BLUE WATERS

SUSTAINED PETASCALE COMPUTING

# A Classification of Parallel I/O Toward Demystifying HPC I/O Best Practices

Rob Sisneros


GREAT LAKES CONSORTIUM
FOR PETASCALE COMPUTATION

CRAY

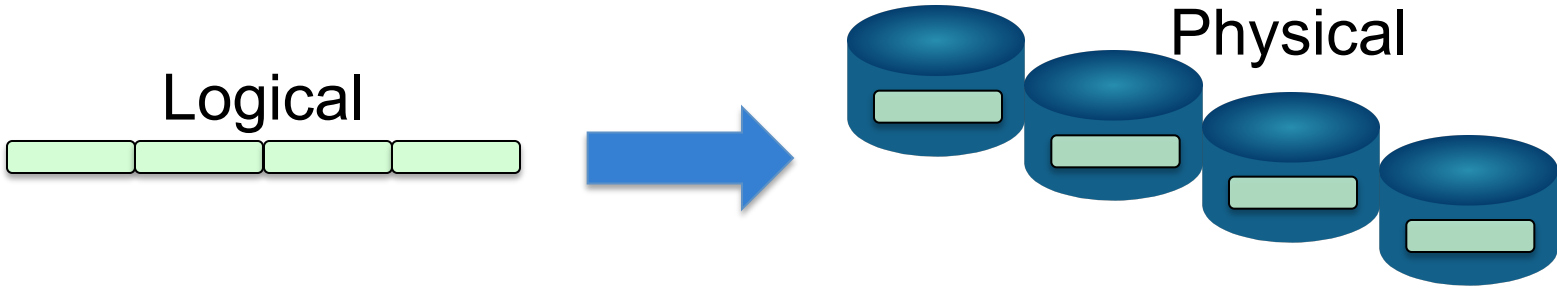
# Summary

- Communicating parallel I/O
  - Training
  - Performance
  - Best practices
- The problem
- The work
  - Shuffling assumptions
  - A solution

Some slides I gave to new Blue Waters users

# THE INTRODUCTION TO PARALLEL I/O

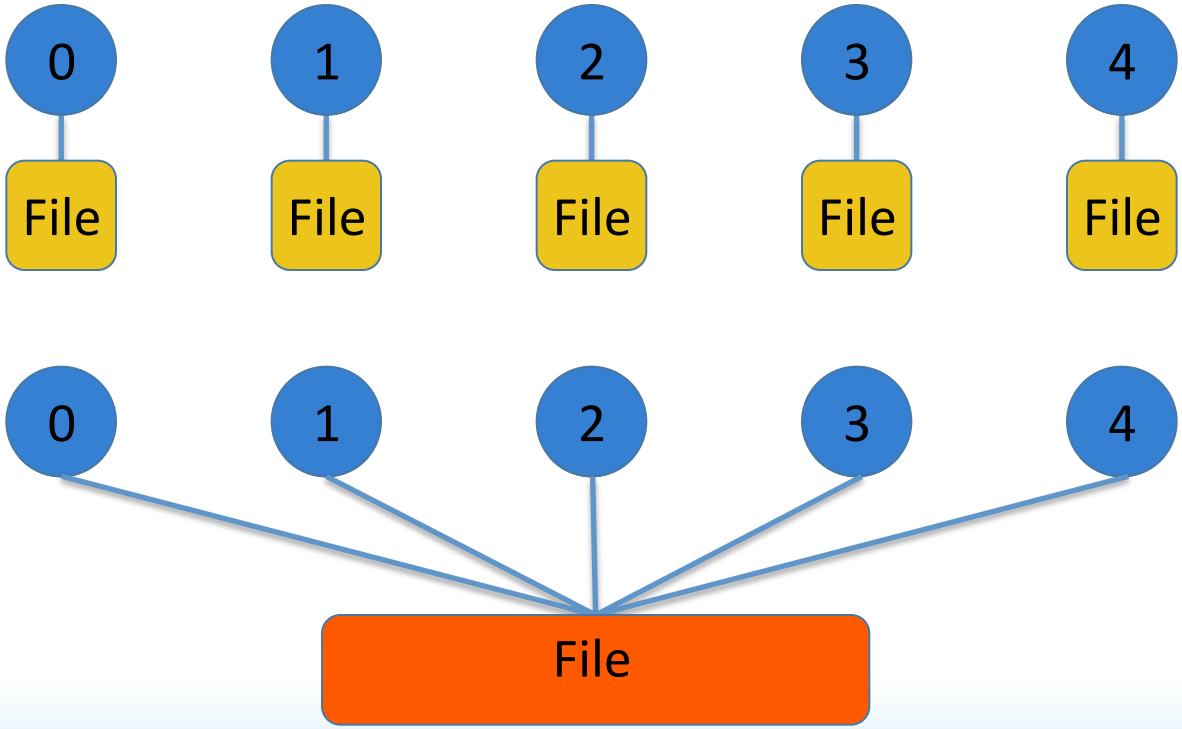
# Lustre File System: Striping (Intro 1)




- **File striping:** single files are distributed across a series of OSTs
  - File size can grow to the aggregate size of available OSTs (rather than a single disk)
  - Accessing multiple OSTs concurrently increases I/O bandwidth

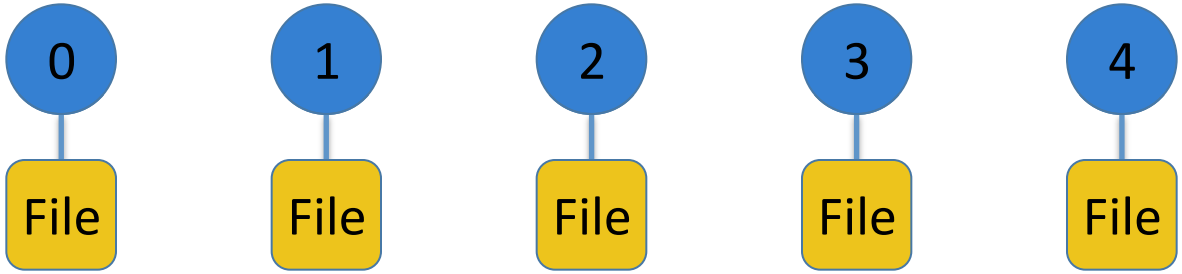
# Large Scale I/O in Practice (Intro 2)

- Serial I/O is limited by both the I/O bandwidth of a single process as well as that of a single OST
- Two ways to increase bandwidth:



# File-Per-Process (Intro 3)

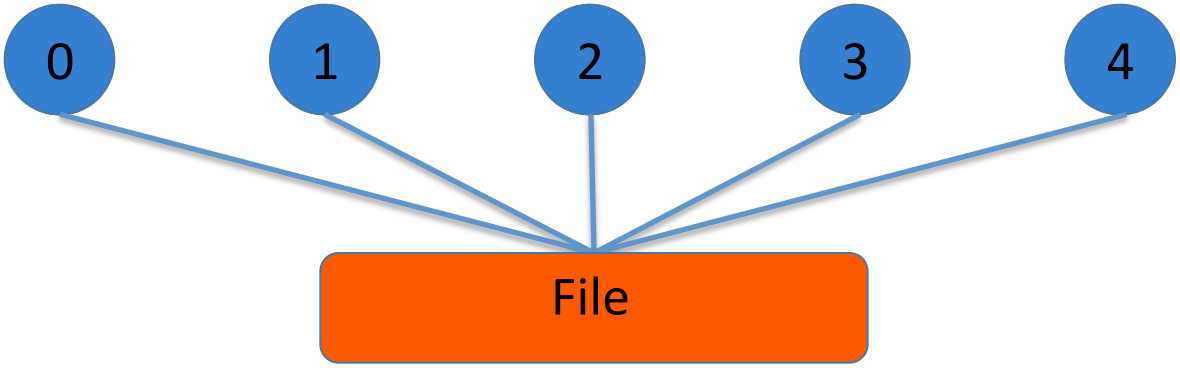
- Each process performs I/O on its own file




- Advantages
  - Straightforward implementation
  - Typically leads to reasonable bandwidth quickly
- Disadvantages
  - Limited by single process
  - Difficulty in managing a large number of files
  - Likely requires post processing to acquire useful data
  - Can be taxing on the file system metadata and ruin everybody's day

# Shared-File (Intro 4)

- There is one, large file shared among all processors which access the file concurrently




- Advantages
  - Results in easily managed data that is useful with minimal preprocessing
- Disadvantages
  - Likely slower than file-per-process, if not used properly
  - Additional (one-time!) programing investment

# Performance Impact: Configuring File Striping (Intro 5)

- `lfs` is the Lustre utility for viewing/setting file striping info
  - **Stripe count** – the number of OSTs across which the file can be striped
  - **Stripe size** – the size of the blocks that a file will be broken into
  - **Stripe offset** – the ID of an OST for Lustre to start with, when deciding which OSTs a file will be striped across
  - A pool of OSTs to stripe across may also be specified – unlikely to be useful as you can not create a pool
- Configurations should focus on stripe count/size
- Blue Waters defaults:

```
$> touch test
$> lfs getstripe test
test
lmm_stripe_count:   1
lmm_stripe_size:    1048576
lmm_stripe_offset:  708
   obdidx           objid           objid           group
      708         2161316       0x20faa4               0
```

# Setting Striping Patterns (Intro 6)

```
$> lfs setstripe -c 5 -s 32m test
$> lfs getstripe test
test
lmm_stripe_count:   5
lmm_stripe_size:    33554432
lmm_stripe_offset:  1259
   obdidx           objid           objid           group
     1259         2162557       0x20ff7d               0
     1403         2165796       0x210c24               0
      955         2163063       0x210177               0
     1139         2161496       0x20fb58               0
      699         2161171       0x20fa13               0
```

- Note: a file's striping pattern is permanent, and set upon creation
  - `lfs setstripe` creates a new, 0 byte file
  - The striping pattern *can* be changed for a directory; every *new* file or directory created within will inherit its striping pattern
  - Simple API available for configuring striping – portable to other Lustre systems

There is no winner

# THE HIGH LEVEL ADVICE

# And the Winner is… Neither? (Intro 7)

- Both patterns increase bandwidth through the addition of I/O processes
  - There is a limited number of OSTs to stripe a file across
  - The likelihood of OST contention grows with the ratio of I/O processes to OSTs
  - Eventually, the benefit of another I/O process is offset by added OST traffic
- Both routinely use all processes to perform I/O
  - A small subset of a node's cores can consume a node's I/O bandwidth
  - This is an inefficient use of resources
- The answer? It depends… but,
  - Think aggregation, a la *file-per-node*

# MORE ADVICE: BEST PRACTICES

- Use large data transfers and buffer when possible
- Consider using MPI-IO and other I/O libraries
  - Portable data formats vs. unformatted files
- Limit the number of files in a single directory
- Use system specific hints and optimizations
  - Avoid misaligned operations
- Exploit parallelism using striping
  - Focus on stripe alignment, avoiding OST contention
- Don’t default to file-per-process model
  - Use aggregation and reduce number of output files

Following best practices

# THE WARNING

# Variation of the Following

- From “Application Scalability and Parallel I/O” presentation by William Gropp
  - No easy recipe
  - Performance can be lost anywhere
  - Rules of thumb can be misleading
  - Specifics depend on the application

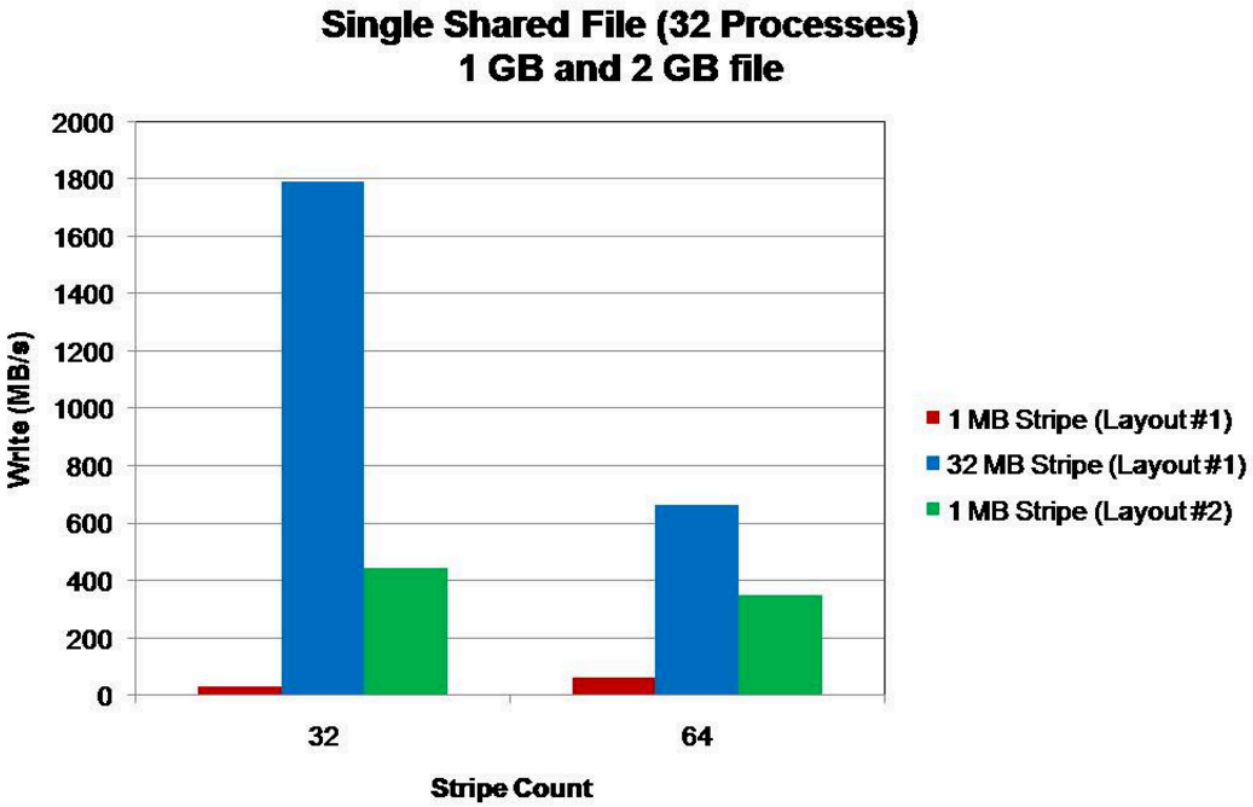
Backing up our claims

# BENCHMARKING I/O

# IOR

- Benchmark performance of various libraries for shared file I/O and file-per-process I/O
- Processors write data “blocks” in series of “transfers”
- These things are tuned along with different Lustre stripe settings to display performance results

# Example Result (From NICS)



# How to Determine Stripe Settings?

- Subtle, and completely left out of my “intro” talk!
- From NERSC:

**Striping Shortcut Commands**

| | Single Shared-File I/O | File per Processor |
|---|---|---|
| Description | Either one processor does all the I/O for a simulation in serial or multiple processors write to a single shared file as with MPI-IO and parallel HDF5 or NetCDF | Each processor writes to its own file resulting in as many files as number of processors used (for a given output) |
| **Size of File** | **Command** | |
| < 1GB | Do Nothing. Use default striping. | keep default striping |
| ~1GB - ~10GB | stripe_small | keep default striping |
| ~10GB - ~100GB | stripe_medium | keep default striping |
| ~100GB - 1TB+ | stripe_large | Ask consultants |

# PROBLEMS

# Best Practices, In Practice

- Really two categories
  - Experience based, made vague to give *idea* of how to improve performance
  - General to system, provided from perspective of fully loaded machine, not necessarily from single user's experience
- Benchmarking rarely provides clear evidence of either

# The Result

- How many applications are exceptions to the rules? Probably all of them.
- We hand tune applications to adhere to the best practices we all agree on
  - Ensuring I/O is stripe aligned
  - Avoiding OST contention

# THIS WORK

# The Idea

- Our Efforts as Creators of “Best Practices”
  - Too focused on aspects that we are unable to usefully generalize
  - Too reliant on deep knowledge of I/O intricacies
  - Size is underrepresented
- Let’s forget explaining stripe settings and start the generalization at the application level
  - Categorize an application entirely by size of I/O
  - Generalize I/O Models

# The Hope

- Filling in the gaps in current I/O models (between file-per-process and shared file) will
  - Create valuable performance data
  - Allow us to categorize I/O models in an understandable context
- We can provide understandable evidence that backs up our best practices

# Gaps in I/O Models

- Likely exist for good reason: maybe “file per 2 cores per 3 nodes” doesn’t make sense
- Complicated to generate and benchmark
- IOR is not usable in this case, need custom benchmark code

# The Custom Code

- Input arguments: I/O size (to match with an application's write phase), maximum nodes and processors per node to use
- Called with single aprun with maximum nodes/ppn
- Iterates through non-crazy I/O patterns keeping write size consistent

# Examples of Non-Crazy Patterns

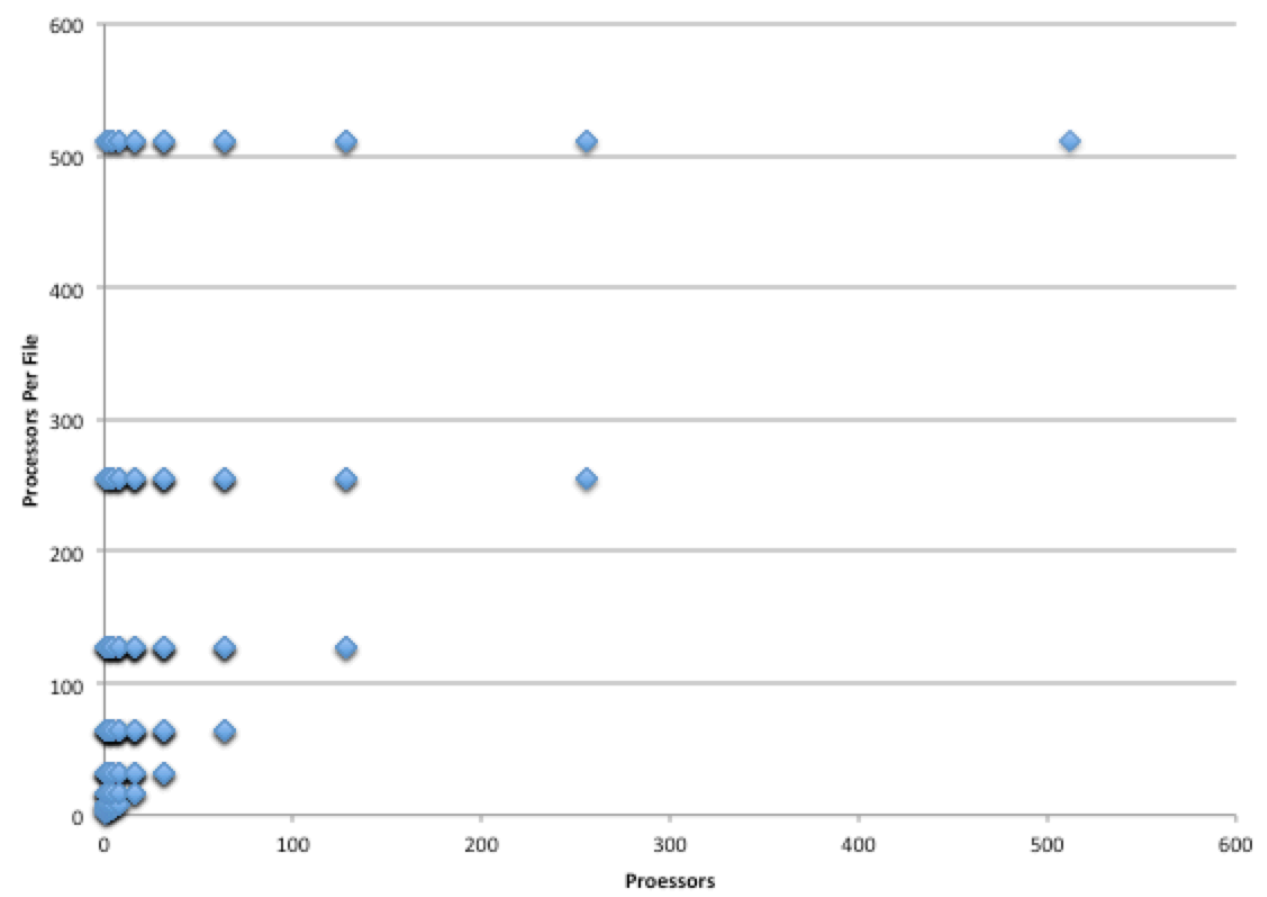
- File per process
- Single shared file
- File per node
- Combinations of *f* files per node shared across *m* nodes
- Measures only write time

# Two Tests

- Both conducted on Blue Waters test and development system: JYC on 32 nodes, 16 ppn
- I/O size 1: 33554432 (32 MB)
- I/O size 2: 34359738368 (32 GB)

- Number of tests in each case: 315!

# The Classification: Processors vs. Processors per File



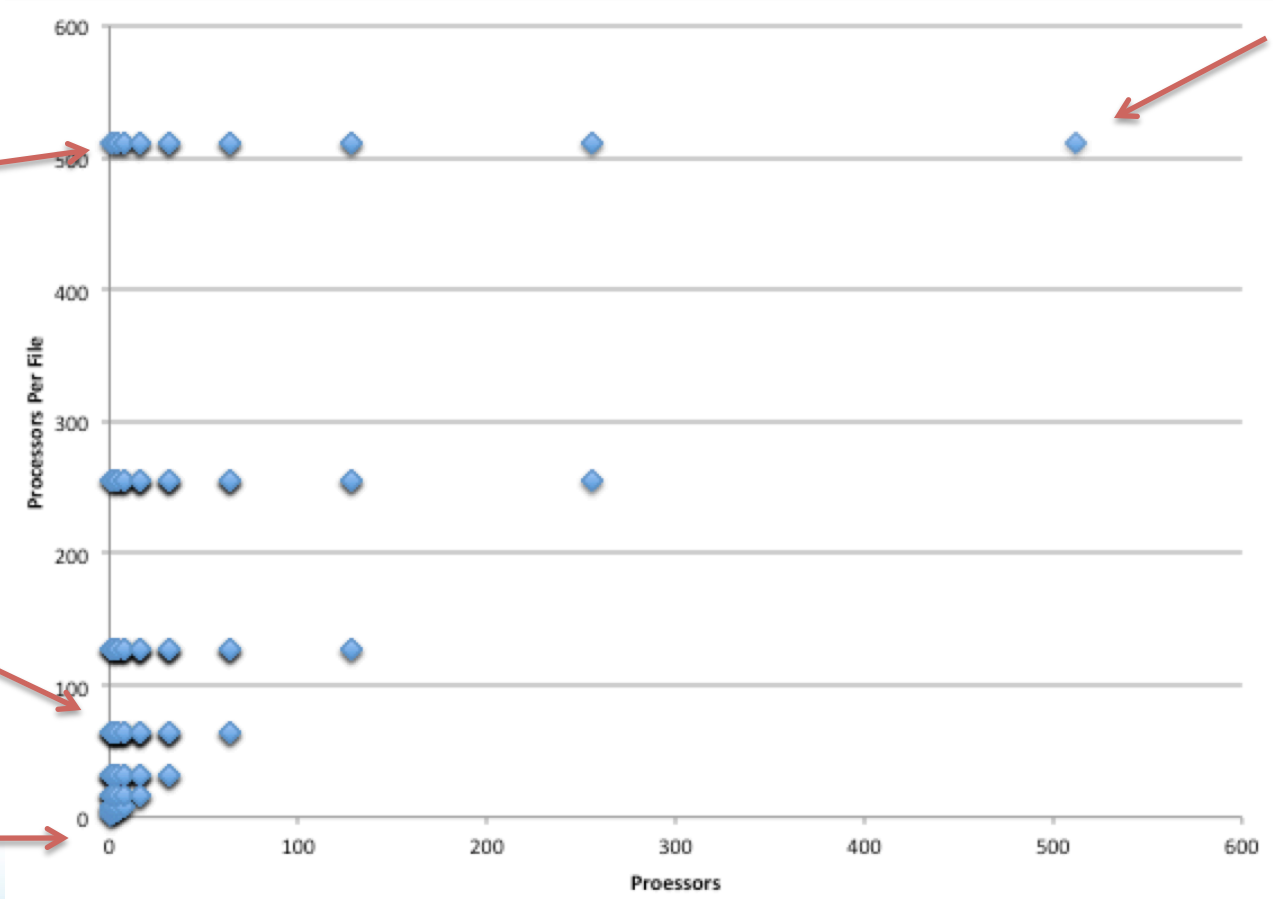
# Common Patterns

File per process

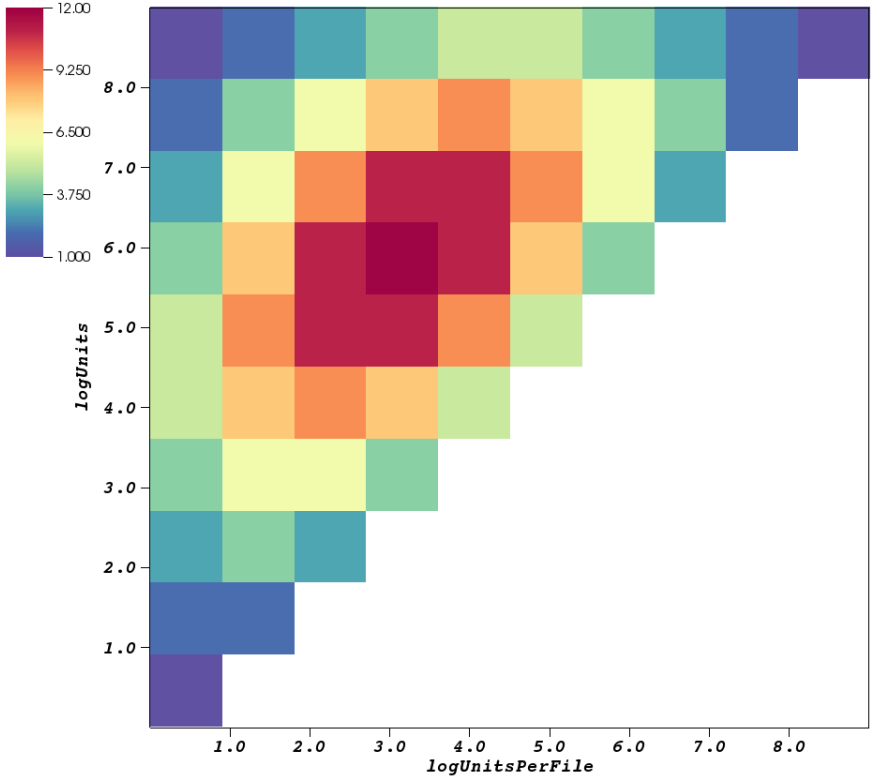
Single shared file

File per Node

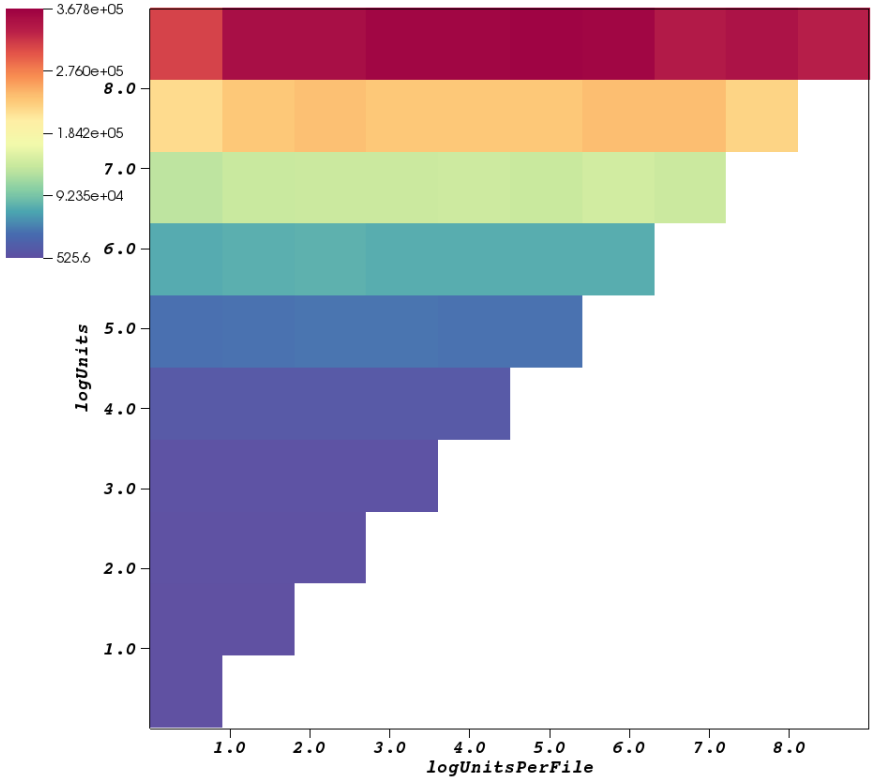
Serial



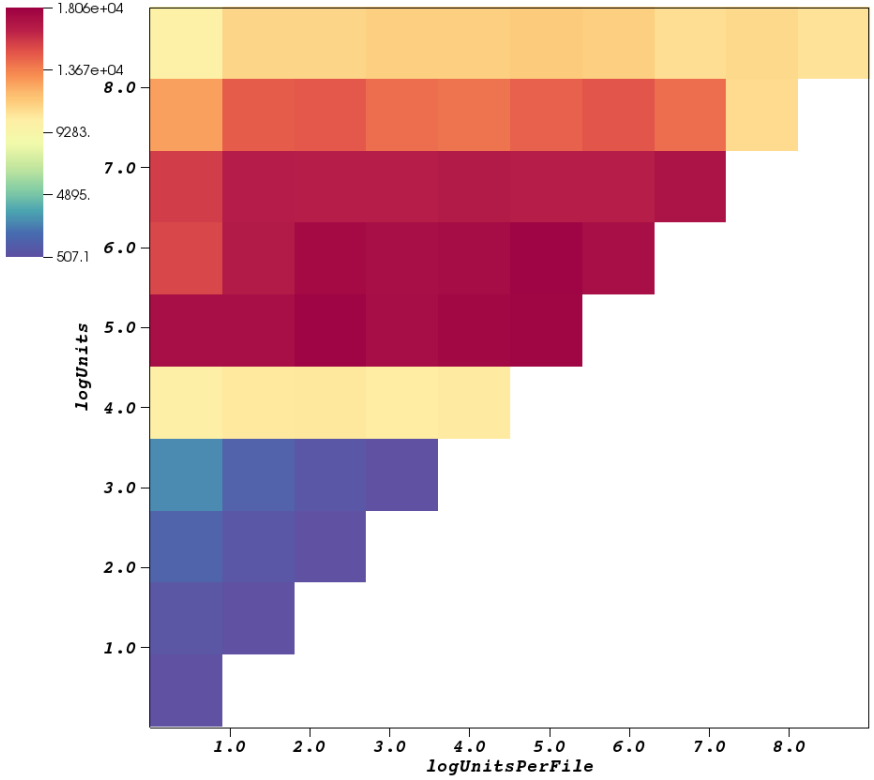
# Log It, 2D Histogram It: Distribution of 315 Tests



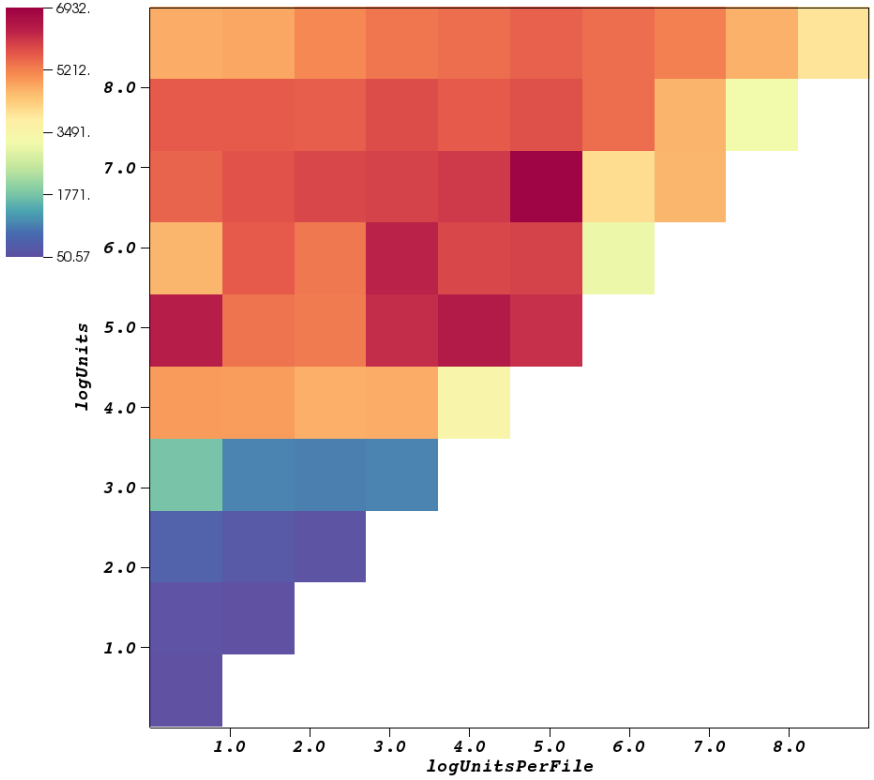
# 32MB Throughput



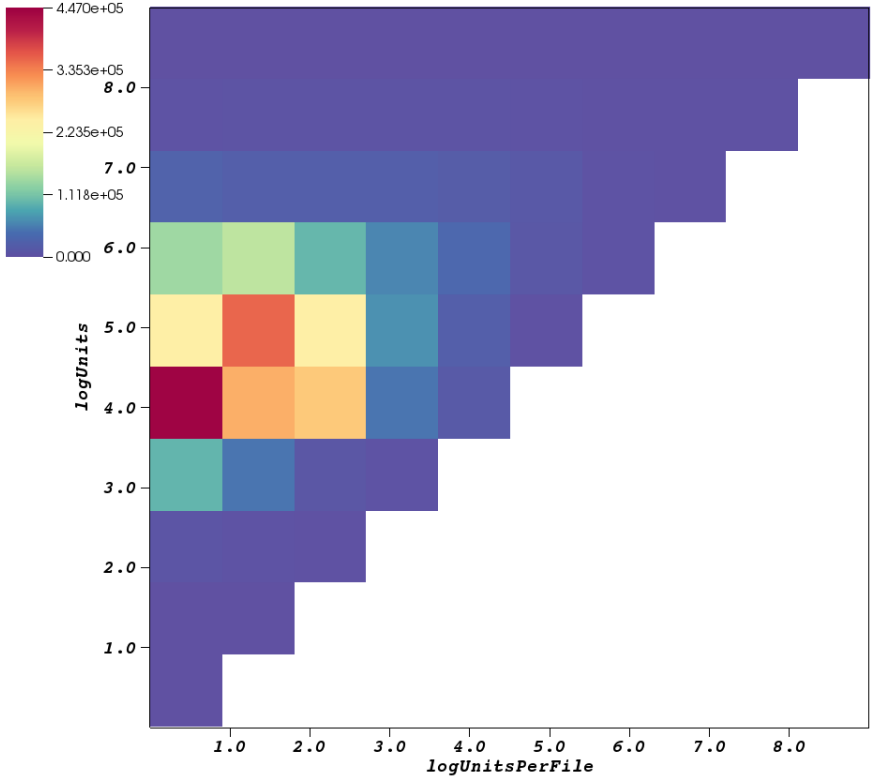
# 32MB Efficiency: Throughput per Node Hour



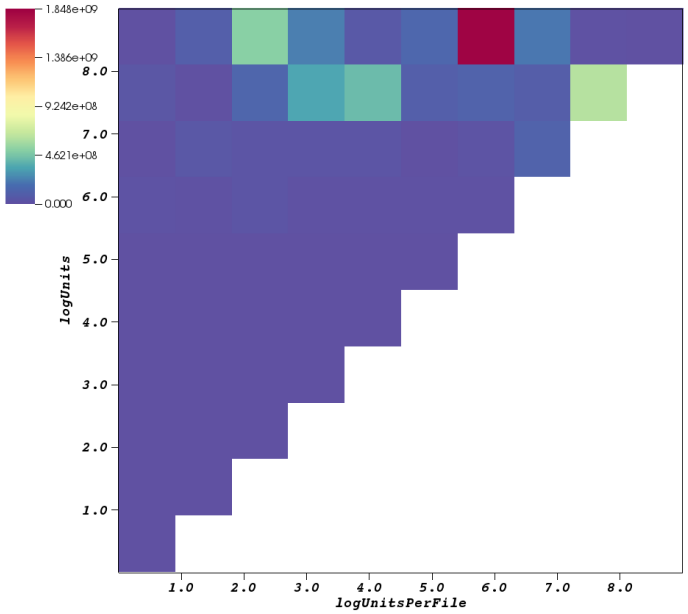
# 32GB Throughput

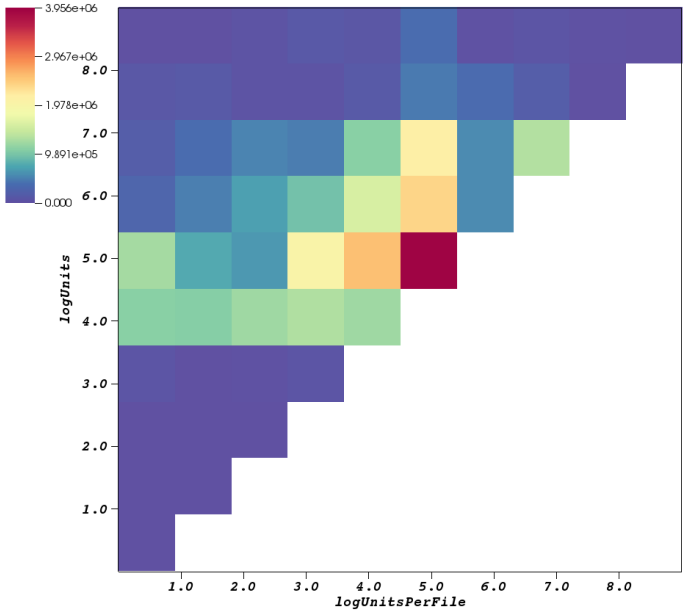


# 32GB Efficiency: Throughput per Node Hour



# Bonus Mysteries: Standard Deviation



# Wrapping it Up

- A context where I/O models are compared directly against each other for a specific application
  - Valuable for an application
  - Backs up best practices
- Provides upper bound on expected benefit to put the effort into creating I/O aggregation
- Models can be run machine wide to provide additional measurement of increased OST contention
- Stored typical benchmarking (altering sizes) results can easily be incorporated – maybe similar sizes are good enough?