CSE 5449: Intermediate Studies in Scientific Data Management

Lecture 15: Automatically tuning parallel I/O performance

Dr. Suren Byna

The Ohio State University

E-mail: byna.1@osu.edu

https://sbyna.github.io

03/02/2023

Today’s class

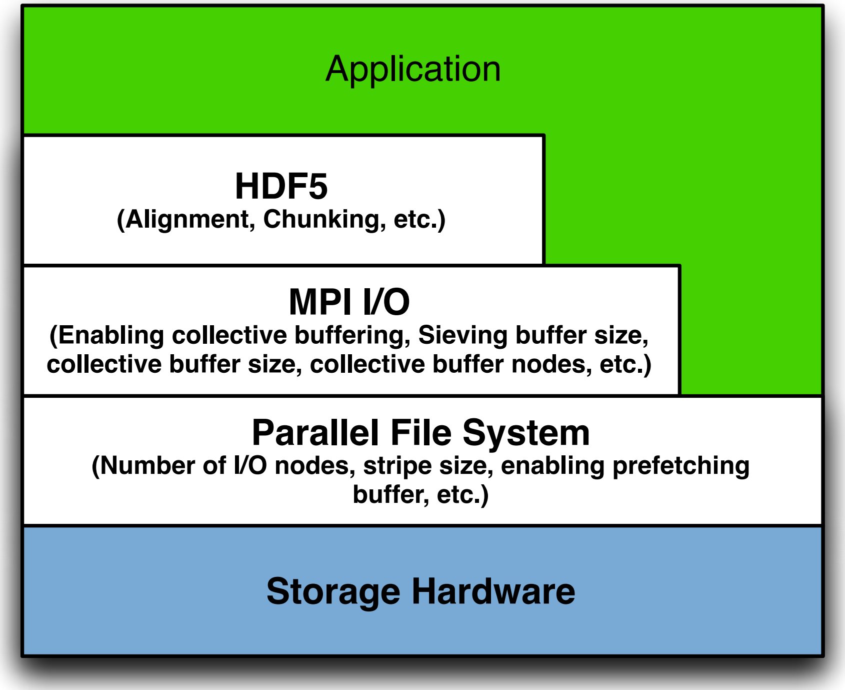
- Any questions?

- Progress of the project

- Today’s class –
  - How to tune I/O performance automatically?

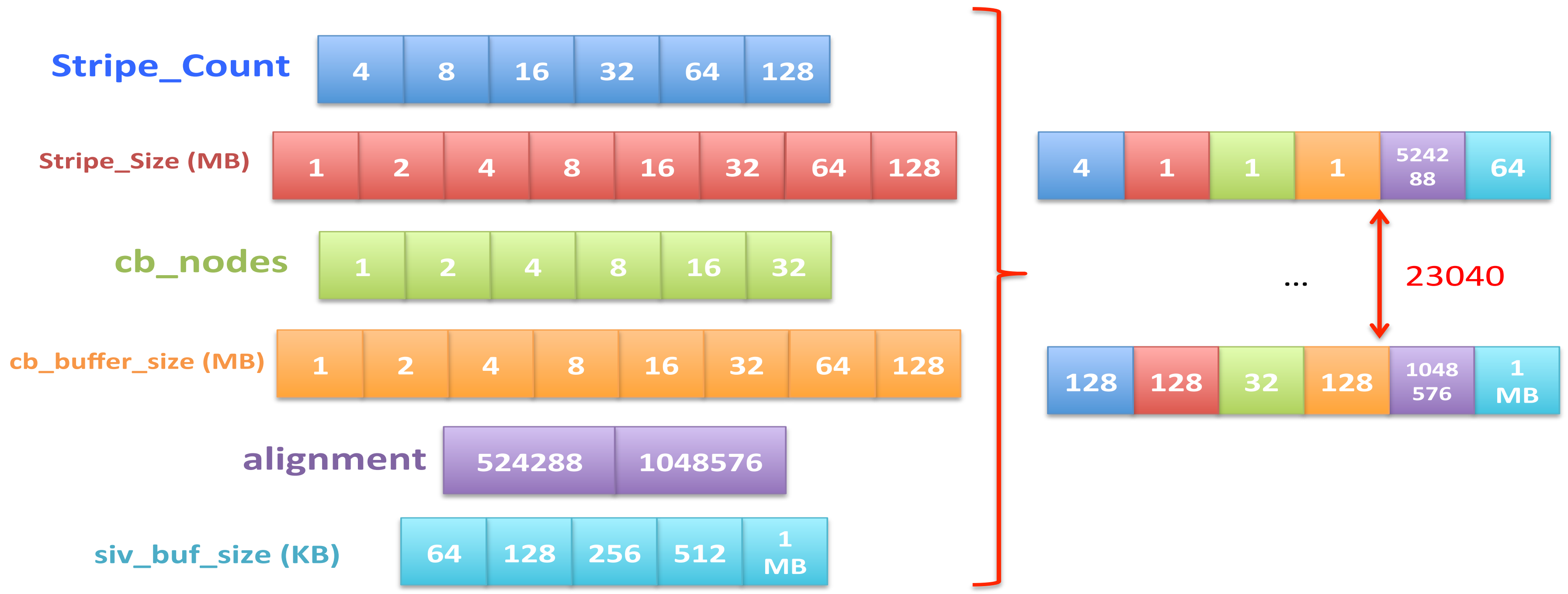
Complexity of Parallel I/O Sub-system

- **Parallel I/O software stack**
  - Application
  - High-level I/O libraries and data models
  - I/O middleware
  - Parallel file system
- **Options for performance optimization**
- **Complex inter-dependencies among layers**


Application
HDF5
(Alignment, Chunking, etc.)
MPI I/O
(Enabling collective buffering, Sieving buffer size, collective buffer size, collective buffer nodes, etc.)
Parallel File System
(Number of I/O nodes, stripe size, enabling prefetching buffer, etc.)
Storage Hardware

Tuning parameter space

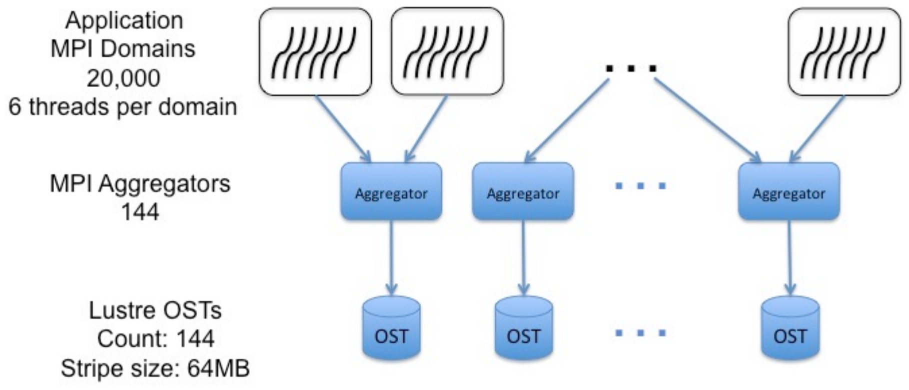
The whole space visualized

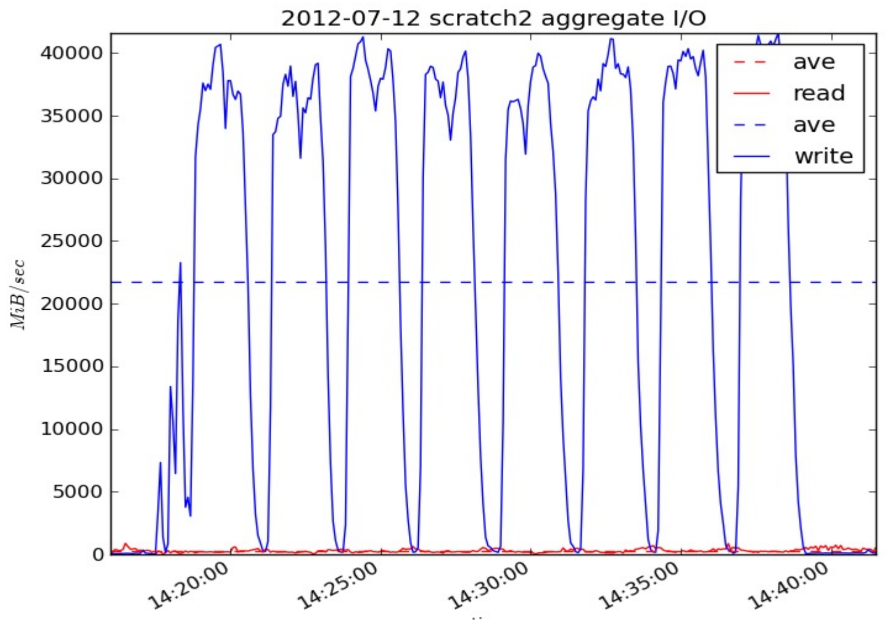


Manual tuning for writing trillion particle datasets

- Simulation of magnetic reconnection (a space weather phenomenon) with VPIC code
    - 120,000 cores
    - 8 arrays (HDF5 datasets)
    - 32 TB to 42 TB files at 10 time steps
- Extracted I/O kernel
- M Aggregators to 1 shared file
- Trial-and-error selection of Lustre file system parameters while scaling the problem size
- Reached peak performance in **many** instances in a real simulation

More details: SC12 and CUG 2013 papers



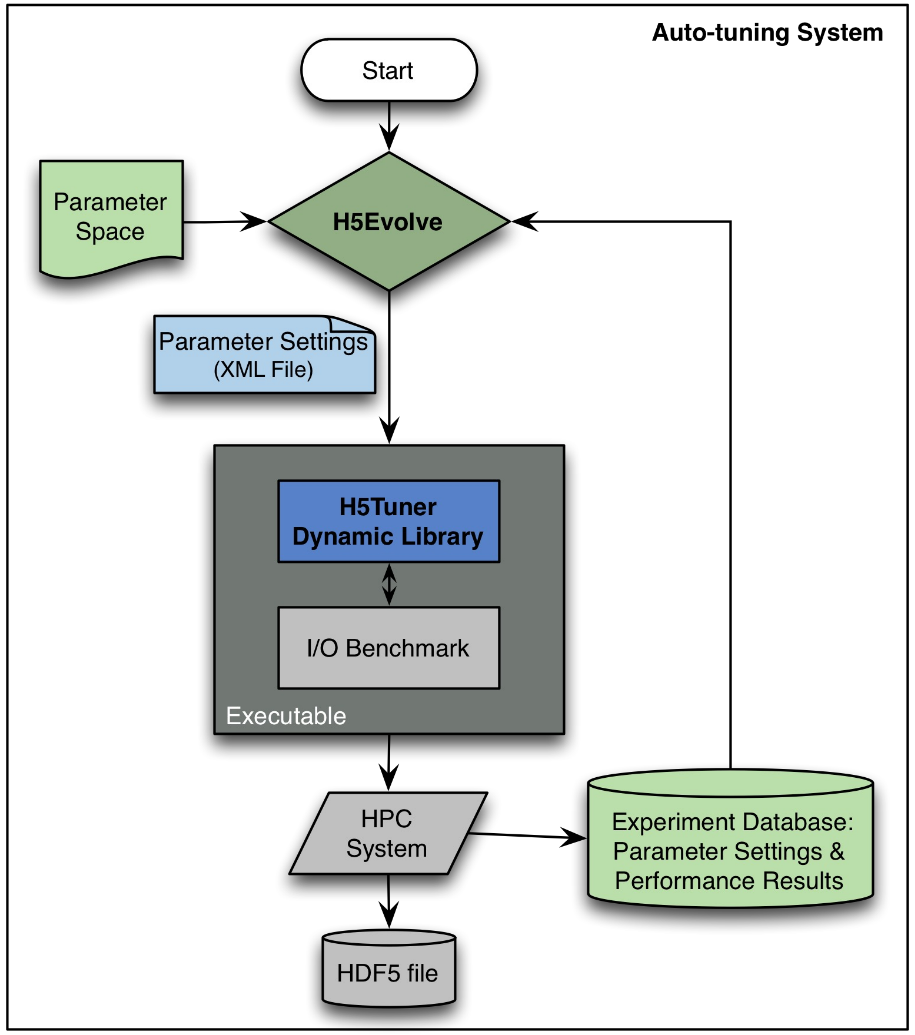


Tuning combinations are abundant

- Searching through all combinations manually is impractical
- Users, typically domain scientists, should not be burdened with tuning
- Performance auto-tuning has been explored heavily for optimizing matrix operations
- Auto-tuning for parallel I/O is challenging due to shared I/O subsystem and slow I/O
- Need a strategy for reduce the search space with some knowledge

Our solution: I/O Auto-tuning

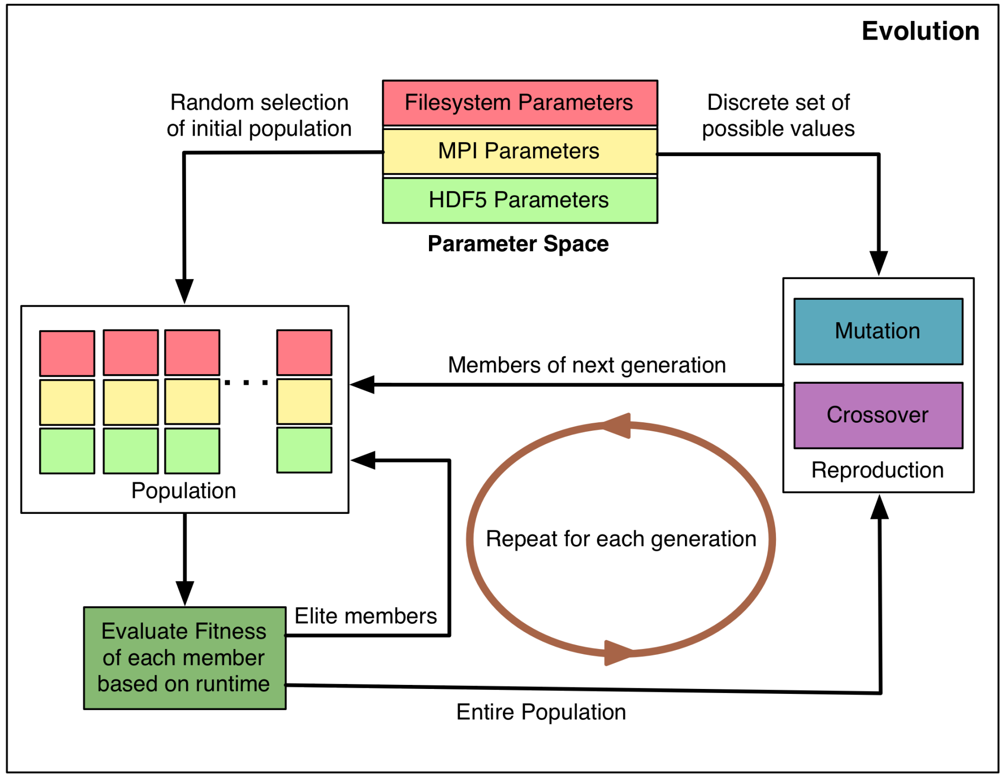
- Auto-tuning framework to search the parameter space with reduced number of combinations
- HDF5 I/O library sets the optimization parameters
- H5Tuner: Dynamic interception of HDF5 calls
- H5Evolve: Genetic algorithm-based selection



Earlier version: Genetic Algorithm-based

- GA evaluates fitness (I/O performance) and selects members based on least runtime and on mutation of various optimization parameters

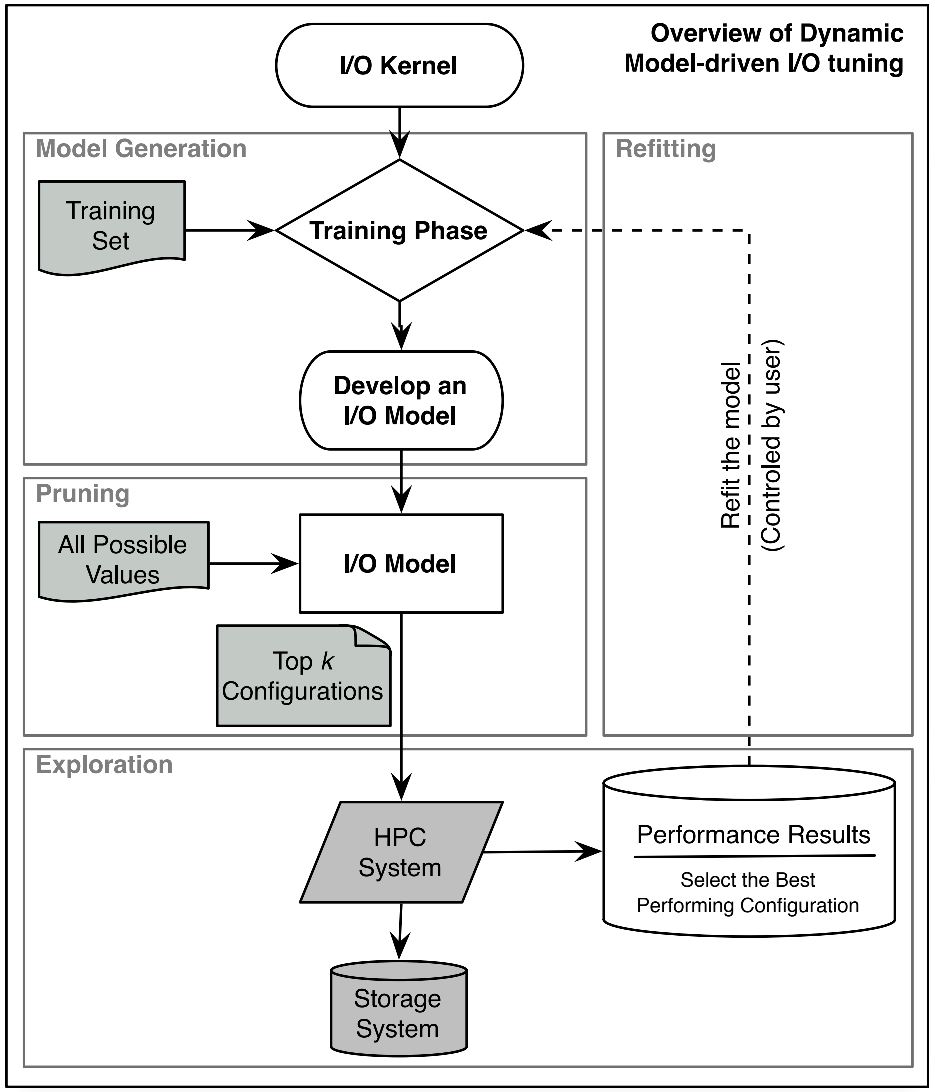
  - Problems
    - Long search time (more than 12 hours)
    - Limited general purpose applicability for different problem sizes


Evolution
Random selection of initial population
Filesystem Parameters
MPI Parameters
HDF5 Parameters
Parameter Space
Discrete set of possible values
Mutation
Crossover
Reproduction
Members of next generation
Population
Repeat for each generation
Elite members
Evaluate Fitness of each member based on runtime
Entire Population

B. Behzad et al. “Taming Parallel I/O Complexity with Auto-Tuning”, SC13

Dynamic Model-driven Auto-tuning

- Auto-tuning using empirical performance models of I/O
- Steps
  - Training phase to develop an I/O model
  - Pruning phase to select the top-K configurations
  - Exploration phase to select the best configuration
  - Refitting step to refine performance model



Training phase: Developing I/O models

- Faster reduction of search space: A statistical approach for generating empirical prediction models for parallel I/O performance
- Our goal is not to predict I/O performance accurately, but to reduce the parameter search space
- I/O layers and parameters considered
    - Application: File size
    - HDF5: Chunking size, alignment
    - MPI-IO: Number of aggregators, collective buffer size
    - Lustre: Stripe size and stripe count

Performance Model - Parameters

- Independent variables/parameters (e.g., the stripe count)
    - $x = [x_1, \cdots, x_{nx}]$
- Scalar-valued output/dependent variable (i.e., write time)
    - $y(x)$
  - Data collected from a set of experiments is of the form $\{\square(x^j, y^j) : j = 1, \ldots, n_y\}$

Empirical Performance Model

- **Non-linear regression model**

$$\mathrm{m}(x;\beta) = \sum_{k=1}^{n_b} \beta_k \phi_k(x)$$

  - Linear combinations of n_b non-linear, low polynomial basis functions (ϕ_k), and hyper-parameters β (selected with standard regression approach) for a parameter configuration of x
  - For example:

$$m(\mathbf{x}) = \beta_1 + \beta_2 \frac{1}{s} + \beta_3 \frac{1}{a} + \beta_4 \frac{c}{s} + \beta_5 \frac{f}{c} + \beta_6 \frac{f}{s} + \beta_7 \frac{cf}{a},$$

  - f: file size; a: number of aggregators; c: stripe count; s: stripe size

$$\beta_i = [10.59, 68.99, 59.83, -1.23, 2.26, 0.18, 0.01]$$

Model Training

- **Developed empirical model based on small-scale experiments**
    - **Time for pruned search space exploration: ~2 hours**
    - **6X to 12X improvement over GA for small-scale training phase**

# of cores	file size (GB)	training set size
128	32	216
256	64	120
512	128	72
1024	256	60
2048	512	60
4096	**1024**	**0**
8192	**2048**	**0**

Experimental Setup: Platforms

- **NERSC/Hopper**
    - Cray XE6
    - Lustre file system (156 OSTs, 26 OSSs)
    - Peak I/O BW: 35 GB/s
- **NERSC/Edison**
    - Cray XC30
    - Lustre file system (96 OSTs, 24 OSSs)
    - Peak I/O BW: 48 GB/s
- **TACC/Stampede**
    - Dell PowerEdge C8220
    - Lustre file system (160 OSTs, 58 OSSs)
    - Peak I/O BW: 159 GB/s

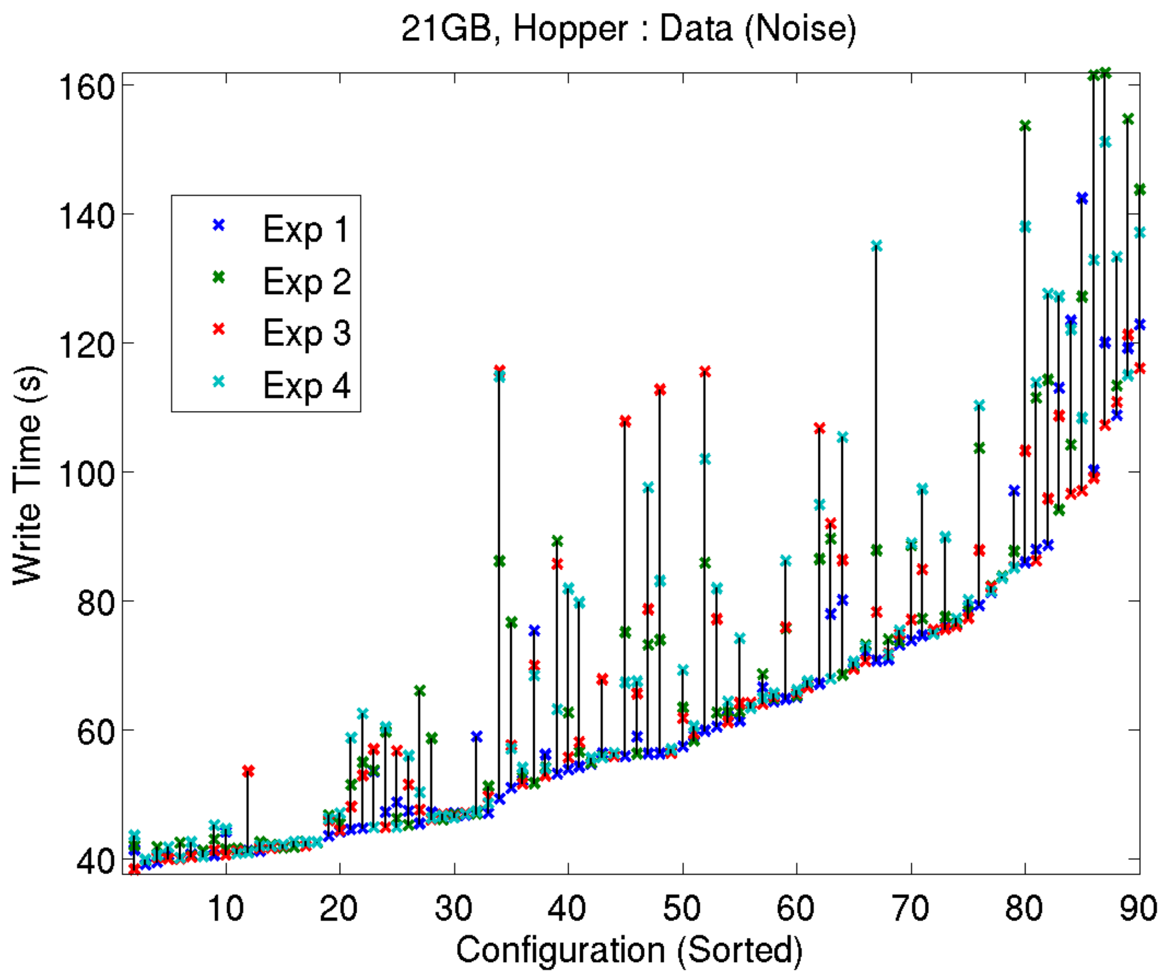
Experimental Setup: Application I/O Kernels

- **VPIC-IO**
    - IO-Kernel manually derived from VPIC plasma physics application
    - Writes 8 1D arrays
- **VORPAL-IO**
    - IO-Kernel manually derived from VORPAL accelerator modeling
    - Writes 3D block-structured grid
- **GCRM-IO**
    - IO-Kernel manually derived from GCRM atmospheric model
    - Writes 3D block-structured mesh

Parallel I/O – Performance variation

Parallel I/O performance varies significantly due to interference from other users

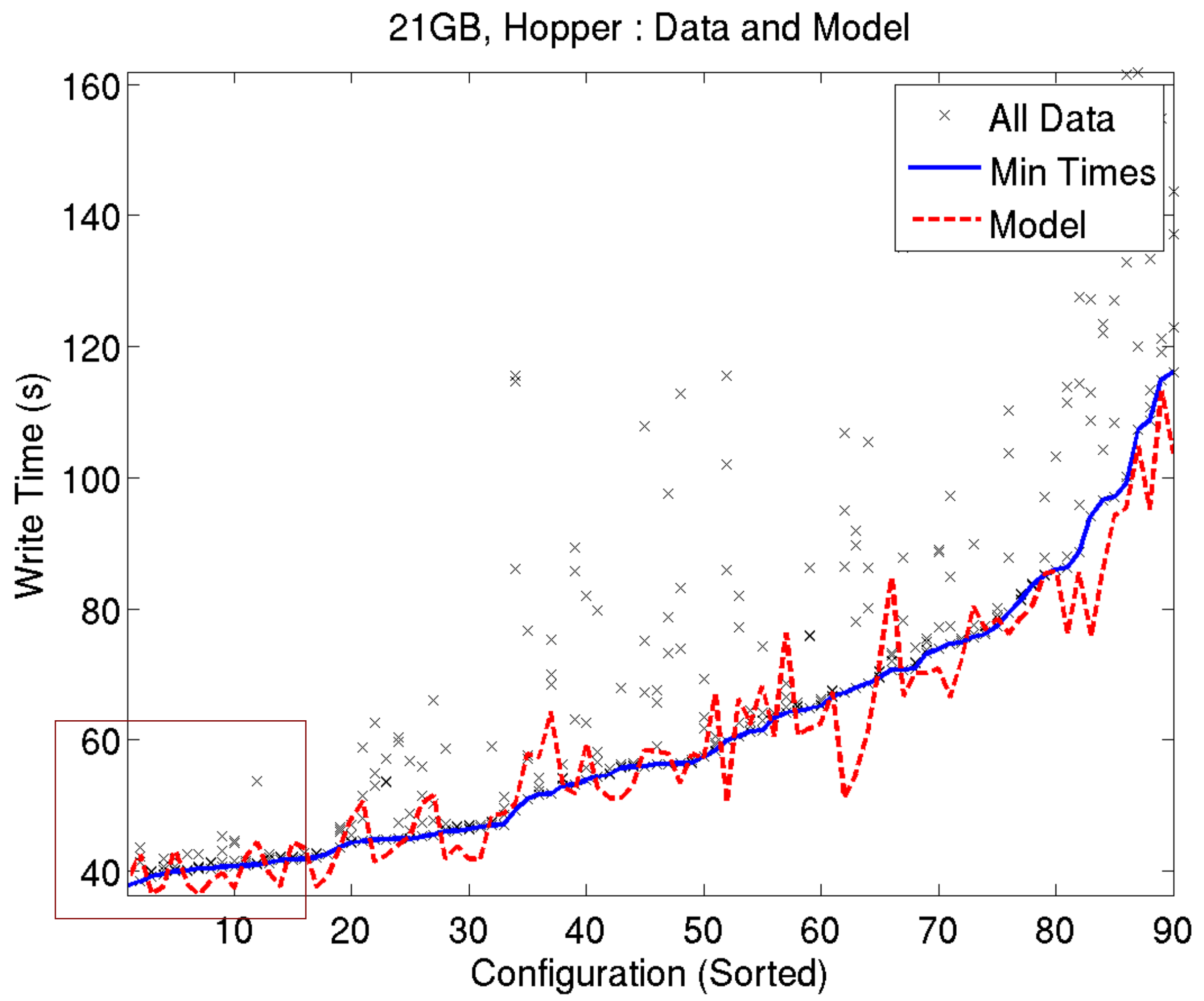
Variation is low for high-performant configurations



VPIC-IO on a single Lustre OST

Performance model accuracy

For high-performant configurations, model is accurate.

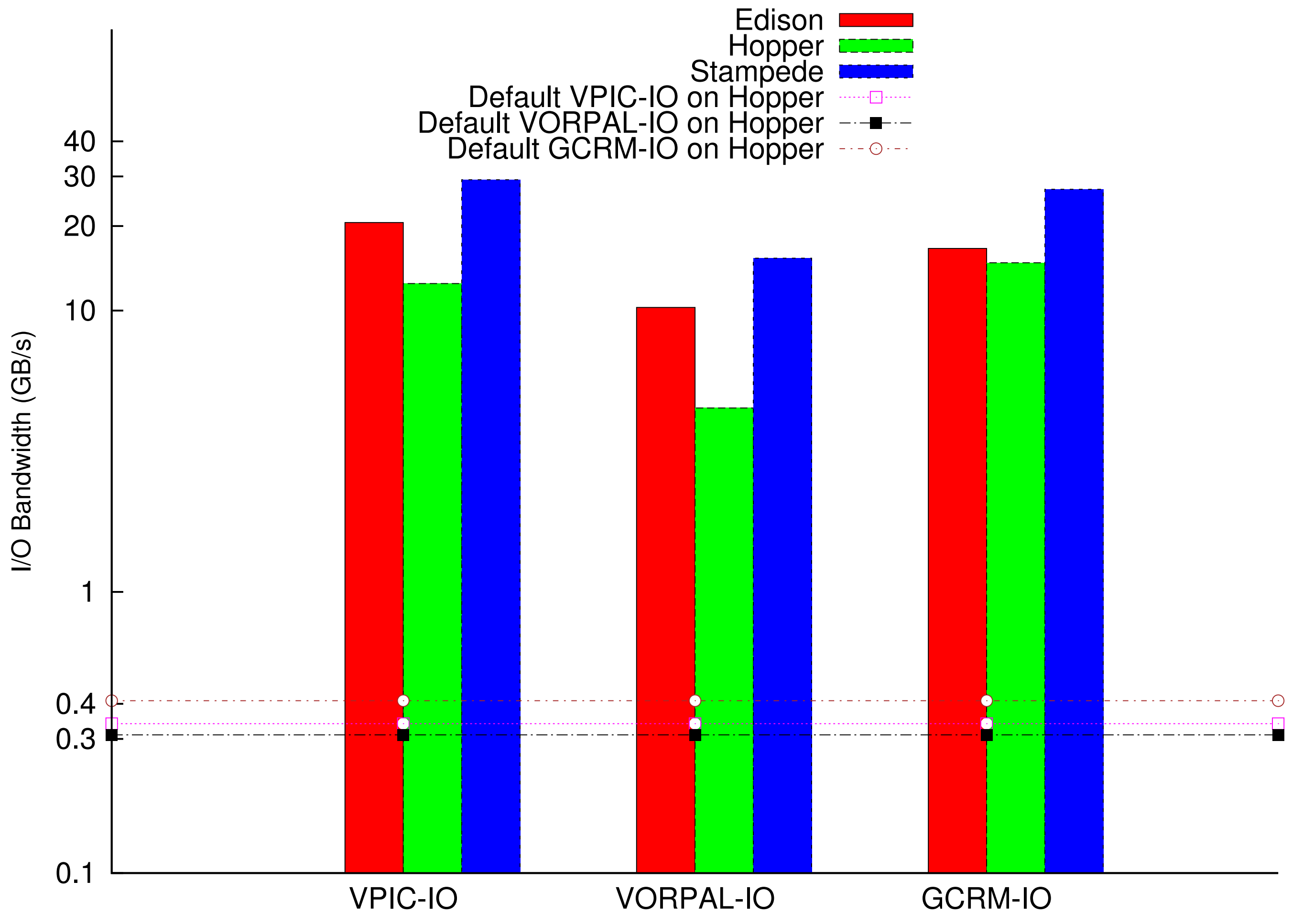


VPIC-IO on a single Lustre OST

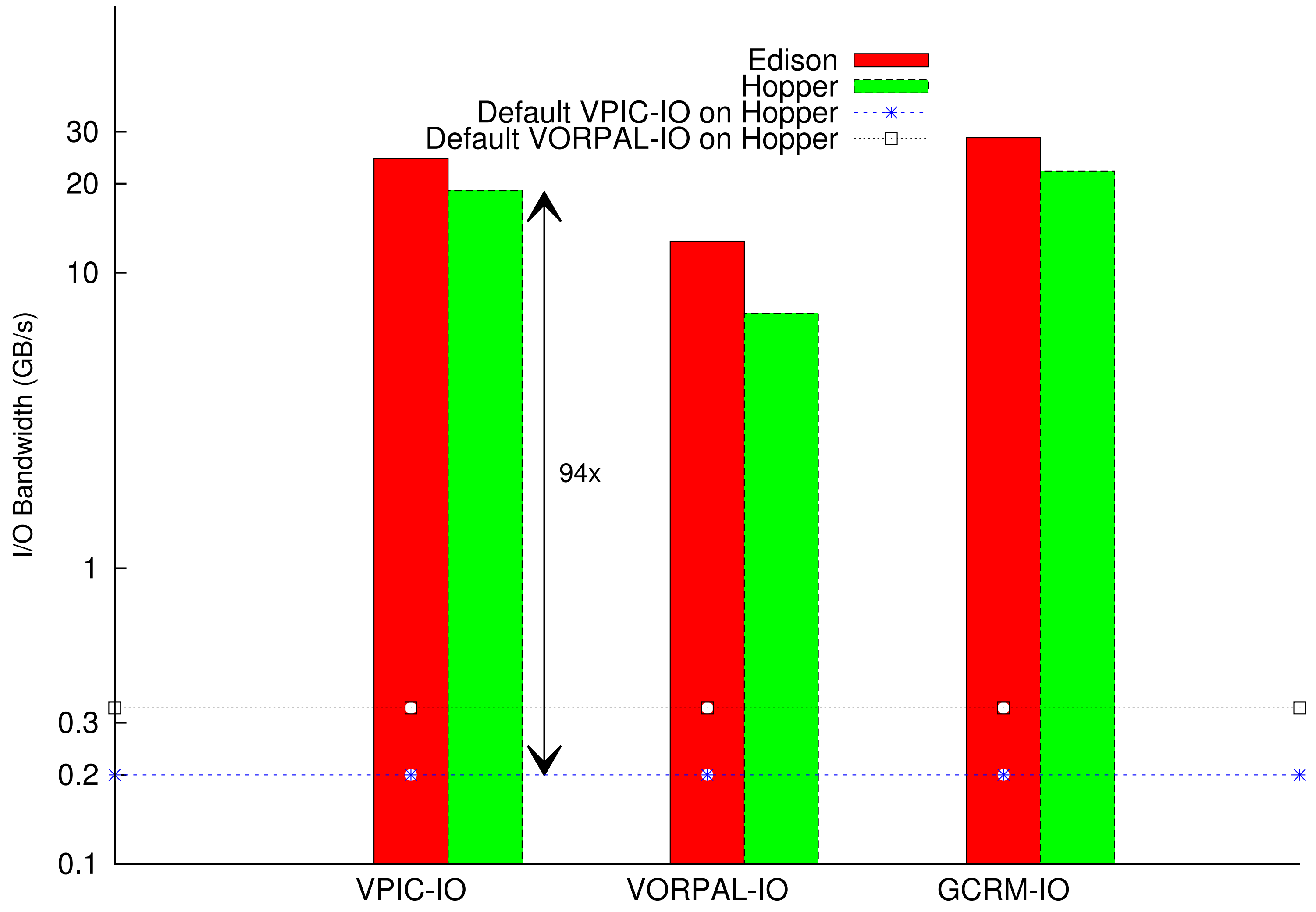
Performance Results – VPIC-IO

# of cores	File Size (GB)	Modeling Bandwidth (MB/s)	GA Bandwidth (MB/s)	Default Bandwidth (MB/s)	Speedup
128	32	2075	3034	472	**4.4X**
512	128	5185	-	409	**12X**
1024	256	6182	-	337	**18X**
2048	512	11422	14900	412	**28X**
4096	1024	14892	17620	365	**41X**
8192	2048	18857	-	345	**54X**

Performance Improvement: 4K cores



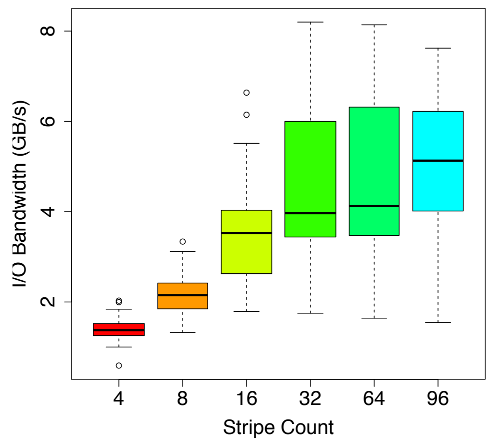
Performance Improvement: 8K cores



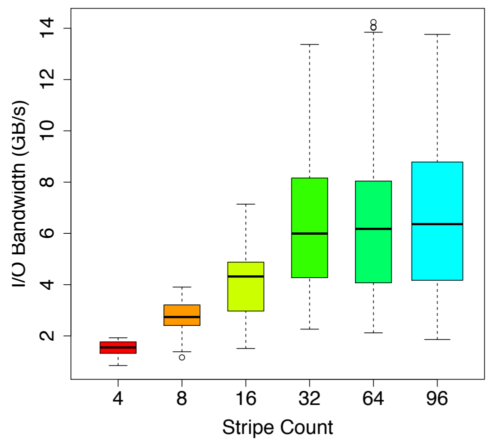
Time to prune search space

Method	Training Phase	Applying the Model	Per App. & Scale Tuning	App. Runtime (VPIC-8192 on Hopper)
Genetic Algorithm	N/A	N/A	$>$ 10 hours	118 seconds
Model Fitting	$>$ 10 hours (can reuse)	$<$ 1 minute (automatic)	$<$ 1 hour	100 seconds
Default Config.	N/A	N/A	N/A	$>$ 3 hours

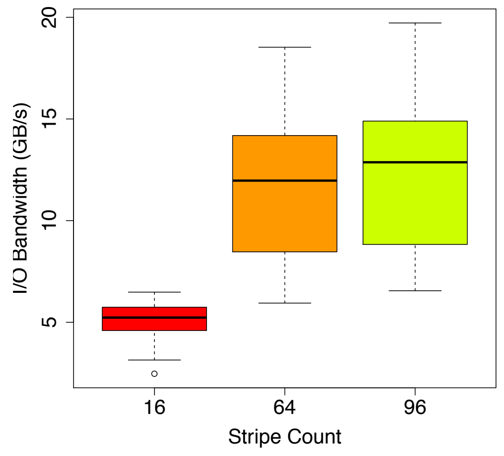
Analysis of Inter-dependencies: Stripe Count



(a) 512 cores - 128 GB



(b) 1K cores - 256 GB

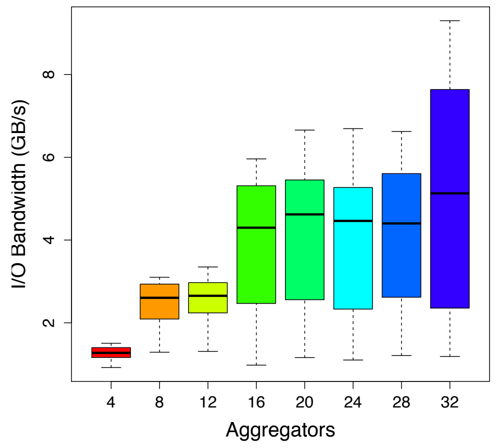


(c) 2K cores - 512 GB

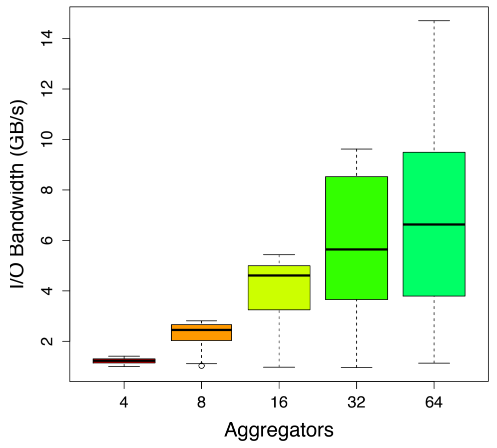
Effect of Lustre's stripe count at three scales of VPIC-IO on Edison

- **As we increase the problem size, increasing Lustre's stripe count leads to more parallelism and therefore results in an improvement in the I/O bandwidth**
- **This applies to all platforms**

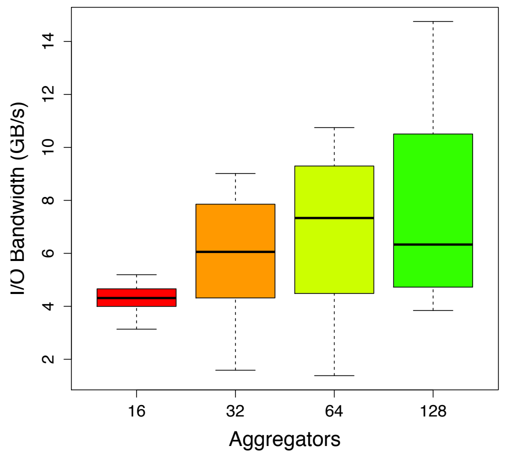
Inter-dependencies: Number of aggregators



(a) 512 cores - 128 GB



(b) 1K cores - 256 GB

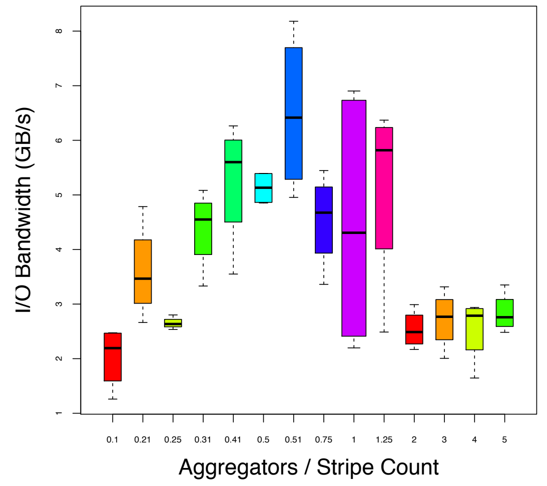


(c) 2K cores - 512 GB

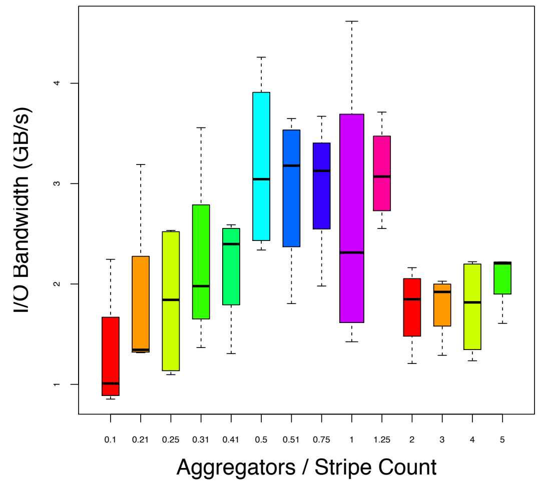
Effect of MPI-IO aggregators at three scales of VPIC-IO on Stampede

- **As we increase the problem size, increasing MPI-IO aggregators gives better performance**

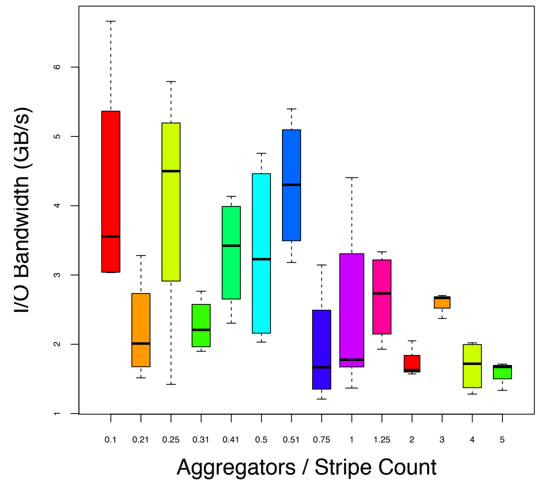
Inter-dependencies: Aggregators to Stripe count ratio



(a) VPIC-IO



(b) VORPAL-IO



(c) GCRM-IO

Ratio of MPI-IO's aggregators and Lustre's stripe count on three different applications on 2K cores of Hopper - 512 GB of data

- **The number of aggregators each OST handles has an impact on concurrency of Lustre and the communication between an aggregator and an OST**

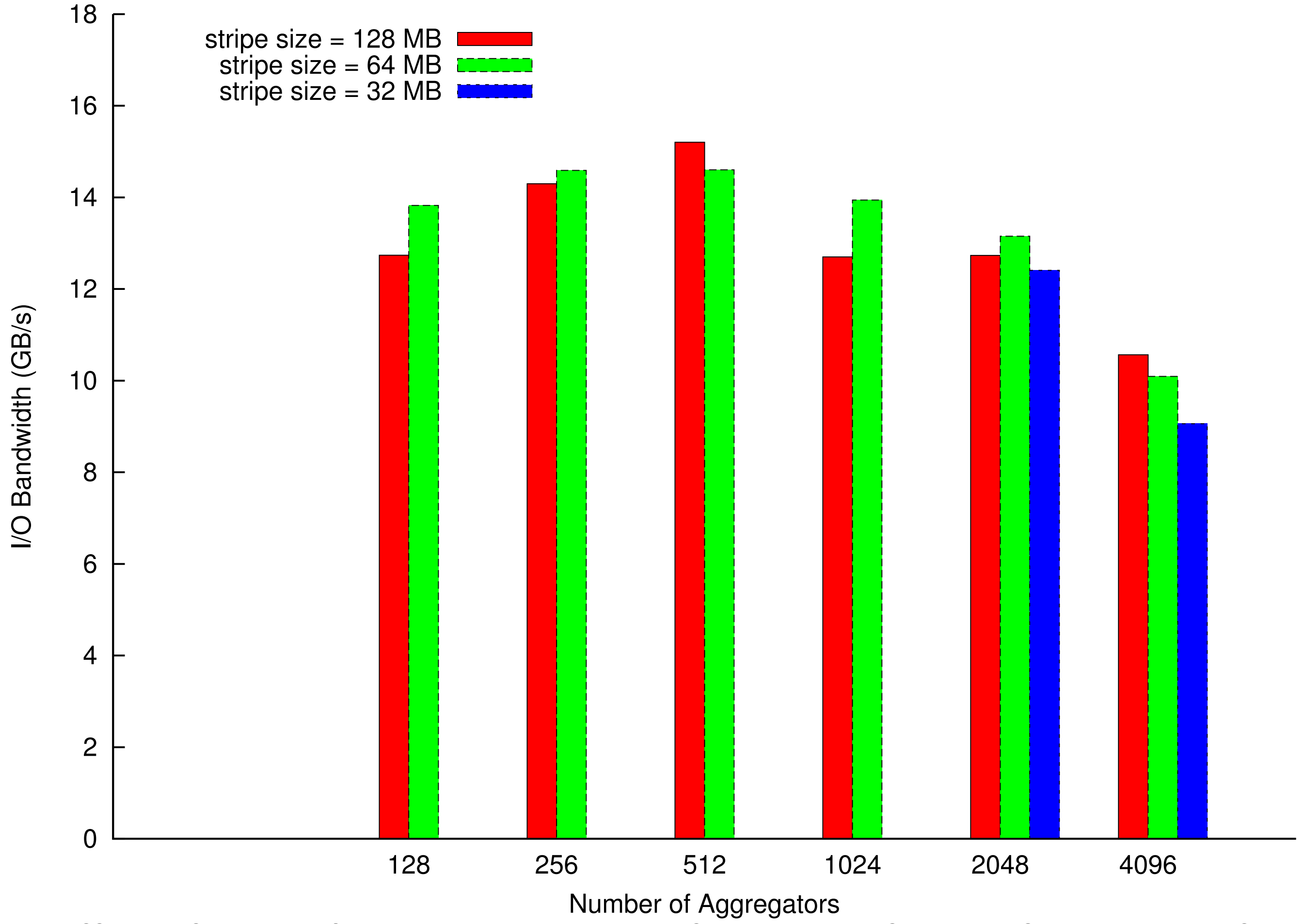
Inter-dependencies: Stripe size matters

exp_id	c	s	a	f (GB)	time (s)	bandwidth (GB/s)
0	156	1	1024	1024	58.87	17.39
1	156	2	1024	1024	49.84	20.54
2	156	4	1024	1024	47.06	21.75
3	156	8	1024	1024	42.11	24.31
4	156	16	1024	1024	38.99	26.25
5	156	32	1024	1024	40.28	25.41
6	**156**	**64**	**1024**	**1024**	**35.06**	**29.20**
7	156	128	1024	1024	44.96	22.77
8	128	1	1024	1024	61.33	16.69
9	128	2	1024	1024	65.87	15.54
10	128	4	1024	1024	58.94	17.37
11	128	8	1024	1024	54.72	18.71

Table: Top twelve configurations predicted by our model for VPIC-IO on 4K cores of Stampede

Inter-dependencies: Aggregators and stripe size

The number of MPI-IO aggregators should be specified carefully and not blindly minimized or maximized.

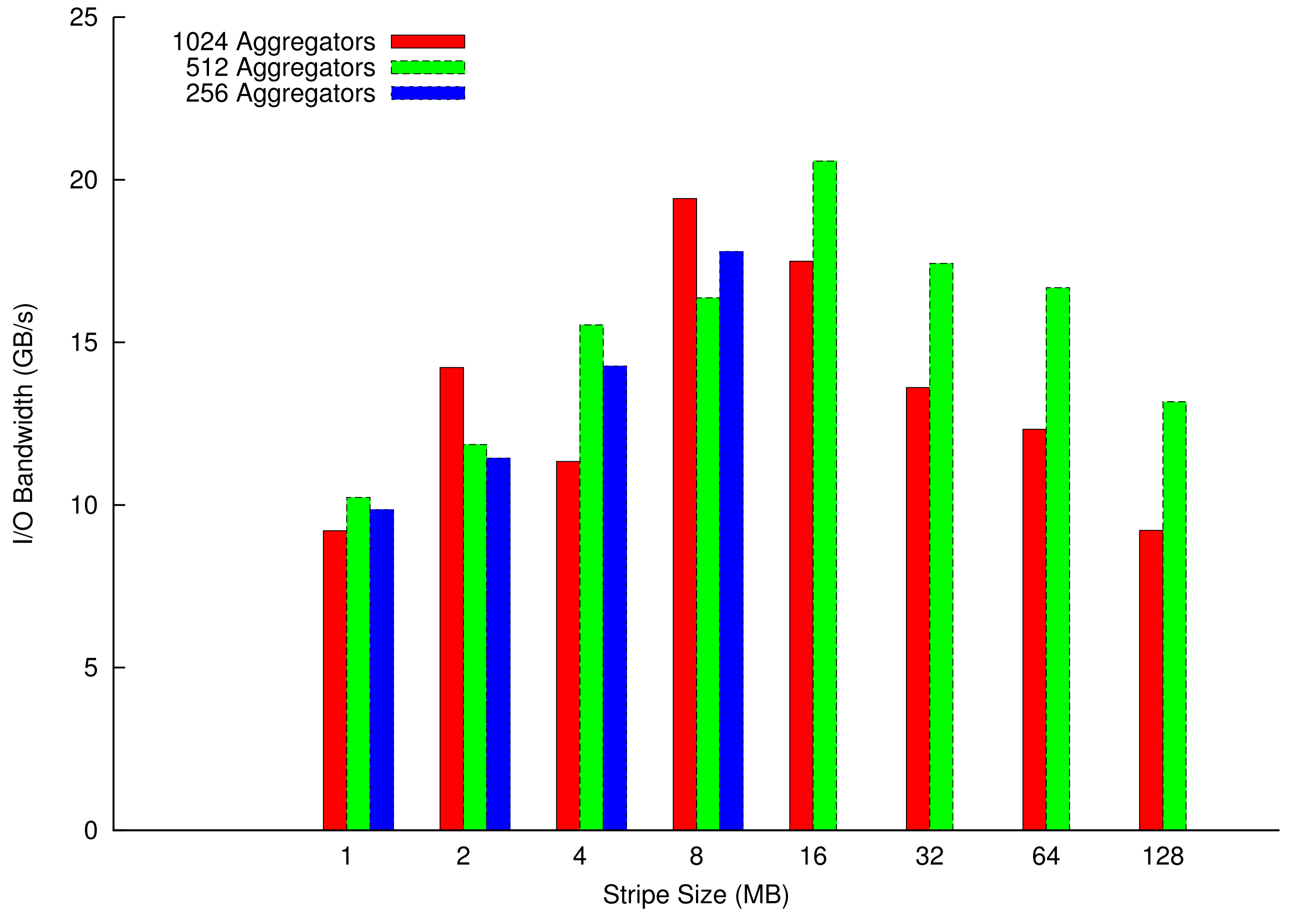


Effect of MPI-IO's aggregators on performance of 14 configurations of VORPAL-IO on 16K cores of Edison. Stripe count is fixed at 96

Inter-dependencies: Stripe size and aggregators

Two-fold difference between poor and best performing

On Edison, best stripe size was 16MB while on Stampede it was 64MB



Effect of stripe size on the Top 20 VPIC-IO configurations on 4K cores of Edison. Stripe count is fixed at 96

Conclusions

- It is challenging to obtain maximum performance from I/O subsystems due to interdependencies among software libraries and their tuning parameters
- Introduced a model-driven tuning framework that uses non-linear regression models to find the top performing configurations
- Achieve significant portion of the available I/O performance on various HPC platforms for a range of applications
- We shed some light on the complex interdependencies of different parallel I/O tunable parameters

Summary of today's class

- Combinations of tuning options is significantly large
- Used genetic algorithms to find tuned combination – takes a long time to train
- Using analytical modeling-based approach is faster, but applying to different scales and different applications is difficult
- Next class: Recap of the first half