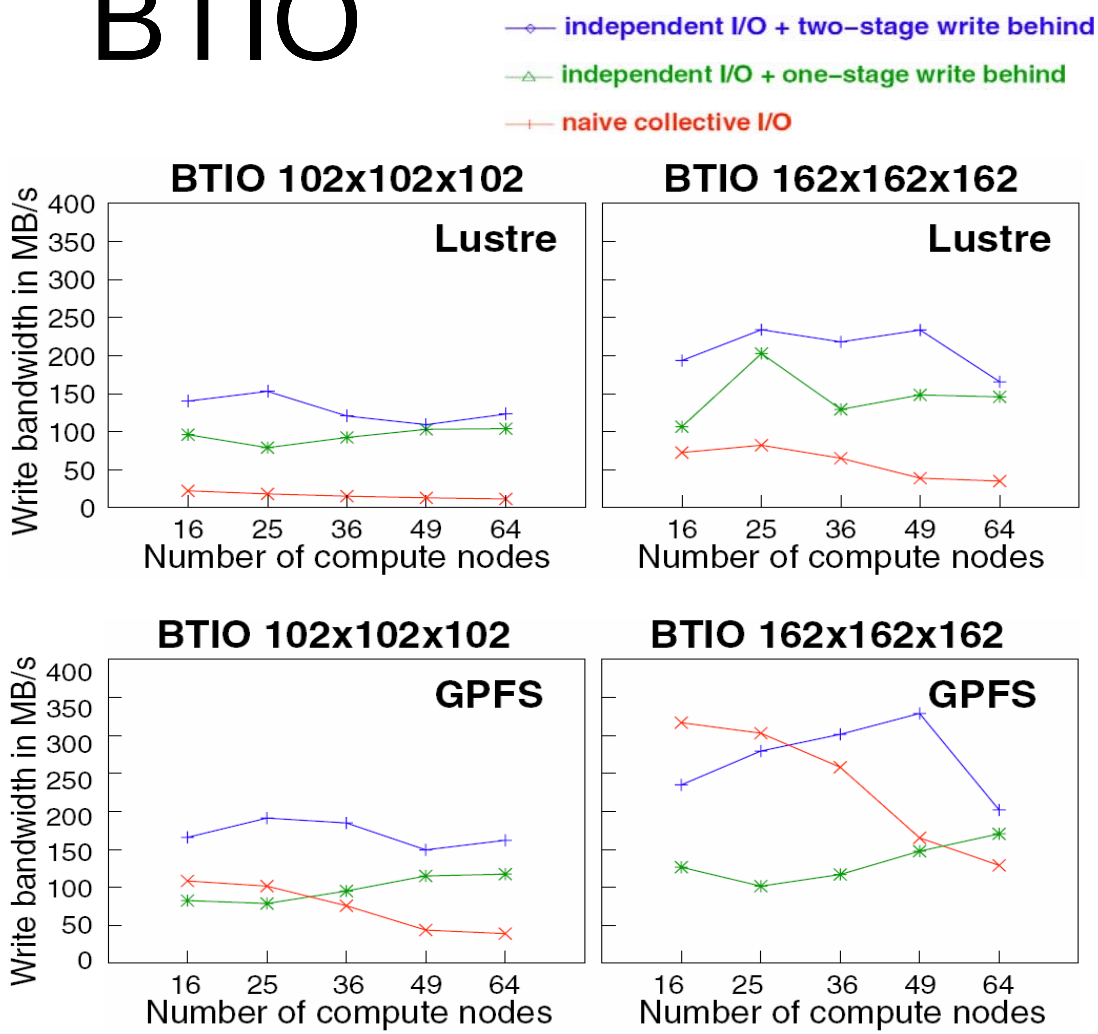
- Platforms
  - Tungsten, a Linux cluster @ NCSA running Lustre
  - Mercury, an IBM cluster @ NCSA running GPFS
- MPICH
  - Configured with using Gigabit Ethernet (currently MPICH's multi-threading is only supported for socket channel)
- BTIO Benchmark NASA
  - Traditionally, using collective I/O has been reported much better than using independents
  - Independent BTIO has significantly larger number of requests

Number of write requests per MPI process

Number of compute nodes	BTIO Class B		BTIO Class C	
	collective	independent	collective	independent
16	40	104040	40	262440
25	40	83240	40	209910
36	40	69360	40	174960
49	40	59400	40	150000
64	40	52000	40	131240

BTIO





- One-stage write behind uses only the 2nd-stage buffering
- The bandwidth for native independent I/O (not shown) is **less than 5 MB/sec**

Summary

- MPI independent I/O
  - Usually, independents perform worse than collectives
  - Most of the existing optimizations for collectives is not applicable
- Write-only pattern
  - Occupies 90% of the I/O activities in scientific applications
  - Can be improved by write behind
- File locking in parallel file system
  - Ensures cache coherence
  - The main cause of I/O bottleneck
- Improvement by two-stage write-behind method
  - In our experience, MPI independent I/O can even outperform collectives