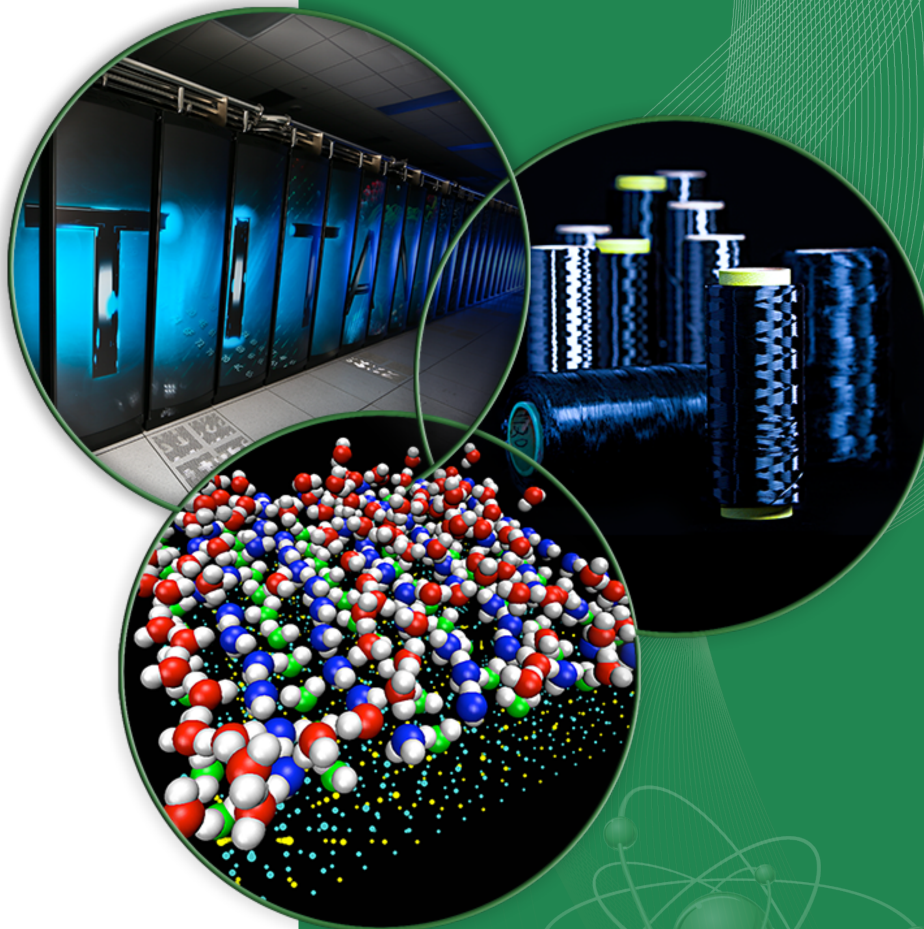
Oak Ridge National Laboratory
Computing and Computational Sciences Directorate



Evaluating Progressive File Layouts for Lustre

Rick Mohr (University of Tennessee)

Michael Brim (ORNL)

Sarp Oral (ORNL)

Andreas Dilger (Intel)

ORNL is managed by UT-Battelle
for the US Department of Energy

OAK RIDGE National Laboratory

Agenda

- Progressive File Layout (PFL) Overview
- PFL Prototype Implementation
- Streaming I/O Tests
- Comparison to Synthetic Dynamic Striping
- Object Placement Testing

Lustre File Layouts Today

- Several Lustre parameters control file layout
  - Stripe size
  - Stripe count
  - Stripe index
  - Pool
- In practice, stripe size and count are primarily used
- Layout constraints
  - One set of parameters for entire file
  - Parameters chosen at time of file creation
  - Only one layout type (RAID-0) supported

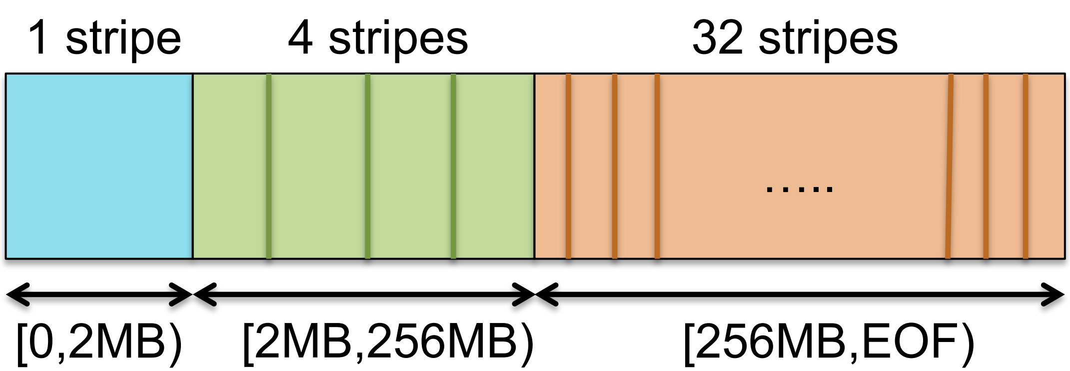
Layout Enhancement

- Under OpenSFS contract SFS-DEV-003, Intel's High Performance Data Division produced a new design for file layouts
- High Level Design document describes several new layouts
  - Composite layouts
  - RAID layouts
  - Compact layouts
  - Large layouts

http://wiki.lustre.org/Layout_Enhancement_High_Level_Design

Progressive File Layout (PFL)

- Progressive File Layout feature is built using Composite Layouts
  - File layout is described by a series of components
  - Each component covers a non-overlapping extent of the file
  - Each component has its own striping parameters



Progressive File Layout Goals

- Single layout definition for multiple file sizes
  - Reasonable performance for a variety of I/O patterns
  - Simplify Lustre usage for novice users
- Change stripe layout as file grows
- More striping options for advanced users
  - Customization for non-uniform files
  - Different regions of file on different storage
- Stepping stone to more features in the future
  - HSM for file components
  - New uses for Composite Layouts

PFL Prototype Implementation

- Intel, under contract from ORNL, has been developing PFL feature
- A prototype implementation was delivered in the first half of 2015 for evaluation and testing
  - Some limits on functionality
    - No dynamic allocation of new components
    - No support for setting PFL on directories
- Continuing development
  - PFL inheritance from parent directory
  - Integration with existing Lustre code base

PFL Evaluation Tests

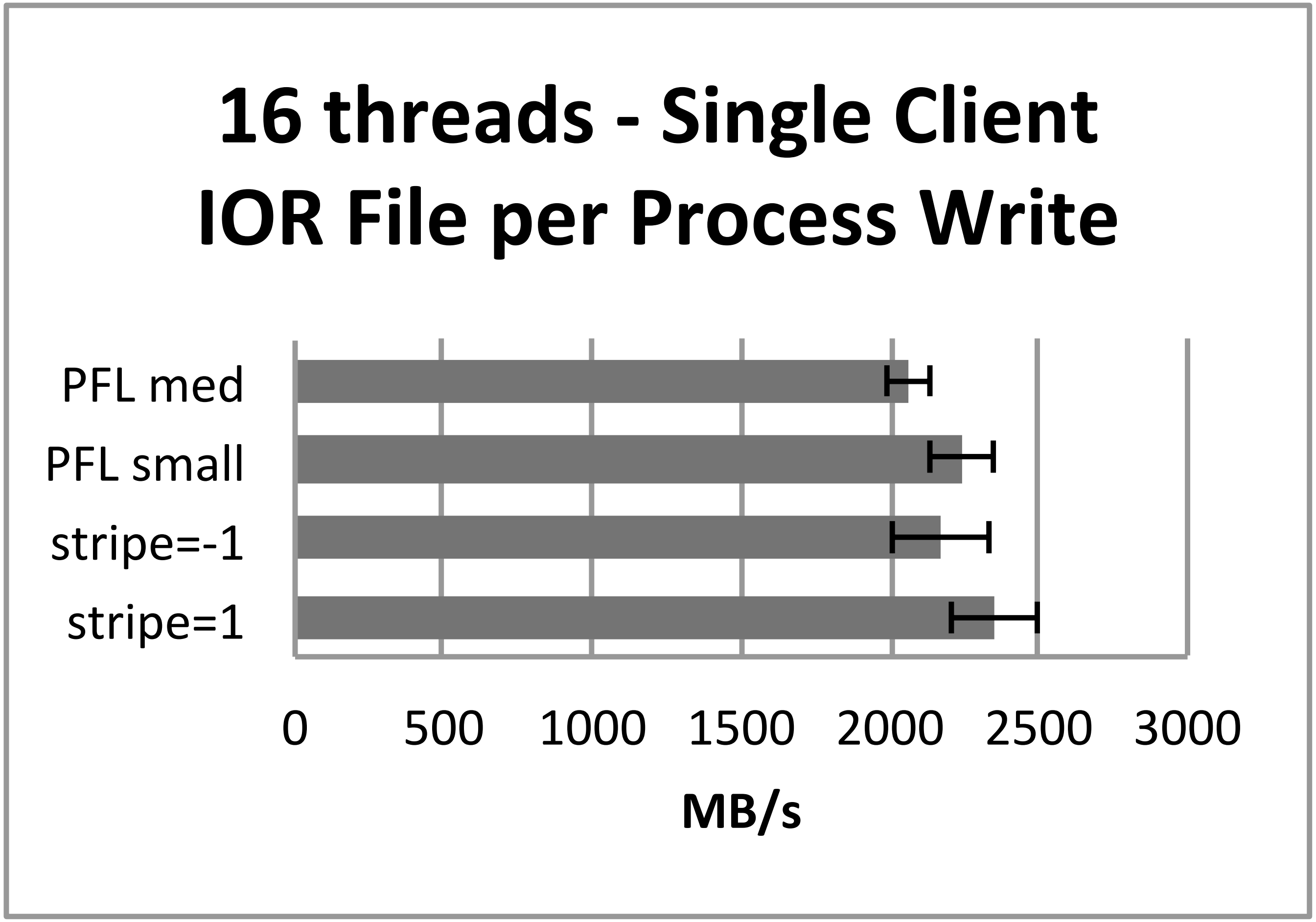
- Several different tests were run to evaluate the functionality and performance of PFL prototype
- Streaming IOR and mdtest (Intel)
  - Comparison with traditional Lustre striping
  - Single client and multiple clients
  - Shared file and file per process
- Comparison to Synthetic Dynamic Striping (ORNL)
- Object placement (ORNL)
  - Difference in OST object placement between PFL and traditional striping

Intel Testing

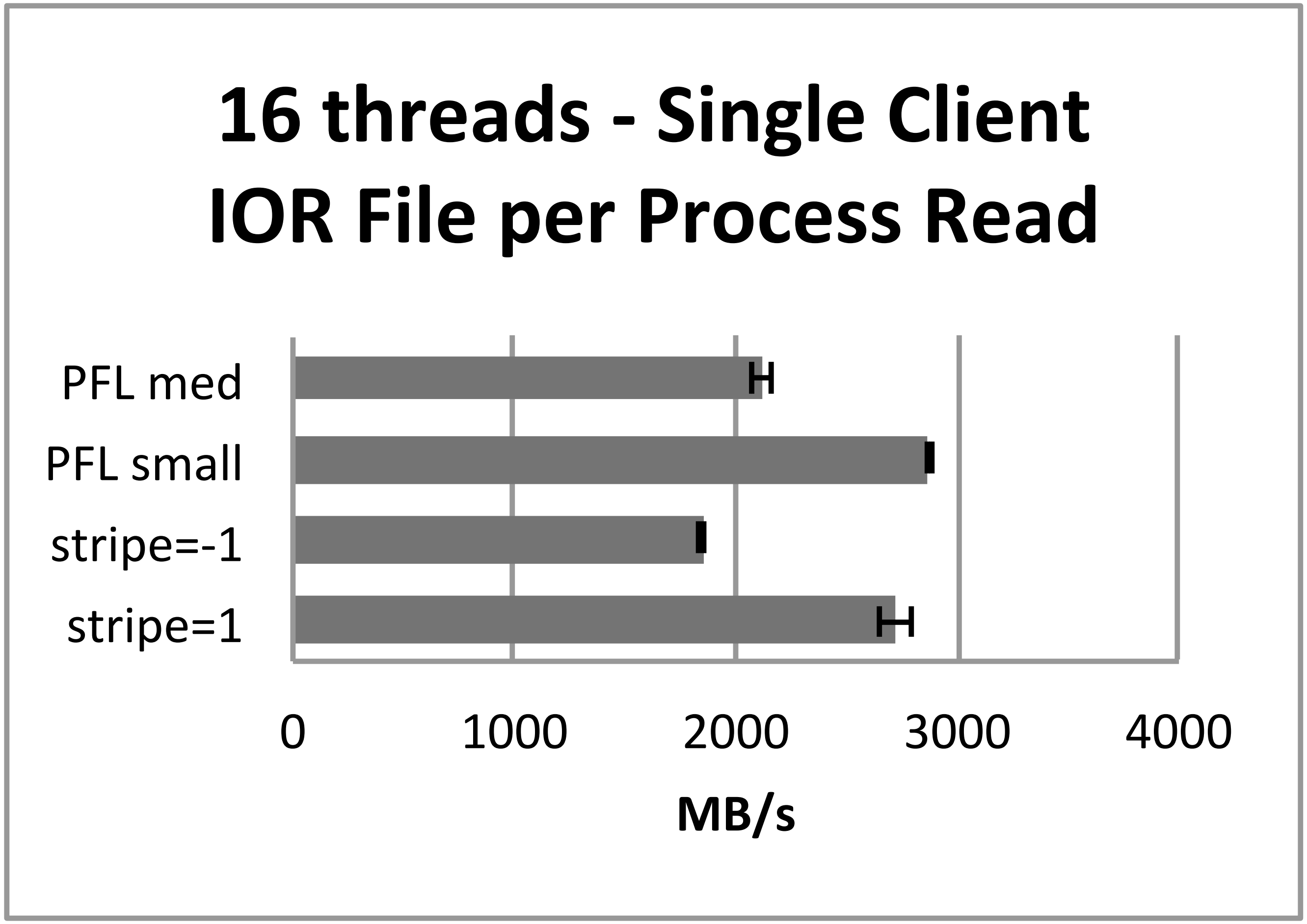
- Intel’s tests were run on Hyperion at LLNL
  - 32 Lustre clients
  - 16 Lustre servers with 52 OSTs (ldiskfs)
  - Mellanox DDR Infiniband
- IOR
  - Single client (16 threads) file per process
  - 32 clients (512 threads) file per process
  - 32 clients (512 threads) shared file
  - Each thread reads/writes 4 GB of data

Test File Layouts

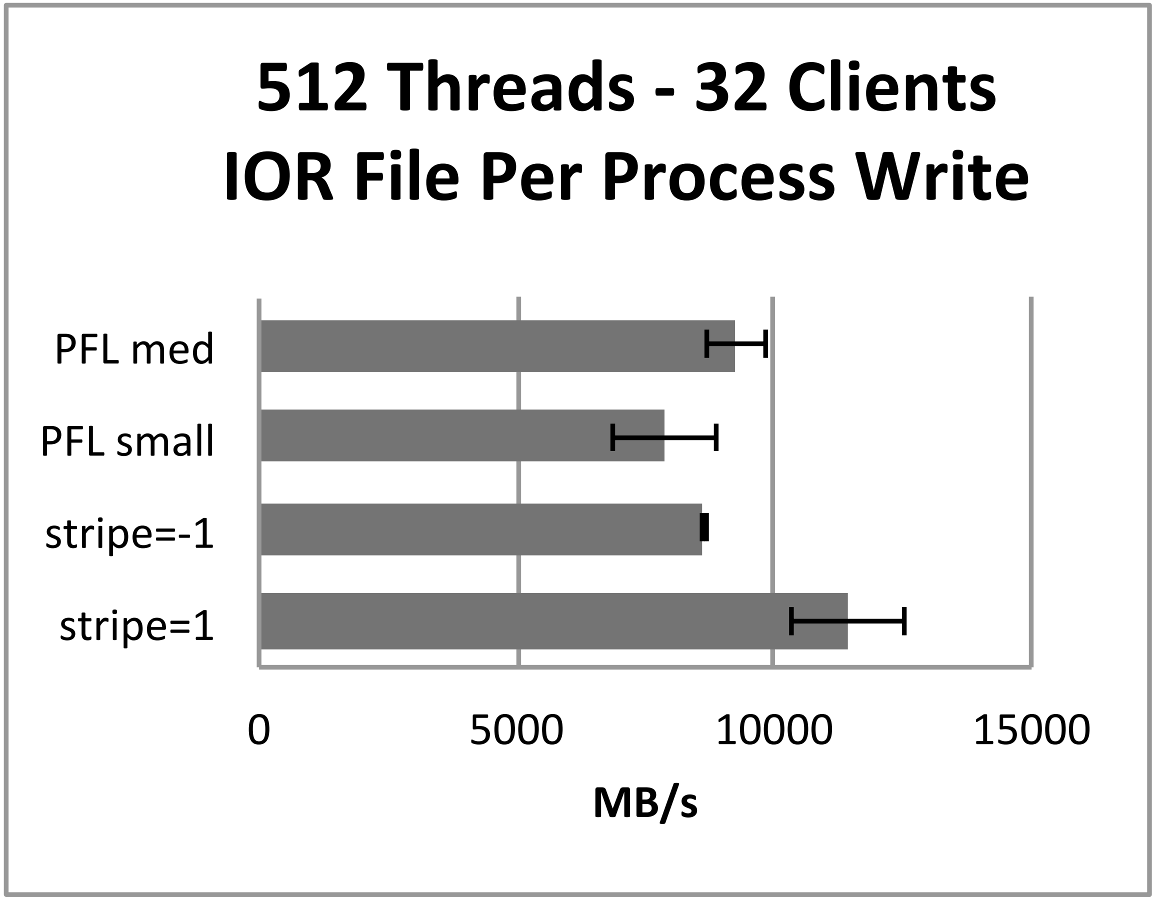
- Tests compared several different layouts
  - Traditional
    - stripe_count = 1 and stripe_count = -1
  - PFL small
    - [0,EOF) → stripe_count=1
  - PFL medium
    - [0,16M) → stripe_count=1
    - [16M, EOF) → stripe_count=4
  - PFL large
    - [0,16M) → stripe_count=1
    - [16M, 128M) → stripe_count=4
    - [128M, EOF) → stripe_count=47



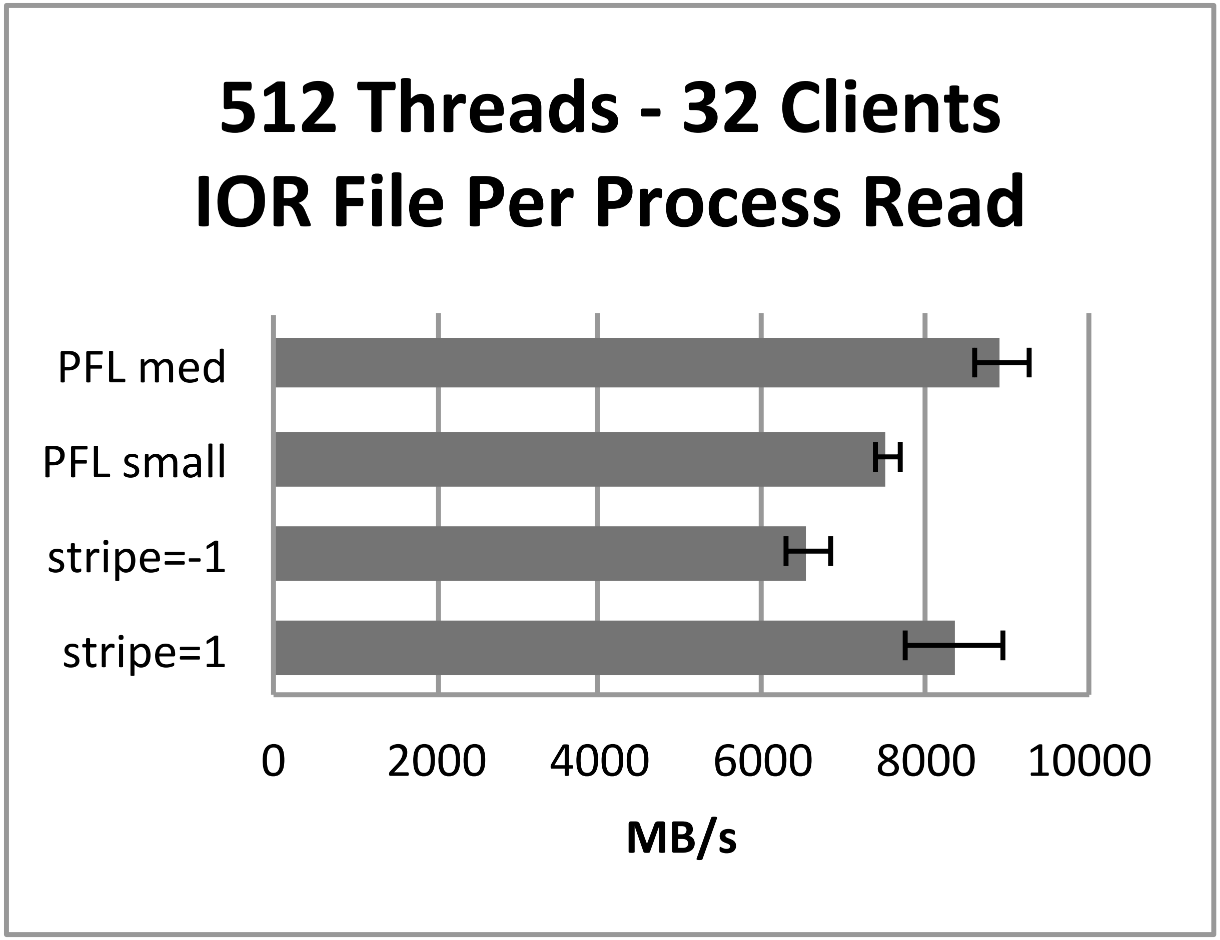
- Performance of PFL small and stripe=1 are comparable
- Performance of PFL med is lower than stripe=-1, but they are within error margins



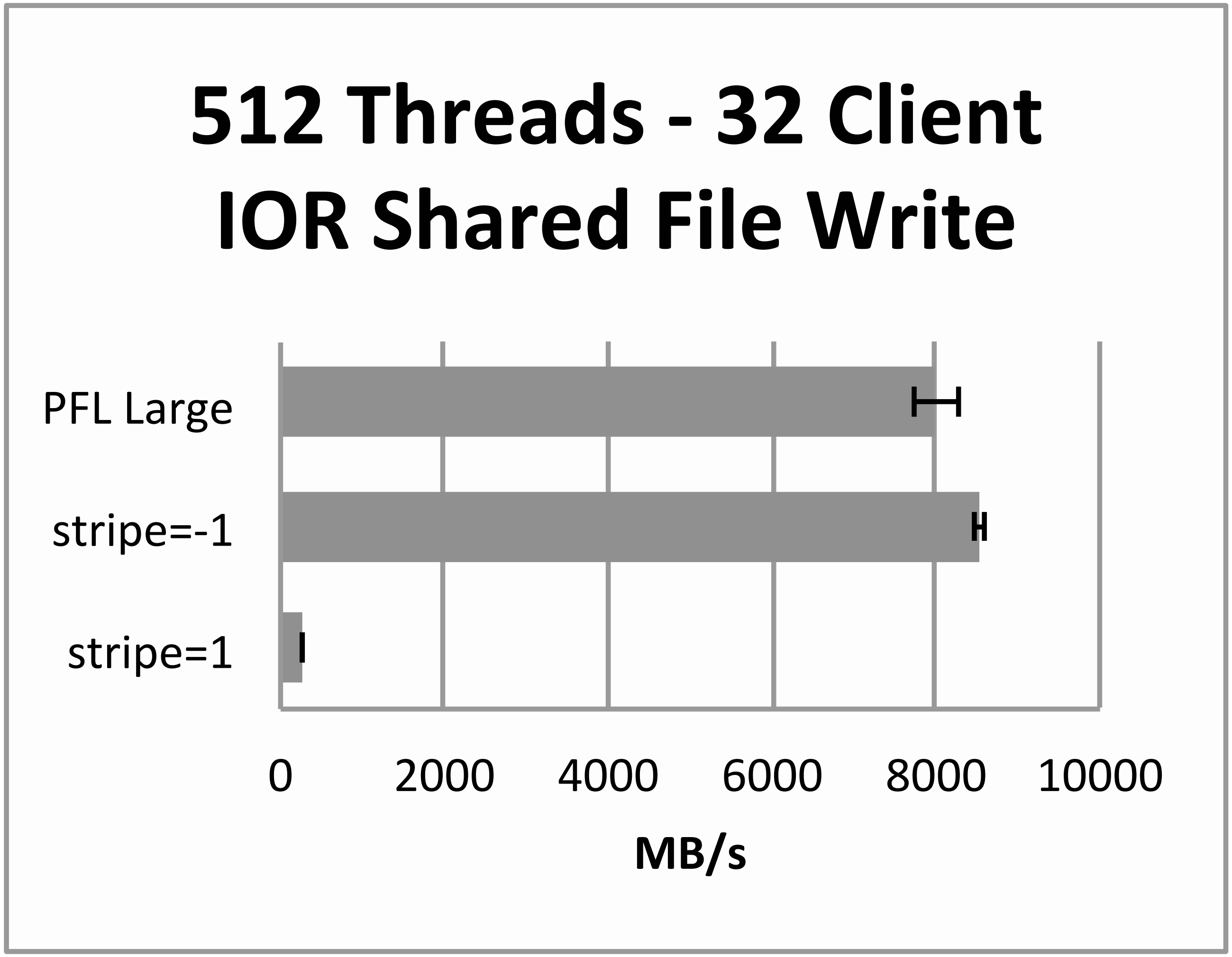
- Performance of PFL small is slightly better than stripe=1, but still comparable
- Performance of stripe=-1 is lower than PFL med due to more contention at OST level



- Performance of stripe=-1 is less than stripe=1 due to increased contention on the OSTs
- Performance for PFL small is less than stripe=1 (which is not expected). Large variances may indicate contention from other processes on test file system.



- Performance of PFL small is similar to stripe=1 although somewhat less. (Again, large variance for stripe=1 may indicate contention.)
- Performance of PFL med is better than stripe=-1 due to less contention on OSTs



- Performance for stripe=1 much less than stripe=-1 (as expected)
- Performance of PFL large is on par with stripe=-1

ORNL Lustre Testbed

- 8x OSS Servers (Dell R720)
  - 2x Intel Xeon 2630 v2
  - 64 GB RAM
  - 250 GB 7.2K RPM SATA3 drive
  - Mellanox ConnectX-3 FDR HCA
- 2x MDS Servers (Dell R720)
  - Same as OSS servers except:
    - 128 GB RAM
    - 6x 300 GB 15K RPM SAS drives
- 8 OSTs per OSS server
  - OSTs use ZFS backend

Synthetic Dynamic Striping

- Prior to the PFL prototype, a simulated form of dynamic striping was tested
  - Files were split into smaller components
  - Each component was created in a different directory with different stripe counts
  - Applications were modified to perform I/O to file components
    - IOR
    - BLAST

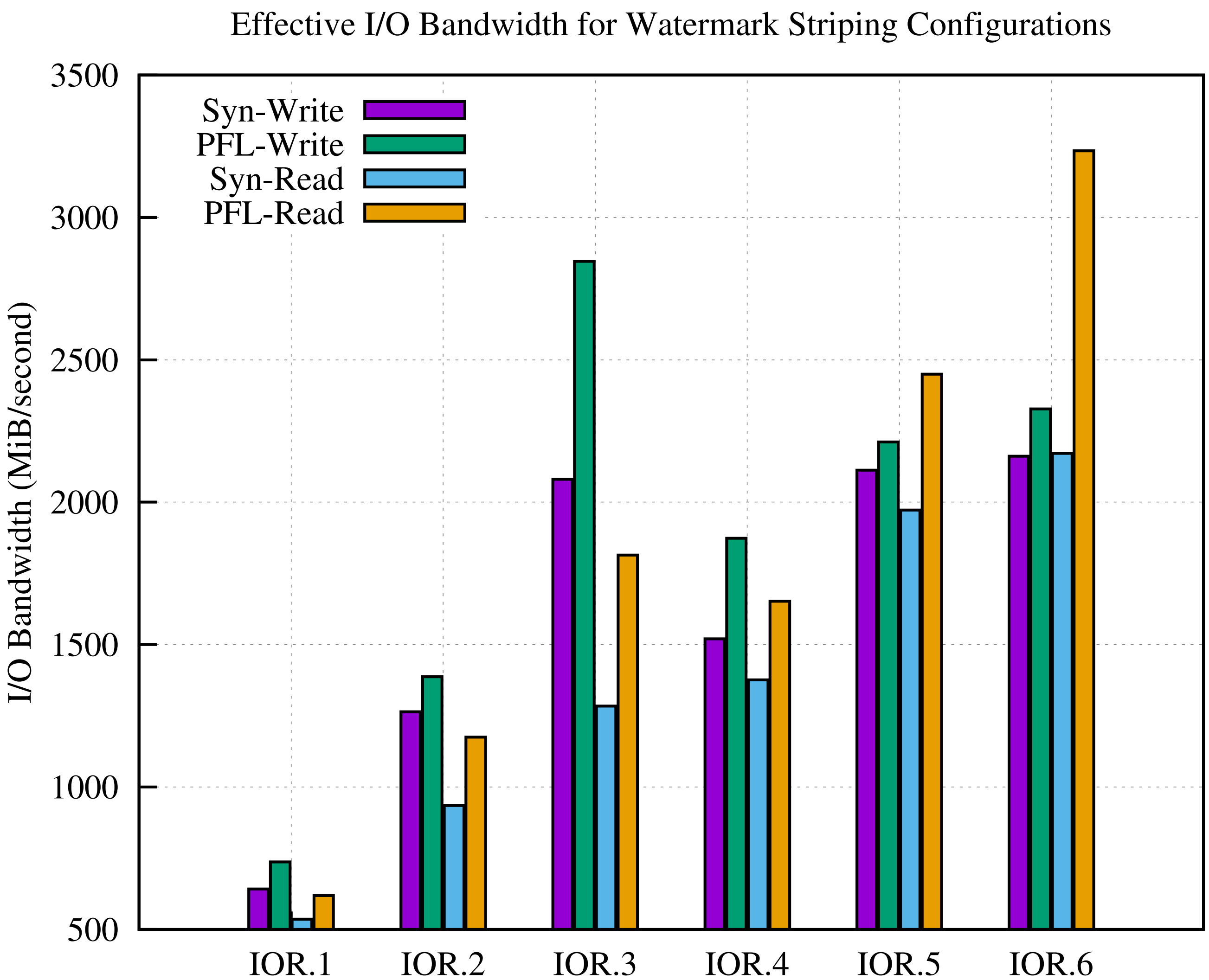
http://arxiv.org/pdf/1504.06833v1.pdf

Dynamic Striping Test Cases

- IOR POSIX shared file
  - 16 compute nodes with 4 processes per node
  - 4 TB total file size

Standard	PFL
[IOR.1] Entire file: stripe_count=4	[IOR.4] 0-1 TB: stripe_count=4 Remainder: stripe_count=8
[IOR.2] Entire file: stripe_count=8	[IOR.5] 0-1 TB: stripe_count=4 Remainder: stripe_count=16
[IOR.3] Entire file: stripe_count=16	[IOR.6] 0-1 TB: stripe_count=4 1-2 TB: stripe_count=8 Remainder: stripe_count=16

PFL vs. Synthetic Results



Object Placement Testing

- Users often choose poor striping patterns
  - Large file, small stripe count →Imbalanced OST usage
  - Small file, large stripe count → Sub-optimal performance
- PFL can use increasing stripe count to accommodate multiple file sizes
  - How does using a single PFL layout for all files compare to “ideal” traditional striping?
- Test scenario:
  - Create files with traditional striping
  - Create files with single PFL layout
  - Compare distribution of OST usage

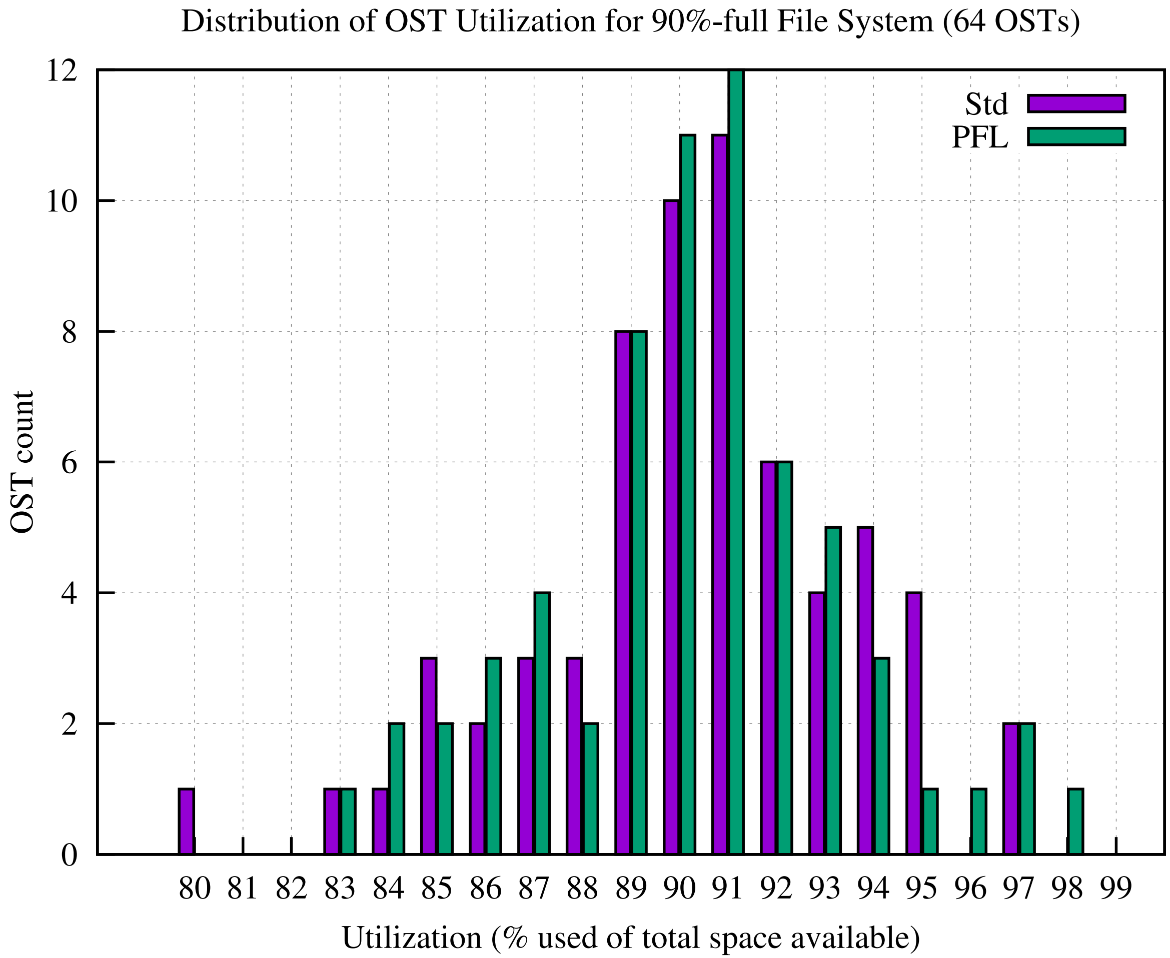
Object Placement Testing (cont.)

- Choose file size distribution (based on OLCF stats)

File Size	Percentage	Stripe Count
1 MB	70%	1
64 MB	20%	4
128 GB	9%	16
4 TB	1%	48

- Choose PFL Layout
  - [0, 1MB) stripe_count=1
  - [1MB, 64MB) stripe_count=4
  - [64MB, 128GB) stripe_count=16
  - [128GB, EOF) stripe_count=48
- Fill file system to 90% capacity

Object Placement Results



- Distribution of OST utilization for PFL files is very similar to the distribution seen using ideal striping parameters

Summary

- Progressive File Layouts provide additional flexibility when defining the striping configuration for a file
- PFL performance appears to be on par with traditional Lustre (and in some cases better)
- Single PFL layout can be effectively used for files of widely varying sizes
  - Can help simplify Lustre usage for users
  - Will save some headaches for sys admins

Future Work

- PFL Implementation
  - Layout inheritance from parent directory
  - Define PFL layout as default for file system
  - Improved OST allocator
  - Dynamic component instantiation

- PFL Testing
  - More data intensive workloads
  - Increase scaling

Acknowledgements

This work was supported by the United States Department of Defense (DoD) and used resources of the Computational Research and Development Programs at Oak Ridge National Laboratory.

Questions?