

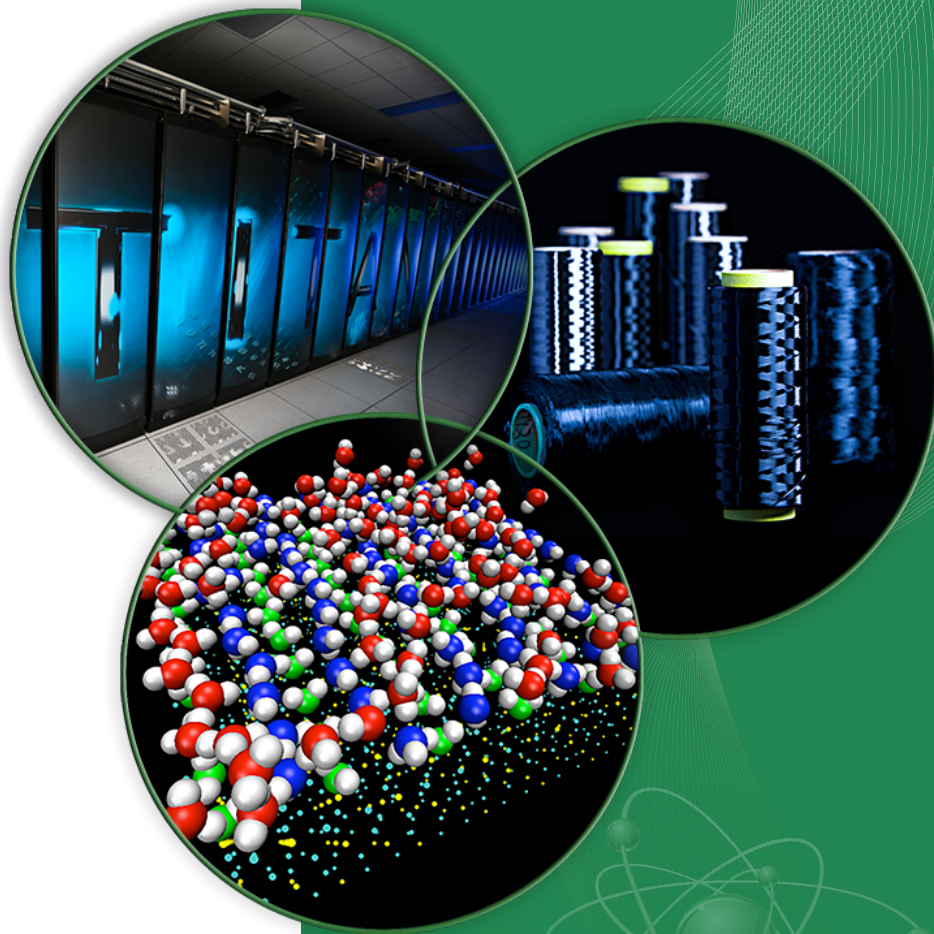
Evaluating Dynamic File Striping for Lustre

Joel W. Reed

Jeremy Archuleta

Michael J. Brim

Joshua Lothian



Outline of Presentation

- Motivation
- Possible Solution
- Experimental Method
- Experiment 1: Isolated Testbed
 - Description
 - Modified IOR results
 - Netflow workload results
- Experiment 2: Center-wide Shared Resource
 - Description
 - Blast workload results
- Conclusions

Motivation

- Data analytic workloads are becoming more prevalent
 - IO patterns and storage requirements differ from traditional scientific workloads
 - Read intensive
 - Data size and IO process quantity not known a priori
 - Often characterized by a sequential indexing scan followed by random analysis phase
 - Data collected from a streaming source makes predicting size for a given collection interval difficult
- Determining the ideal Lustre striping parameters is difficult

Dynamic Striping: A Possible Solution

- The ability to apply different striping parameters to contiguous segments of a file as it grows
 - Initially set stripe parameters, but modify stripe parameters as the file grows beyond one or more thresholds (watermarks)
- If the file ends up being small, it is spread over a small number of servers and utilization is good
- If the file ends up being large, much of it is spread over a larger number of servers and contention is reduced
- Other potential benefits of dynamic striping
 - Fault tolerance: adapt to OSS/OST outages by reducing stripe count
 - Easy expansion onto newly added OSTs
- **Hypothesis: dynamic striping on Lustre should increase read performance of analytic workloads where the file size is unknown in advance**

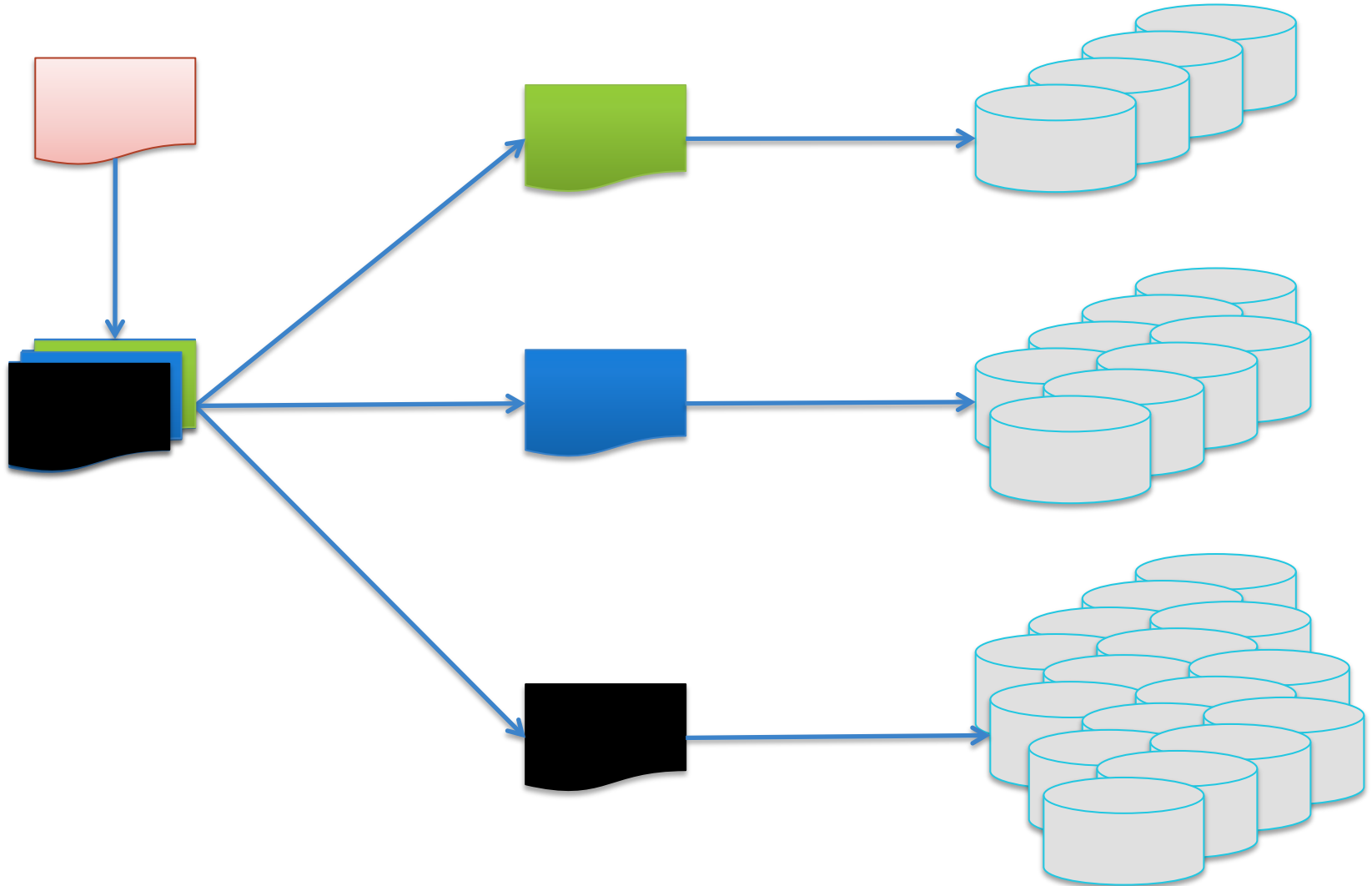
Experimental Method

- Simulate dynamic striping by splitting data file into segments at predetermined watermarks and store these segments in directories with different striping parameters.
- Evaluate using three different workloads
 - Modified IOR
 - A netflow analysis workload
 - The blastn algorithm from NCBI Blast
- Remove caching of results as much as possible
 - Drop client page caches between runs
 - Read large, unrelated file on each client between runs

Simulated Dynamic Striping

Manually split target file into multiple segments

Specify different *stripe_count*, *stripe_width*, for each segment



Experimental Platforms

- ORNL ESSC Lustre Testbed
 - Experiments had exclusive use of the Lustre filesystem
 - Workloads
 - Modified IOR benchmark
 - Netflow workload
- OLCF Titan and Spider
 - Experiment shared Lustre file system with other jobs
 - Workloads
 - NCBI Blast

ORNL ESSC Lustre Testbed

- 35 nodes
 - 2 Lustre MDS
 - 8 Lustre OSS
 - 4 Lustre router (LNET)
 - 1 cluster management
 - 16 Lustre clients
 - 2 large-memory Lustre clients
 - 1 login/scheduling node
- Software
 - Servers: Lustre 2.5.3 with ZFS and Mellanox OFED
 - Clients: Lustre 2.5.2 with Mellanox OFED

ESSC Lustre Testbed - Node Hardware

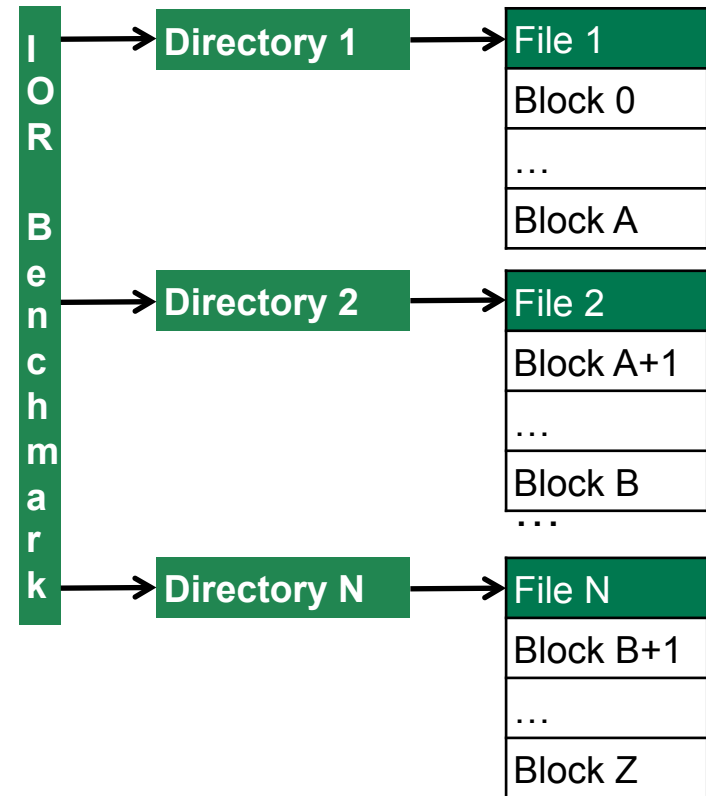
- All nodes based on Dell R720
 - Two Xeon 2630v2 2.6GHz processors
 - 64GB RAM
 - Mellanox ConnectX-3 FDR Infiniband HCAs
- Client RAM: some with 128GB and others with 384GB
- MDS: 128GB RAM and 15K SAS drives for metadata storage
- Connected with Mellanox SX6036 FDR IB switches

ESSC Lustre Testbed - Storage Hardware

- Storage consists of 8 re-purposed LSI 2680 controller pairs
- Each controller pair has two 12-disk shelves with 7200 RPM SATA drives
- Connected to OSS nodes via 8 Gb/s fibre channel through redundant fiber channel switches
- Current configuration is 8 OSTs per OSS
 - each OST is a RAID-0 zpool on top of 3 SATA drives

IOR Benchmark Modifications

- IOR modified to take an ordered list of paths with associated segment sizes and split the file is creates across the directories while abiding by the provided segment size.



```
ior -a ctest -o /path1,1024G,/path2,2048G,/path3 -b 64G -t 4M
```

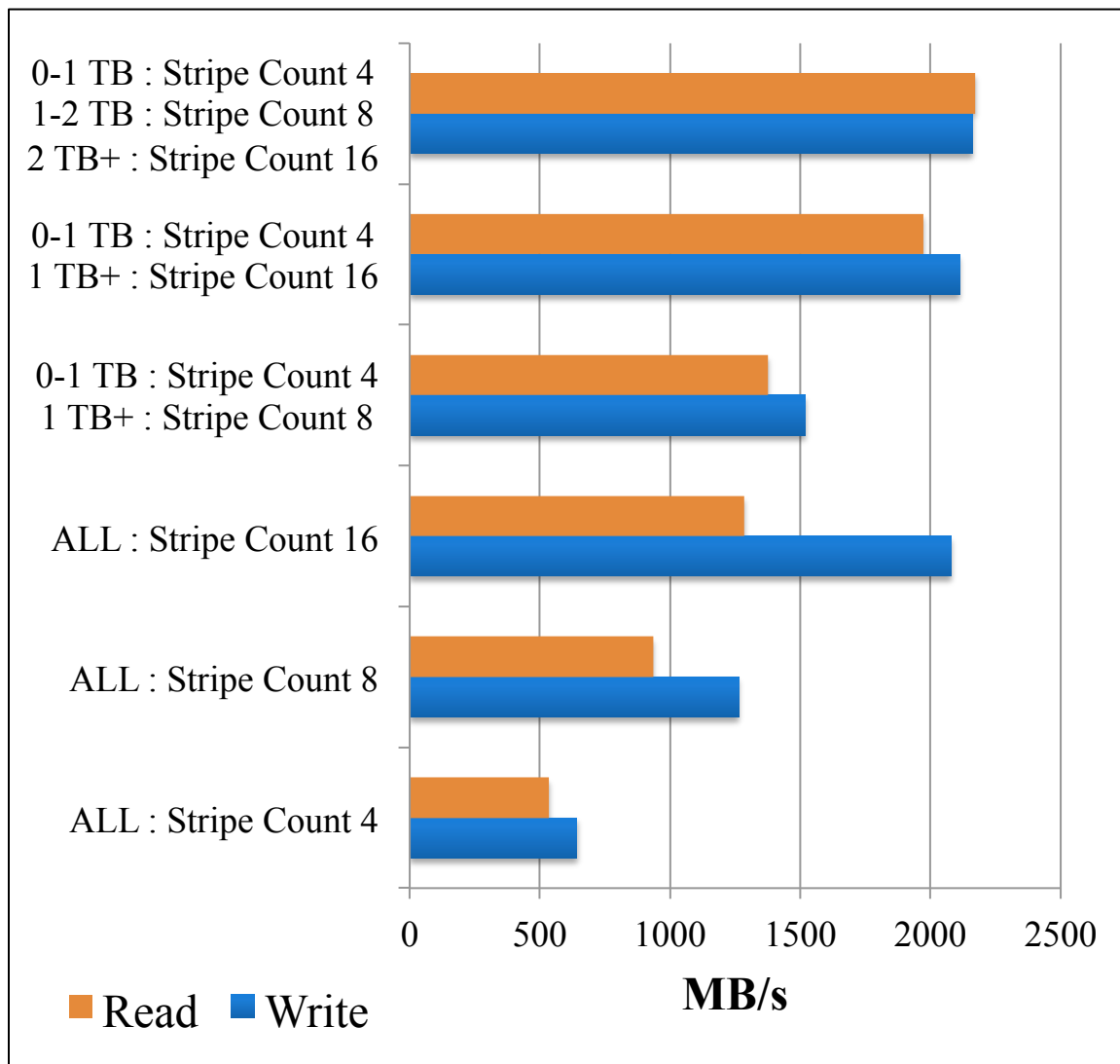
IOR Results / ZFS Cache Enabled

Test parameters:

- 16 testbed nodes
- 64 tasks (4 per node)
- IOR block size of 64 GB
- Total file size 4 TB
- 6 stripe count/watermark configurations; executed five times each

Test results:

- Synthetic dynamic striping improves IOR sequential read performance significantly
- Synthetic dynamic striping does not hurt IOR sequential write performance



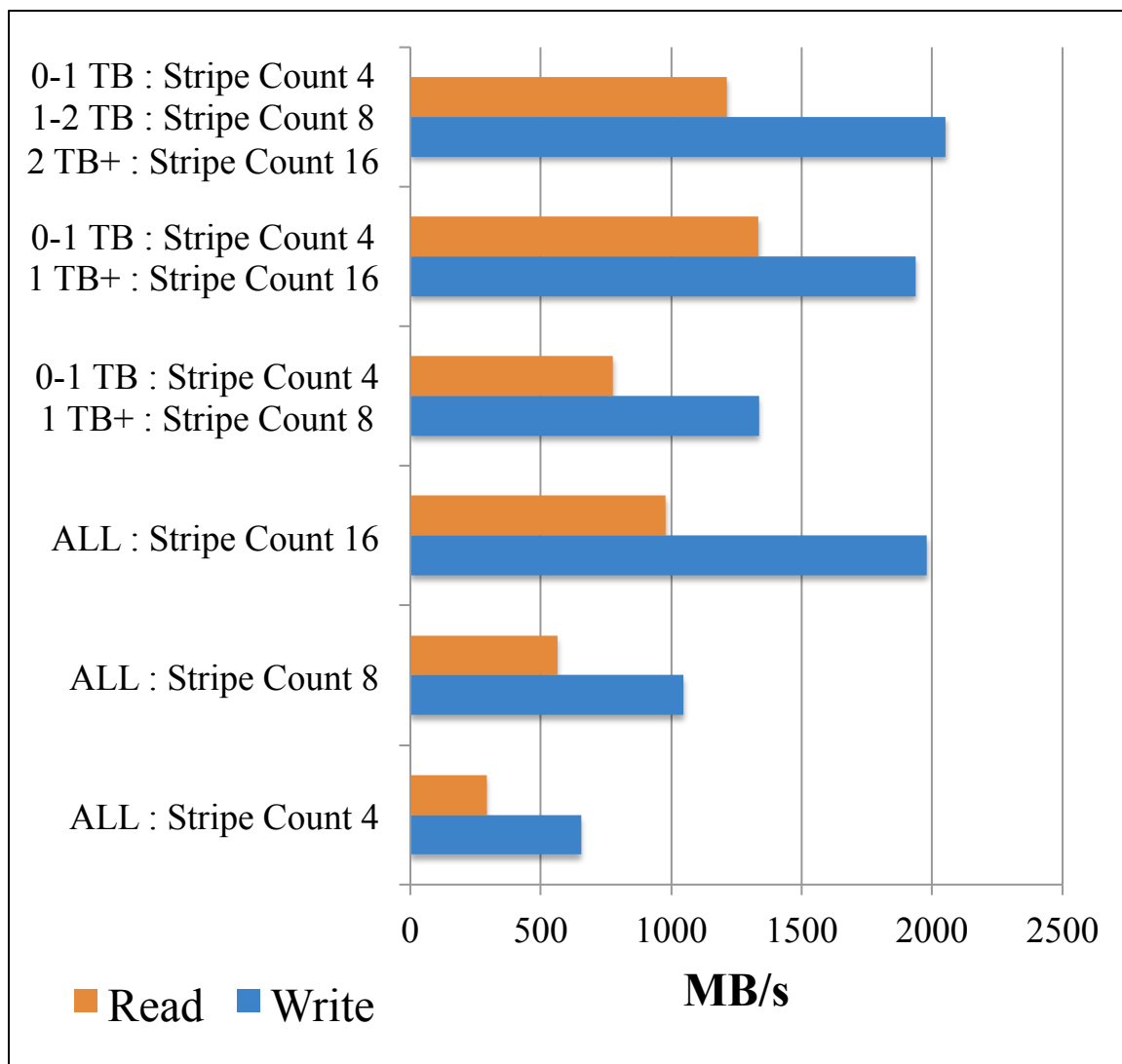
IOR Results / ZFS Cache Disabled

Test parameters:

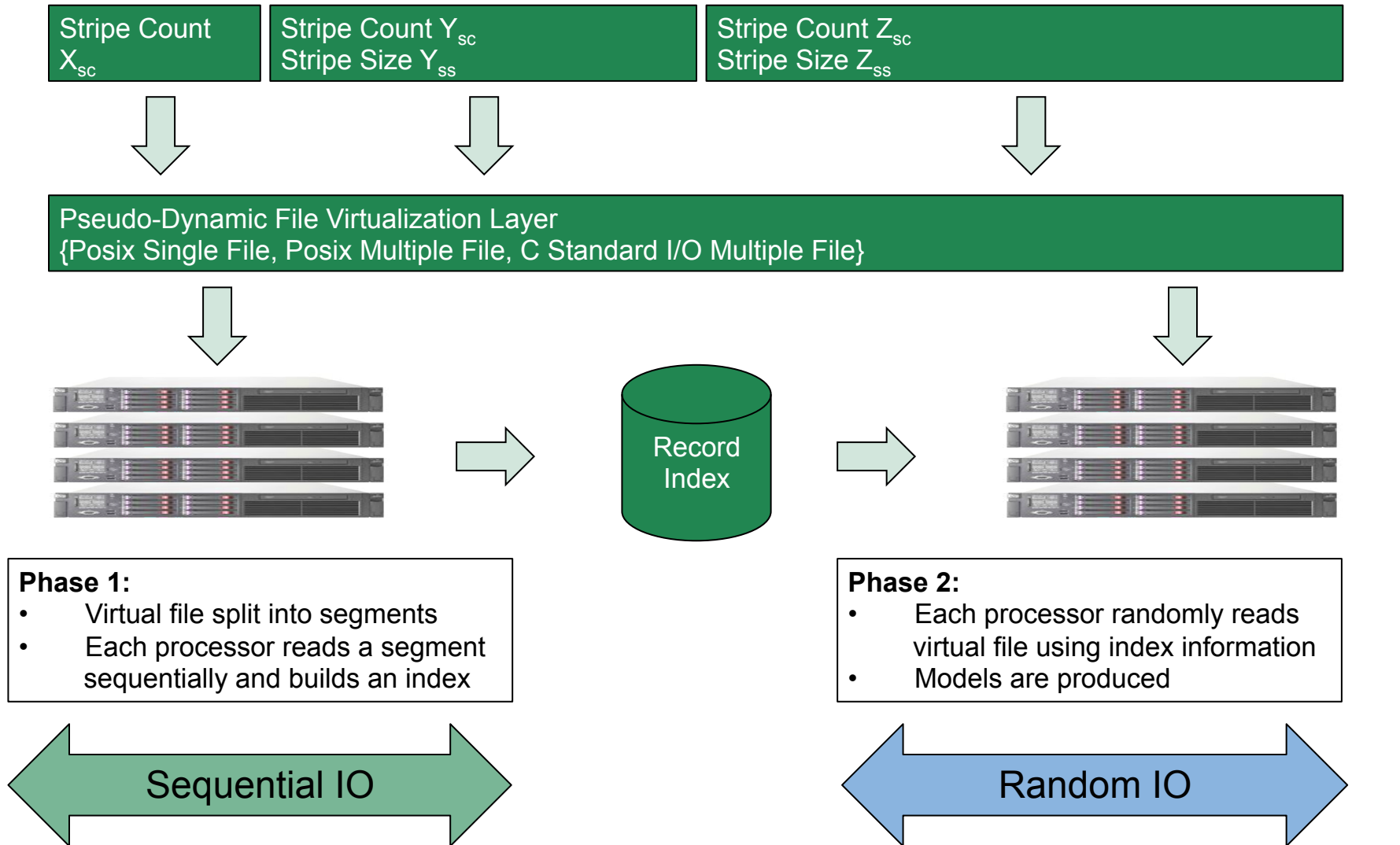
- Same as previous

Test results:

- ZFS Cache – decreased write performance by ~7% and read performance by about ~38% on average
- Synthetic dynamic striping still improves IOR sequential read performance significantly
- Synthetics dynamic striping does not hurt IOR sequential write performance



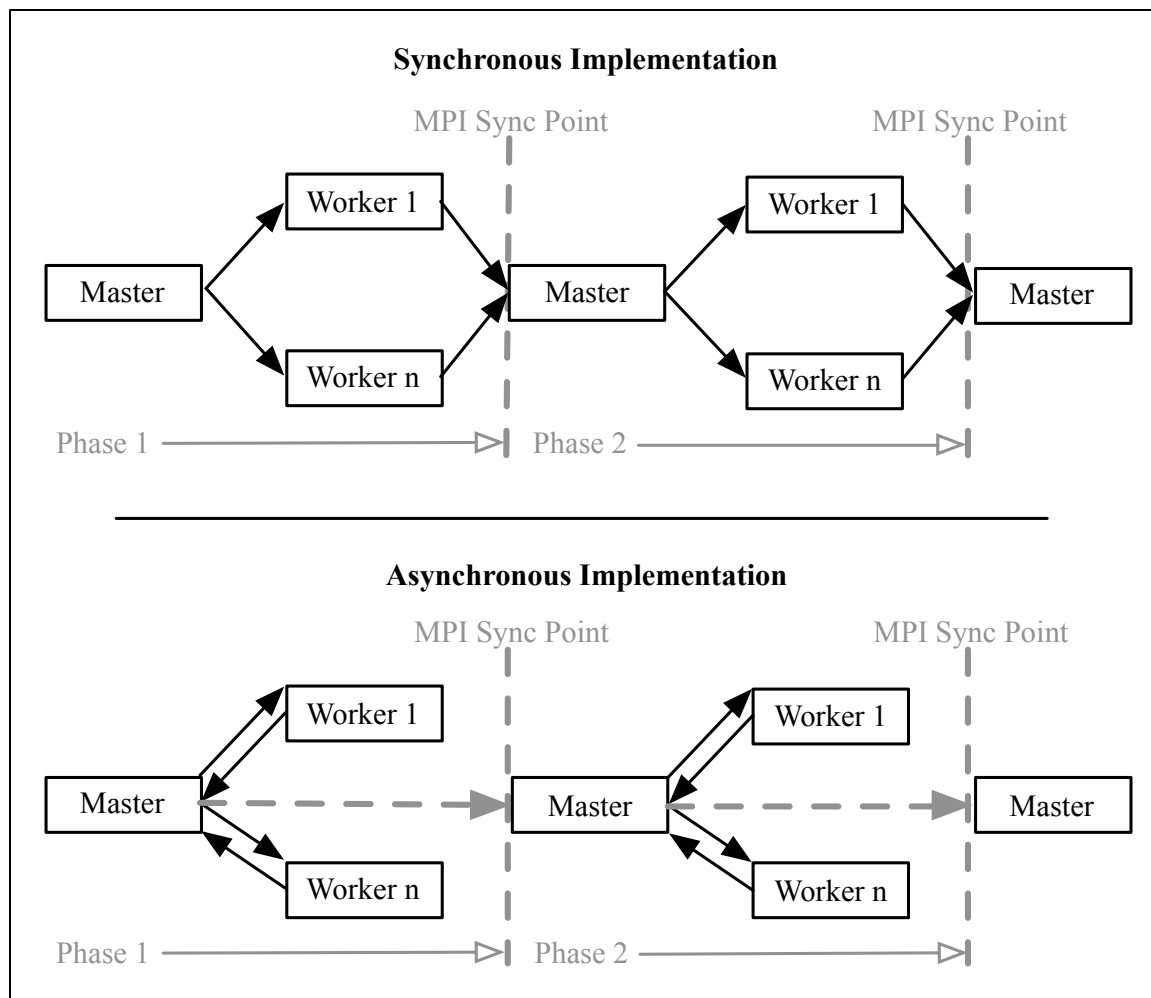
Netflow Workload



Netflow Workload Implementations

Implemented two versions of the netflow workload:

- *Synchronous* -- corresponds to analysis of a static historical data set
- *Asynchronous* -- corresponds to near real time analysis of a data set being actively collected



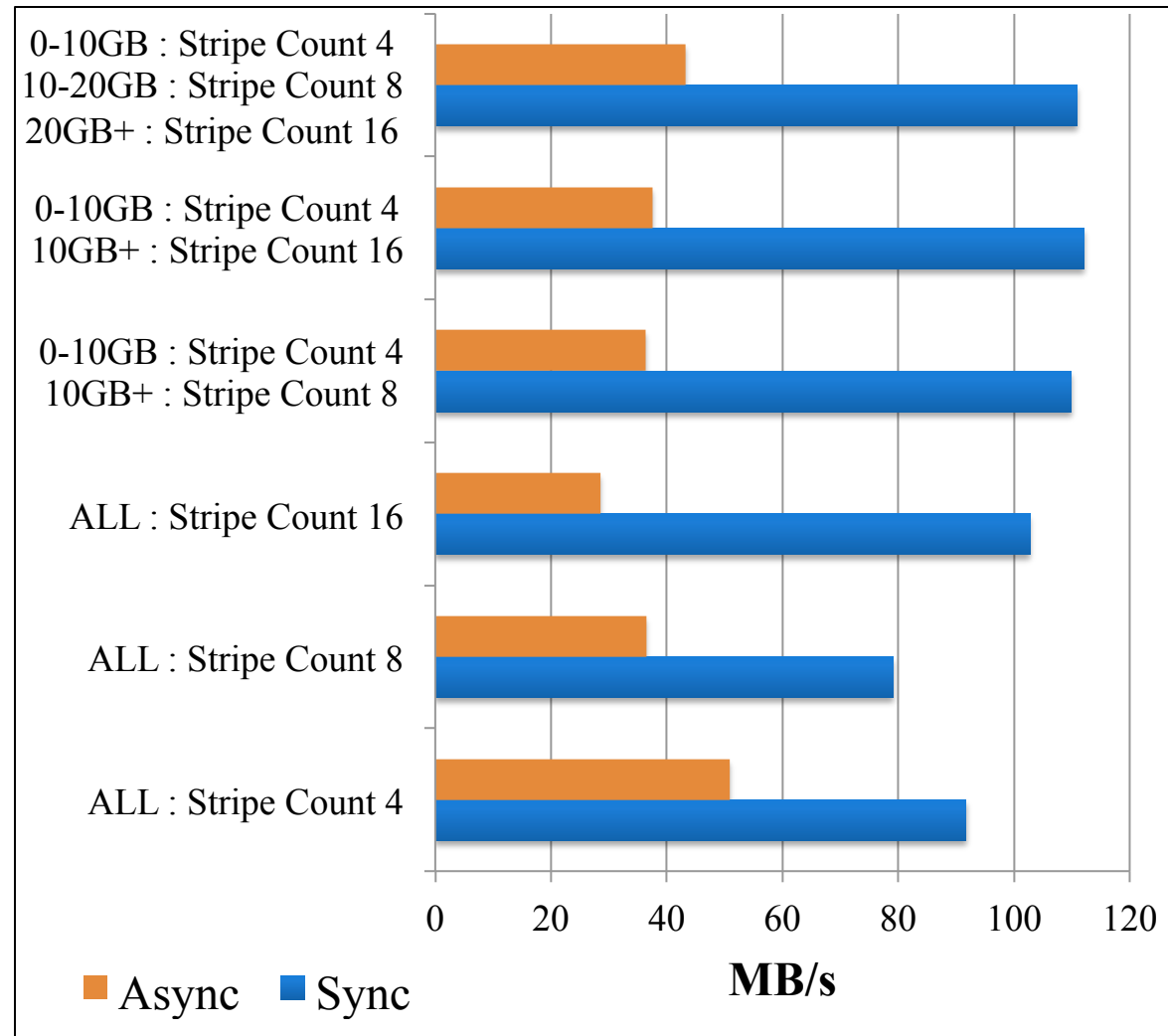
Netflow Phase 1 Results / ZFS Cache Enabled

Test parameters:

- 12 testbed nodes
- 48 tasks (4 per node)
- 56 GB Argus netflow
- 6 stripe count/watermark configurations; executed 5 times each
- Median result presented

Test results:

- Synthetic dynamic striping slightly improves performance
- Less dramatic improvement vs. IOR likely explained by smaller data set and unaligned IO
- Asynchronous performance worse due to issues isolating IO time from compute time – being reworked



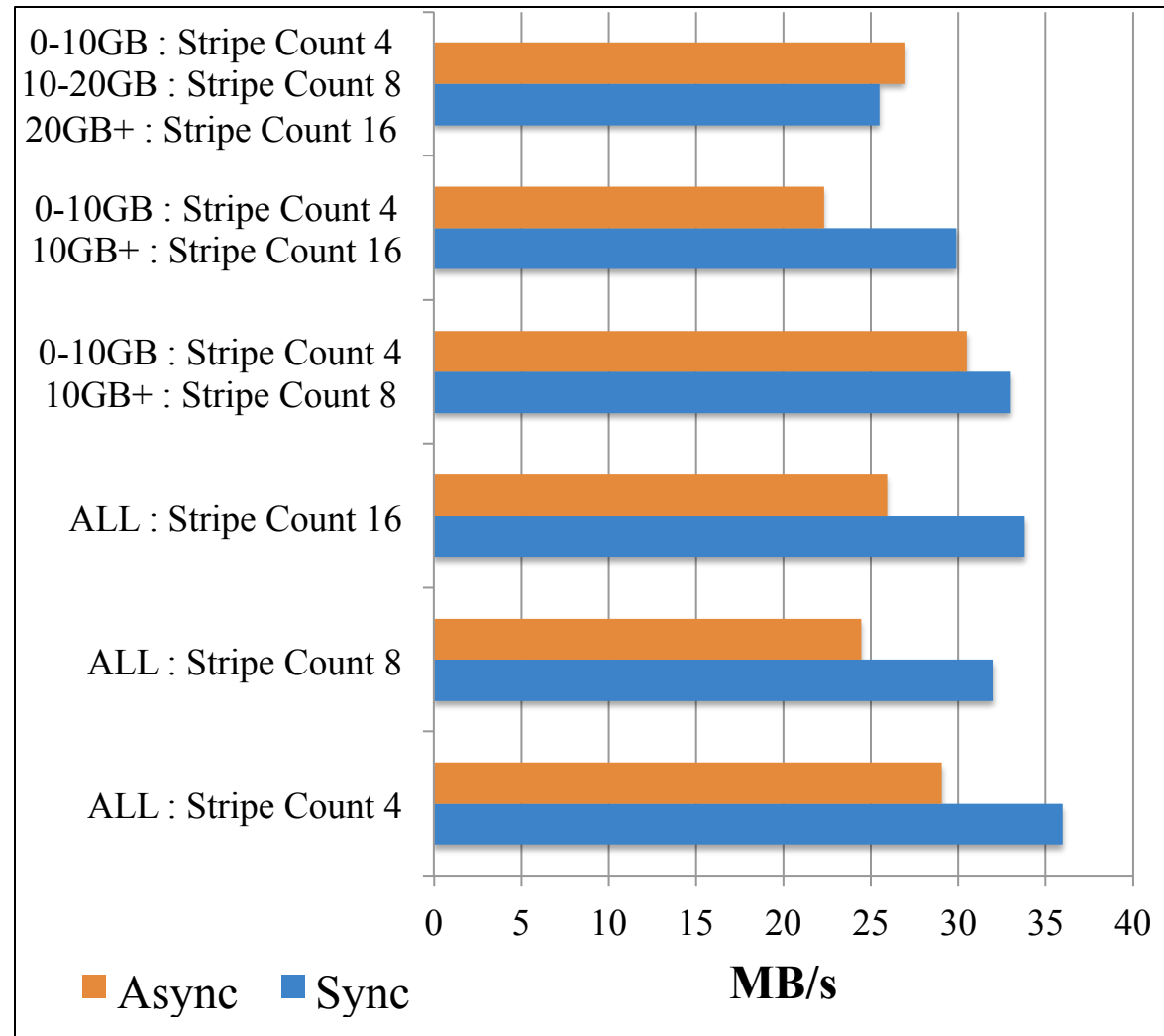
Netflow Phase 2 Results / ZFS Cache Enabled

Test parameters:

- Same as previous

Test results:

- Synthetic dynamic striping slightly degrades performance
- A large number (77 million in this case) of small variable-length reads may not benefit from dynamic striping



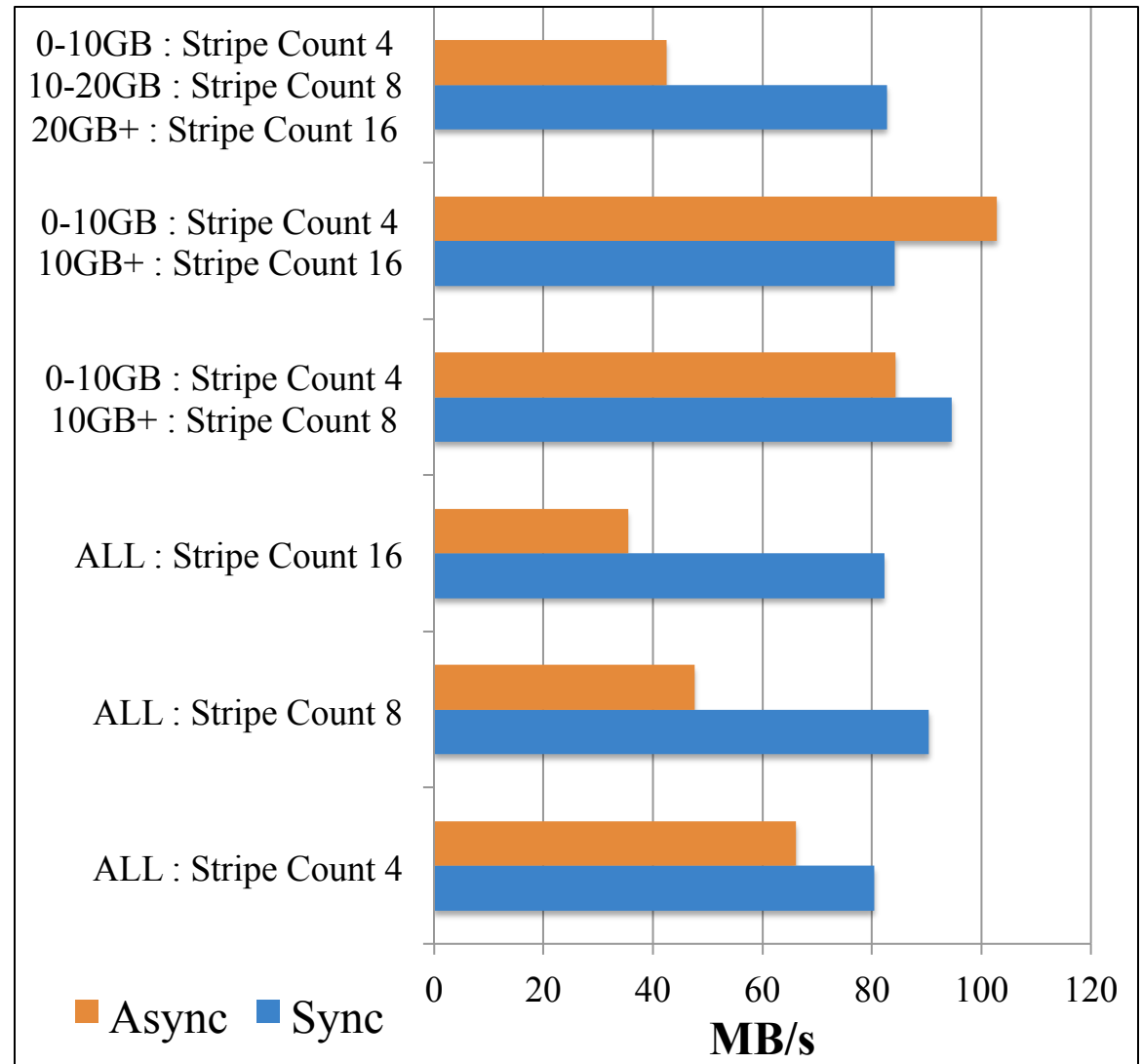
Netflow Phase 1 Results / ZFS Cache Disabled

Test parameters:

- Same as previous, but 3 tests for each stripe count/watermark combination

Test results:

- Synthetic dynamic striping slightly improves performance
- Less dramatic improvement vs. IOR likely explained by smaller data set and unaligned IO
- Asynchronous performance worse due to issues isolating IO time from compute time – being reworked
- ZFS Caching being disabled reduced performance ~15% on average for synchronous implementation and was very similar for asynchronous implementation.



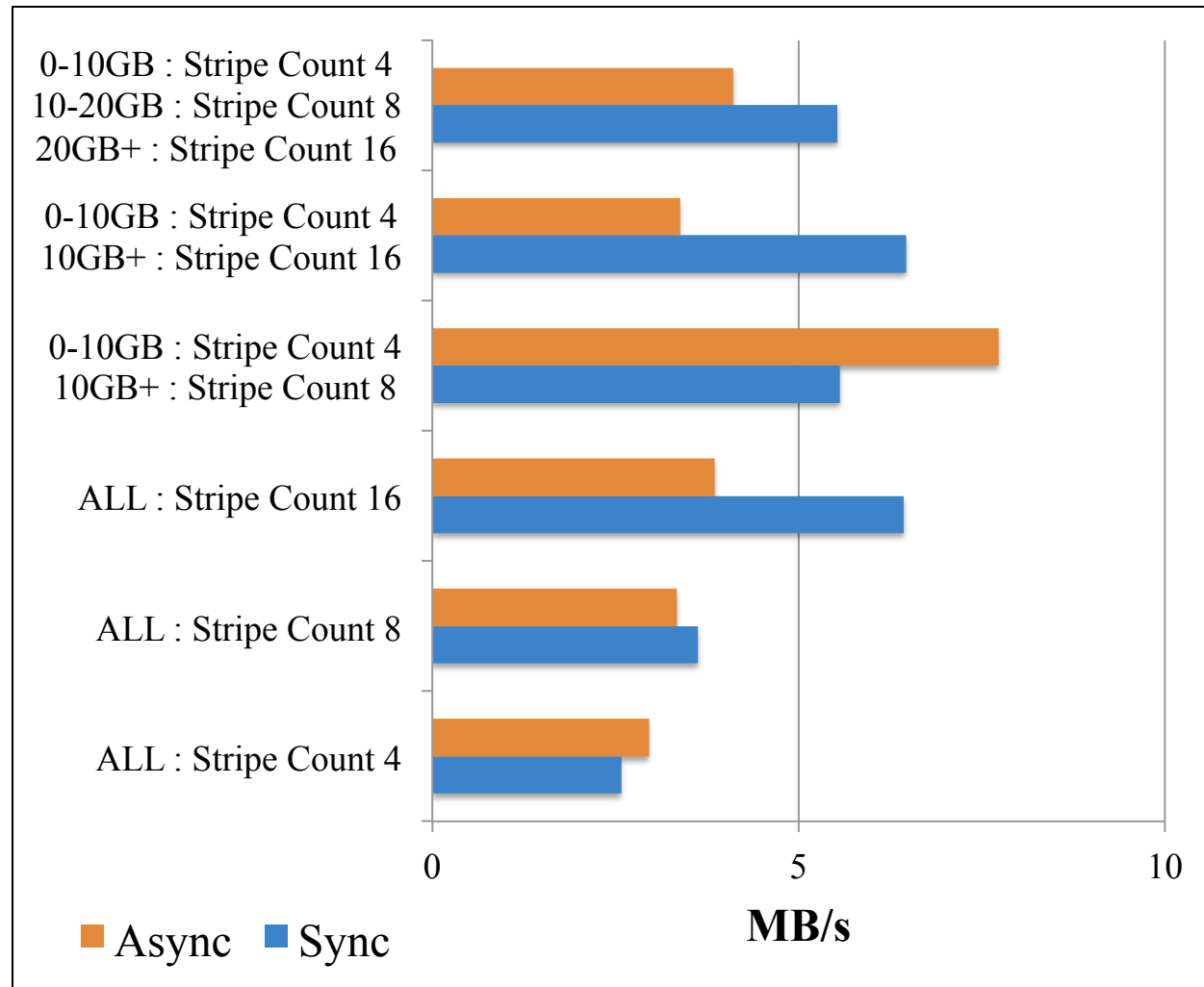
Netflow Phase 2 Results / ZFS Cache Disabled

Test parameters:

- Same as previous

Test results:

- Synthetic dynamic striping slightly degrades performance
- A large number (77 million in this case) of small variable-length reads may not benefit from dynamic striping
- ZFS caching being disabled reduced performance ~26% on average for the synchronous implementation and ~22% for the asynchronous implementation



OLCF Production - Titan

- Cray XK7 supercomputer
 - 18,688 compute nodes, each with
 - 16-core AMD Opteron, 32GB host RAM
 - NVIDIA K20 GPU with 6GB DDR5 device RAM
 - Cray Gemini interconnect (3-D torus)
 - 432 LNET router nodes (XIO)
 - Each handling up to 2880 MB/s of I/O traffic
 - Connected to Spider via FDR Infiniband

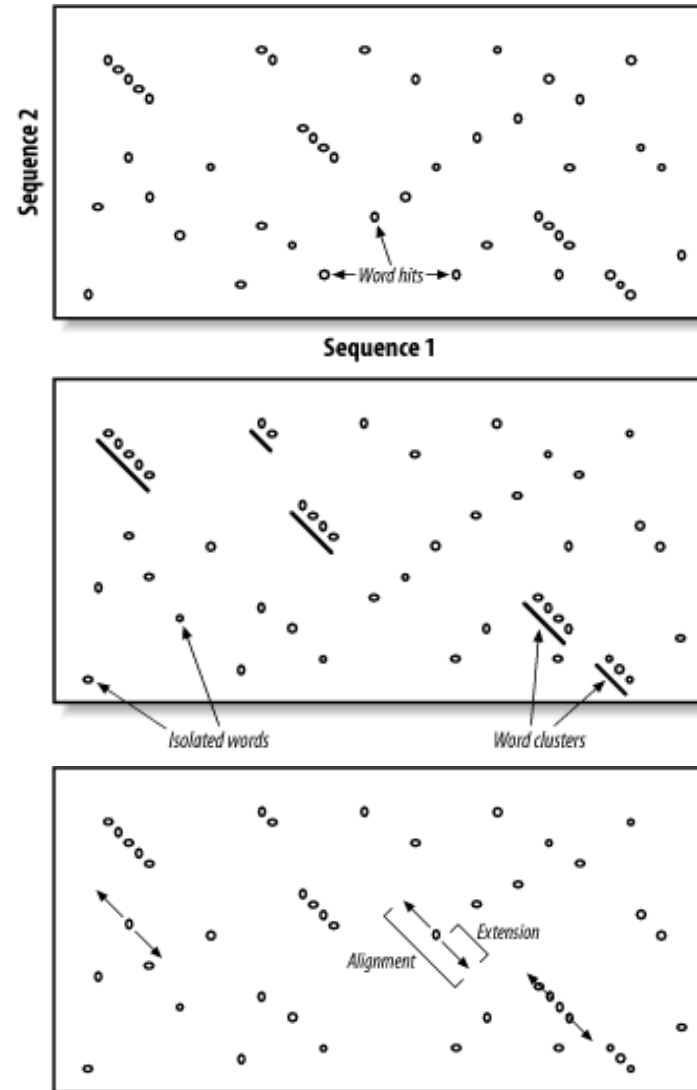
OLCF Production - Spider

- Center-wide Lustre deployment
 - Over 25,000 clients spread across several compute and service resources
 - 32PB and peak I/O rate of over 1 TB/s
- Two filesystems backed by 4 identical I/O clusters
 - each filesystem has:
 - 14 PB usable space
 - 144 OSS, 7 OST per OSS (total of 1008 OST)
 - each I/O cluster has:
 - 72 OSS nodes
 - 9 DDN SFA12K40 couplets
 - 560 2TB near-line SAS disks per couplet

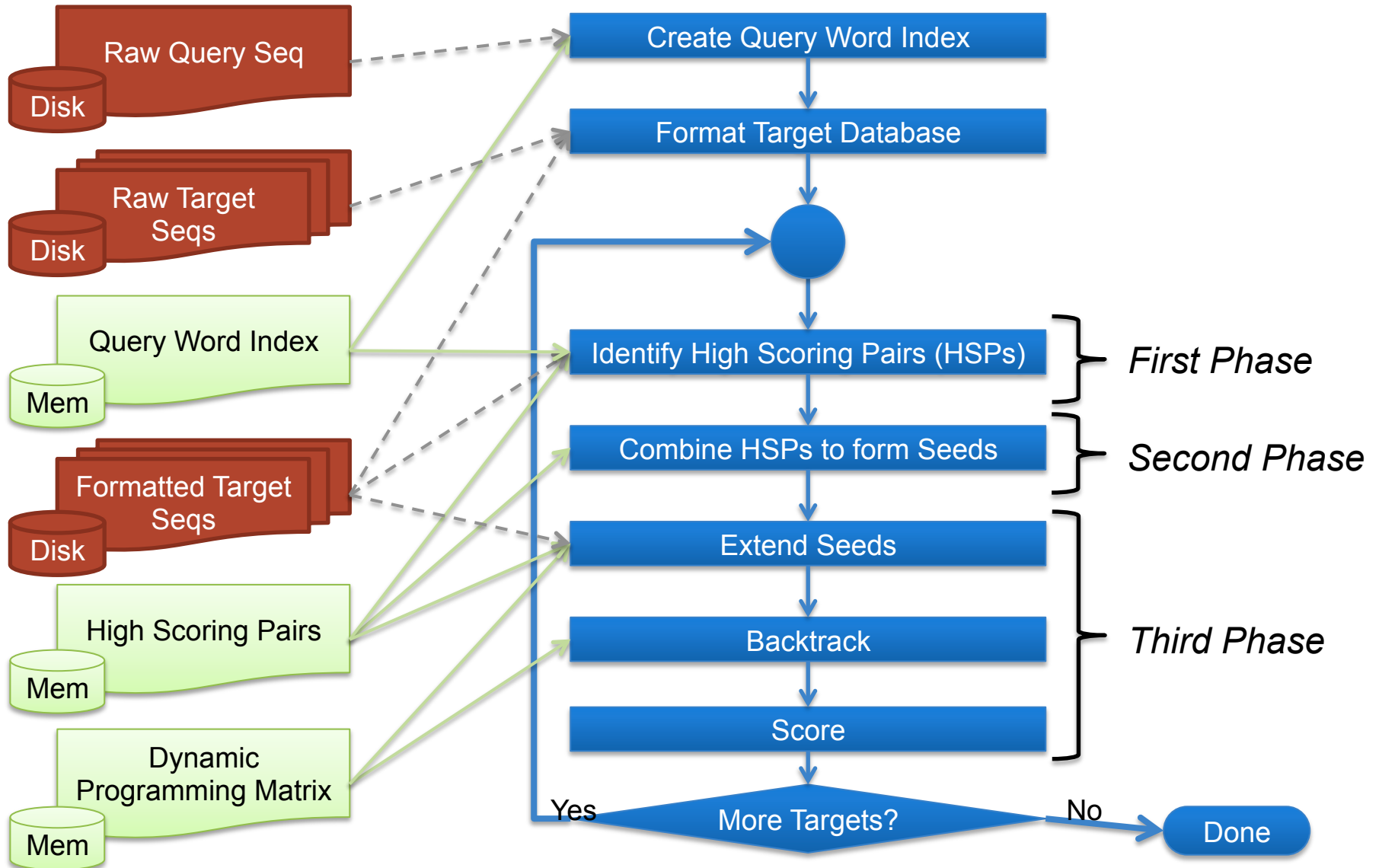
BLAST Sequence Alignment Workload

Three Phases:

1. Identify word hits (seeds)
2. Combine “close” word clusters
3. Extend each match as required



BLAST Algorithm



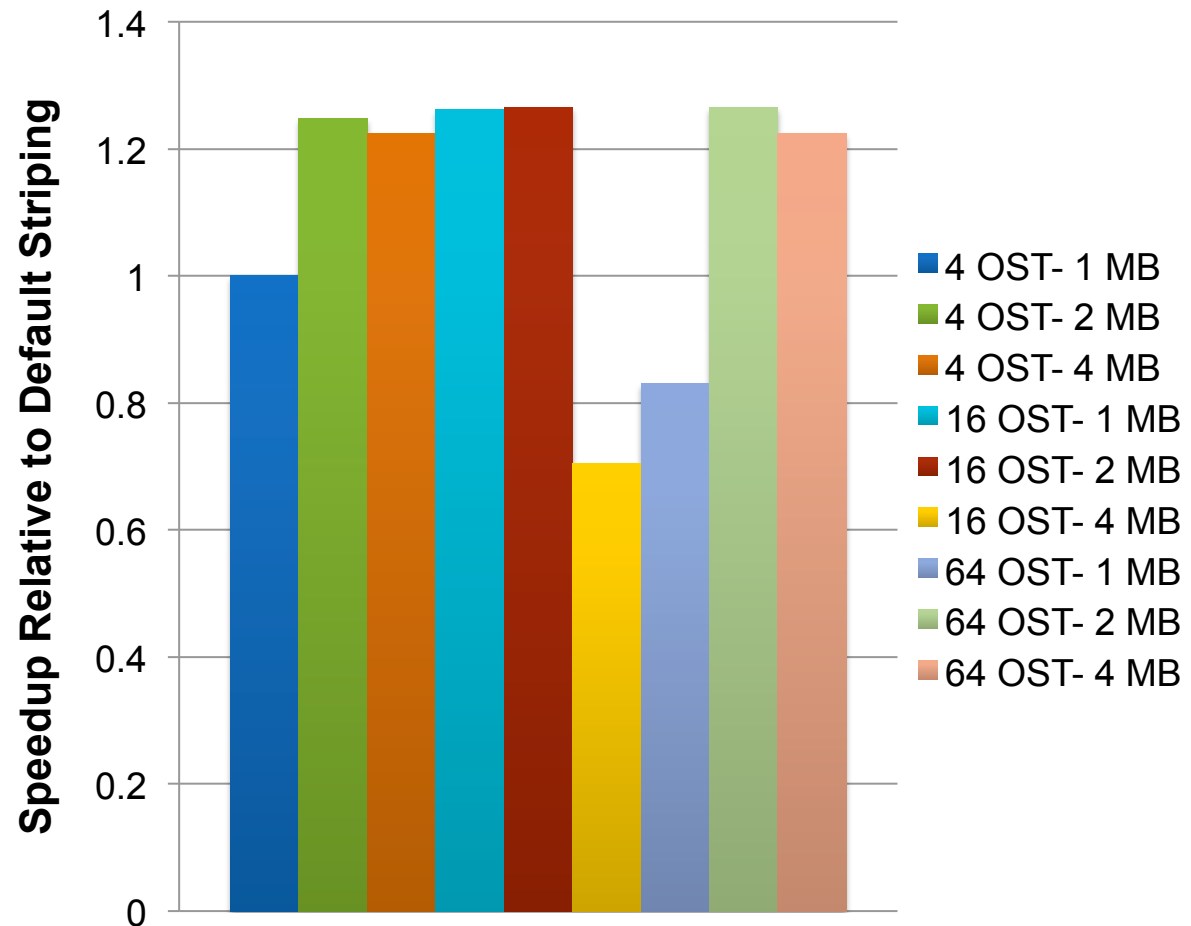
BLAST: Baseline Results w/ Static Striping

Test parameters:

- 8 Titan nodes
- 64 tasks (8 per node)
- 79 GB target database
- 4, 16, 64 OST stripe count
- 1MB, 2MB, 4MB stripe width
- Watermark: 33%-33%-33%
- Each executed three times

Conclusions:

- Default striping pattern of 4 OST with 1 MB stripes is non-optimal
- Variation between static striping configurations is small
- Anomalies at 16OST-4MB & 64OST-1MB likely due to other HPC job executing simultaneously on Titan.



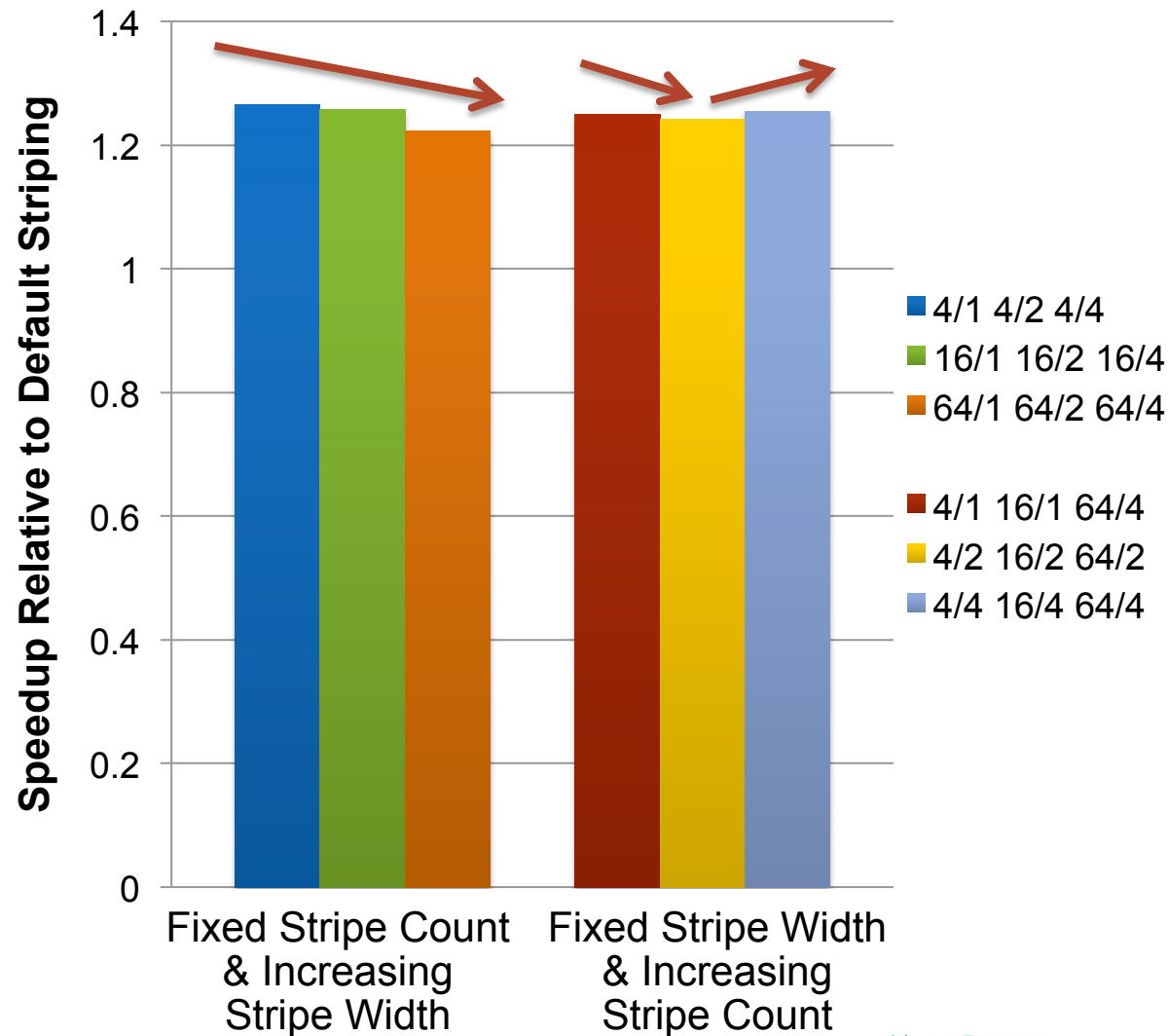
BLAST: Fixing Stripe Count or Stripe Width

Test parameters:

- 8 Titan nodes
- 64 tasks (8 per node)
- 79 GB target database
- 4, 16, 64 OST stripe count
- 1MB, 2MB, 4MB stripe width
- Watermark: 33%-33%-33%
- Each executed three times

Conclusions:

- Fixing Stripe Count and increasing Stripe Width decreases performance
- Fixing Stripe Width and increasing Stripe Count provides inconsistent performance
- Will run additional tests on larger number of nodes to increase contention



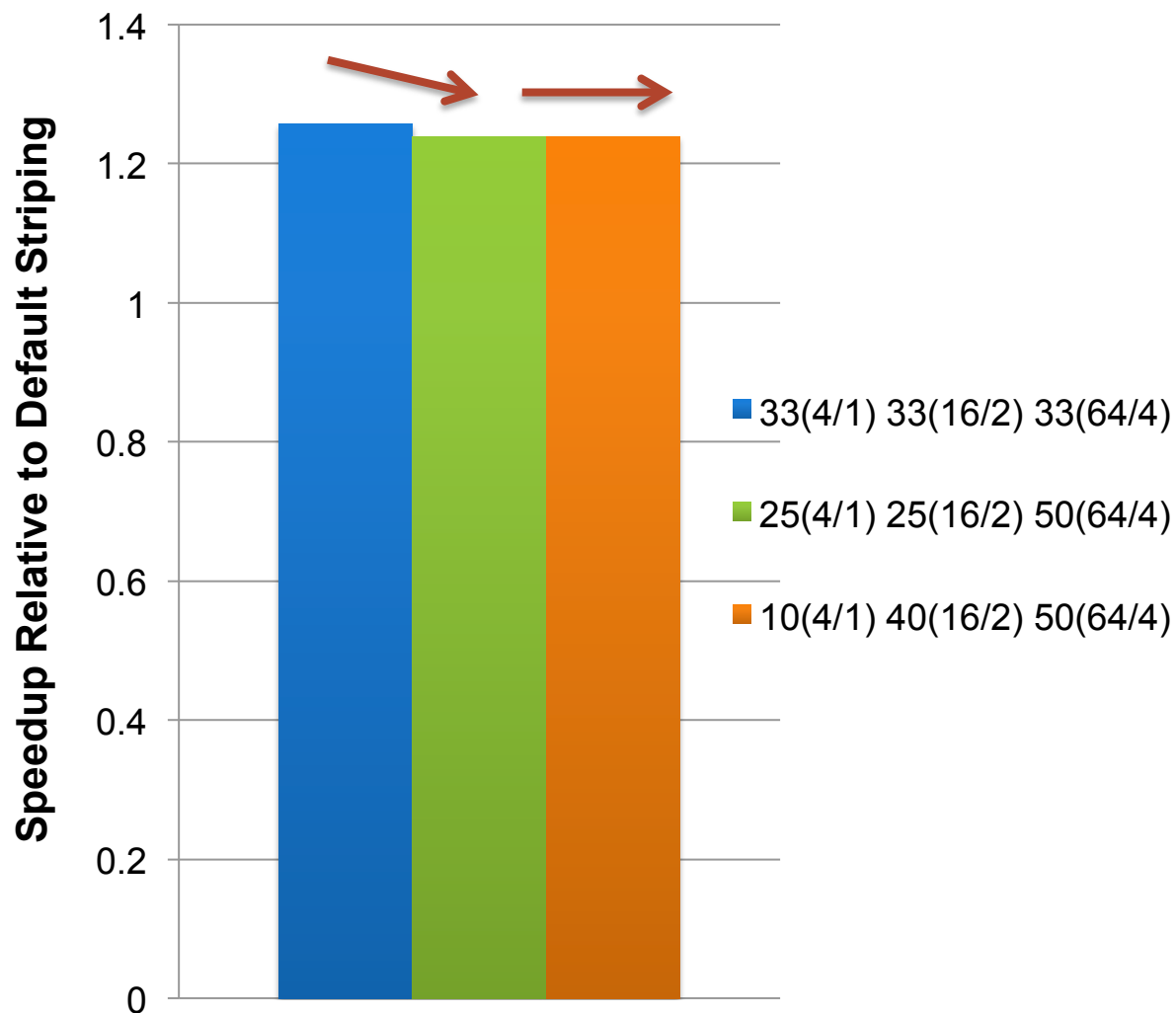
BLAST: Changing Watermark Location

Test parameters:

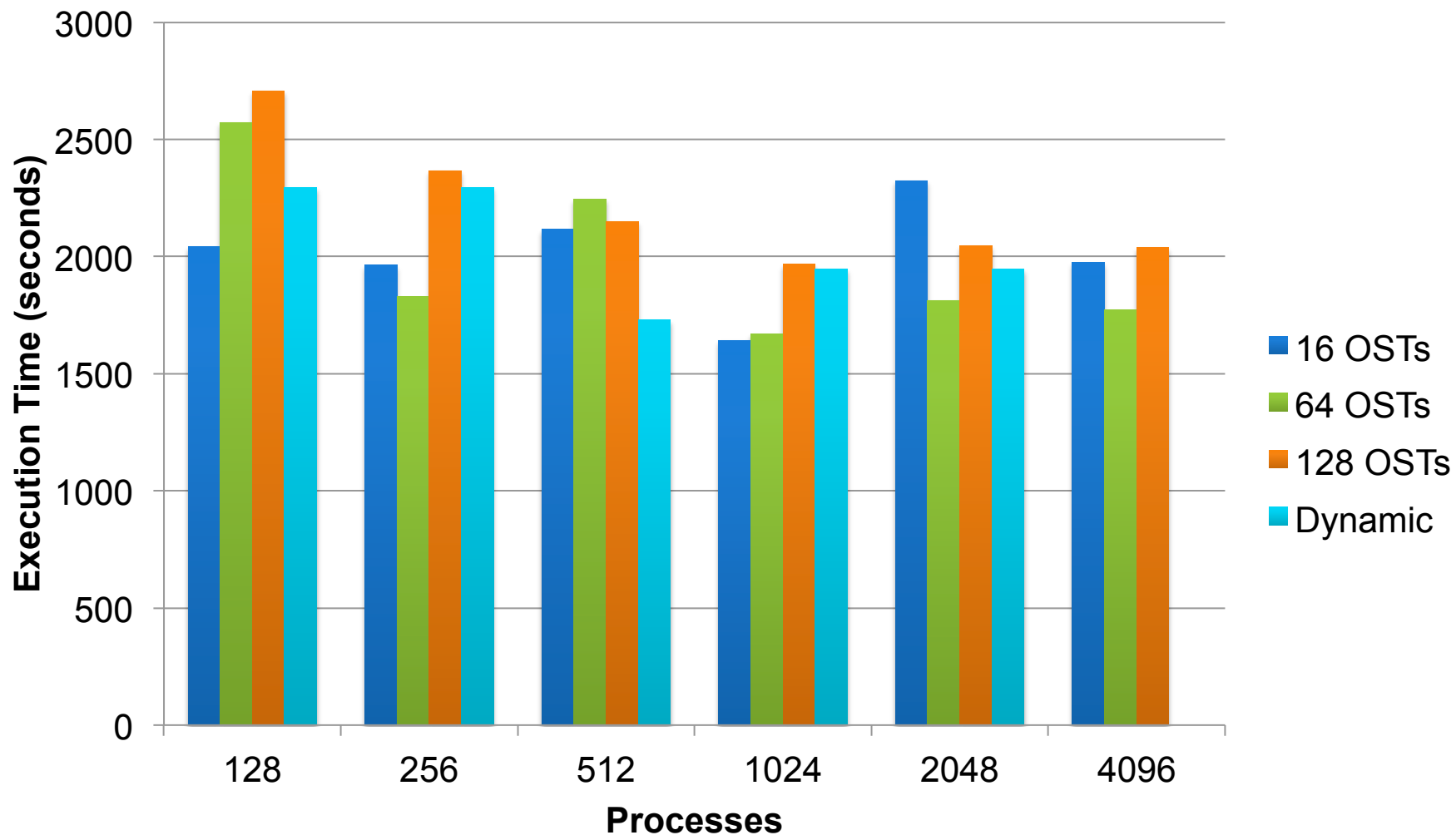
- 8 Titan nodes
- 64 tasks (8 per node)
- 79 GB target database
- 4, 16, 64 OST stripe count
- 1MB, 2MB, 4MB stripe width
- Watermarks
 - 33% - 33% - 33%
 - 25% - 25% - 50%
 - 10% - 40% - 50%
- Each executed three times

Conclusions:

- Inconsistent trend when changing watermark location with 2 watermarks
- Will run additional tests to evaluate changing 1 watermark location
- Will run additional tests on larger number of nodes to increase contention



Larger Blast Benchmarks



Conclusions

- Dynamic striping will likely benefit sequential read performance
- Dynamic striping will likely harm performance only on large numbers of very small random reads
- Much larger tests need to be done to definitively characterize the performance of dynamic striping, but given other benefits like fault tolerance and easy expansion, implementation should proceed.

Thank You.

Questions?