BLUE WATERS

SUSTAINED PETASCALE COMPUTING

How Deep is Your I/O? Toward Practical Large-Scale I/O Optimization via Machine Learning Methods

Rob Sisneros, Jonathan Kim,
Mohammad Raji, Kalyana Chadalavada

GREAT LAKES CONSORTIUM FOR PETASCALE COMPUTATION

The Point

- Motivation
  - I/O is a routine bottleneck, there is obvious benefit to optimizing
  - Modeling Parallel I/O
- This work
  - Explore machine learning methods for modeling HPC I/O
  - Create practical utility for I/O optimization

This Talk

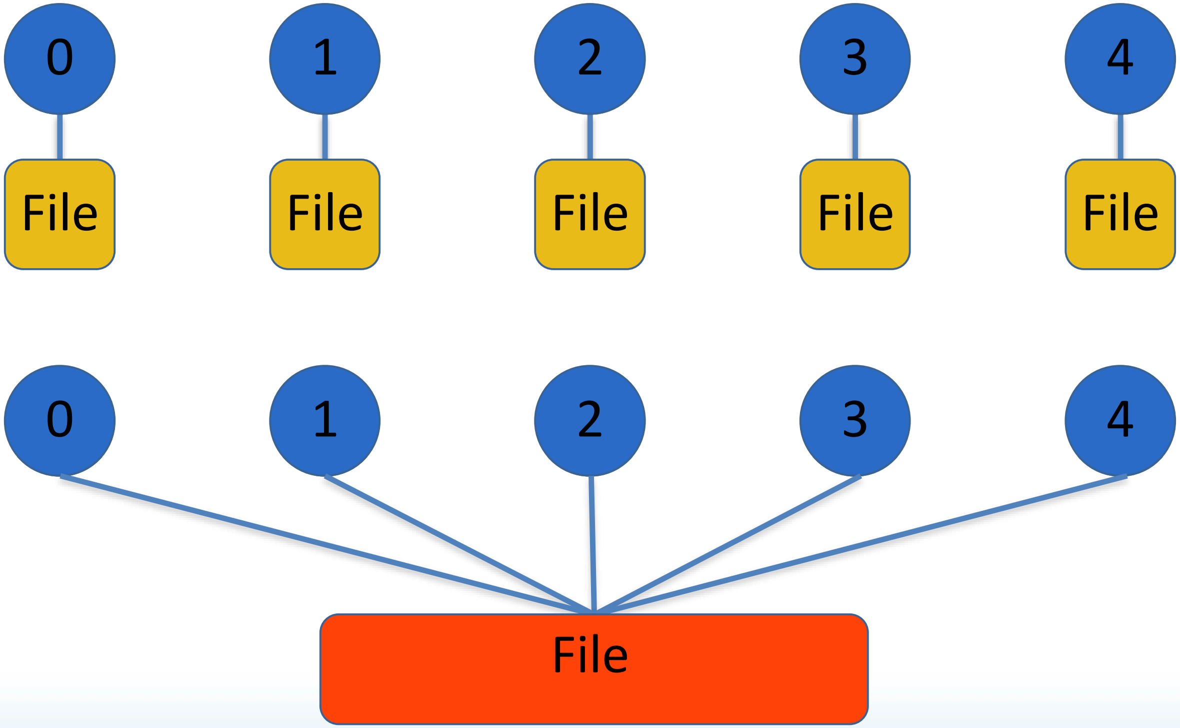
- Modeling Parallel I/O
  - HPC I/O considerations
  - Configuration space
  - ML approaches
- Generating training/test data
- An optimization utility
- Evaluation

Motivation I

HPC I/O CONSIDERATIONS

Large Scale I/O in Practice

- 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:



The Typical Discussion

- 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

The Typical Followup

- 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

Motivation II

I/O CONFIGURATION SPACE

What Feels Right

- It is possible to statistically model I/O for an HPC system, it's just impossible
  - There are just too many possible configurations
  - Would require prohibitive benchmarking
  - Would be similarly expensive to update
- Machine Learning may provide decent model training on significantly reduced configuration space

HPC Machine Learning

- Machine Learning seems to have many applications to HPC diagnostic data, but…
  - Ground truth data required on which to train
    - Possible but not practical to hand classify this data
    - Often, data understood well enough to hand classify does not represent *interesting* applications of ML
  - Data including certain metrics have built in "classification" – downtime, utilization, etc.
- I/O as a function of configuration -> throughput fits perfectly

So, How Big is This Configuration Space?

- Many parameters
  - Job size and I/O size
  - Aggregation level/type
  - Number of files
  - File system settings (stripe size, count, etc.)
  - I/O library (and its freakin' metadata)
- Interdependencies of the above make it difficult to clearly iterate through the space (I wrote some broke-ass code to try)

Back to the Back of the Envelope

Different sizes of I/O operations:

- "Practical" upper bound
  - Every XE node writes 50GB
  - (22,636 * 50) = 1,131,800GB
- Technically, we could write in byte increments
  - 1 byte, to 1.13PB
  - 1,215,260,996,403,200 different sizes
- Less obnoxious (minimum Lustre file size)
  - 512 byte increments
  - 2,373,556,633,600 sizes

Back of the Envelope (II)

Forget size, number of configurations:

- Job sizes: [1, 22636] nodes, [1, 32] PPN
- Job I/O assumptions
  - Each writing core writes the same amount
  - An operation can go to [1, nodes*PPN] number of files
- Stripe count possibilities on Blue Waters [1, 360]
- Stripe sizes {64K, 128K, … , 512M, 1G}, 15 total

The Grand Total

- 11,687,892,956,467,200
- The apparent overlap
  - Example: 128K I/O is both
    - 64K from 2 cores on a single node
    - 64K from single cores on two nodes
- We want this distinction, number of nodes has significant impact
  - Affects throughput to I/O nodes, and likewise
  - Network contention

The Grand Total (II)

- 11,687,892,956,467,200 does however include several bone-headed Lustre striping settings
- Let’s assume the “right” choice is always made
- New grand total: 2,164,424,621,568
- A conservative number restraints
  - Power of 2 node counts, PPN counts
  - Only write to same-size files
  - Correct striping
- New new grand total: 38,132,160

The Grand Total (III)

- 38,132,160 is still prohibitive for adequate benchmarking for statistical modeling
- 38,132,160 perspective
  - There are still other settings, e.g. stripe offset
  - This nor any of the other totals include multiple I/O libraries!!
- ML modeling hope
  - Adequate training on very reduced configuration set
  - “Adequate” as in at least providing a heuristic for optimization

THE APPROACH

ML Modeling

- Trying two approaches
  - SVR
    - Support Vector Regressor
    - Regression predicting continuous ordered variables
    - Natural fit to our data
    - Python sklearn.svm.SVR
  - DNN
    - Deep neural network
    - Selected because “DEEP LEARNING!!!!”
    - 3 dense layers
    - Rectified linear unit activation
    - Mean squared error loss function
    - Python Keras (on TensorFlow)
- Incremental testing to analyze how things are actually working
- Optimization utility leveraging trained models

Training and Testing

I/O BENCHMARK DATA

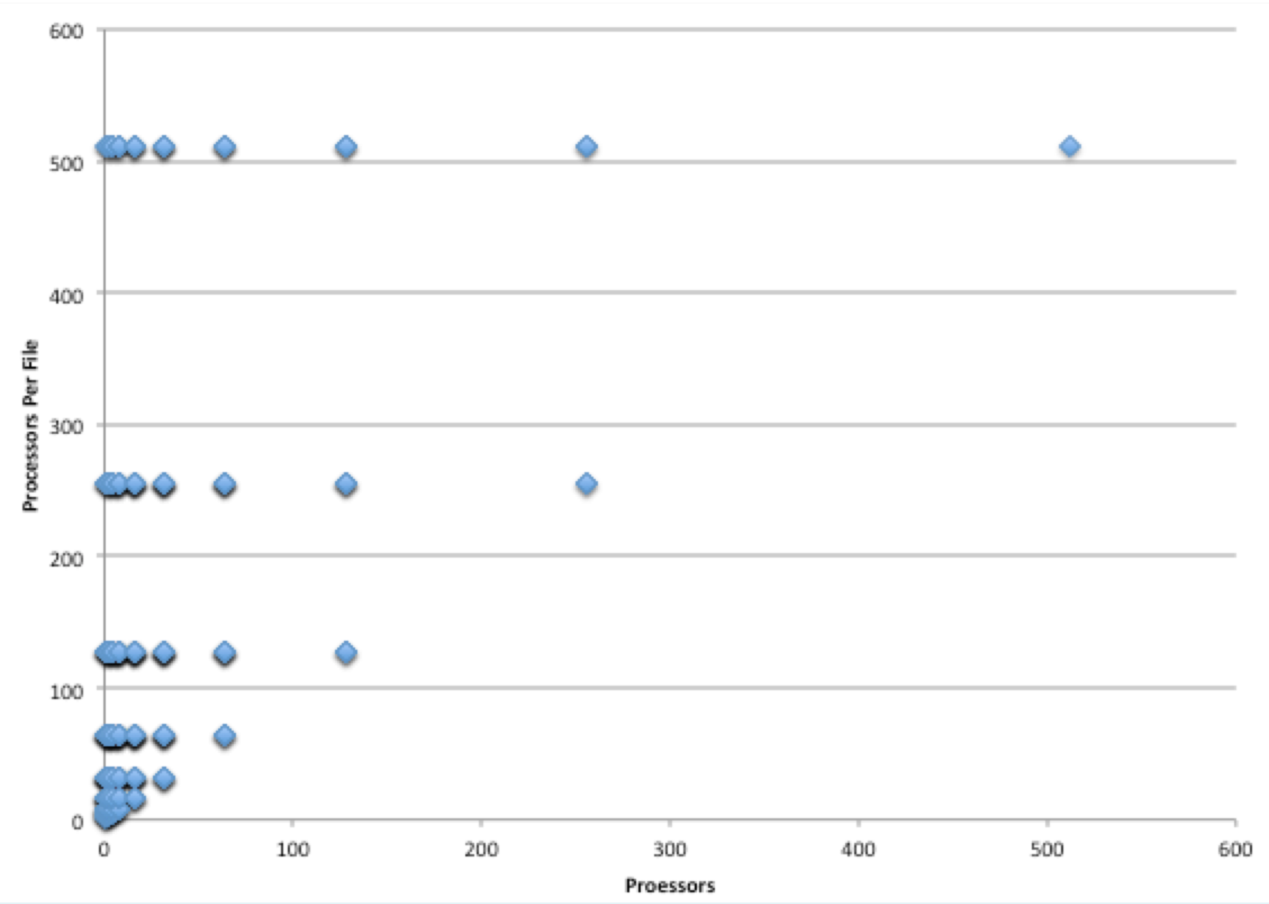
Benchmarks

- Custom aggregation benchmark
  - Combinations of *f* files per node shared across *m* nodes
  - Not possible with common benchmarks
  - Measures only write time
- 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

The Custom Aggregation Benchmark

- 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 aggregate write size consistent

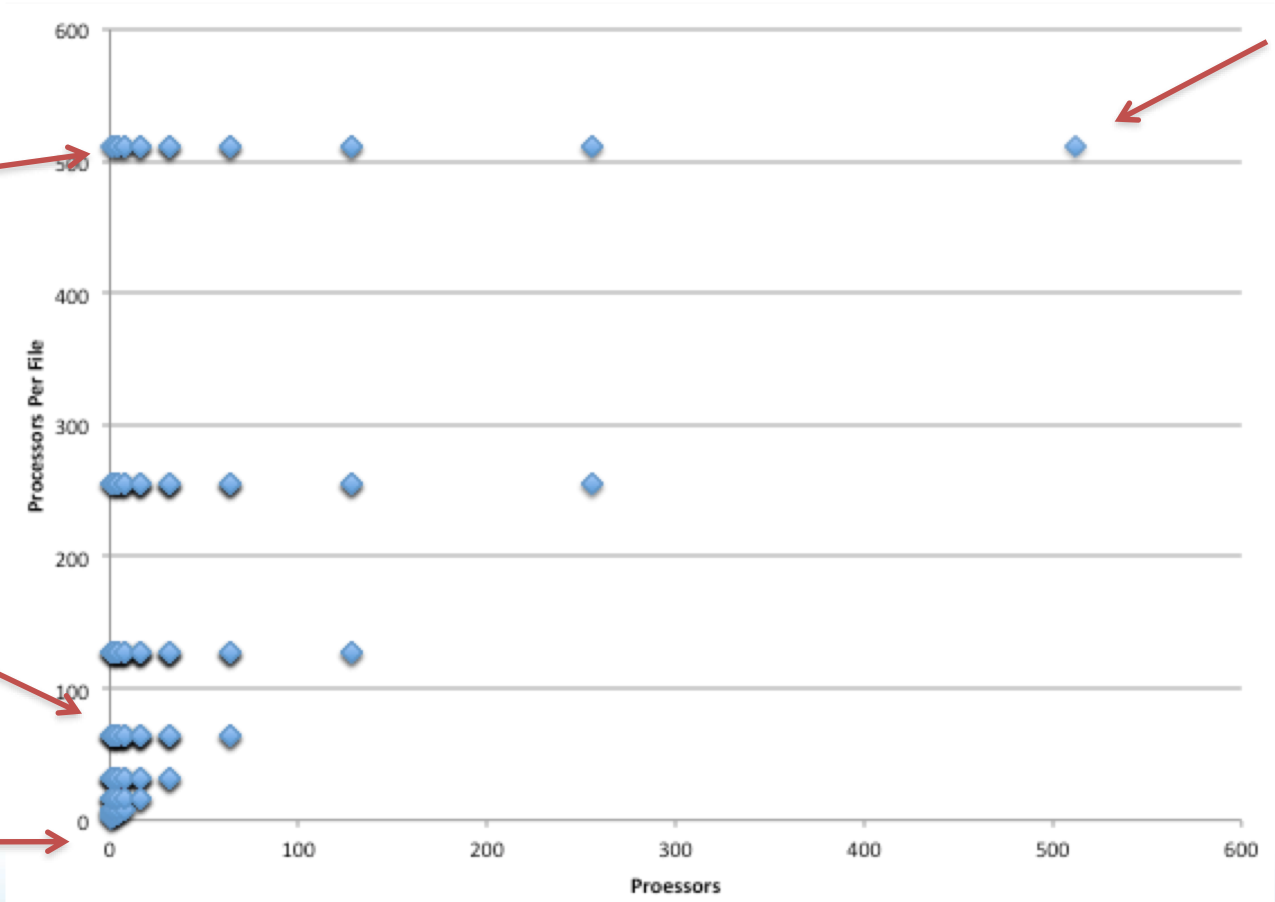
The Classification:
Processors vs. Processors per File



Common Patterns

File per process

Single shared file



File per Node

Serial

The Aggregate Benchmark Runs

- Run 1
  - 32 nodes, 16PPN, I/O size: 1MB/node (32MB)
  - 315 separate tests
- Run 2 – same as run 1, but 1GB/node (32GB)
- Run 3
  - 512 nodes, 16PPN, I/O size: 1MB/node (512MB)
  - 826 tests
- Run 4 – same as run 3, but 1GB/node (512GB)
  - 675 tests (too large to aggregate to single node, etc.)
  - Slow. Only collected 296 of the tests.

IOR Benchmark Runs

- File per process extravaganza
  - Stripes: {512K,1M,2M,4M,8M,16M,32M,64M,128M,256M,512M,1G}
  - Procs: {2,4,8,16,32,64,128,256,512,1024,2048,4096,8192,16384}
  - PPN: 16
  - Also
    - For stripe size of 32MB
    - PPN: {1,2,4,8}
    - Procs: {2,4,8,16,32,64,128,256,512,1024,2048,4096}
- run on 16384/16 = 1024 nodes
- 200 tests, 3 iterations each (600 tests)

IOR Benchmark Runs (II)

- Single Shared File
  - Stripes: {512K,1M,2M,4M,8M,16M,32M,64M,128M,256M,512M,1G}
  - Procs: {2,4,8,16,32,64,128,256,512,1024,2048}
  - PPN: 16
  - Also
    - For stripe size of 32MB
    - PPN: {1,2,4,8}
    - Procs: {2,4,8,16,32,64,128,256,512,1024,2048,4096}
  - Stripe count set to number min(procs, 360)
    - 360 is number of available OSTs
    - If Procs>360, 256 stripe count also tested
- run on 2048/16 = 128 nodes
- 151 tests, 3 iterations each (453 tests)

Training/Test Data Format

- Span configuration space with series of parameters with minimal interdependencies
- Used the following parameters
  - Nodes
  - PPN
  - Nodes per file
  - Files per node
  - Unit size
- To model Throughput

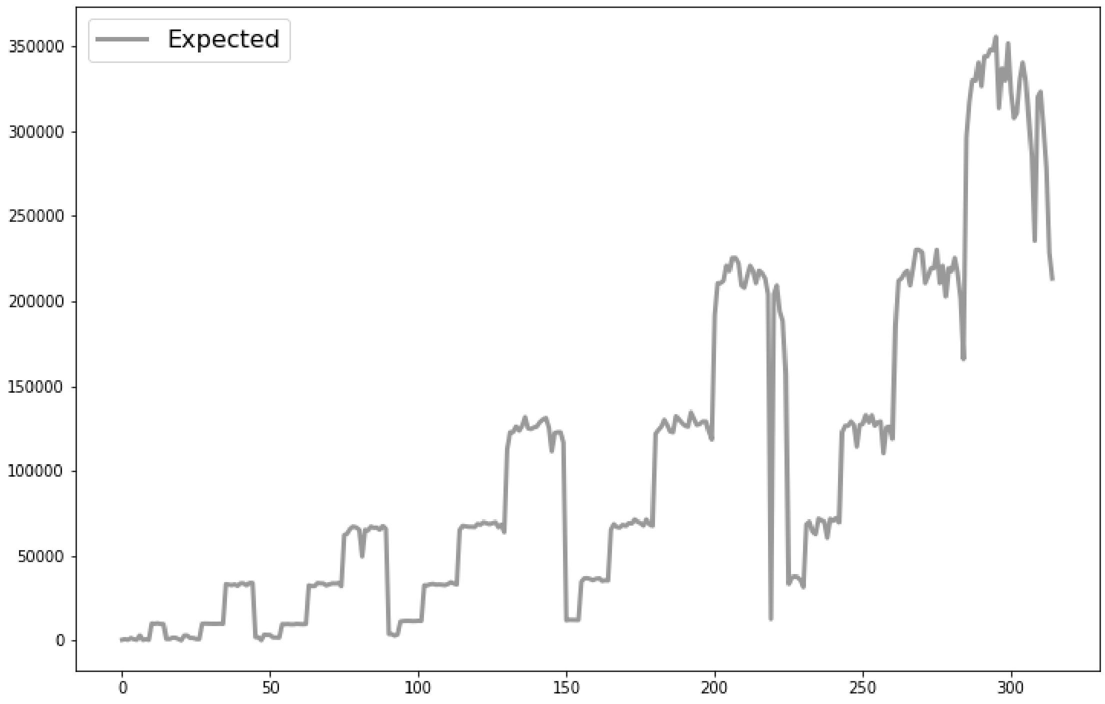
TRAINING/TESTING

Test 1

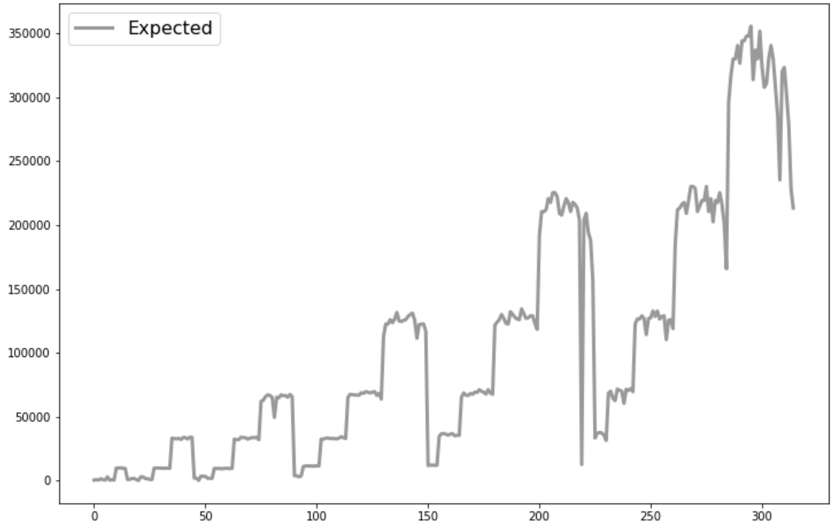
- Training data: custom benchmark runs 2 and 3
- Testing data: run 1

- Purpose of test
  - Possible to run early (while collecting bigger runs)
  - Runs 2 and 3 don't cover some of the smaller tests in run 1

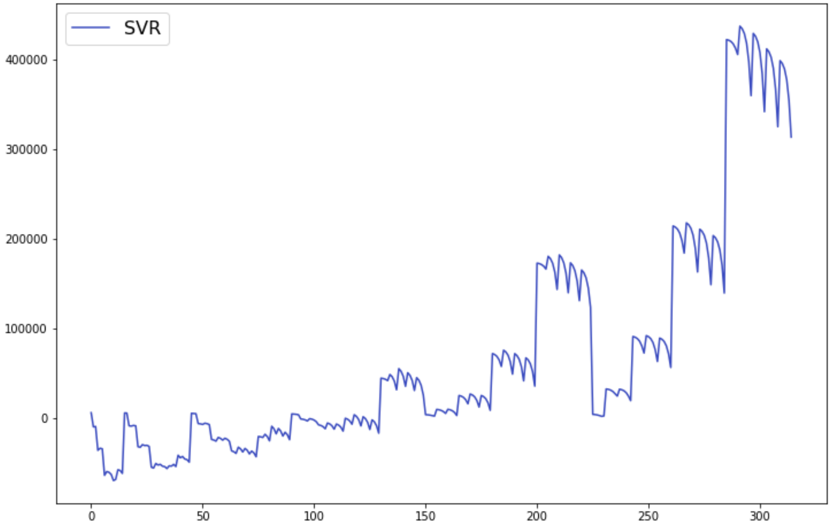
Expected: Run 1 Throughput Measurements



SVR Model

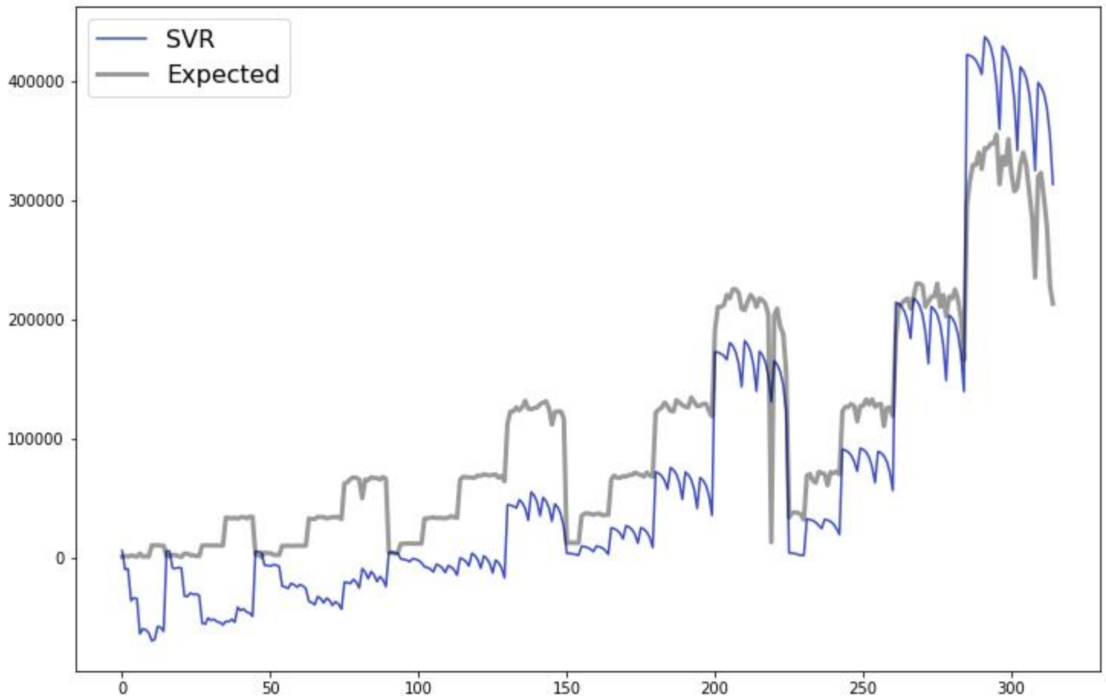


Expected
350000
300000
250000
200000
150000
100000
50000
0
0
50
100
150
200
250
300

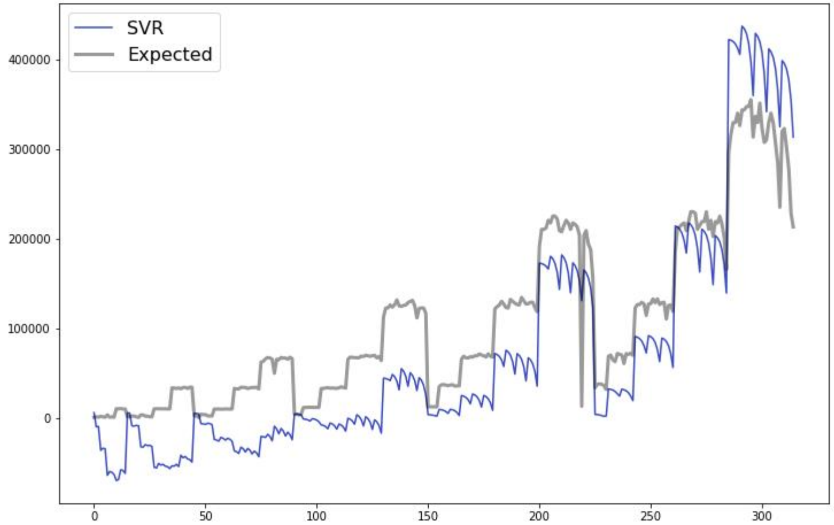


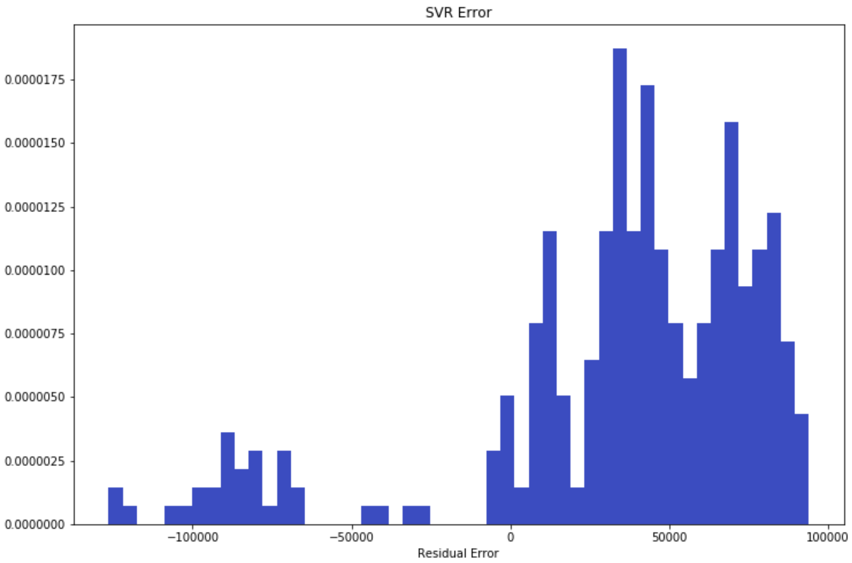
SVR
400000
300000
200000
100000
0
0
50
100
150
200
250
300

SVR Model

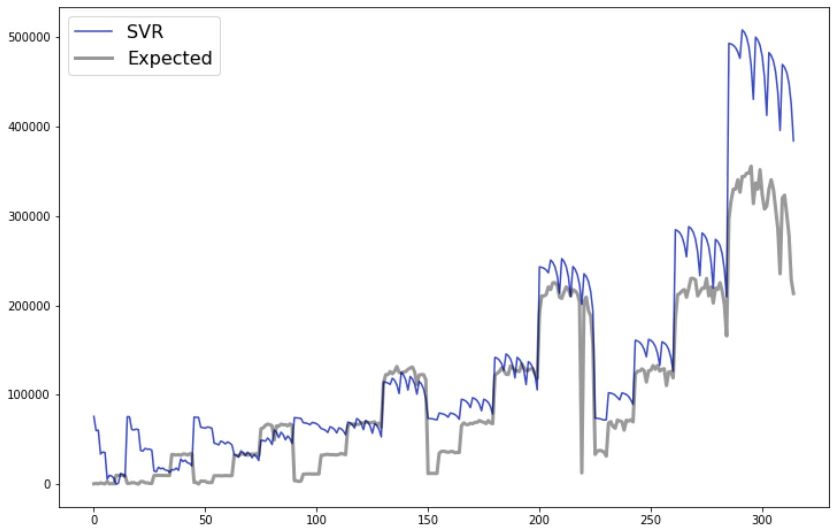

SVR
Expected
400000
300000
200000
100000
0
0
50
100
150
200
250
300

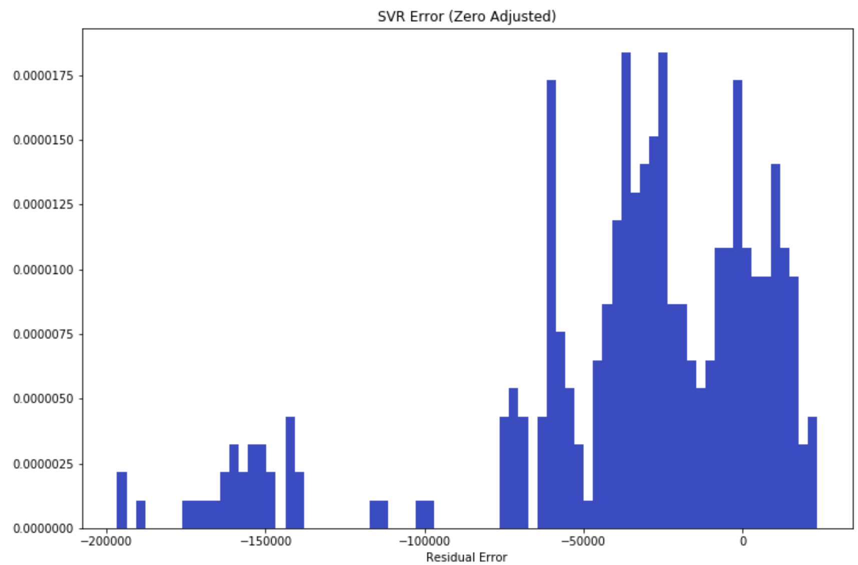
SVR Model, Error



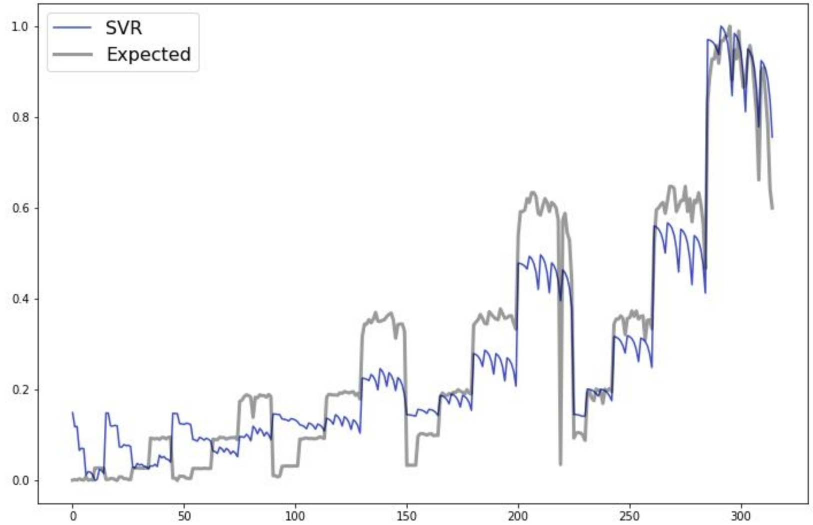


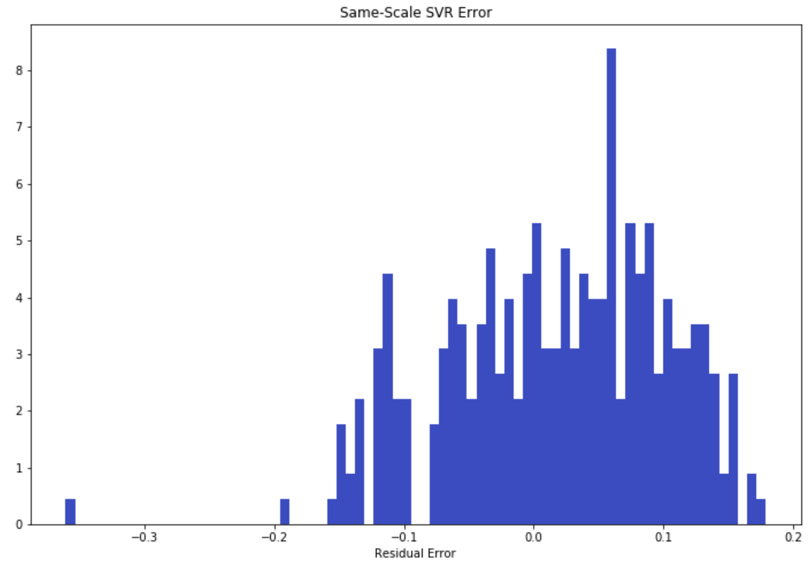
SVR Model, Zero Adjusted



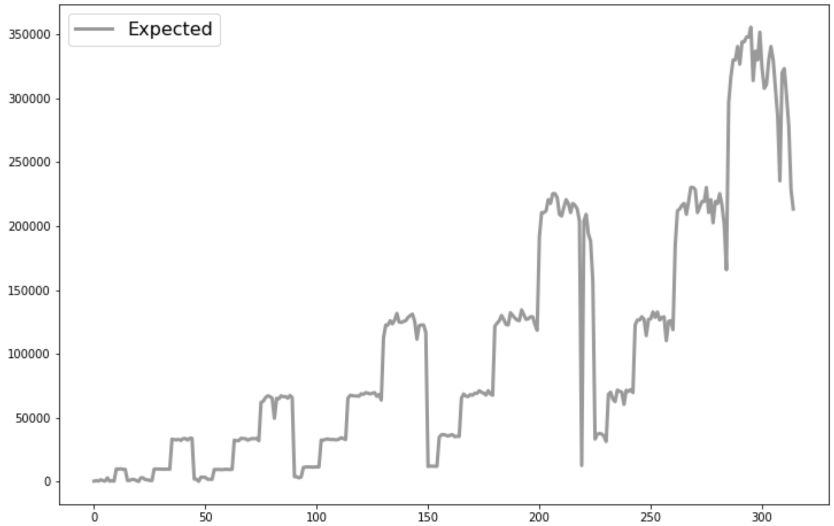


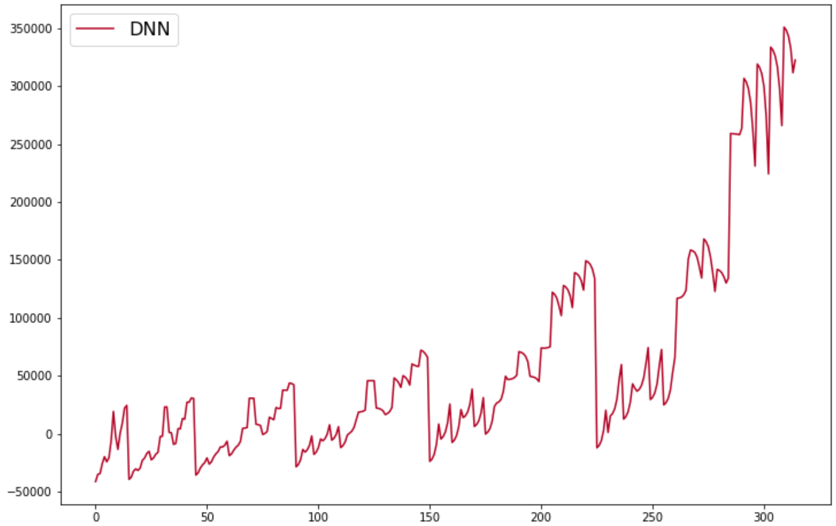
SVR Model, Independent Scale



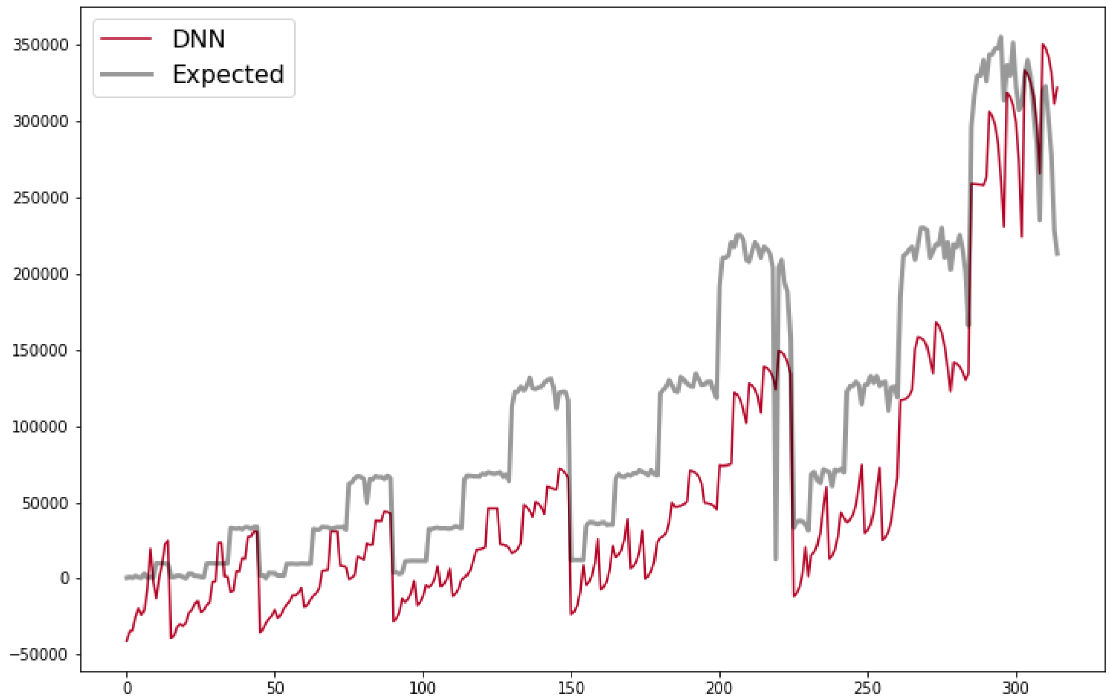


DNN Model

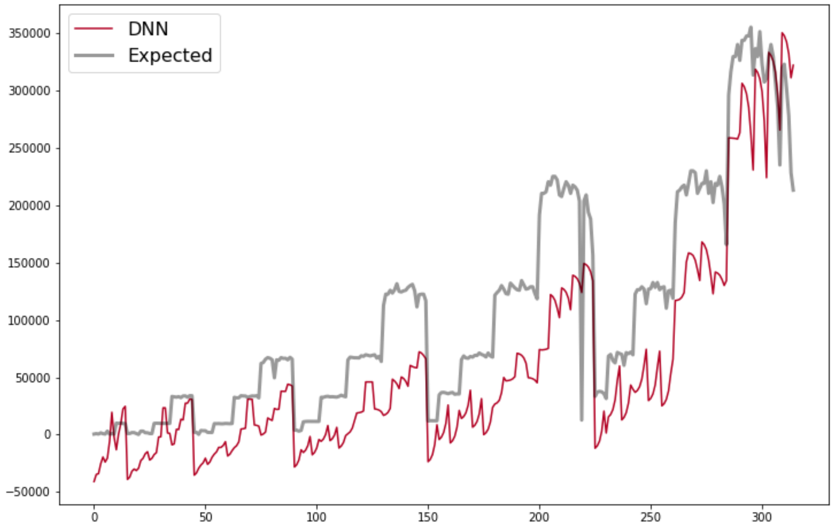

Expected
350000
300000
250000
200000
150000
100000
50000
0
0
50
100
150
200
250
300

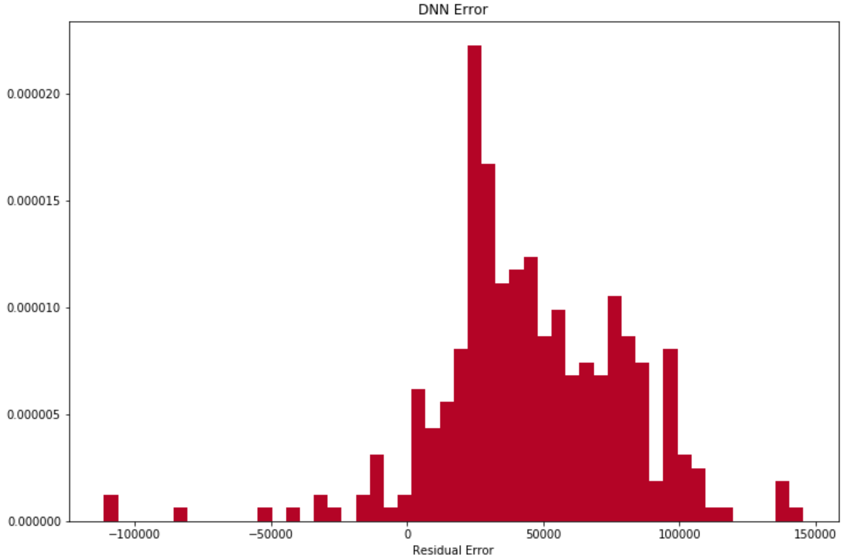

DNN
350000
300000
250000
200000
150000
100000
50000
0
−50000
0
50
100
150
200
250
300

DNN Model

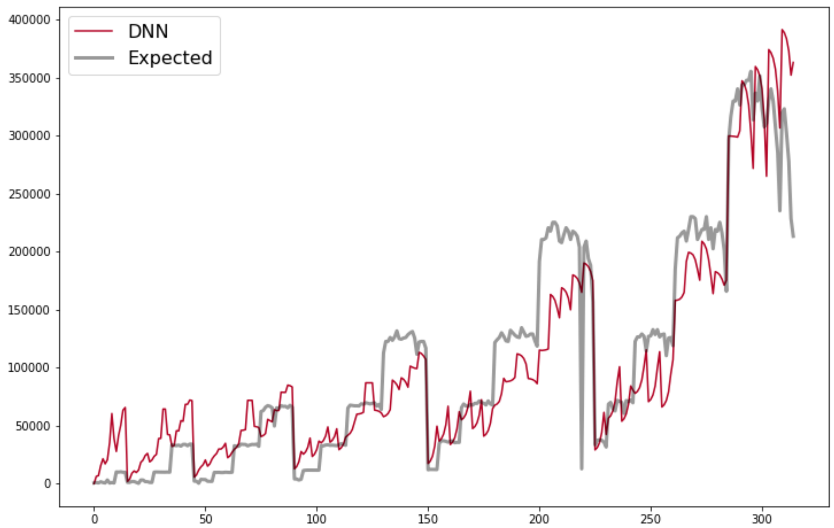

DNN
Expected
350000
300000
250000
200000
150000
100000
50000
0
−50000
0
50
100
150
200
250
300

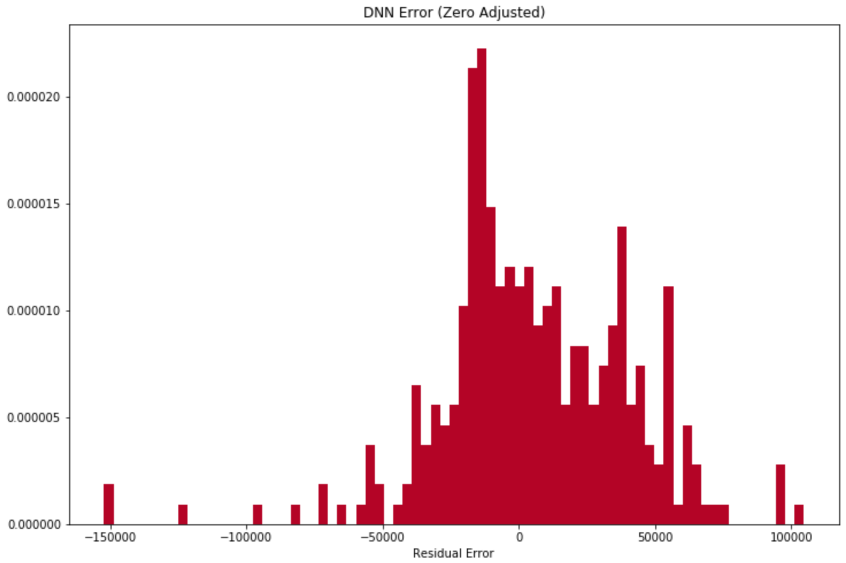
DNN Model, Error



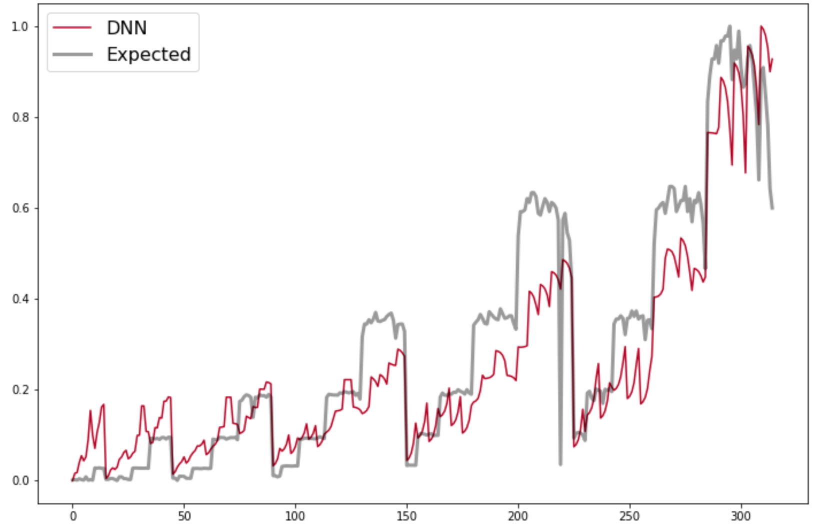


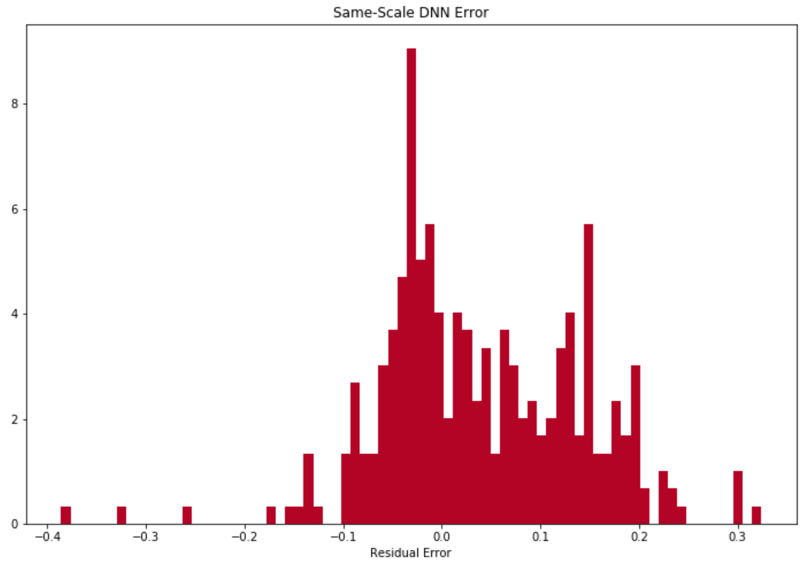
DNN Model, Zero Adjusted

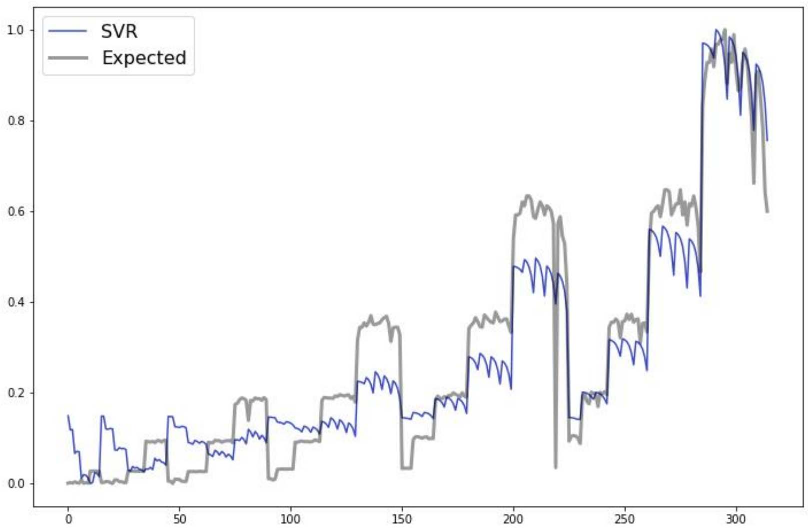


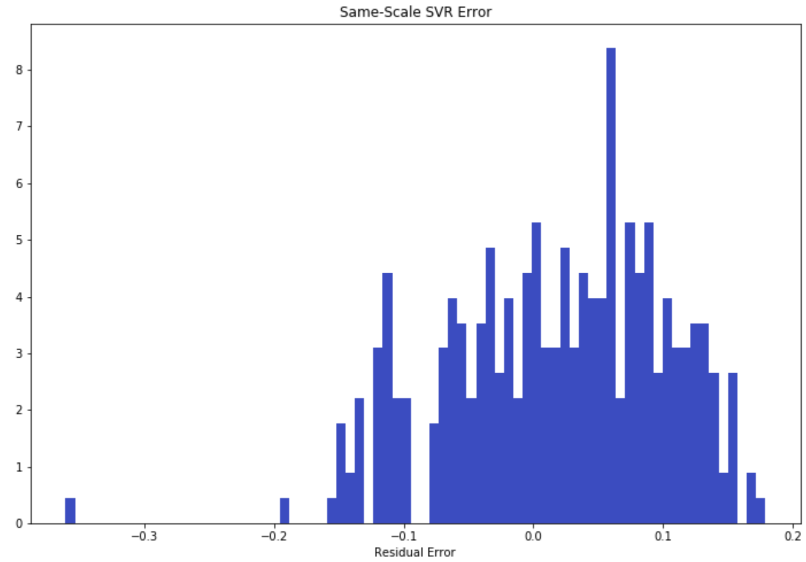


DNN Model, Independent Scale

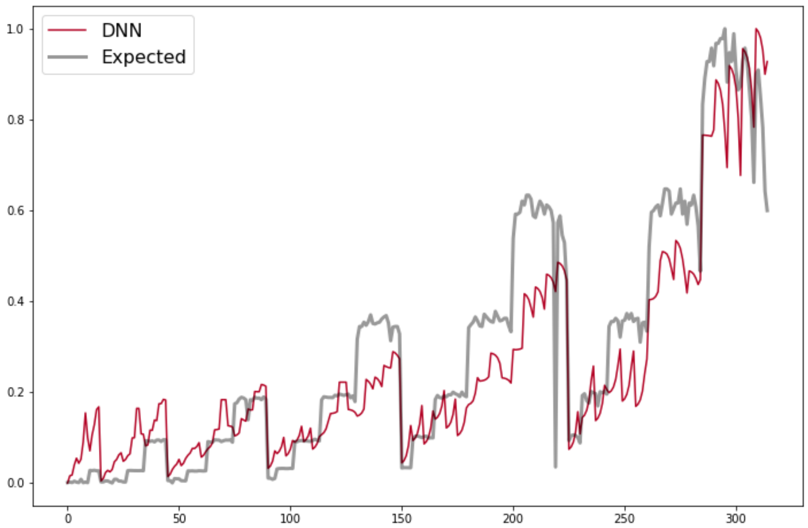


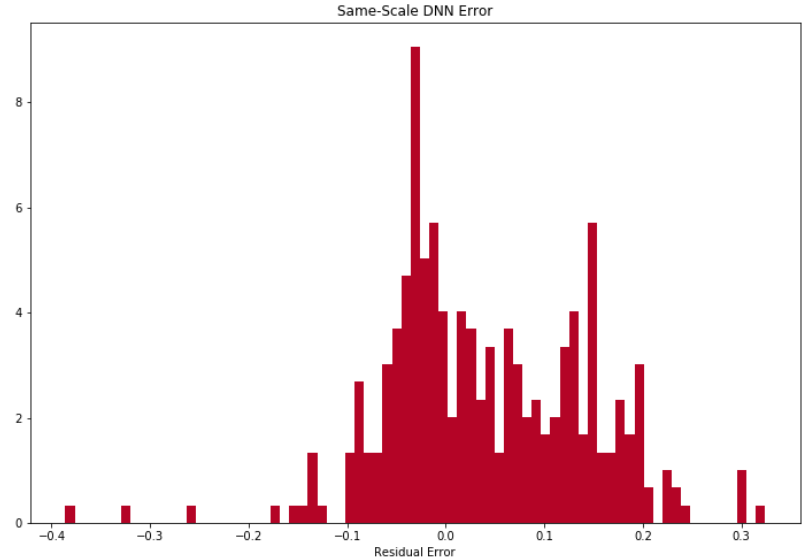






SVR, DNN Comparison





Test 1 Takeaways

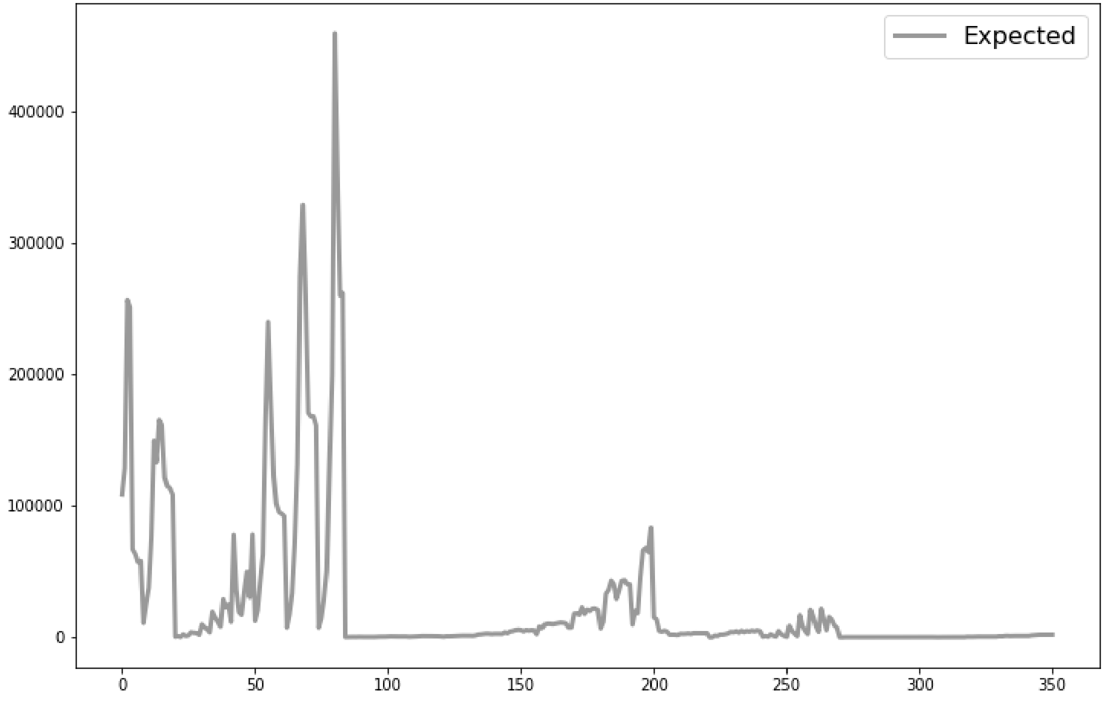
- Scales of models are off, but distributions look good
- DNN better modeled features that were underrepresented in training data
- DNN model had better error distribution, while accuracy needs work it doesn't seem something is *missing* from model
- Change of goals
  - Scales might be from our ignorance in using ML, we'll skip this problem
  - Look to improve distribution, not scale with subsequent tests
  - Develop method to optimize based solely on distribution

Test 2

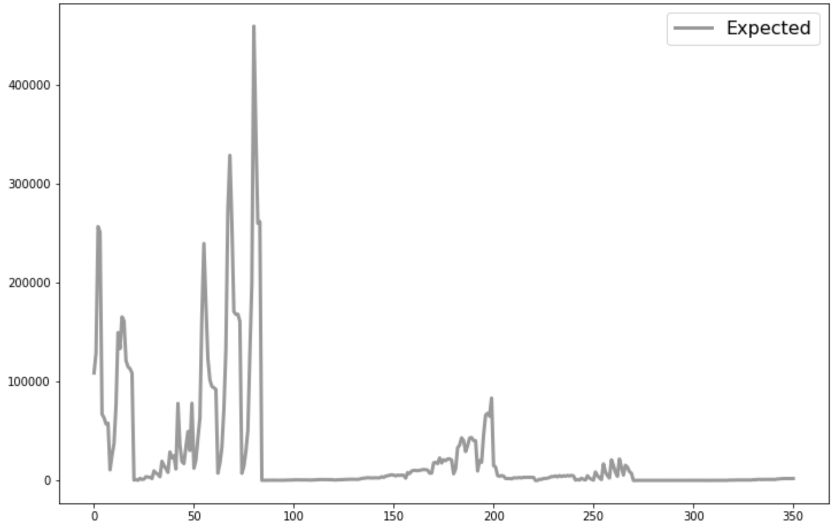
- Training data: All custom benchmark runs
- Testing data: All IOR runs

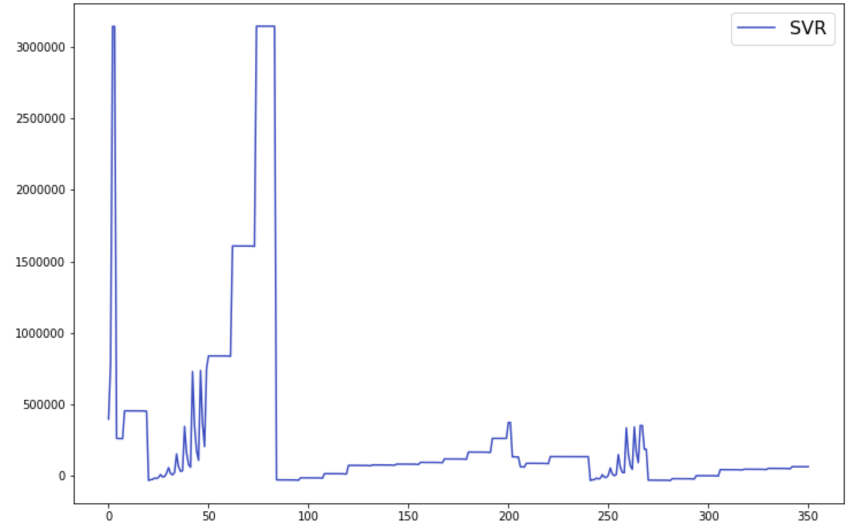
- Purpose of test
  - Custom benchmark is far less “practical” than IOR
  - Proof of concept for general HPC modeling

Expected: IOR Throughput Measurements

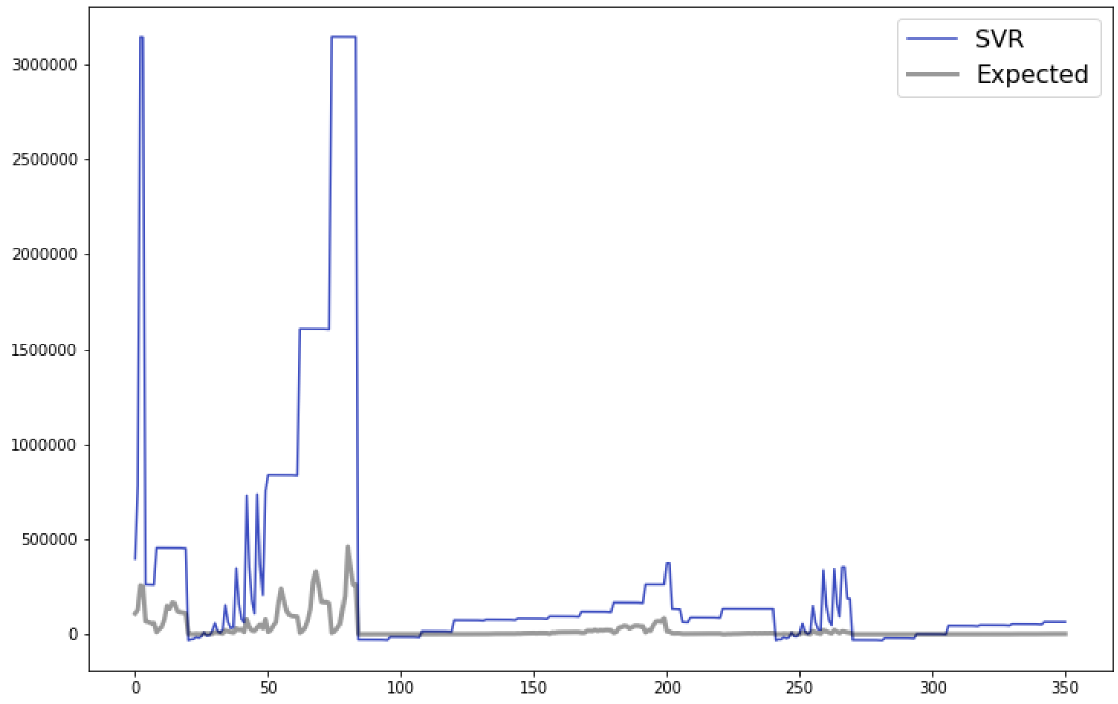


SVR Model


Expected
400000
300000
200000
100000
0
0 50 100 150 200 250 300 350

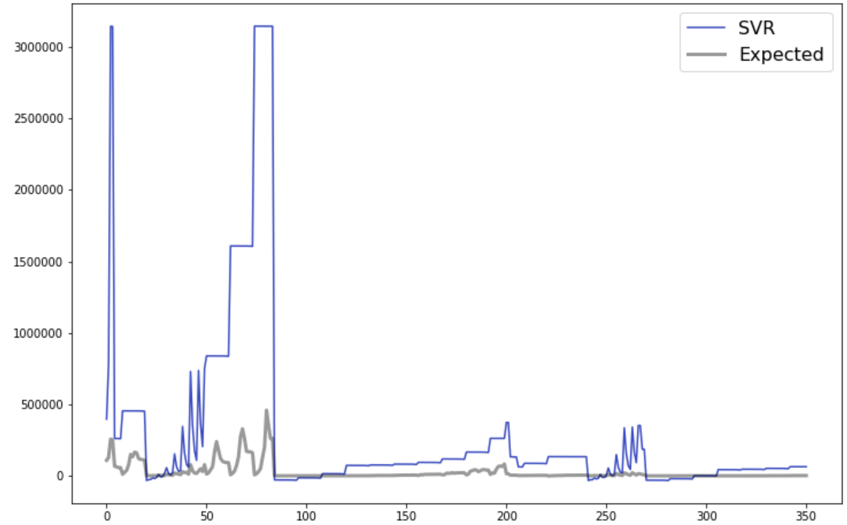

SVR
3000000
2500000
2000000
1500000
1000000
500000
0
0 50 100 150 200 250 300 350

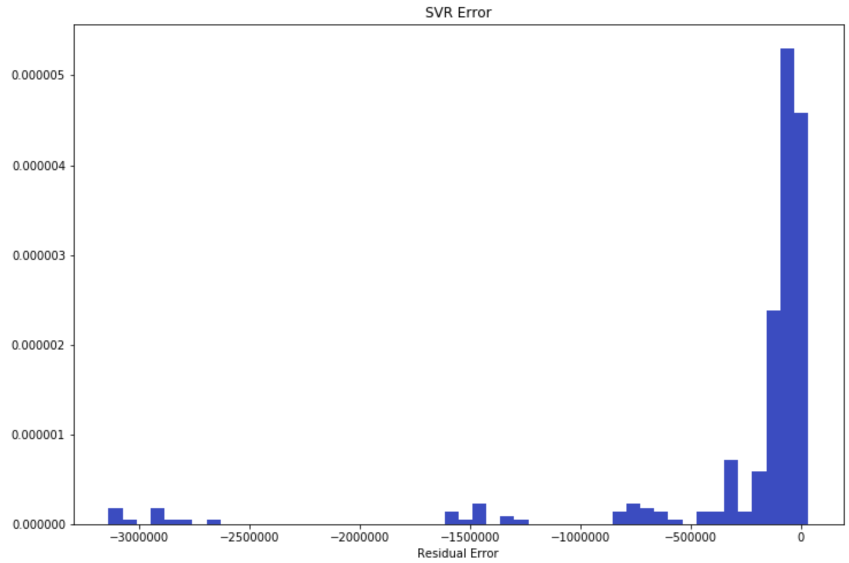
SVR Model



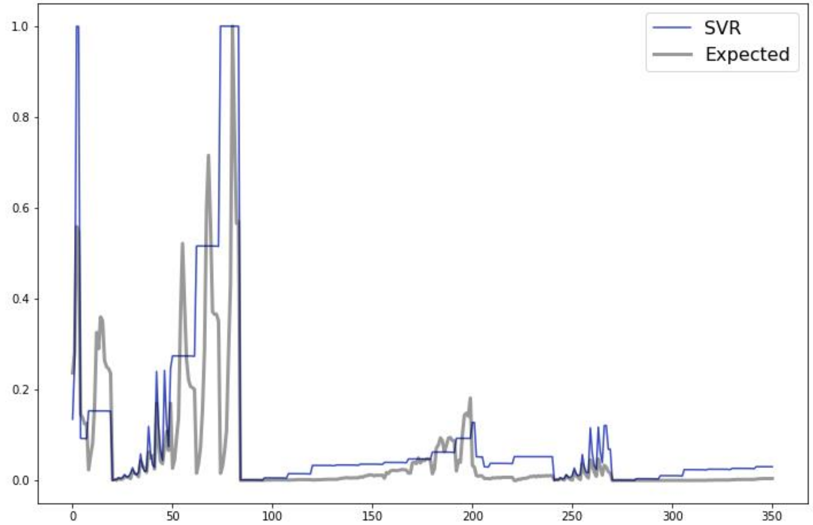
SVR
Expected
3000000
2500000
2000000
1500000
1000000
500000
0
0
50
100
150
200
250
300
350

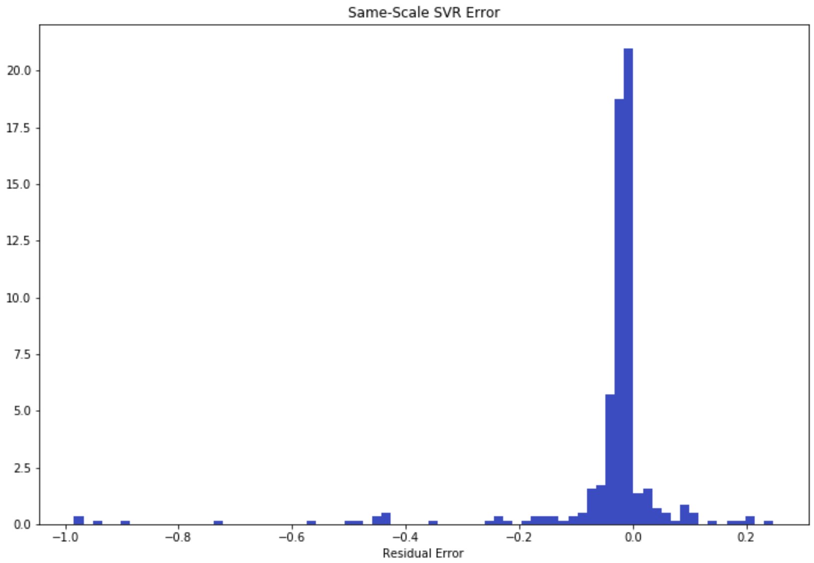
SVR Model, Error



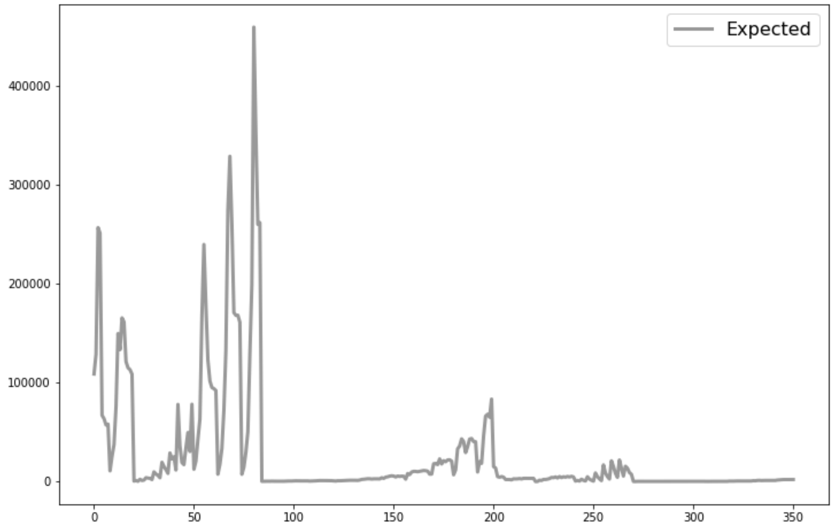


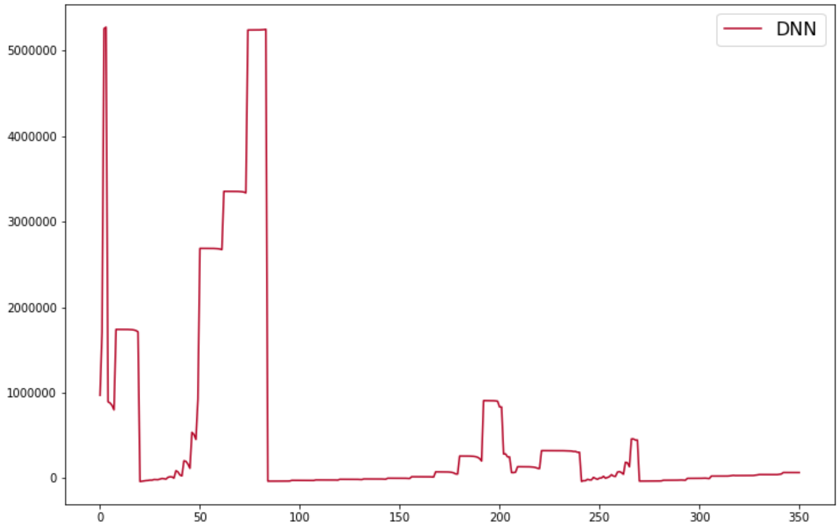
SVR Model, Independent Scale



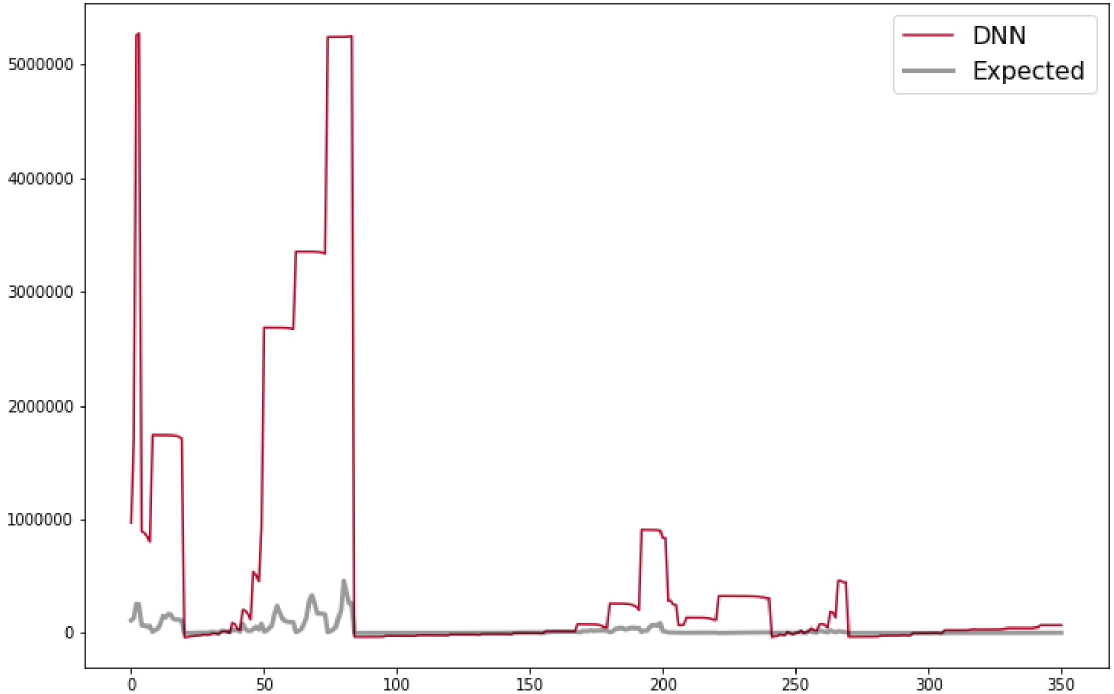


DNN Model

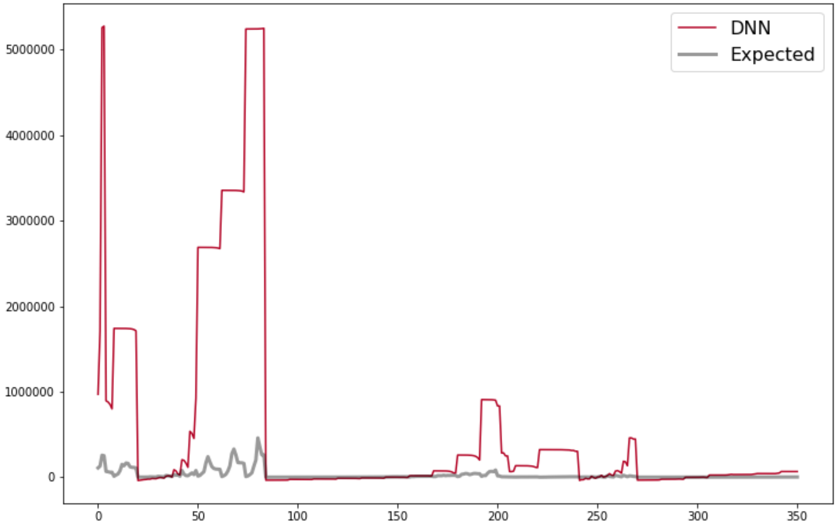

Expected
400000
300000
200000
100000
0
0
50
100
150
200
250
300
350

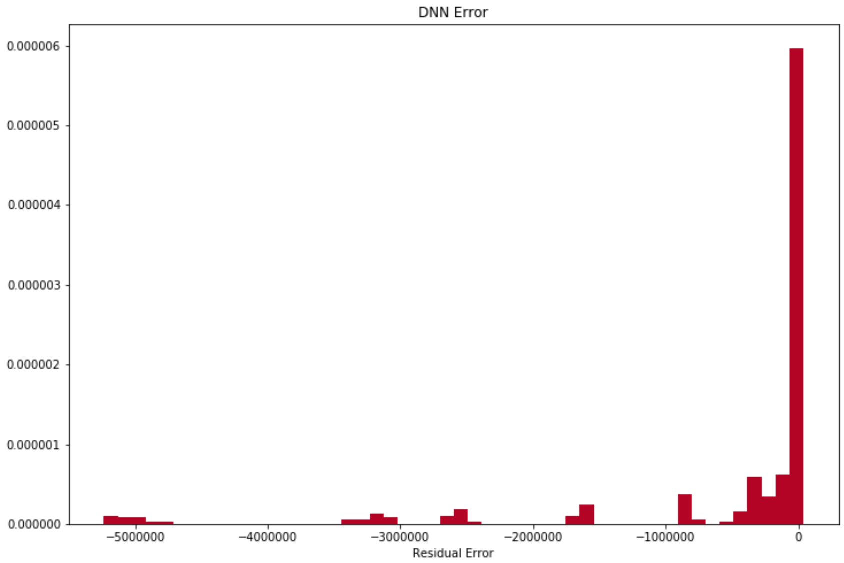

DNN
5000000
4000000
3000000
2000000
1000000
0
0
50
100
150
200
250
300
350

DNN Model

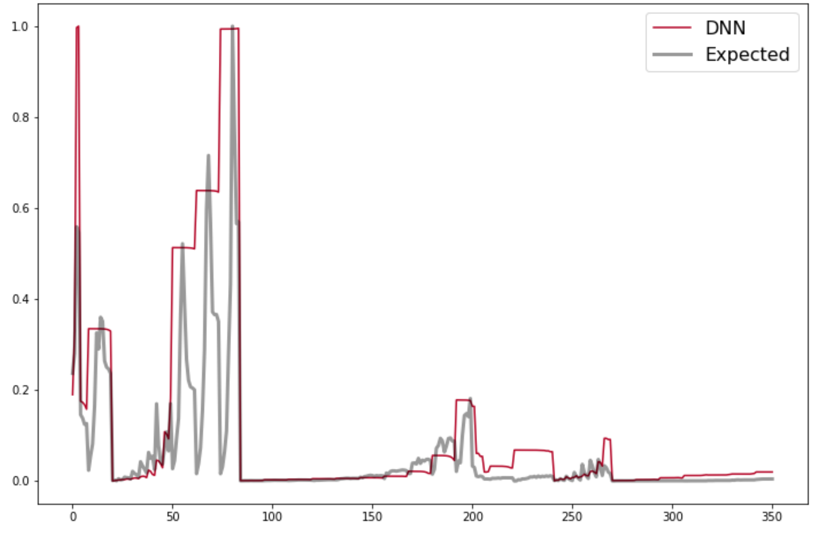

DNN
Expected
5000000
4000000
3000000
2000000
1000000
0
0
50
100
150
200
250
300
350

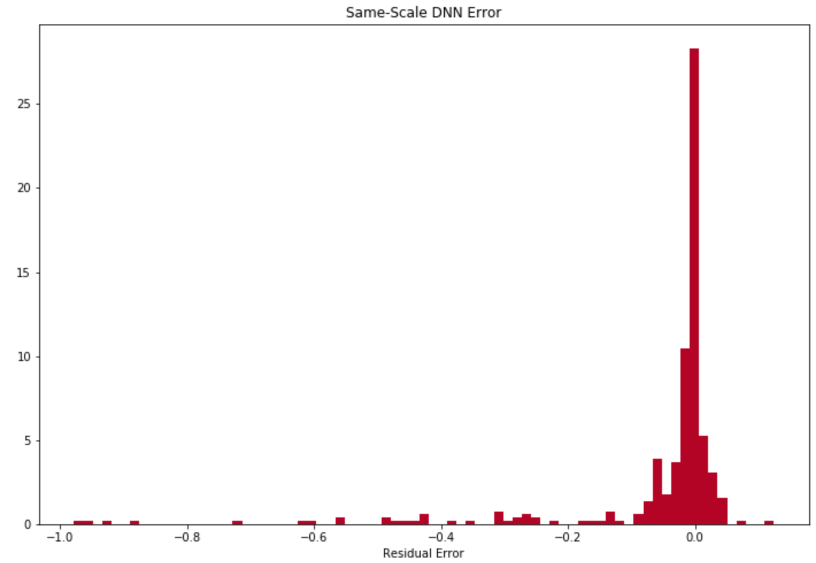
DNN Model, Error

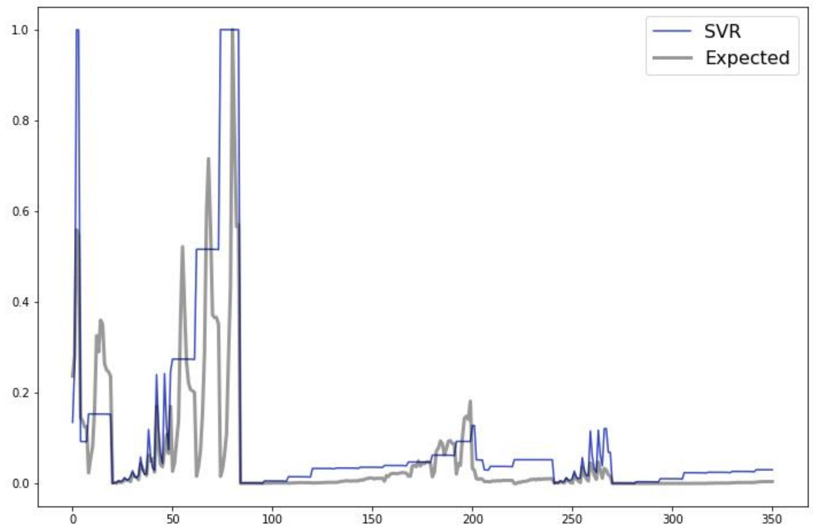


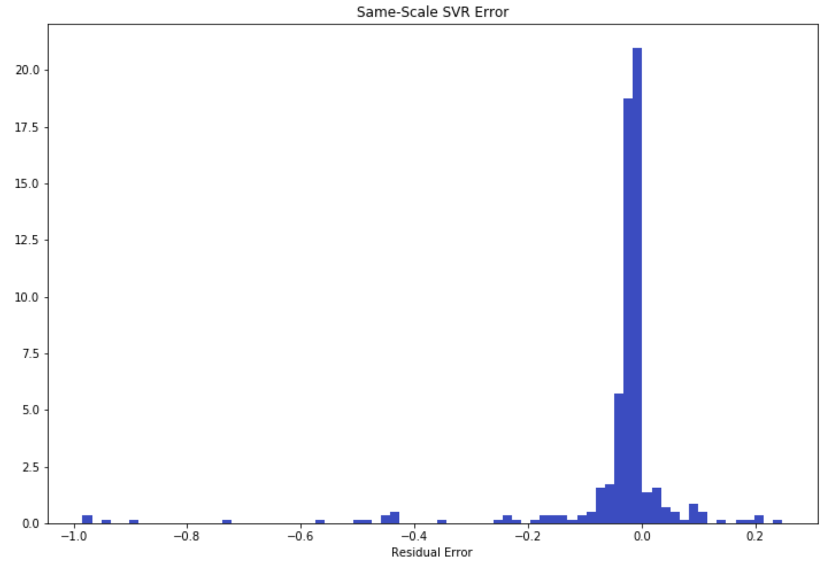


DNN Model, Independent Scale

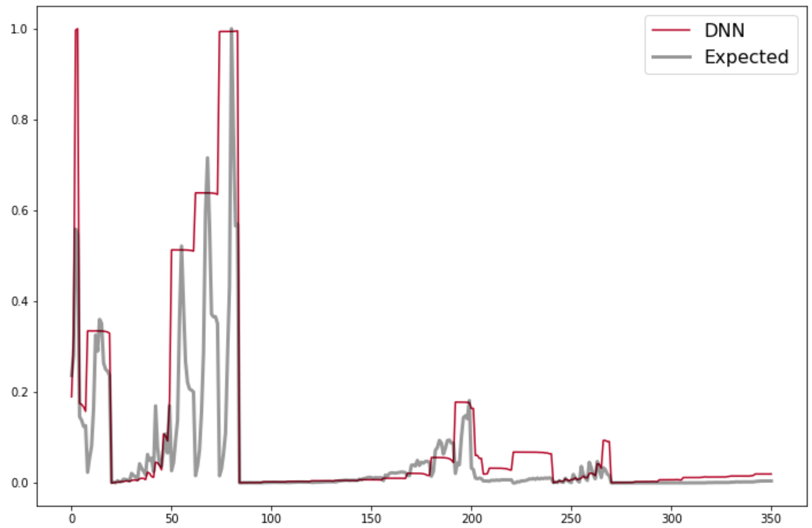


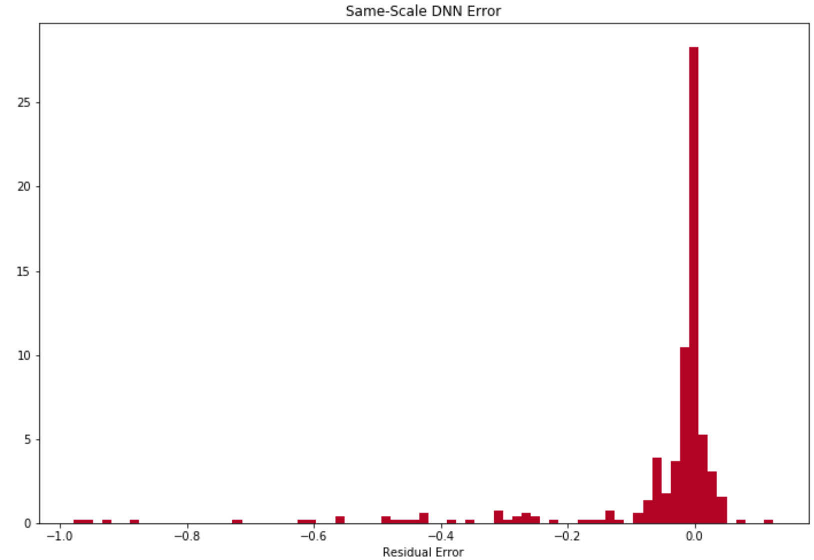






SVR, DNN Comparison





OPTIMIZATION UTILITY

About the Optimization Utility

- Meant to provide insight of actual simulation *relative* to its current I/O model
  - Avoids scale
  - Our stranger parameters may be derived from basic application specification
- Input: nodes, PPN, number of files, I/O size, cnodes, cppn
  - Nodes, PPN refer to how simulation is running, while
  - Cnodes, cppn refer to subset of above actually participating in I/O
  - I/O size of a single operation, think writing at end of simulation loop step
- Iterates through all configurations, predicting throughput from loaded model Shows relative expected performance of current configuration and other I/O patters (FPP, shared file, and file per node)
- Outputs configuration predicted as optimal
- Calculates efficiency (in terms of node hours) and provides above in that context

Utility Evaluation – The Problem

- Input from personal experience working with BW science team
- Originally, code was FPP which was leading to difficulties for postprocessing
- After transition to a shared file, performance tanked
- I/O parameters
  - 512 nodes, 8192 cores, single file, 5GB write per iteration

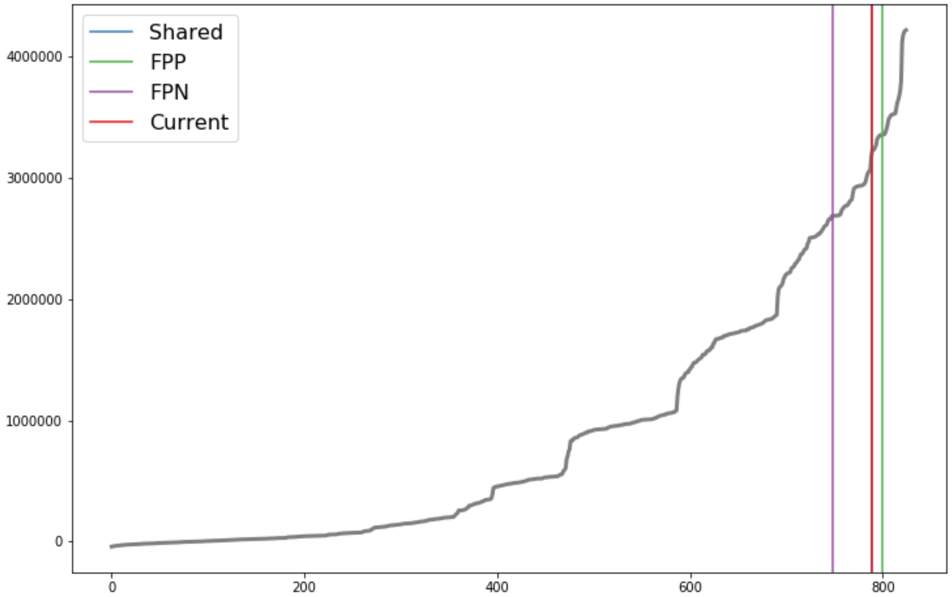
The Problem (cont.)

- Performance issue was due to fact that the 5G weren’t written at once
  - There were 20 variables in the simulation, each written separately
  - Each core was only writing 32K
- Purpose of this test
  - Rare case of such small I/O that typical patterns result in poor performance and efficiency
  - Expect suggestion for how to aggregate

Test 2 DNN Model (Aggregation Training)

Throughput Predictions:

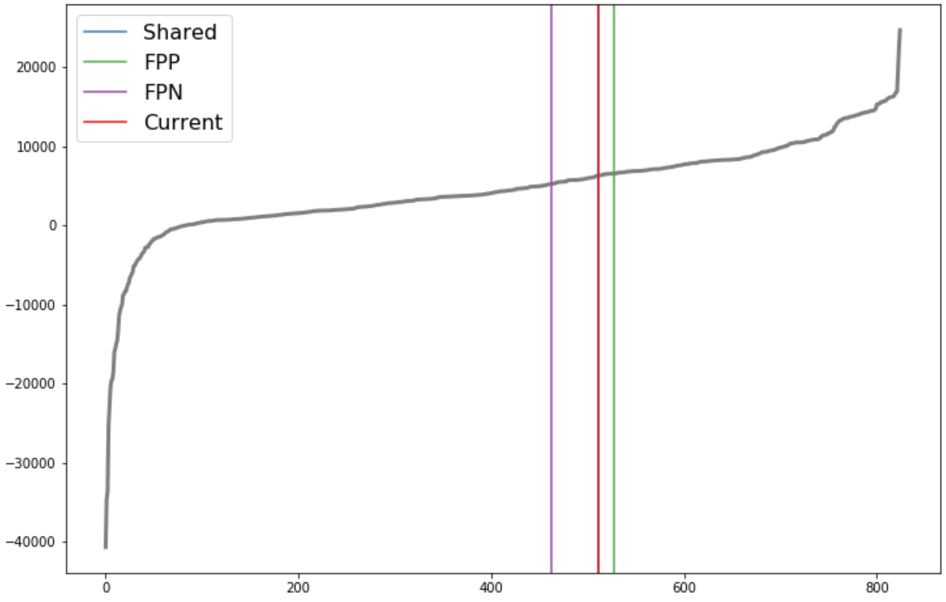
- Suggest it is unlikely to improve by much
- Predicted optimal configuration:
  - 4 files per node
  - 64K stripes



Test 2 DNN Model (Aggregation Training)

Efficiency Predictions:

- Predicted optimal configuration:
  - 1 node writes to
  - Single shared file
  - From 16 cores



Test 2 Optimization

- Seems incorrect
  - For such small I/O, very parallel solutions are predicted to have too high of a throughput and efficiency
  - Expect diminishing returns on adding files/nodes, but even for this size an improvement can be made over single file on a single node
- Does however suggest some aggregation based on throughput prediction

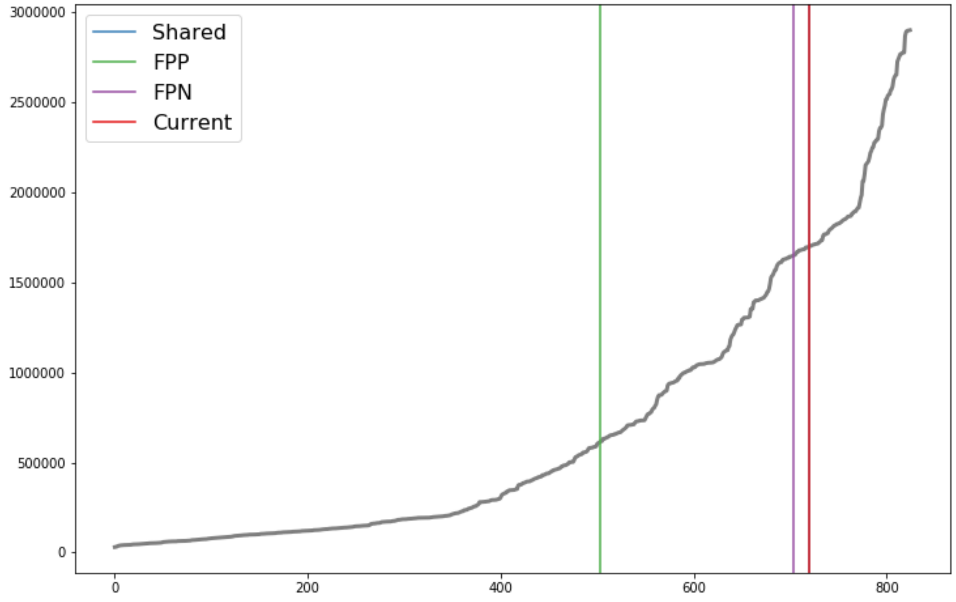
Bringing it all Together

- Final training: all benchmark data (custom + IOR)
- IOR benchmark represents common, practical I/O configurations
- Updated training with IOR may help model common sense I/O considerations

Final DNN Model (Aggregation/IOR Training)

Throughput Predictions:

- Now show FPP as poorer performance
- File per node better throughput as expected



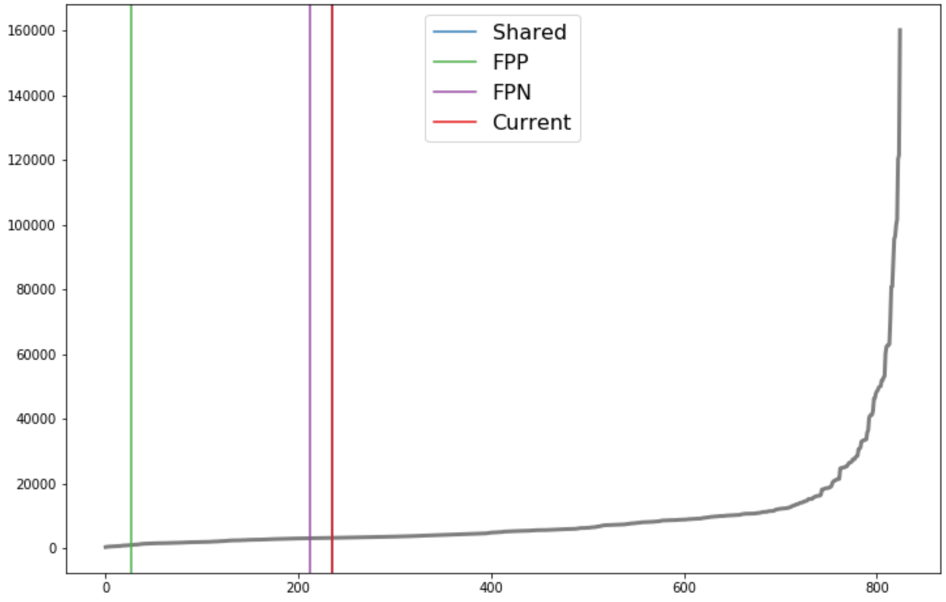
Predicted Optimal Configuration

- 256 nodes, 16 PPN
- 512 separate files
- Complex aggregation
  - Each node writes to eight files
  - Each file is written to by 4 separate nodes
- Would be a difficult implementation, but not dissimilar from other highly tuned applications
- Better representation of diminishing returns for over parallelizing small I/O
- Points out “trick”
  - Each node hits multiple OSTs and multiple nodes hit each OST
  - This is likely to improve injection bandwidth as well as better utilize I/O server bandwidth

Final DNN Model (Aggregation/IOR Training)

Efficiency Predictions:

- Aligned with expectations
  - Highly parallel is bad
  - For small I/O, most are not efficient
- Predicted optimal configuration:
  - 1 node writes to
  - Eight files
  - From 16 cores



The End

- Scales need work, but
- ML seems viable for I/O modeling
- Distributions are valuable for optimization purposes
- ML leads to practical frameworks
  - Training done locally, based on data generated on HPC machines
  - Improved models can be dropped in without changing utility

Questions?

- Now? Ask Away.
- Later? sisneros@Illinois.edu