Y-Lib: A User Level Library to Increase the Performance of MPI-IO in a Lustre File System Environment

Phillip M. Dickens

Jeremy Logan

Department of Computer Science

University of Maine

Data Intensive Applications

- Large-scale computing clusters are being increasingly used to execute large, data-intensive applications.
  - Data sets ranging from gigabytes to terabytes and beyond.
  - I/O requirements are becoming a significant bottleneck in application performance.
- Led to very powerful parallel file systems that can accommodate concurrent file system accesses by thousands of clients (e.g., Lustre, GPFS).

MPI-IO

- Scalable performance also requires flexible parallel I/O interfaces with high-performance implementations to optimize access.

- MPI-IO generally considered de-facto parallel I/O API.
  - MPICH-2 is one important implementation of MPI.
  - ROMIO implements parallel I/O API (perhaps most widely used implementation).

Lustre File System

- Lustre consists of three main components:
  - File System Client: (Request I/O services).
  - Object Storage Servers (OSSs, provide storage services).
    - Each OSS can manage multiple Object Storage Servers (handle object storage and management).
  - Meta-data servers (manage the namespace).

Lustre File System

- Lustre is object-based file system where OSTs manage the objects they control.

- Two features lead to enhanced performance.

  - Meta-data stored separately from file data.
    - Once meta-data acquired can interact directly with the OSTs.

  - Files can be striped across multiple OSTs.
    - Provides concurrent access to shared files by multiple application processes.

Problem

- Quite often MPI-IO performs very poorly in Lustre file systems.
- One obvious reason is that Lustre exports the POSIX file system API.
  - Difficult to implement high-performance parallel I/O.
- Less obvious reason is that the assumptions upon which most important parallel I/O optimizations are based do not hold in a Lustre environment.
  - Key assumption: ***Performing large, contiguous I/O operations in parallel provides the optimal parallel I/O performance.***

Collective I/O

- Collective I/O operations
  - All processes make the I/O call and provide their individual I/O requests.
  - Provides significant information to implementation about aggregate I/O request.
  - Implementation can often combine small, non-contiguous I/O requests into larger, contiguous requests.
- Implemented in ROMIO using *two-phase I/O.*
  - *First phase: I/O requests are combined and data is redistributed among aggregator processes to put into correct order.*
  - *Second phase: Data is written to disk.*

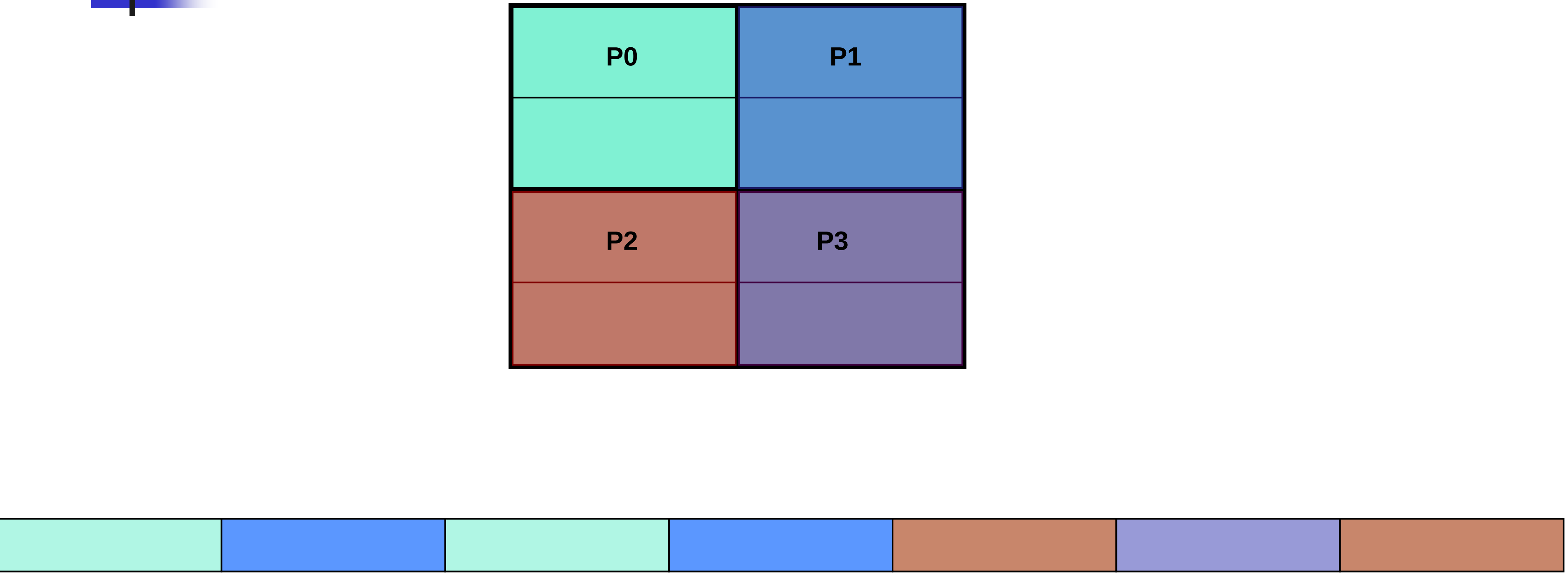
Collective I/O

- Implemented in ROMIO using *two-phase I/O.*
  - *First phase: I/O requests are combined and data is redistributed among aggregator processes to put into correct order.*
  - *Second phase: Data is written to disk.*

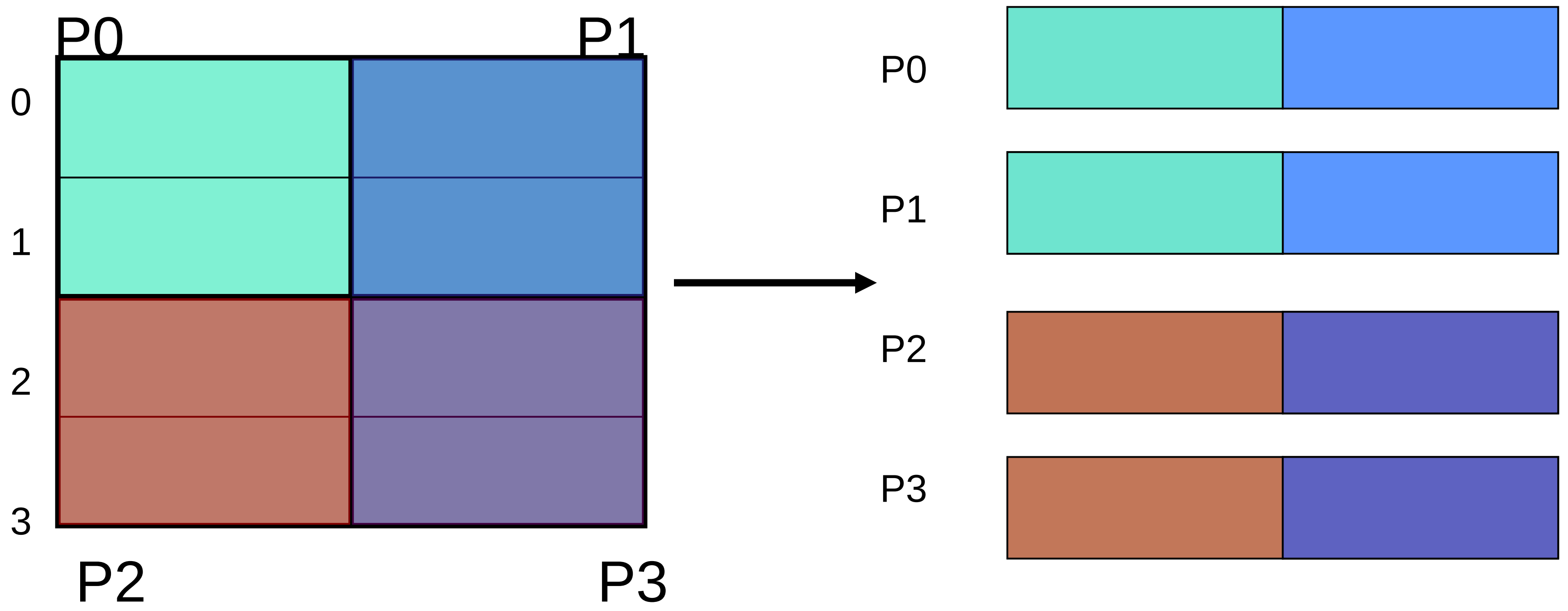
Collective I/O

- Implemented in ROMIO using *two-phase I/O.*
  - *First phase: I/O requests are combined to obtain picture of the aggregate I/O request.*
    - *Then redistribute data among aggregator processes to put into correct order.*
  - *Second phase: Data is written to disk concurrently.*

Simple Example

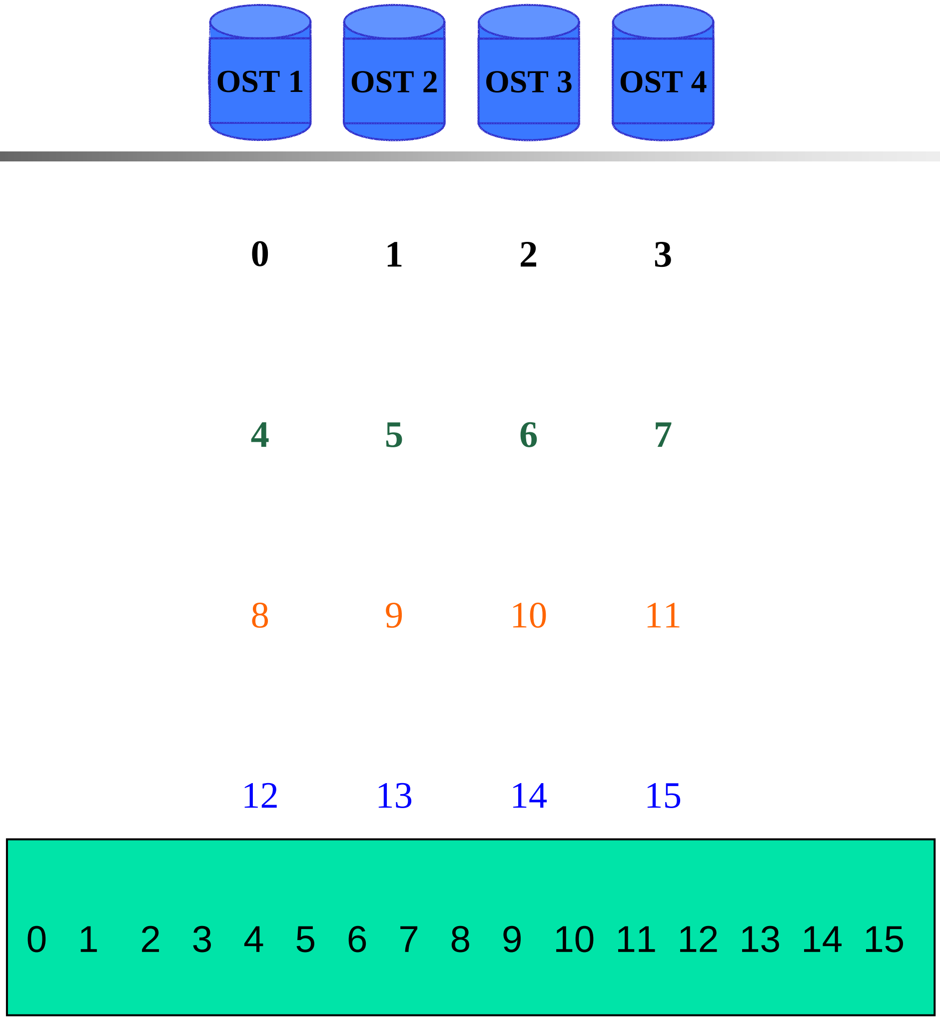


Conforming Distribution

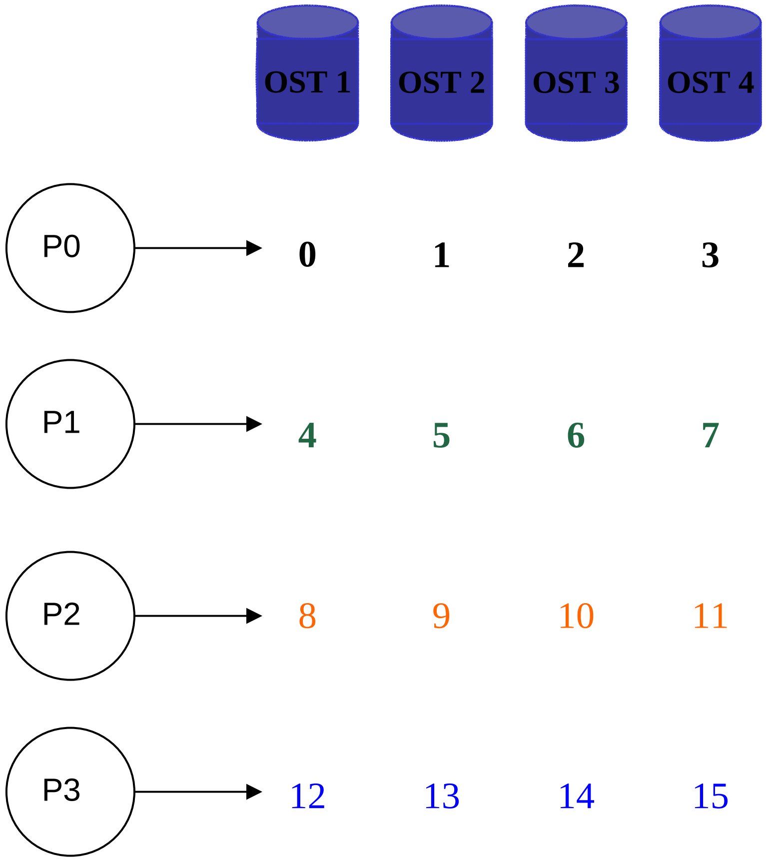


Lustre API

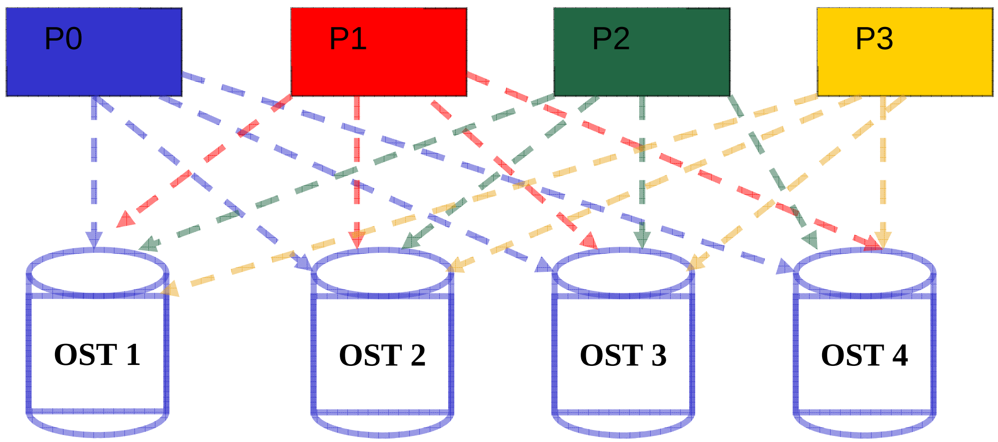
- User can set:
  - stripe size
  - stripe width (number of OSTs across which file is striped)
  - beginning OST
- File system stripes objects across OSTs in a round-robin fashion, starting from user-specified start.


OST 1
OST 2
OST 3
OST 4
0 1 2 3
4 5 6 7
8 9 10 11
12 13 14 15
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

ROMIO redistributes data into conforming distribution.



All-to-All Communication Pattern



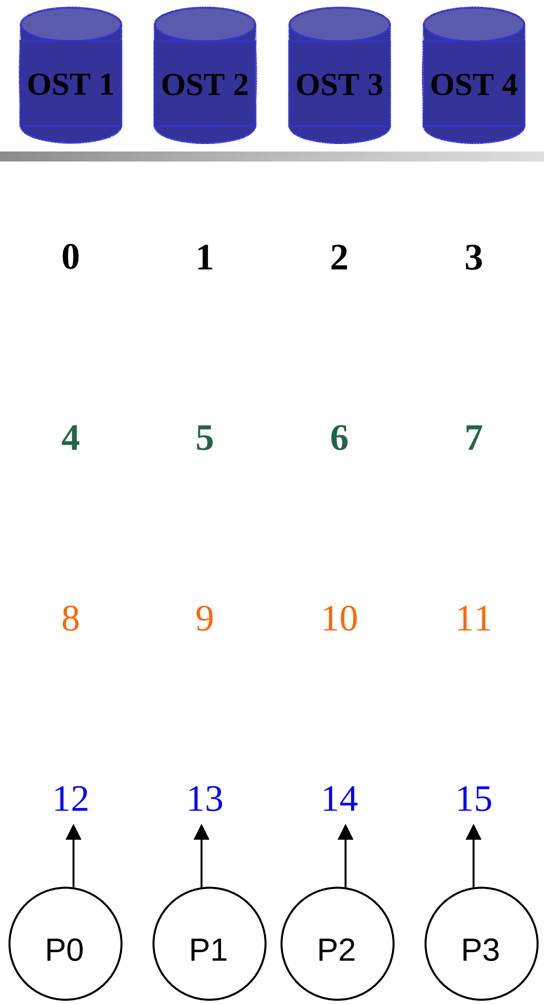
All-to-All Communication Pattern

- Well known problem
- Generally suggested to limit stripe width
  - Frequently see default and suggested width of four.

- Reduces contention but significantly limits parallelism and parallel I/O performance.
- Problem is that this data aggregation pattern is not well aligned with Lustre’s object-based storage architecture.

Y-Lib

- Believe it is possible to stripe across large numbers of OSTs and minimize contention.
- Accomplished by a user-level library termed Y-Lib that redistributes data in a different pattern.
  - Redistributes data such that the number of OSTs with which a given process communicates is limited (one-to-one OST pattern).
  - Process write data to OSTs by performing a set of independent writes concurrently.


OST 1
OST 2
OST 3
OST 4
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
P0
P1
P2
P3

Trade Offs

- All-to-All make few large requests.
- One-to-One make a large number of small I/O requests.

Experimental Design

- Conducted experiments on Ranger at the Texas Advanced Computing Center (TACC, University of Texas).
- Lustre file system used here consisted of 50 OSSs, each of which hosted 6 OSTs (total of 300 OSTs, 1.73 Petabytes of storage).
- Studied the throughput obtained in collective write operations.

Parameters

- Varied three key parameters:
  - Data redistribution pattern.
  - Number of aggregator processes (128 - 1024).
  - File size
  - Each process wrote 1 Gigabyte
    - File size varied from 128 Gigabytes to 1 Terabyte.
  - Maintained constant 128 OSTs.
  - Bottleneck was 1-Gigabyte/second throughput from the OSSs to the network.
  - Results are mean of 50 trials taken over 5 days.

Parameters

- Stripe size was constant at 1 Megabyte.
- Each process wrote 1 Gigabyte
  - In the case of Y-Lib each process performed 1024 independent write operations.
  - File size varied from 128 Gigabytes to 1 Terabyte.
- Maintained constant 128 OSTs.
- Bottleneck was 1-Gigabyte/second throughput from the OSSs to the network.

Parameters

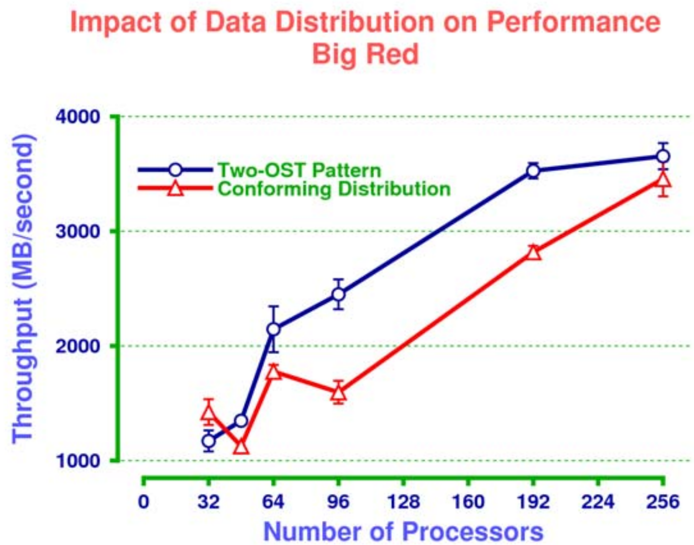
- Maintained constant 128 OSTs.
- Bottleneck was 1-Gigabyte/second throughput from the OSSs to the network.
- Each data point is the mean of 50 trials taken over 5 days.
  - Also show 95% confidence intervals.

Data Aggregation Patterns

- Redistribution required
  - MPI: Assigned data in one-to-one OST pattern and set the hint to use two-phase I/O.
  - Y-Lib: Initially in conforming distribution with collective call to Y-Lib.
- No redistribution required
  - MPI data in conforming distribution
  - Y-Lib data in one-to-one OST pattern.

MPI I/O Write Strategies

- Can the performance of MPI-IO itself be improved using this technique?
  - Forced to use one-to-one OST pattern with concurrent, independent writes.
  - Set the file view specifying one-to-one OST pattern and disabled two-phase I/O and data sieving.



Impact of Data Distribution on Performance
Big Red
Two-OST Pattern
Conforming Distribution
Throughput (MB/second)
4000
3000
2000
1000
0
32
64
96
128
160
192
224
256
Number of Processors