UC Santa Cruz
UC Santa Cruz Electronic Theses and Dissertations

Title
Dynamic performance enhancement of scientific networks and systems

Permalink
https://escholarship.org/uc/item/57z5f9dq

Author
Bel, Oceane

Publication Date
2020

Copyright Information
This work is made available under the terms of a Creative Commons Attribution-ShareAlike License, availalbe at https://creativecommons.org/licenses/by-sa/4.0/

Peer reviewed|Thesis/dissertation

eScholarship.org
Powered by the California Digital Library
University of California

UNIVERSITY OF CALIFORNIA
SANTA CRUZ

DYNAMIC PERFORMANCE ENHANCEMENT OF SCIENTIFIC NETWORKS AND SYSTEMS

A dissertation submitted in partial satisfaction of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

by

Oceane Mireille Sandra Bel

December 2020

The Dissertation of Oceane Mireille Sandra Bel is approved:

Professor Darrell Long, Chair

Professor Faisal Nawab

Professor Maria Spiropulu

Doctor Nathan Tallent

Quentin Williams
Acting Vice Provost and Dean of Graduate Studies

Copyright © by

Oceane Mireille Sandra Bel

2020

Table of Contents

List of Symbols vi

List of Figures vii

List of Tables ix

Abstract xi

Dedication xiv

Acknowledgments xv

1 Introduction 1
- 1.1 **Optimizing system modeling with feature selection** . . . 3
- 1.2 **Applying our modeling techniques to data layouts** . . . 4
- 1.3 **Identifying modeling techniques for network routing** . . . 5
- 1.4 **Our contributions** . . . 6

2 Motivation and background 8
- 2.1 The problem with data layouts . . . 8
- 2.2 Reinforcement learning versus supervised learning . . . 10
- 2.3 Optimizing system modeling via feature selection (WinnowML) . . . 11
  - 2.3.1 Dimensionality reduction and feature selection . . . 12
  - 2.3.2 Feature ranking . . . 15
  - 2.3.3 Combined algorithms for effective feature reduction . . . 17
- 2.4 Optimizing data layouts . . . 18
  - 2.4.1 Data layout strategies . . . 18
  - 2.4.2 Moving the code to the data . . . 19
- 2.5 Dynamic route selection . . . 20
  - 2.5.1 The LHC program performance drawbacks . . . 21
  - 2.5.2 Optimizing computer network throughput using models . . . 21
  - 2.5.3 The TAEP Project . . . 21

3 Lowering training error and overhead through feature selection 24
- 3.1 WinnowML Design . . . 25
  - 3.1.1 Input data . . . 27

3.1.2 Feature ranking . . . 27
3.1.3 Feature group creation . . . 28
3.1.4 Measuring a group's accuracy . . . 31
3.1.5 Ranking groups of features . . . 32
3.1.6 Manually setting importance . . . 34
3.2 Experimental setup . . . 35
3.2.1 Dataset . . . 36
3.2.2 Experiments . . . 38
3.3 Results . . . 39
3.4 Conclusion . . . 46

4 Applying modeling techniques to data layouts 47
4.1 Motivating workloads . . . 48
4.2 Geomancy Design . . . 49
4.2.1 Architecture . . . 50
4.2.2 Online Learning . . . 52
4.2.3 Modeling target filesystem throughput . . . 54
4.2.4 Discovering Features . . . 55
4.2.5 Smoothing/Normalizing Features . . . 58
4.2.6 Predicting effects of file remapping . . . 59
4.2.7 Hyperparameter Tuning . . . 61
4.2.8 Checking Actions . . . 66
4.3 Experimental Setup . . . 67
4.3.1 Experiment 1: performance improvements . . . 67
4.3.2 Experiment 2: drawbacks of placing all data on a single storage point . . . 72
4.3.3 Experiment 3: impact on other workloads . . . 73
4.4 Evaluation . . . 73
4.5 Overhead Study . . . 76
4.6 Related work . . . 77
4.6.1 Static approaches . . . 78
4.6.2 Applying dynamic solutions to tuning system performance . . . 79
4.6.3 Automated data placement techniques . . . 79
4.7 Conclusion . . . 80

5 Identifying modeling techniques to improve network routing at the switch level 81
5.1 Design . . . 82
5.1.1 Training data . . . 83
5.1.2 Prediction Engine . . . 85
5.1.3 Port selection . . . 86
5.1.4 Site-level coordination . . . 87
5.1.5 Security challenges . . . 87
5.2 Model and hyperparameter search . . . 88
5.2.1 Selecting the Neural Network Model . . . 88
5.2.2 Structured and unstructured data flows . . . 90
5.2.3 Neural network model and heuristic selection . . . 91
5.2.4 Hyperparameter search, determining M, N and O . . . 92
5.3 Experimentation: contention reduction on a simulated network . . . 97
5.4 Results . . . 103
5.4.1 Study 1 results . . . 103

5.4.2 Study 2 results 106
5.4.3 Study 3 results 106
5.4.4 Study 4 results 108
5.5 Related Work 110
5.6 Conclusion 111

6 Conclusion and Future Work **112**
6.1 Future work in optimizing system modeling via feature selection 113
6.2 Future work in applying modeling techniques to data layouts 113
6.3 Future work in identifying modeling techniques for network routing 114

Bibliography **115**

List of Symbols

$r(x, y)$	Correlation coefficient between features x and y
μ	Mean
ω	correlation weight when calculating ranking of features
σ	Standard deviation
η	Variable-share penalizing rate
θ	Pool's influence on weight update
α	L1 learning rate
β	Limit between bottlenecked and underused ports

List of Figures

2.1 Placement of a Tofino switch in Caltech Tier 2 network. All links are bi-directional. . . . 22

3.1 WinnowML system design. Blue and gray arrows represent the data flowing into and out of the neural network. Red solid arrows represent the dataflow of the log and target ID given by the user through the data processing section of WinnowML. The dotted arrows represents the data given by the user. 26

4.1 Geomancy's architecture. The control agents and the monitoring agents are located on each available location on the target system. Thick arrows represent the communication between the target system and Geomancy. Thin arrows represent the communication between the agents of Geomancy. Red arrows represent the data flow during the decision process. Blue arrows represent the data flow during the performance data collection. . . . 51

4.2 Neural network architecture. Z is the number of performance metrics used to describe an access. In the BELLE II experiment, we used 6 performance metrics. In the experiment provided by CERN, we used 13 performance metrics. Discussion of the layer sizes can be found in Evaluation. 53

4.3 Correlation between the raw access features found in the EOS logs and the throughput. We choose features (orange) that are commonly found in scientific systems that also happen to be positively correlated. 56

4.4 A visual representation of the storage devices on the BlueSky system. This system is shared among many other users. 68

4.5 Geomancy's performance compared to dynamic and static solutions on the live system. Gray dotted lines represent data movements done by Geomancy dynamic. Size of the data that Geomancy moves ranges from 583 KB to 1.1 GB. We focused on showing the number of files moved by Geomancy to show that it does not move a significant amount of data at once, and it only moves the data when needed. 74

4.6 If a new workload is started in parallel to running workloads, Geomancy is able to actively learn how to place data according to this new change. Doing so, Geomancy is able to respond to the changes and attempt to push performance back to what it once was. . . 78

5.1 High level design of the Diolkos system and target system. The outlined arrows correspond to data exchanged between the network of switches and Diolkos. The thick dotted arrow is the flow taken by the new calculated port by the model. The thin arrows are data paths between the components of Diolkos. The thin dotted arrow is the path taken by data during the training round. Undotted thick and thin arrows are the paths taken by data collected from the target system. 83

5.2 We define M, N and O to identify how much training data is used, how far into the future we predict, and how many timesteps should be predicted. P represents the port number on the switch. 85
5.3 Auto-correlation for ports 0, 4, 8 and 12, values close to 0 and in the blue region indicate that data at that lag has no correlation. 90
5.4 Average median and lower to upper quartile values of the absolute error across all the models as N increases from 1 to 10, M=5 and O=1 for port 8 (structured data). As N increases, EWMA does not change since the amount of data for the average stays consistent. In contrast, the Dense model focuses too much on the additional variations present in the data, hence as N increases the noise present in the data causes error to increase for certain values of N. 93
5.5 Average median and lower to upper quartile values of the absolute error across all selected models as M increases from 3 to 7, N=1 and O=1 for port 8 (structured data). As M increases, EWMA's median of the absolute error increases exponentially and the Dense model's median of the absolute error decreases slightly. 95
5.6 *ID* is the number of the Tofino. *22* and *11* are IDs for the available routes. Incoming links into switch 1 (s1) are 100Mbit links. The rest of the links are all 40Mbit links. We are using a mesh network just to represent the a single path on the Caltech Tier 2, which uses a star topology, system within the Caltech campus. 97
5.7 The Tofino switches are similar to that described in Figure 5.6. Each link in the network are 100 Mbit links. The thicker solid links are the outer paths numbered 00 and the thinner solid links are the inner path numbered 11. The dotted link is the link between the hosts and switches 99
5.8 Average throughput per port over each switch, depending on the modeling technique used when we added an additional Tofino switch between Tofino 5 and hosts 2 (H2) and 3 (H3). 104
5.9 When Diolkos controls all the switches, including the non-Tofino switches, all approaches underutilized half of Tofino 5. This reduces the overall throughput of the network. However, the Dense model allowed the network to use the rest of the path. 105
5.10 A ring topology still demonstrates the Dense model's superiority over other modeling techniques, demonstrating that Diolkos is network topology agnostic. 107
5.11 Increasing the number of flows from 10 to 100 and letting Diolkos only control the Tofino switches demonstrates its ability to react to surges in demand. 108
5.12 Rises and falls in demand does not cause the Dense model to bottleneck in performance. Rather, it responds to it to maintain high average throughput. 109

List of Tables

3.1 Accuracy of WinnowML vs model used. Each model is described in Table 3.2. Each row corresponds to a round of training with new data. 36
3.2 Model description 38
3.3 Number of features selected for each round versus the feature selection technique used . 40
3.4 Feature ID selected for each round versus the feature selection technique used 42
3.5 Feature ID selected for each round versus the data used for the prediction 44
3.6 Mean absolute error (%) of selected feature subset for each round versus the feature selection technique. Error reported is standard deviation of mean absolute error to show error fluctuation. When the standard deviation overshoots the average, this shows that the model failed to capture the variation of the target value 45

4.1 Model architectures 61
4.2 Model comparisons on predicting performance over all mounts. The time was measured using the time package in python. We started it before the model started training and ended it when the model finished training. We repeated the process for the prediction time. Additionally here we represent the standard deviation of each value to show how it fluctuates over time. 63
4.3 Prediction accuracy of model 1 on each individual BlueSky's storage points. Model 1 has no worse than 43.15% mean absolute error (23.62 + 19.53 max deviation from mean). 65
4.4 Performance and utilization of storage points available to Geomancy. In this table, we demonstrate the variation of the throughput at every mount and their respective usage observed by Geomancy. In some cases, the standard deviation, showing the variation of the performance, does surpass the mean, indicating that the performance varies significantly over time. Additionally, it shows that in many cases the mounts are contended, and when they become less contended the throughput shoots up. This creates an extreme difference in performance. 76

5.1 μ_{AE} for port 8 for the 7 models with the lowest prediction error out of the 19 tested. Our selected model is in bold. EWMA and Holt-Winter perform similarly, however we have chosen to compare our Dense network against EWMA since EWMA had the highest accuracy. 91
5.2 Beyond values of O=1, neither approach produces predictions that are accurate. Thus, every prediction cycle should produce at most one prediction. 96

5.3 *partial* means that Diolkos will only control the Tofino switches, while *full* means that Diolkos controls all switches. For mesh networks, Diolkos will only control the Tofino switches. In ring toplogies, Diolkos will only partially control the Tofino switches (the side that does not contain a host). When all the switches are controlled, there is a version of Diolkos on each switch, including non-Tofino switches. 102

Abstract

Dynamic performance enhancement of scientific networks and systems

by

Oceane Mireille Sandra Bel

Large distributed storage systems such as High Performance Computing (HPC) systems used by national or international laboratories must deliver performance for demanding scientific workloads that pose challenges from scale, complexity, and dynamism. Ideally, data is placed in locations and network routes are set to optimize performance, but the aforementioned challenges of large scientific systems and networks inhibits adaptive performance tuning on a large scale.

If the data is not placed or routed with the potential future access patterns, the system risks having popular data stored on a slow file system. This can cause a steep drop in performance when access demand shifts to that data. Ideally, determining when and how to move data around in anticipation of future demand spikes can prevent steep drops in performance, and we refer to these drops as performance bottlenecks. We can leverage the fact that workloads shift through time, creating access patterns that can be used to forecast demand. Past accesses can be used to reveal important information when determining future accesses.

An existing solution for determining where to move the data is to use access logs to create system performance models that can predict how the system reacts to fluctuations in demand. The drawback is that the search space of how to build an effective model is massive. The more there are locations where the data can be stored to larger the search space becomes. Additionally, applying the model to a system can potentially negatively impact any performance benefits of dynamically moving the data around the system. If a model is not set up correctly, the added overhead of using the model on the system may overshadow the benefit of applying the model on a target system. Ideally, the models

created by the system engineers place and route the data to optimize performance, but workloads shift can be hard to predict using traditional methods such as heuristics.

This dissertation applies and enhances online learning to predict and accelerate the performance of data movements and accesses in large scale storage systems. We focus on large scientific systems and networks like the CERN EOS, Pacific Northwest National Laboratory's (PNNL) BlueSky system and the Caltech Tier 2 system. These systems allow for us to stress test our models with real data. Additionally it allows us to collect information about how accesses are distributed and executed on real systems. From the collected information we are able to create models based on real workloads. We are also able to reduce the training overhead of our models by determining which features may not bring added information to a model during training, and removing them. We developed a methodology to select features over time that is resilient to data collection noise added by new data collected from the system.

Access logs created by system engineers often track hundreds of metrics describing its daily operation, and we must sift through these metrics to discover the most relevant metrics for the modeling task at hand. To explore different feature selection techniques, we have developed WinnowML, a system of automatically determining the most relevant feature subset for a specific modeling methodology when modeling system performance over time. We use WinnowML to determine what combination of existing techniques allow us to get a stable selected subset of features which will not vary significantly over time while keeping a low prediction error. Using the created list of features, system analysts can determine what features should be used when modeling an aspect of their system over time. Using WinnowML lowered the resulting mean absolute error by 13.6% on average compared to the closest performing approach such as L1-regularization or Principal Component Analysis (PCA).

To optimize the placement of data, we developed Geomancy, a tool that models the placement of data within a distributed storage system and reacts to drops in performance. Additionally to optimize the network routing decisions, we developed Diolkos, a tool that dynamically reroutes data flow

in response to drops in performance. Using a combination of machine learning techniques suitable for temporal modeling, Geomancy and Diolkos determines when and where a bottleneck may happen due to changing workloads and suggests changes in the layout and routing decisions to mitigate or prevent them.

Using WinnowML to determine which features to use when training our Geomancy tool, the predicted data layouts of Geomancy offered benefits for storage systems such as avoiding potential bottlenecks and increasing overall I/O throughput from 11% to 30%. It managed to free up resources that other workloads running concurrently could use. We then moved on to tackle the data transfer overhead in scientific networks. There exist several techniques that reroute data within scientific networks while improving network performance, leveraging latent parallel data transfer capabilities. The issue with these techniques is that they require a central authority, which may add computational and network overhead to the system. We propose a decentralized rerouting technique that exists at the switch level. When we applied Diolkos to the Caltech Tier 2 network, we found that our most accurate model, a dense model with one hidden layer trained using all the ports, increases throughput of switches up to 49% compared to the best performing heuristic approach, exponentially weighted moving average, and up to 28% when using a traditional controller.

To my parents who were always there for long talks, my husband who made my long nights a bit more bearable and everyone who supported me through this journey, I want to say thank you. I could not have made it through this journey without you.

Acknowledgments

I would like to thank my committee members, all my lab colleagues, collaborating professors, and Caltech's high energy physics department for their help in reviewing my papers. I would also like to thank our collaborators at Pacific Northwest National Laboratory and CERN for providing usage of their systems and workloads. We are grateful for funding support from the U.S. Department of Energy's (DOE) Office of Advanced Scientific Computing Research as part of "Integrated End-to-end Performance Prediction and Diagnosis." I would like to thank Caltech's High Energy Physics department for collaborating with us on the networking section of this thesis. I also thank NVIDIA Corporation for their donation of a TITAN Xp GPU which was used as part of the development of the projects described in this thesis. This research was supported by the NSF under grant IIP-1266400 and by the industrial members of the NSF IUCRC Center for Research in Storage Systems. Part of this work was conducted at "*iBanks*" the AI GPU cluster at Caltech. We acknowledge NVIDIA, SuperMicro and the Kavli Foundation for their support of "*iBanks*" We acknowledge support from Caltech's Intelligent Quantum Networks and Technologies (INQNET) research program and AT&T's Palo Alto Foundry.

Chapter 1

Introduction

High-Performance Computing (HPC) and High Throughput Computing (HTC) systems deliver ever-increasing levels of computing power and storage capacity; however, the full potential of these systems is limited by the inflexibility of data layouts to rapidly changing demands. A shift in demand can cause a system's throughput and latency to suffer, as workloads access data from contended regions of the system. In a shared environment, computers may encounter unforeseen changes in performance. Network contention, faulty hardware, or shifting workloads can reduce performance and, if not diagnosed and resolved rapidly, can create slowdowns around the system.

Allocating more resources to mitigate bottlenecks does not always resolve contention between workloads [1], and it is not always economically possible to add more system resources. We define bottlenecks in distributed storage systems as any situation that results in reduced performance due to contention. To mitigate contention, system designers implement static or dynamic algorithms that place data based on how recently the files have been used similar to the caching algorithm *Least Recently Used* [2] which placed least recently used files on slower performing locations in the system. However, existing strategies require manual experimentation to compare various configurations of data which is expensive or in some cases infeasible. These algorithms are not sufficient for all workloads because they

do not adapt as workloads change, and they may not be optimal for all workloads.

Other than storage bottlenecks, networks can severely impact the performance of accesses to a system. Scientific networks endure constant heavy traffic at all hours of the day, and the types of workflows seen on these networks do not allow for traditional network enhancement techniques to have full effect. In non-scientific networks, engineers use techniques such as Holt-Winter [3] predict the return time of a packet in a network, and use this information to route packets to destinations with lower return time. Predicting in such a manner requires as much information about the network to be available as possible to produce the best type of predictions, and this often requires a global view of the network.

With a global view of the network, engineers can use any modeling technique (neural network based or not) to make accurate predictions as to where data should do. However, central authorities allow such a view at the cost of scalability: as a network grows the overhead of using these systems increases [4]. Additionally, a central authority risks leaking sensitive information [5], since it needs to store data from the network to determine how to direct the traffic. In scientific networks, these drawbacks are a considerable hurdle towards improving performance. Where there is latent parallelism to be exploited, there is an equal challenge in discovering how to best execute and route parallel data flows.

Another problem with modeling network and storage performance comes from selecting the features to use. Performance tuning systems that rely on continuous retraining of a neural network suffer greatly from retraining overheads. The benefit of such systems is that they can quickly diagnose a performance drawback in the system and prevent it before the performance drop happens. Problematically, the overhead of using all the available training features builds up significantly over time, making them computationally intractable. Therefore, the model must minimize the need for retraining. The performance data used for training may also contain noise from data movements, concurrent accesses, and other sources of entropy which impacts the model's accuracy and training efficiency. If the selected features vary, the model must retrain with the new features, discarding any knowledge gained from past training cycles. Thus, and selection of training features must not change often, and must sufficiently

represent the system being tuned.

The traditional solution has been using larger and deeper networks and letting the neural network select features has been the solution. However, machine learning problems, datasets, and models are becoming larger and relying entirely on the model to extract features is becoming infeasible and prohibitively expensive if scientists don't have the resources to run these larger networks. To remedy the lack of resources, with a sparse feature set, researchers use wrapper methods such as Permutation Feature Importance (PFI) [6] or Self Organizing Maps (SOM) [7] to return a ranked list of features from most to least important. Additionally, people have used dimensionality reduction techniques, such as Principal Component Analysis (PCA) [8] or autoencoders [9] to reduce features before training the neural network. These techniques use mathematical and probabilistic relationships between features to reduce the number of features needed to model a target value accurately. The drawback of PCA, SOM, and PFI is that it does not output the actual features that should be used or creates unrealistic feature groups. Further, when new features or entries are added to the dataset, these techniques may choose different features than what was already suggested. Additionally autoencoders need the user to set a compression ratio, which mayproduce sub par results when determining the final feature group since the user will have to look through a large amount of values for the compression rate and may not pick the value that the most optimized.

1.1 Optimizing system modeling with feature selection

Many existing feature selection techniques select new features when new data is added to a dataset. However, changing the selected features may lead to confusion when the user is trying to determine which feature impacts the target value the most. Recently researchers developed techniques to deal with streaming data, situations when not all the data is available during the initial feature selection, such as online feature selection (OFS) [10] and Feature Selection on Data Streams (FSDS) [11] algorithms.

These algorithms do not tackle regression problems such as modeling system performance through time.

To handle feature selection for regression problems, we have developed WinnowML, a system to winnow down features to a set of features that does not vary significantly when new data becomes present and maintains a low prediction error rate. WinnowML's goal is to determine a way to keep the feature ranking and selection as stable as possible when faced with new data and noise. It automates the feature selection and reduction process given a dataset and intended modeling technique. The system combines existing feature ranking techniques and an iterative approach to return a list of features that maximizes prediction accuracy. An increase in prediction accuracy is represented by a drop in the mean and standard deviation of the absolute error between the predicted and measured values, also called target values. WinnowML not only remains stable in its selection of features, it also kept a 13.6% lower mean absolute error when compared to the feature selection approach that performed the closest to WinnowML as described in section 3.

1.2 Applying our modeling techniques to data layouts

To demonstrate improvements at the storage layer of HPC, we have developed *Geomancy*, a tool that improves system performance by finding efficient data layouts using online learning in real-time. Geomancy use the selected features of WinnowML to model the throughput of accesses in large scientific systems and adapt the data layout in response to change in the performance. It targets systems that serve and process petabytes of data, such as particle collision analysis [12]. Workloads on these systems are commonly spread across multiple storage devices which can lead to storage devices becoming contended over time. If a storage device becomes contended, the delay can be felt across the system. Geomancy analyzes the system's reported performance metrics to build an understanding of how file layouts affect performance, and moves files to more suitable storage points when necessary.

Performance data includes parameters such as average access latencies, remaining storage

space, number of previous reads and writes, restrictions on reads or writes, file types that are read or written, and number of bytes accessed. Using this data, we build a predictive model using artificial neural networks that relates system time, data location, and performance. Geomancy's neural network uses this model to forecast when and where a bottleneck can happen due to changing workloads. Additionally, it preempts future drops in performance by moving data between storage devices. If the model predicts an improved location for a piece of data, Geomancy sends the new location ID to the target system, which moves the data to the new location. We experimentally test our method in a small scale system against algorithmic modeling, and observe an 11% to 30% performance gain in our experiments including moving overhead compared to policies that place data dynamically or statically according to how frequently or recently the data has been used, as seen in Section 4.

1.3 Identifying modeling techniques for network routing

Determining where the data should be placed is only the first step to improving the performance of scientific systems. We must also carefully control how the data flows from one point to another in the network. Since a global controller may not be optimal for large scientific networks, we turn to using a switch level approach. Our selected model will not have any information of the entire network, and must learn to tune it given a limited view of the network's properties.

To measure any predictive technique's effectiveness at the switch level, we implemented a network routing tool called *Diolkos*. It monitors and reroutes network traffic at the switch level to increase the throughput of the network, avoiding contended switches as necessary. Our goal is to compare the accuracy and efficiency of various modeling techniques when improving network performance in a simulated system.

We first present an analysis of various network performance modeling methodologies, analyzing their effectiveness at modeling data flows. Second, we demonstrate the effectiveness of these

methods on a simulated network, since we did not have access to the original switch that we used to collect data for the hyperparameter and model search. The application on a simulated network shows that rerouting data based on network load can reduce network contention. We found that several neural network models produce performance benefits on switches that had data flows with trends, while heuristics produce performance benefits on switches with data flows that had chaotic trends. Overall, we observe a throughput increase of up to 18% at the switch level when we use a Dense model over the Exponentially Weighted Moving Average (EWMA) heuristic to select output ports.

1.4 Our contributions

Through this dissertation we have developed a system that consistently chooses similar features even when new data for the features become available. Additionally our system is able to maintain a high accuracy even when compared to commonly used techniques. Models used for time series prediction would benefit from such a system since it allows for reduced training overhead and wasted training iterations on useless features.

Using the features selected by our WinnowML system, we were able to validate that the features used to model the performance of data placements were the best ones to use. With an ideal set of features, we developed a system that updates the data layout of a system in response to performance drops. We also demonstrated that there is a need to update the data layout of a system to prevent performance bottlenecks from happening.

After determining a technique for placing data, we turned to controlling how the data is moved in a network, specifically scientific networks such as the Caltech Tier 2 network that links multiple research sites in the US to the CERN LHC system in Switzerland. Because we were dealing with such a large network that has consistent traffic flowing through it, a global controller would add unnecessary overhead of the network because of the added data flow. We developed a system that works at the

switch level, rerouting data in a scientific network without the need for a global view on the network. We demonstrated that the predictions of our model allowed the network to outperform similar networks using traditional controllers, and predictions from other predictive models and heuristics

Chapter 2

Motivation and background

2.1 The problem with data layouts

In High Performance Computing systems, there is a mix of storage devices and computation nodes, and storage hardware may not necessarily be local to computation. Additionally, each storage hardware may not have the same performance. If certain data is suddenly demanded and is located on slow storage hardware, all the workloads demanding that data may stall until the data can be either moved to faster hardware or the demand times out. Further, since these systems are often utilized by researchers across the world, accesses do not follow traditional growth and decline patterns. Hence, moving data in a system risks creating additional bottlenecks in the system if the data movement is not done with regards to future demand.

Commonly, researchers place data with regards to how often it is accessed. Algorithms such as Least Recently used (LRU), Most Recently Used (MRU) [2], and Least Frequently Used (LFU) [13] can be used to monitor changes in accesses at the file level. When accesses change, files that are no longer accessed may be moved to a different (likely slower, higher volume) storage mount to make room for data that is rising in demand. Some researchers have attempted to merge LRU and LFU to create a

more comprehensive cache replacement algorithm [14], but these approaches do not keep track of how well the data was served when it was in its previous location. By not knowing how the placement of data affects future performance, predicting how data should be provisioned risks placing data on a storage mount that may have a bottleneck in the next demand shift.

Placing data among many storage devices produces another complication: how to best provision the act of moving data between devices. Beyond simply placing data, researchers have worked on enhanced versions of the previously described algorithms that also take into account how to best move the data on a network. Wen *et al.* [15] has developed a new algorithm based on LRU and Bloom filters to classify large flows and update the network in response to these flows. Murugan *et al.* [16] developed a hybrid LRU algorithm which prioritizes data that is heavily reused, demonstrating a significant performance benefit at the cache level. Alzakari *et al.* [17] proposed to use a Randomized LFU algorithm for network caches, and demonstrated that it outperforms standard LRU, FIFO and Random replacement strategies. However, the problem of better data layouts has recently shifted towards improved network transfer rather than improved data placements.

Our goal is to build a solution that bridges the problems of data placement and data transfer: a system that can optimize an HPC system's performance over time with improved data placement and network routing. Additionally, we aim to lower the retraining time of the model that places the data while maintaining a high training accuracy through feature selection. We run our system outside of the system it improves, minimizing any negative performance impact on the system. Additionally, at the network level, it does not need a global view on the available switches, working at the switch level to lower the impact of data transfers between storage points.

2.2 Reinforcement learning versus supervised learning

We treat the problem of determining a suitable data layout as a reinforcement learning problem. In short: moving data around while learning how a placement affects performance, slowly learning how to best move the data with regards to total system throughput. Reinforcement learning problems use agents that move throughout an environment, first taking random decisions to learn about the environment and slowly transitioning to calculated decisions taken from the output of a neural network. Supervised learning takes as input a labeled dataset, and can predict future values based upon historical knowledge of how the data changes. Supervised learning, importantly, does not interact with the environment it is predicting, and is not trying to maximize a specific target. Our learning technique does use a labeled dataset, however unlike traditional supervised learning, we use a reward function and environment interaction with the goal of maximizing the reward. We take actions upon the environment to learn how these actions affect the rewards, and continue to build this labeled dataset with the feedback from the environment. The steps that our reinforcement learning model takes are variations of the data layout in a system applied at even time intervals. Our reward function is how high our policy can make storage throughput go, however there is no exact number the policy must optimize towards. Although supervised learning is commonly used to analyze a system's performance, we do not necessarily know target performance values ahead of time. It would be prohibitively difficult to know prior to training that the system's performance will be some value in the future given a slight variation in the data layout.

In a sense, we are training the neural network with small batches of information until the model can effectively move data with the understanding that movement will maximize reward (throughput). The training data for such models is performance information such as throughput values over time. Our observation phase takes elements from traditional reinforcement learning. During observation, random actions are applied to the system so the results of such actions are recorded for experience replay. This experience is saved such that we can train a neural network with replays of what could happen if an action

is taken. Once the observation phase is done, training the neural network is done with knowledge gained from observation, and from the results of applying those changes. Therefore, we use online learning which is a combination of reinforcement learning and supervised learning.

The observation phase allows our system to gather real measured data. Even when our model starts taking decisions, our system still monitors the system it is tuning to understand how the actions taken by our system impacted the performance. This unsupervised learning technique allows us to make one technique that is applicable across systems: instead of relying on the model to be transferable, the technique can be widely used and adapted to any performance tuning situation.

2.3 Optimizing system modeling via feature selection (WinnowML)

A online learning based approach to placing data requires effective training data to provide to a neural network, and failure to do so will make predictions inaccurate. Inaccurate predictions could lead to sub-optimal data placements, and thus reduced performance gains. Though we do not investigate this in this thesis, we will investigate this link in future work. Although it may be tempting to provide the neural network with an overwhelming amount of data about the system it must learn about, training a neural network with all features may cause an increase in training time and lower accuracy. When presented with an enormous number of features, most neural networks begin by considering each feature to be equal then adjusting the weights for each feature. Additionally, some features may not add any information to model the target value. These features add unnecessary bias and noise to the prediction calculation, reducing the neural networks' ability to model the target values. Some systems have to react fast to changes in the target values to update the target system. The inability to model these changes can reduce the model's effectiveness on the target systems. By removing the extra features, we can reduce the error when predicting future values, as demonstrated in Section 3.3.

We focus on timeseries modeling as used by system analysts, where information changes over

time thus affecting the relationship between features. Most methods end up selecting new features as data gets collected. Selecting a similar subset of features and maintaining high accuracy as new data becomes available allows system analysts to gain an understanding of what features are impacting their target values. Additionally, by lowering the number of features, we can lower the neural network size which lowers the computational overhead. There are several approaches we can take to reduce the number of features, and extract only those needed to adequately train the neural network.

2.3.1 Dimensionality reduction and feature selection

The goal of dimensionality reduction is to avoid the costly time expenditure of experimenting with different sizes of feature subgroups. Dimensionality reduction techniques opt to declare a subgroup size instead of selecting specific features. They then let the algorithm fit the data to the subgroup. Other feature selection techniques, similar to WinnowML, use the model property to directly select the features that the user should use to model their target values. The selected feature subset allows a user to gain more in depth understanding on which features impact their measured values. In the case of performance enhancement systems, the subset can be used to indicate which features might be impacting the performance of the system. Popular techniques for dimensionality reduction, and feature selection, are Principal Component Analysis [18], UMAP [19], t-SNE [20], Incremental Principal Component Analysis [21], Sparse Principal Component Analysis [22] and L1-regularization [23]. These techniques find mathematical relationships between features to create a feature subset that represents the entire dataset. As part of future work, we will compare WinnowML to the use of autoencoders for feature selection. We do not investigate this here because the compression ratio is very difficult to determine, and we aim to compare our technique against common approaches.

2.3.1.1 Principal component analysis (PCA)

PCA [18] is a technique that reduces a high dimensional dataset into a lower dimensional dataset (from many features to fewer features). The low dimensional dataset contains the "principal" components of the original dataset, hence reducing its size while keeping the parts that are the most relevant. An application of PCA is image embedding for image recognition, where large images are reduced to discover the relevant features within it [24]. The drawback of such an approach is that it requires the user to know with certainty how many features they want a dataset to be reduced to. Researchers have since then improved on the initial PCA design with the incremental and sparse PCA algorithms.

Incremental PCA As described by Ozawa *et al.* [21], incremental PCA builds a low-rank approximation of the relationship between the input features regardless of the number of input data samples. It allows the features that are selected to not vary as input data samples are added to the model. The drawback of Incremental PCA is it ends up removing too much data from the features and the model starts losing crucial data to predict future values.

Sparse PCA As described by Zou *et al.* [22], the benefit of Sparse PCA is that it is less strict than the original PCA. The original PCA only keeps features that have non-zero coefficients when expressed as linear combinations of the original variables. The drawback is that there is a need for an expert to determine an optimized number of selected features. As more data points are collected, some features might start gaining importance, and a later analysis may find that more features are required. If so, the model must be redone.

UMAP and t-SNE UMAP [19] and t-SNE [20] can compute future values rapidly which is a benefit for live systems that rely on rapid response of the system to apply any corrections or changes in real time. One main drawback about t-SNE is that it can only reduce the set of features down to less than three features. For log data, there is often a need to have a larger dataset to describe what actions need to

be taken. The drawback of UMAP is similar, it merges all the data into a subset of data thus not revealing the specific features that were the most beneficial at modeling the target values.

2.3.1.2 Feature selection

L1-regularization L1-regularization [23] returns the cost associated with using each features in the case of a simple linear regression. The cost of each feature is representative of the impact it had on modeling the target features. A zero cost means that the feature had no impact. Compared to the PCA approaches, it allows a system analyst to identify specific features that impact a target value. Additionally, L1 can be used with the modeling approach that the user wants to use to model the target values. The resulting features that are selected by the algorithm are the ones that are found to be optimized for the modeling approach used to model the target values.

Other feature selection techniques deal with feature streams, a situation when the number of instances per feature remains consistent but the candidate features arrive one at a time. In these cases, the feature selection techniques need to select in a timely manner the features that should be used for the modeling task. Certain feature selection techniques [25–28] can determine a new subset of the features even when new features are added. The drawback of such approaches is that it was developed mainly for classification problems.

In most cases, when new timesteps are added the relationship between features might change over time. Some features that were originally seen as important become less so when new data is added. The first approach is the Online Feature Selection [10]. It focuses on feature selection for classification as features gain more data over time. Methods like unsupervised feature selection on data streams [11] deal with regression problems with unlabeled data. Huang *et al.* demonstrated that the feature coefficient when training the data to rank the features can be used as a indicator of which feature to select. Using only the coefficient of the features may lead to unreliable selection because of the noise present in performance data.

A third approach is the use of Support vector machines in features selection as demonstrated by Chang *et al.* [29] and Bradley *et al.* [30]. Using SVM works well for unbalanced classification problems as demonstrated by Maldonado *et al.* [31]; however, compared to concave minimization [30], SVM selects more problem features which then decreases the overall prediction accuracy.

2.3.2 Feature ranking

A common way to select features is to rank the features by importance or relevance to a specific model. Researchers use feature ranking to determine the strength of the relationship between training and target features. Some commonly used approaches are ranking features using correlation between the training and target features, Permutation Feature Importance or Self Organized Mapping. The last two feature ranking methods organize features by how each feature affects the calculation of the target values when using the modeling approach provided by the user. Other feature ranking methods such as Univariate Selection [32] and Recursive Feature Elimination [33] uses a series of statistical tests to determine each feature's relevance to the target values. However, these methods are prone to removing features too readily, and produce feature rankings that may not include the most important ones.

2.3.2.1 Correlation

The correlation between features represents the similarity in changes in values between the features. By using features that change in the related way or inversely to the target value, the model can predict more accurately changes that will happen in the future timesteps. We use Pearson's correlation [34] coefficient r, where features are ranked based upon their correlation with the target values. We use the following equation to determine the correlation between the training feature x and target feature y. The drawback of such a ranking is that not every dataset has every feature that is heavily correlated with the target values, such as system performance data, as seen in Figure 4.3.

$$r(x,y) = \frac{\sum_{i=0}^{n}(x_i - \mu_x)(y_i - \mu_y)}{\sqrt{\sum_{i=0}^{n}(x_i - \mu_x)^2}\sqrt{\sum_{i=0}^{n}(y_i - \mu_y)^2}} \tag{2.1}$$

2.3.2.2 Permutation Feature Importance (PFI)

The benefit of PFI [6] is that it is model agnostic and can be calculated many times with different permutations of the features. Using this technique allows the rankings to be consistent even when the user needs to change the modeling technique. PFI, as seen in Algorithm 1, permutes the features at each round, then removes a feature from the list and associates the increase in error as the rank to the feature that was removed. The drawback of this approach is that PFI does not tell us how many features to use, only what are the most highly ranked. Further, the ranking is not as stable as Self-organizing Mapping, and the feature ranks may change with the same feature data.

Algorithm 1: Permutation Feature Importance Algorithm.

```
Data: Trained model: NN, Training features: X, Target feature: Y, Loss function: L
Result: List of features weights for ranking: FR
Prediction = NN.predict(X) ;
initial_loss = L(Y, Prediction) ;
for i in length(X) do
    Perm_feature_list = Permute(X, X_i) ;
    New_loss = L(Y, Perm_feature_list) ;
    FR.append( initial_loss-New_loss );
end
Return FR ;
```

2.3.2.3 Self-organizing Mapping (SOM)

Self-Organizing Maps (SOM) [7], as described in Algorithm 2, uses the weights from the neural network trained with all the features to determine a rank for all the features. Once each feature is assigned a rank, it is sorted from highest to lowest, and the features of highest rank are considered to be most important for modeling the target value. Using these ranks enables the user to select only correlated features with the target and to remove features that have ranks of zero, features that are completely unrelated to the target values. The drawback of such an approach is that, similar to PFI, it does not determine the exact number of features to be used. Without knowing how many feature to use, there will be a decrease in accuracy if we choose the wrong number of features.

Algorithm 2: Permutation Feature Importance Algorithm.

```
Data: Trained model: NN, Training features: X, Target feature: Y
Result: List of features weights for ranking: FR
av_feature_weights = NN.get_weights() ;
// the get_weights() function get the average weights of
   each features from the model
for i in length(av_feature_weights) do
    FR.append( av_feature_weights[i] );
end
Return FR ;
```

2.3.3 Combined algorithms for effective feature reduction

Our solution to feature selection is a combination of dimensionality reduction and feature ranking. WinnowML lowers the number of selected features to a subset that changes minimally over time for online learning problems. The selected feature subset also keeps accuracy high and consistent

as new training data is measured from a target system. WinnowML does update the feature subgroup if new features are needed to increase the overall accuracy; however, it does so over time to ensure the new feature is relevant to the target over a long period of time. Online Learning models dealing with time series can use WinnowML to lower the number of selected features, lowering the amount of noise and biases included during the training of the model.

2.4 Optimizing data layouts

Choosing the features that do not vary over time enable models to efficiently train without wasting training cycles or using unrelated features. Performance enhancement systems need to react quickly to changes in performance in a target system. If they use all available feature they will not be able to react to drops in performance thus rendering the new layout useless. Additionally accuracy plays a big role in the resulting performance benefits. If a location is incorrectly predicted to have a high performance, moving the data to that location may result in worse performance compared to leaving the data where it is currently located.

2.4.1 Data layout strategies

Adjusting a system's data layout adjusts I/O performance based on the data distribution across storage devices. In a way, placing data is a multi-dimensional knapsack problem [35] where performance changes based upon how data is placed in locations that individually could hold all the data. Letting users decide where to put data puts the system at risk of one storage point being contended because it is popular. Several solutions have been proposed, and many have attempted to solve the complex problem with equally complex solutions such as particle swarm optimization [36] and the Sliding Window algorithm [37]; however, these solutions are rarely seen in production systems.

Heuristics cannot determine if a change in latency is temporary or long lasting. Some heuris-

tics distributes the files all over the system which may cause the system to incur penalties from moving many files to attempt to achieve a small speed-up. Also, distributed systems may be designed for a single purpose, in which case their data layout algorithms may only work in that environment. Chervenak *et al.* [38] discuss how data layout algorithms are developed in scientific computing environments, highlighting how many of these algorithms are developed in isolation for the workloads available at that institution. Thus, the performance gain from those algorithms may vary from one system/workload to another.

2.4.2 Moving the code to the data

Moving "code to the data" is normally achieved by spreading computational resources among many computation nodes linked to shared storage. Frameworks such as Hadoop [39], MapReduce [40], and Twister [41] move the code to the computation, and flow the data needed for collaborative computation along a mesh network. Such approaches achieve remarkable gains in computational and I/O performance, however make several assumptions about the workflow executed. One, the workflow software can be placed on hardware close to storage hardware; two, the inter-software flows of network data runs on a purpose-built network with little other traffic; three, the workload is not storage bottlenecked. Some workloads, such as physics workloads, require hundreds of terabytes of data. In such situations, it remains storage bottlenecked, and thus we look to improve workloads that cannot be improved by moving code. Such workloads may also run on a shared network, where mice flows (small bursts of tiny packets of data) from highly distributed computing may impact the work done by other users not conducting the same work. Another approach to optimizing the performance of a workload is the EMU [42] architecture, a processor-in-memory architecture for data objects in shared memory. Such an approach works excellently on data that can fit in memory, however Geomancy targets large data sets in a distributed computing environment.

Additionally, moving compute to the data requires specialized system architectures. Most

institutions will not support the ability to allocate a node per user. In highly distributed environments, the ratio of storage and compute power is highly variable (e.g., storage versus compute servers), so it is often necessary to move data. Further, in many workloads, some data is used more than other data. In these cases, moving compute maximizes locality but lengthens makespan due to less parallelism; load balancing then requires data movement. Geomancy would be useful in those situations since it rearranges the data for a workload with regards to other workloads running on the system. Hence, Geomancy targets the problem space where it is more desirable to move data rather than compute.

2.5 Dynamic route selection

Knowing where to place data solves the access performance problem; however, moving the data to the new location does add additional overhead. If data is not moved fast enough to prevent the performance drop then moving the data becomes unnecessary. It even may make the performance worse since it adds additional strain on the network for no particular immediate performance benefit. Additionally, existing workloads on networks can also cause contention if the data flows are not balanced.

People use traditional best-effort network connectivity services that use techniques from Global First Fit to Simulated Annealing [43]. In some cases, these services are unable to meet the emerging requirements of exascale computing workflows [44]. These techniques show that dynamic routing can deliver bandwidth gains if given a global view of the system. However the larger the network gets, the more overhead a global controller adds on the network.

We argue that optimizing at the switch level with neural networks provides increased bandwidth for data layout redistribution. This goal is motivated by the need to optimize scientific systems such as the US-CMS scientific networks, for which it is not practical to use a global controller since they are distributed across the world and are consistently being accessed by researchers. If we are to optimize how data is laid out, we must also optimize how the data is moved, and we do so by improving network

throughput.

2.5.1 The LHC program performance drawbacks

The increasing traffic of Large Hadron Collider (LHC) programs has severely affected the network. The US-CMS [45] Facilities (Tier1 [46] and Tier2 [47]) have upgraded their network to 100 Gbit connection. Yet, some of the US campus links remain not upgraded, and are heavily loaded at the 12–40 Gbit level [48–50]. Data flows should be intelligently routed through the ports to prevent the network from becoming saturated, especially as the LHC experiments continue to take data. It is evident that optimization must be taken local to where the work is being done: the networking hardware that serves the network.

2.5.2 Optimizing computer network throughput using models

Several works have attempted to enhance the throughput of network by modeling how a network's throughput changes with minor adjustments to the system. Nahm *et al.* [51] determined that increasing the TCP congestion window using a fractional increment can increase the sustained throughput. However, an issue arises when determining a good value for that fractional increment. Li *et al.* [52] demonstrated that using models to tune the parameters of a system can achieve up to 45% increase in the systems sustained throughput. As these works have shown, deep learning has merit when tuning large network and storage systems.

2.5.3 The TAEP Project

The TAEP project [53] was the first foray by Caltech and AT&T researchers into using Tofino programmable switches [54] to enhance networks. Tofino switches are standard networking switches with user-facing routing software, and users can run custom code on these switches for metric collection

or executing custom routing methods. Two experiments were conducted on a Tofino switch connected on the Tier 2 network at Caltech as demonstrated in Figure 2.1: a reinforcement learning-based load balancer and a heavy hitter detector. The Caltech Tier 2 network is a open network, a WAN visible in education and commercial network without private paths between campuses. Both experiments were conducted on one switch located on the network. The switch had four available ports that the data could get rerouted through.

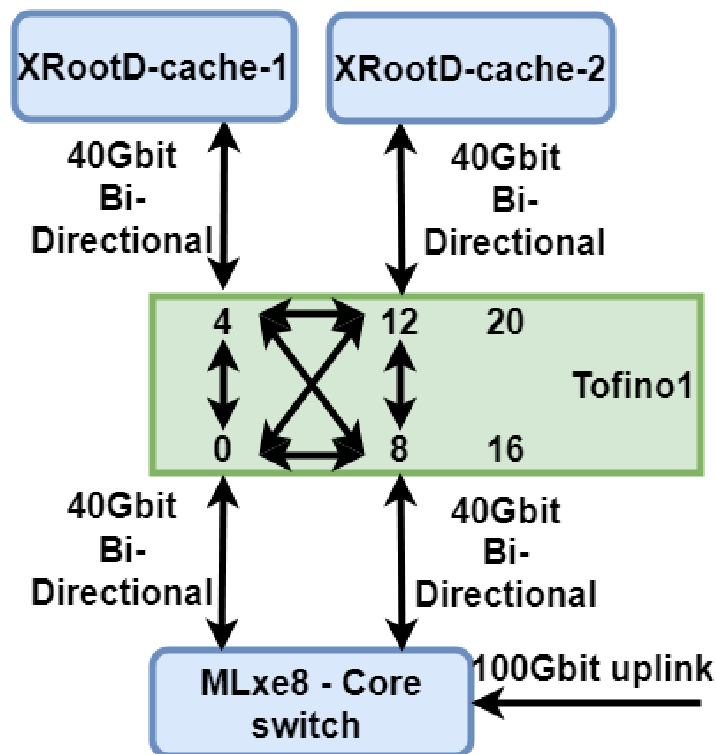


Figure 2.1: Placement of a Tofino switch in Caltech Tier 2 network. All links are bi-directional.

Their first approach was a reinforcement learning based load balancer that balances incoming network traffic across all outbound ports. The resulting traffic was routed 30% on one route and 70% onto the two remaining routes. Their second approach was to create a heavy hitter detection program that uses deep learning to identify heavy requests, and reroutes the network packets to reduce congestion. Heavy hitter detection is the detection of large flows of data within a network, referred to as "heavy

hitters". A sudden increase in data flows can cause contention and bottlenecks in a network [55]. Using programmable network switches, an agent running on the switch can re-route the heavy hitters across network links that have less contention, thereby reducing the impact of the heavy hitter. From their initial experimentation, they concluded that additional inputs must be considered for better load balancing. It is not sufficient to only know that a heavy hitter is occurring without an intelligent solution to resolve the heavy hitter. Therefor, we focus on dynamic routing as a solution to network contention.

Chapter 3

Lowering training error and overhead through feature selection

To lower the need to retrain the model when feature relationships change, we need to select a subset of features that accurately represents the target value and minimally changes as new data becomes available. If we do not, then our model may need to consistently retrain on new feature groups, thus making it hard for a system analyst to determine what aspect of the system is impacting the target values. Because we focus on time series modeling problems, our system should be able to identify features that consistently impact the target value even when new data is introduced. The knowledge must be built up over time. For example, a performance enhancement system needs to react quickly to changes in the system or it risks trying to tune a system with an outdated model. Additionally, using features that are unrelated to the measured value that the user wants to model may add additional noise to the model, which increases the prediction error. A model with high prediction error causes the predictions to become unreliable. For example, if a model suggests a wrong route in the system, then the data may never reach its destination. Lowering the prediction error prevents these mistakes from happening in the first place.

There are multiple approaches to select a representative feature subgroup from a larger body of data, such as looking at the correlation between the features and the target values. Algorithmically, we can also use dimensionality reduction approaches such as Principal Component Analysis (PCA) or feature selection technique such as L1 regularization. For this thesis, when we are modeling performance of a system, we model its I/O throughput; however, these approaches do not change when modeling for latency.

3.1 WinnowML Design

WinnowML, as illustrated in Figure 3.1 is a system used to maximize accuracy and lower biases through feature selection for systems that re-train as new data points become available. Usually as workloads shift, models need to be retrain with new features to represent the change in relationship. WinnowML lowers this overhead by continuously selecting a similar subset of features while keeping a low prediction error. The prediction error is measured by the absolute difference between the target and predicted values. Additionally, WinnowML can run in parallel to the model it is tuning and only update the model when there is a significant change in the features selected, thus further reducing the overhead.

To use WinnowML, the user starts by submitting a dataset, the target features they wish to model, and the neural network architecture to model the features. WinnowML uses the provided information to rank the features using a combination of the sum of the neural network's weights and the correlation between the input and target features. When calculating the ranking of the features, it prioritizes the correlation between the training and target features and then uses the sum of the weights the rank the rest of the features. The benefit of using our combination of the correlation and the sum of weights is that WinnowML finds a ranking that changes minimally as new data becomes available. Our ranking remains stable but may change when enough evidence shows that accuracy would improve with the addition of new features.

From the ranking, it identifies a grouping of features that produces accurate predictions com-

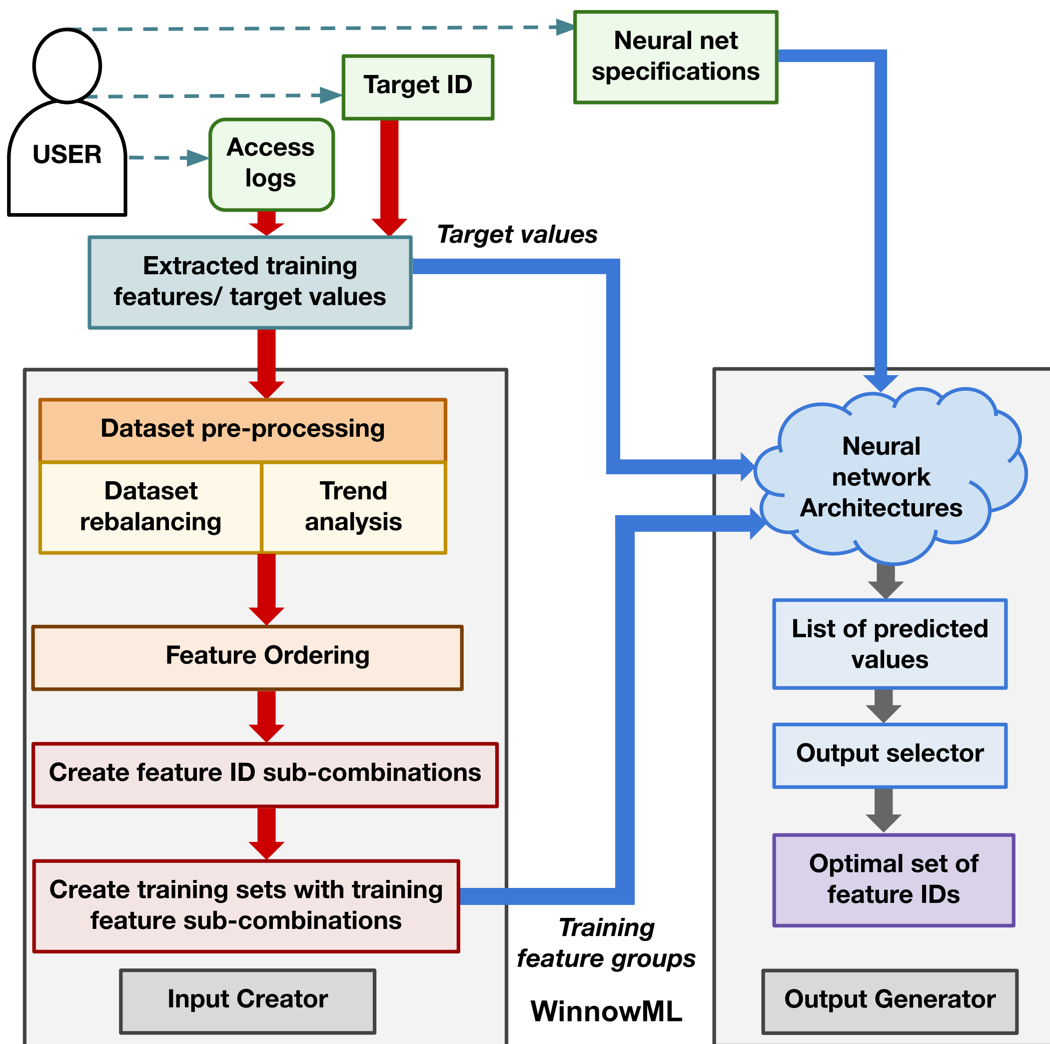


Figure 3.1: WinnowML system design. Blue and gray arrows represent the data flowing into and out of the neural network. Red solid arrows represent the dataflow of the log and target ID given by the user through the data processing section of WinnowML. The dotted arrows represents the data given by the user.

pared to other feature groups. When the group is identified, WinnowML return the feature IDs to the user. Using the resulting selected features, the user can gain important insight on whether or not the

modeling approach is right for their target values. Since the features selected remain consistent over time, the features might give the user additional insights on how to optimize their system.

3.1.1 Input data

The dataset that WinnowML accepts must contain information to identify training and target features. The data should be represented in a comma-separated value (CSV) format where the first row contains the labels of each column. We will be expanding to other formats but currently we are starting with this one for the first version of our tool because it is commonly used. The labels of the column are used to identify the features of the dataset. They are also used to select the training and target data points. The target value ID and python model is provided by the user as well. Additionally, the data needs to contain some amount of correlation between the training and target values at the beginning of the dataset. That correlation is used for the initial feature ranking. Once the features are ranked once using the correlation, even if for the next timesteps the training features have no correlation with the target features, the ranking remains relatively consistent.

3.1.2 Feature ranking

WinnowML ranks features with high absolute value of correlation at the top of the ranking. It combines the original score of each feature with the sum of the neural network's weights, or the resulting score using PFI, and the current correlation of the features to get the final score for each feature. The goal of each method is to identify how much each feature affects the resulting prediction. WinnowML keeps track of past ranking and uses them in the current ranking calculation, thus keeping the ranking consistent through time. Information gain from past ranking limits the potential changes in the feature ranking since high-rank features have an advantage for future rankings.

Other techniques, such as feature ranking techniques using linear SVM [29], are model specific. Since most time-series predictions utilize utilizes different prediction techniques, it is essential to

come up with a ranking mechanism that works for a variety of modeling approaches. We focus on models created using Keras [56] and Scikit-learn [57] libraries, which are commonly used by researchers. Additionally, many approaches change their feature selection when new data is introduced. If the selected features change, the model needs to retrain on new data when it becomes available. In many cases, time series modeling relies on long term trends to accurately model future data changes. By having to retrain on new data, the model loses the ability to use past trends in the prediction calculation.

3.1.3 Feature group creation

Algorithm 3: Feature ranking.

```
Data: Feature list: FL_in, Modeling Target: T, Model:NN, dataset:data
Result: Ordered Feature list FL
rank_of_features = get_rank(FL_in, data, NN) ;
// The get_rank() function calls one of the two wrapper
   methods available and returns the rank value for each
   features
// each rank is a value between 1 and 0
Final_rank = [] for i in length(FL_in) do
    new_rank = ω ×corr_{FL_in[i],T} + rank_of_features[i];
    Final_rank.append(new_rank);
end
FL = organize(FL_in, Final_rank);
// organizes the features with the feature with the highest
   rank is placed at index 0 and the feature with the lowest
   rank is placed at index n
Return FL ;
```

WinnowML starts by ranking the features using Algorithm 3; however, individually ranking each feature is insufficient to determine what group of features is needed to maximize prediction accuracy. WinnowML creates groups of features starting with only the highest-ranked feature determined in the previous step and trains the neural network. WinnowML then predicts the target value, assigning the prediction accuracy measured to that group, using Algorithm 4. One feature subgroup's prediction and accuracy index in the returned array matches the subgroup's index in the Feature_subgroups array. Next, WinnowML adds the next highest-ranked feature to the first group, creating a new group. With the new group (now with two features), WinnowML trains the neural network to create a new prediction and accuracy measurement. The group making process continues as WinnowML adds more features with decreasing rank until a final group is created with all of the features available in the dataset. Once the groups are created, the model is trained using 35,000 timesteps for each features of the current group. For the purpose of this paper, we trained the model with 200 epochs and early stopping with an allowance of 50 epochs to prevent overfitting. Additionally, we set the batch size to 100. In practice, however, the model and how it's trained will be determined by the user. The data used for training is split into 3 sections. The first section contains 60% of the data is used for training, the second section which contains 20% of the data is used for validation, and the last section which contains 20% of the data is used for testing the model. In the end, each group is assigned an accuracy score calculated from the resulting prediction of the 3rd section of the data.

Algorithm 4: Feature group creation and accuracy calculation.

Data: Feature list: FL, Modeling Target: T, dataset: data

Result: Feature subgroups, group predictions and accuracy

Function AE (P, T) **:**

  result_AE = [];

  foreach *i in length(P)* **do**

    result_AE += $\frac{|T_i - P_i|}{P_i}$;

  end

  return result_AE;

accuracy = [] ;

predictions = [] ;

Feature_subgroups = [] ;

for *i in length(FL)* **do**

  Feature_subgroups.append(FL[0:i]) ;

  NN.train(Feature_subgroups[i], T, data) P = NN.predict(Feature_subgroups[i], data) ;

  predictions.append(P) ;

  accuracy.append([($\mu_{AE(P,T)}$ or $\mu_{\text{precision}(P,T)}$), ($\sigma_{AE(P,T)}$ or $\mu_{\text{recall}(P,T)}$)]) ;

end

Return Feature_subgroups, predictions, accuracy ;

If WinnowML attempted to create every possible combination of features available, it would have created $f(n) = 2^n - 1$ subgroups, where n is the number of features available, and r is the size of the group. Our grouping benefit is that it lowers the search space for the best feature group to n groups. Although there is no guarantee that this is the best group to search, it at least reduces the search space considerably to a smaller set of plausible candidate groups. Additionally, the groups that were removed are the ones with more lower ranked features. The lower ranked features bring less information about potential changes in the target values compared to higher ranked features. By removing them, we reduce

the risk of adding additional bias and noise to the model, as demonstrated in Section 3.3.

3.1.4 Measuring a group's accuracy

We rank the subgroups of features to maximize the prediction accuracy when modeling a measured value. Focusing on groups that contain the highest ranked features helps WinnowML reduce the number of groups analyzed for accuracy. We prioritize highly ranked features because using the most amount of high ranked features increases how the weights change towards modeling a target.

Ranking each group requires WinnowML to consider several factors that describe each group's performance as it relates to prediction accuracy. For regression, WinnowML uses mean absolute error (μ_{AE}) to measure on average how close the predicted values are to the real values that the neural network is modeling. It also uses the standard deviation of the absolute error (σ_{AE}) to measure how the average error between the prediction and target fluctuates. A high σ_{AE} indicates that the model is not capturing how a target value rises and falls, either overpredicting or underpredicting the value. Both μ_{AE} and σ_{AE} are important factors to consider when measuring a group's accuracy: not only does a group need to accurately model the target, but it must do so consistently. For classification, instead of using μ_{AE} and σ_{AE}, we use *recall* and *precision* [58]. Both Prediction and Recall return a value from zero to one. The closer the precision and recall value is to one, the more correctly the predictions are classified.

Kurtosis and skew are common metrics for evaluating model quality. Kurtosis is the fourth moment when describing a function. It measures how heavy the tails of a distribution differ from the tails of a normal distribution. Skewness on the other hand is the third moment when describing a function. It measures the asymmetry of a distribution. Both approaches could be used to measure if the model over/under predicted; however, kurtosis and skew values are extremely sensitive to noise in the data. When the noise is high enough to impact the correlation between the training and target values, which is the case for most performance data, the kurtosis and skew difference between the prediction and measured values does not correctly capture the similarities between both values. When comparing the

groups both kurtosis and skew end up overshadowing the first two metrics and thus selects a subgroup that does not have a low error rate between the prediction and target values. Therefore, we forgo using both of these metrics.

3.1.5 Ranking groups of features

We use both μ_{AE} and σ_{AE} as described above to create a formula representing each group's effectiveness towards increasing accuracy. Higher effectiveness grants a higher rank to that group. The accuracy is calculated using Algorithm 5, where w_1 and w_2 are the weights of each individual metric used to calculate the accuracy of the prediction. These weights describe how important each metrics is when ranking a group. In the algorithm P is the prediction values, T is the measured value that we are modeling, the problem based function $PB(P,T)$ is the sum of the problem specific accuracy metrics.

Algorithm 5: Ranking calculation

Data: ProblemID

Result: Accuracy of feature subgroup

if *ProblemID is regression* **then**

$PB(P,T) = w_1 \times \mu_{AE(P,T)} + w_2 \times \sigma_{AE(P,T)}$;

end

if *ProblemID is classification* **then**

$PB(P,T) = w_1 \times (1 - \mu_{\text{Precision}(P,T)}) + w_2 \times (1 - \mu_{\text{Recall}(P,T)})$;

// $\mu_{\texttt{Precision}(P,T)}$ and $\mu_{\texttt{Recall}(P,T)}$ represent the mean precision and recall value over all the predictions

end

Rank$(P,T) = \frac{1}{PB(P,T)+1}$;

Return Rank(P, T);

We invert all the previously described elements to penalize them as their value increases. For precision and recall, we penalize how far they are from 1, a perfect precision or recall score. Additionally we added one to the total score of all four metrics so that if all metrics are equal to zero there will not be a division by zero. Doing so, a total rank of 1 indicates that the group perfectly maximizes accuracy.

The formula to rank groups is affected by what type of data is used for training. Because each metric can have a varying degree of effectiveness depending on the data used for training, each metric must be assigned a weight (w_1 and w_2). The higher the weight, the more important that metric is considered when assigning a rank to the group.

We determined the weights w_1 and w_2 using the Variable-Share algorithm [59]. Variable-Share is an algorithm that determines how important an "expert" is among a group when each expert contributes to a score. We use the accuracy of the prediction done by a single metric to determine an updated weight for that feature (the expert). We must assign weights per ranking metric because the plain metric values for each subgroup does not tell us how each group differs. Ties in error, for example, may not tell us which of the other two metrics should be considered. Ties may also hide the fact that the equal mean error of one group to another should not be as important when determining rank as variance in mean error. Hence, we calculate weights per metric to determine which metric is the best descriptor of subgroup importance.

The Variable-Share algorithm is run once the accuracy metrics have been calculated. The goal of the algorithm is for it to minimize the overall error metric. The loss function is used by WinnowML to determine how accurate each feature was in choosing the subgroup, as shown in the following equation.

$$\text{Loss}(Y, X_i) = \sum_{j=1}^{n}(w_j \times X_{i,j}) - \sum_{j=1}^{n}(w_j \times Y_j) \tag{3.1}$$

Y is the subgroup with the smallest sum over all n metrics used to determine the accuracy of the subgroup. Y_j is metric j in the subgroup with the smallest error sum. $X_{i,j}$ is metric j in the subgroup

X_i that is the highest ranked group when only metric i is used to calculate the rank of the subgroup. w_j is the weights given to metric j. Overall, each feature is classified by how close it is to choosing the subgroup with the lowest sum of error metrics with the current set of weights. The difference is used to penalize the weights of features that performed badly, represented by the following equation.

$$w_i' = w_i \times e^{-(\eta \times \text{Loss}(Y, X_i))} \tag{3.2}$$

Once the weights are penalized using the loss equation, the weights are updated using the variable-share equation. The equation allows for features that did not perform well at earlier trials to catch up at later trials if their predictions become more accurate.

$$w_i'' = (1-\theta)^{\text{Loss}(Y,X_i)} \times w_i' + \frac{1}{n-1} \times (\text{Pool} - (1 - (1-\theta)^{\text{Loss}(Y,X_i)}) \times w_i') \tag{3.3}$$

Pool contains past weight estimations. It is uses the following equation:

$$\text{Pool} = \sum_{i=1}^{n} (1 - (1-\theta)^{\text{Loss}(Y,X_i)}) \times w_i' \tag{3.4}$$

By assigning weights to each of the four metrics, WinnowML can accurately rank each subgroup without biasing the rankings from one particular metric. The group with the highest rank is returned to the user. The highest rank group is now considered to be the group of features that most accurately models the target features requested by the user.

3.1.6 Manually setting importance

If a user does not wish to strictly maximize accuracy, WinnowML can set a different optimization target. WinnowML offers two approaches to set the importance of the number of features, the error, and the accuracy in the calculation of the ranking for a feature group: a weighted approach and a threshold approach. The weighted approach uses an additional weights, w_5, as represented in the new

ranking equation below. w_5 represents the emphasis on how the number of features affect ranking and NF represents the number of features in the subgroup that is being ranked.

$$\text{Rank}(P,T) = \frac{1}{w_5 \times NF + (1 - w_5) \times (PB(P,T) + 1)} \tag{3.5}$$

The $PB()$ function remains the same as the ones described in Algorithm 5. w_5 is a value between zero and one. In case the user wants to focus strictly on accuracy, they can set w_5 to zero, which will revert the ranking equation to the one described in Algorithm 5. However in some cases the user might want to limit the number of features selected, or the accuracy to a certain threshold, such as not using more then five features or having an error rate lower then 20%.

To set a threshold for the feature group selection, a user will pass a flag stating that they will be using thresholds for a certain parameter, number of features and/or error rate, and the value of the threshold. WinnowML uses the index of the group in the array of groups as an indicator to check how many features are located in the group. If the user wants to set a threshold to the number of features, WinnowML will only rank feature groups whose index is lower then the threshold, if the user wants the selected groups to have less features then the set threshold. In case the user wants more features than the threshold, WinnowML will only consider the feature group with an index larger then or equal to the threshold.

In case the threshold is not met, WinnowML will return an error message that gives the closest ranked feature group. That error message can be used by the user to fine tune the technique used to predict a target value.

3.2 Experimental setup

In our experiments, we use the access logs that represent accesses on the CERN EOS system. EOS is an open-source storage software solution that CERN uses to manage multi petabyte storage for

Table 3.1: Accuracy of WinnowML vs model used. Each model is described in Table 3.2. Each row corresponds to a round of training with new data.

μ_{AE} of Model 1 (%)	μ_{AE} of Model 2 (%)	μ_{AE} of Model 3 (%)	μ_{AE} of Model 4 (%)	μ_{AE} of Model 5 (%)	μ_{AE} of Model 6 (%)	μ_{AE} of Model 7 (%)	μ_{AE} of Model 8 (%)
22 ± 18	29 ± 30	23 ± 21	22 ± 20	22 ± 18	21 ± 19	22 ± 21	20 ± 20
20 ± 13	21 ± 12	20 ± 13	20 ± 13	18 ± 12	21 ± 13	20 ± 13	20 ± 13
21 ± 13	21 ± 13	22 ± 13	20 ± 12	21 ± 12	21 ± 13	21 ± 13	21 ± 13
21 ± 12	20 ± 12	20 ± 12	18 ± 13	18 ± 14	19 ± 12	18 ± 14	21 ± 12
21 ± 12	21 ± 13	22 ± 13	22 ± 13	21 ± 12	22 ± 13	22 ± 12	23 ± 12
21 ± 12	21 ± 12	21 ± 12	21 ± 12	22 ± 13	22 ± 13	21 ± 12	21 ± 12
21 ± 12	21 ± 12	23 ± 12	21 ± 12	21 ± 12	22 ± 13	21 ± 12	22 ± 12
21 ± 12	21 ± 13	20 ± 13	20 ± 12	21 ± 13	21 ± 13	20 ± 12	20 ± 12
23 ± 12	22 ± 11	23 ± 12	23 ± 12	22 ± 11	21 ± 12	23 ± 12	23 ± 12
21 ± 12	22 ± 12	22 ± 12	22 ± 12	22 ± 12	22 ± 12	20 ± 12	22 ± 12

the Large Hadron Collider (LHC). This system uses metadata within each file to store indexes that provide information about where and how a file is being stored. EOS is used worldwide by the physics community to run experiments on the LHC. Therefore, the dataset contains accesses that are representative of large distributed scientific systems. We are interested in modeling storage throughput performance on this system to discover patterns in throughput in daily operation.

3.2.1 Dataset

The EOS dataset [60] contain a wide variety of features describing the operation of the storage system. Every file access is tracked using timestamps with the *day*, *month* and *year*. To keep track of

what data is being accessed by which user and from what location, the EOS access log uses identifiers like the file ID (*fid*), the path ID (*path*), the filesystem ID (*fsid*), the mapped user ID (*ruid*) and group ID (*rgid*), and client information is described using features like organizations *secvorg*, groups *secgrps*, user roles *secrole* and user applications *secapp*. In our experiments, We used logs from 2019, which has 32 features. In the log, we use $35,000$ timesteps for each of the 32 features to model the throughput at each round. Rounds are new training done when new batches of data are collected from the system, or in our case the dataset.

Datapoints in the EOS dataset contain random variations caused by inconsistencies in system accesses, which may interfere with the performance of other workloads. To reduce these variations, we apply a moving average to the datapoints to remove the inconsistent trends in the data and improve prediction accuracy. Additionally, we enable derived target features to be calculated (*throughput* and *latency*), which are not always recorded directly in the dataset. We use features found in the dataset such as start and end timestamps of an action to a file to calculate such target values.

Using the CERN dataset, we modeled the throughput of data accessed on the CERN EOS system using the CERN EOS access logs. WinnowML is trained with 10 batches of 35,000 data points from the datasets to simulate more data being measured from the system. The throughput of access TP_i is calculated using the following equations. *rb* and *wb* are the number of bytes read/written to the data during the access, *cts* are the closed time stamp in seconds, *ctms* are the closed time stamp in milliseconds, *ots* are the opened time stamp in seconds and *otms* are the opened time stamp in milliseconds.

$$\mathrm{Tp_i} = (\frac{\mathrm{rb_i} + \mathrm{wb_i}}{\mathrm{Time_i}}) \tag{3.6}$$

$$\text{where, } \mathrm{Time_i} = (\mathrm{cts_i} + \frac{\mathrm{ctms_i}}{1000}) - (\mathrm{ots_i} + \frac{\mathrm{otms_i}}{1000}) \tag{3.7}$$

3.2.2 Experiments

In the first experiment, we model the CERN EOS I/O throughput with different modeling techniques. This experiment demonstrates WinnowML's ability to adapt to different modeling approaches while keeping the prediction error low. The goal of the this experiment is to demonstrate that not all models need the same feature subset. Here, we take a number of features and attempt to model throughput using the features given by WinnowML. Since we are treating this is a regression problem, we will measure Mean Absolute Error and Standard Deviation of Absolute Error to show that our model is predicting well. We demonstrate the effectiveness of a model by measuring error, the time it takes for a prediction to be generated, the number of features needed, and the time the model needs to be trained. Here, the model is trained for 200 epochs before making predictions, with early stopping set with a patience of 50 epochs.

Table 3.2: Model description

Model number	Description
1	Dense model with 1 hidden layer
2	Dense model with 2 hidden layer
3	LSTM model with 1 hidden layer
4	LSTM model with 2 hidden layer
5	GRU model with 1 hidden layer
6	GRU model with 2 hidden layer
7	SRNN model with 1 hidden layer
8	SRNN model with 2 hidden layer

In the second experiment, we take the base case dimensionality reduction techniques, L1-regularized, FSDS and OFS, compare the resulting number of selected features to the ones selected by

WinnowML. The goal of this experiment is to demonstrate the consistency in the number and types of features selected by WinnowML. We start by looking at the number of features selected by each approaches. Using the number of features, we can narrow down our search space to the approaches that selected consistently a similar amount of features. Once we have narrowed down the number of reduction techniques down to the ones that remain relatively consistent, we examine the specific features that are selected. The features that are selected can be used to identify whether the base dimensionality reduction techniques are more consistent in the choice of feature when compared to the features selected by WinnowML.

In the third experiment, we monitor the change in accuracy over time. The goal of this experiment is to demonstrate the accuracy gain of WinnowML over commonly used approaches such as L1. We compare the resulting accuracy between the two approaches that gave the most stable selection in the past two experiments. The change in accuracy helps us identify if the features selected by WinnowML are actually producing accurate results or if the other feature selection techniques allow for higher prediction accuracy. Additionally, if the prediction accuracy remain high and constant, that means that the features selected remain important features when modeling the target value even when new data is gathered from the target system. These features can then be used by the system analyst as indicators on how to improve the system.

3.3 Results

In all the following experiments, the models are trained over at most 200 epochs. They use early stopping to prevent overfitting. In experiment 1, we measured the mean and standard deviation of the absolute error between the prediction and target values for each group created by WinnowML. We found that there is not a lot of fluctuation in the accuracy between the models, as seen in Table 3.1. They

averaged between 19% and 22% error. With the resulting accuracy, we demonstrate that WinnowML is able to adapt to a number of models without losing accuracy. When we looked at the features selected by each model, the selected features always contained open timestamp (ots) and close timestamp (cts). The selected features demonstrated that the resulting access performance is related to when the file is being accessed. Sometimes the models also selected the size of the file either when opened (osize) or closed (csize). The features selected show that after the time when the file is accessed, the size of the file affects the overall performance of the access. For example, we observe that a large file will have a slower access then a smaller file. Additionally, the reasoning on which feature to use does not change as the modeling technique changes.

Table 3.3: Number of features selected for each round versus the feature selection technique used

Round number	WinnowML	Incremental PCA	Sparse PCA	PCA	L1 (α= 0.01)	OFS	FSDS
1	3	3	6	3	2	3	2
2	1	1	3	3	2	2	4
3	1	2	1	2	2	1	3
4	6	2	10	1	3	10	4
5	1	1	2	1	1	6	7
6	1	12	1	21	1	14	14
7	1	4	10	5	3	6	9
8	1	27	18	30	4	7	14
9	3	31	10	26	3	3	1
10	1	2	2	3	2	3	27

In experiment two, we show that L1 and WinnowML were able to consistently select a similar

number of features over time, as seen in Table 3.3. Unlike the PCA family, which at some point ended up selecting more then triple the amount of features. Additionally, we observed that on round 9 Incremental PCA chose 31 out 32 features which caused it to get μ_{AE} of 28% $\pm$ 27% which is greater than the μ_{AE} gotten from using the 3 features selected by WinnowML which was equal to 22% $\pm$ 12%. In some cases, we found that the standard deviation of the absolute error surpassed the mean absolute error. This means that the prediction value and the target value swapped places. For example, the target value was greater than the predicted value, it means at some point the prediction overshot the target value. In order words, if the standard deviation is greater than the mean absolute error the model failed to capture the variation of the target data. Most feature selection techniques, including WinnowML, fluctuate in the number of features selected since the workload we are using contains some random noise between accesses, which affects the relationships between features. The OFS and FSDS approaches both started to fluctuate their features at round 4 in the number of features selected. Most approaches except for L1 and WinnowML severely increased the number of features selected by round 6, which is the round with the most amount of noise.

Table 3.4: Feature ID selected for each round versus the feature selection technique used

Round number	WinnowML	L1 (α= 0.01)	OFS
1	ots,cts,osize	fsid,ots	sec.app,ots,cts
2	ots	cts,osize	nwc,ctms
3	ots	wt,nwc	fsid
4	ots,cts,csize, osize,otms,wt	ruid,wb,nwc	nfwds,otms,rb, ruid,fsid,fid, nxlfwds,nwc, ctms,nxlbwds
5	ots	nxlfwds	otms,nrc,fsid, rb,wb,wt
6	ots	fsid	ctms,otms,cts, nbwds,rb,ruid, fid,osize,nwc, fsid,wb,sbwdb, nrc,csize
7	ots	fsid,cts,ctms	sxlbwdb,ots,fid, fsid,wb,nxlfwds
8	ots	fid,fsid,otms,wb	wb,rt,cts,osize, fsid,ruid,nwc
9	ots,cts,wt	ctms,wt,osize	ctms,csize,wb
10	ots	fid,cts	cts,nwc,fid

For the second part of the experiment we explore the features selected by WinnowML and L1. We forgo the PCA family since that family of feature selection techniques fluctuated heavily when looking at the number of features selected. Additionally, we forgo FSDS since it performed similarly as OFS. We focus on L1 with α value equal to 0.01 since it performed the closest to the amount of features WinnowML selected, and OFS since it is commonly used for data streams. As seen in Table 3.4, we observe that WinnowML was more stable when choosing the features for training compared to L1 and OFS. OFS in particular selected a significantly larger number of features every round compared to WinnowML. We demonstrate that WinnowML is not only able to consistently select the same number of features, but it also selects a similar subset of features over time. Since it selects mainly the timestamp of when the data is opened or closed as constant features, it shows that over time the amount of accesses to data in the CERN system have significant impact on the resulting performance. It also demonstrates that between rounds the workloads may shift significantly making the performance harder to model using other features. When looking at the results from experiment 1 and the current experiment, we also notice that depending on the model used, the size of the file when it's open and closed may also impact the resulting performance. We can therefore conclude that a combination of the time when a file is accessed and the size when it's opened and closed form a subset of features that will maintain a low prediction error that will minimally change over time. Hence, we only need to monitor the aforementioned four features (open time, close time, file size when opened, and file size when closed) instead of the entire set of 32 features. This selection of features requires a smaller network, which lowers the computational cost of applying the network to the CERN system.

Table 3.5: Feature ID selected for each round versus the data used for the prediction

Round number	WinnowML October	WinnowML July
1	ots,cts,osize	ots
2	ots	ots
3	ots	ots,cts,ctms
4	ots,cts,csize, osize,otms,wt	cts
5	ots	cts,ctms
6	ots	ots,cts,ctms
7	ots	cts
8	ots	ots
9	ots,cts,wt	ots,cts
10	ots	ots,ctms

For these experiments, we used access data taken from the CERN EOS logs on accesses happening to the Fuse OS of the EOS system on the 19th of October 2019. To ensure that we were not overfitting to the data we also ran WinnowML on data collected on the 5th of July 2019, as seen in Table 3.5, in addition to training every model using early stopping. We can see that on both days WinnowML picked up the importance of the open and closed timestamp when it came to modeling the access throughput to the Fuse OS on the CERN EOS system. The variations in features selected comes from the fact that the weights of the model are randomly initialized. Since we are using self organizing mapping as our wrapper method for these experiments, this initialization could cause some features that were not

initially used to get some attention from WinnowML.

Table 3.6: Mean absolute error (%) of selected feature subset for each round versus the feature selection technique. Error reported is standard deviation of mean absolute error to show error fluctuation. When the standard deviation overshoots the average, this shows that the model failed to capture the variation of the target value

Round number	WinnowML	L1 (α= 0.01)	OFS
1	21 ± 15	33 ± 29	25 ± 17
2	20 ± 13	27 ± 26	59 ± 39
3	20 ± 13	30 ± 38	58 ± 33
4	19 ± 12	38 ± 49	81 ± 67
5	21 ± 12	36 ± 43	62 ± 62
6	21 ± 12	25 ± 26	52 ± 43
7	21 ± 12	44 ± 36	65 ± 57
8	20 ± 12	46 ± 68	107 ± 226
9	22 ± 12	30 ± 32	78 ± 83
10	22 ± 12	34 ± 38	69 ± 47

In the third experiment, we found, in table 3.6, that the subset of features selected by WinnowML kept the absolute error between the prediction and target values low compared to the L1 and OFS approaches. The stable accuracy observed in the table shows that a limited subset of features does not adversely affect the prediction accuracy as workload shifts. We observe that for L1 on occasion the standard deviation overshot the mean error, indicating that the neural network was not able to capture the variation in the target data, contrary to WinnowML's prediction error which kept a small standard deviation. It means that unlike L1 or OFS, WinnowML was able to capture the change in the target value

which then can enable a performance enhancement system to determine with higher accuracy when a file should be moved. Additionally, the OFS approach selected more features, and yet did not match WinnowML's accuracy. Rather, it has much higher error, up to 333% over the true target value. From this, we can conclude that on average WinnowML has a 13.6% lower mean absolute error between the target values and the predicted values when compared to the closest performing approach, L1.

3.4 Conclusion

WinnowML manages to identify a subset of features that consistently impacts the performance of a system over time. We observed that, on average, WinnowML has a 13.6% lower mean absolute error between the target values and the predicted values when compared to the closest performing approach, L1. It outperforms the rest of the approaches by more then 13.6% while maintaining a stable selection of features that minimally changed as new data became available. The stability our tool provides allows system analysis the opportunity to better tune their systems. Additionally, WinnowML reduces the training overhead of models used for online learning by lowering the data needed for training without impacting prediction accuracy.

We developed this system while working on using data layouts to optimize performance on distributed storage systems. We started by using features that are easily found on every system that have high correlation with the throughput of a data access to model the data layout. Using WinnowML, we were able to validate that we selected most of the features that WinnowML suggested. Importantly, we were able to verify that selecting features about when a file is accessed and closed consistently allows for low error rate. In future work, we will continue to work on reducing the resulting error rate as much as possible.

Chapter 4

Applying modeling techniques to data layouts

After identifying features that can be used to accurately model data placement in a system, we turn our attention to updating the layout of the data in the system as performance changes over time. By updating the data layout, we can lower the contention coming from shifts in accesses in the system. Accesses can shift to data pieces that may be located on slower hardware, thus resulting in a significant drop in access throughput. Geomancy is a system that we have developed and tested on the Pacific Northwest National Laboratory's system that predicatively rearranges data among many discrete storage points as it forecasts potential performance drops. By rearranging data, we can efficiently place hot data on high bandwidth storage devices, while moving cold data to slower, higher capacity, drives to free up bandwidth to the high performance drives. We have evaluated this system using workloads provided by Pacific Northwest National Laboratories.

4.1 Motivating workloads

Our motivation steps from the workload analysis of two workloads: traces generated from a Monte Carlo physics workload provided by Pacific Northwest National Laboratories (PNNL) and workload traces from CERN. Both of these systems have varying work loads as well as third party loads, as demonstrated in section 4.4. The exact methods to generate the traces do not matter for the purposes of our experiments, however each trace follows a similar setup. The traces all have features that describe the I/O throughput of the system one wishes to optimize. For example, the CERN EOS trace contains information about when a file was opened, closed and where the action took place. We care about when the file was opened since if a file is opened at a time when the storage device is contended it will affect the access latency. We also care about where since some storage devices are more contended then others.

The EOS file system [61] is a storage cluster with an analysis, archive and tape pool. An analysis pool is a low latency disk base caching storage [62]. We did not get access to this system to test Geomancy live; however, because of the great number of storage devices and wide range of instrumentation for metric collection, workloads from this system provide a unique insight into work conducted on the Large Hadron Collider. Although we are not running a live version of Geomancy on the EOS system, we use workload traces from this system to determine an adequate neural network architecture for the purposes of modeling system performance.

Traces are used as a proof of concept to test out the relationship between placement features and performance features (throughput or latency). The EOS trace trained neural network is not used in the live system experiment, and any training data used to train the neural network is gathered solely from the live system's telemetry.

The PNNL BELLE II [63] workload utilizes data from the specialized particle detector from the SuperKEKB [64] collider in Japan. One BELLE II workload processes gigabytes of data from particle collisions and executes several I/O intensive Monte Carlo simulations. A Monte Carlo simulation

provided to us utilizes 24 ROOT [65] files of size from 583 KB to 1.1 GB on the BlueSky system [66] computation nodes at PNNL. The workload acts as a suite of many applications reading and writing many files individually, not as a singular application. The BELLE II workload is representative of workloads common on the PNNL systems. We created our own measurement software to measure throughput between storage devices in the PNNL provided BlueSky system, generating a workload trace that we use as training data.

The workload emulates an experiment that has run on the BELLE system using ROOT files, a framework for the Monte-Carlo simulations used by most high energy particle detectors. These simulations study the passage of particles through matter and their propagation in a magnetic field, enabling a physicist to easily simulate the behavior of a particle detector. In these read-heavy simulations, each file is accessed 10–20 times in succession. We focused on workloads at for this project since the actions all had patterns that were predefined. We want to show how Geomancy acts in an actual scientific system faced with a common workload for such a system.

4.2 Geomancy Design

Geomancy uses online learning to predict performance fluctuations of different storage devices relative to access pattern and historical load. Geomancy is trained with file access patterns containing features such as the location of the file accessed and file ID to calculate future data layouts that increase throughput. It builds a model of how an entire workload's file access patterns affects total I/O throughput of the system, including transfer overheads of moving files. If file access patterns indicates that moving some data can produce higher I/O throughput, then Geomancy instructs the system to move that data from one storage device to another. Once completed, it measures the new performance of the system, and uses any increase in the throughput of the workload as a positive reward indicating that the new location was beneficial to performance. A negative or 0 reward indicates that moving files to the selected

location will not improve performance. All new performance metrics, positive and negative, are saved into a database to record the results of a data movement. Figure 4.1 illustrates the components that embody the Geomancy system. Geomancy's neural network (*DRL engine*) and database (*ReplayDB*) are decoupled from the target system to lower Geomancy's impact on the target system's performance, and to offer high scalability. We consider Geomancy and the target systems to be separate entities, only communicating via a network, and any performance data that Geomancy requires must be sent to it from the target system.

4.2.1 Architecture

Monitoring agents collect access features from the target system and send back performance information from each I/O operation that happens at their location. We refer to the software located in the target agent that is used to monitor and control data movement as agents. Each monitoring agent only measures the performance of one storage device to allow for parallel data collection, individually communicating all collected metrics to Geomancy. When a file is detected to have been accessed, the monitoring agent flags the start of the access and the end of the access and measures the number of bytes read and written on the file. That is then used by Geomancy to calculate the throughput of the access.

The system administrator can set when Geomancy should train and calculate a new location for the data. When a new data layout is determined, Geomancy sends the updated data layout to Control Agents. Both agents execute on the nodes of a target system; however, they do not interfere with the system's activities except for instructing the target system to move data in the background and measure performance. We limit how often and how much data Geomancy can transfer at once without creating a bottleneck in the network for other workloads which is caused by the transfer cost outweighing the benefits. For this project, we had Geomancy run every 5 runs of the workload since we noticed that it overwhelmed the system if it was running more frequently. In the future, we will have it dynamically select when to move the data as workloads shift.

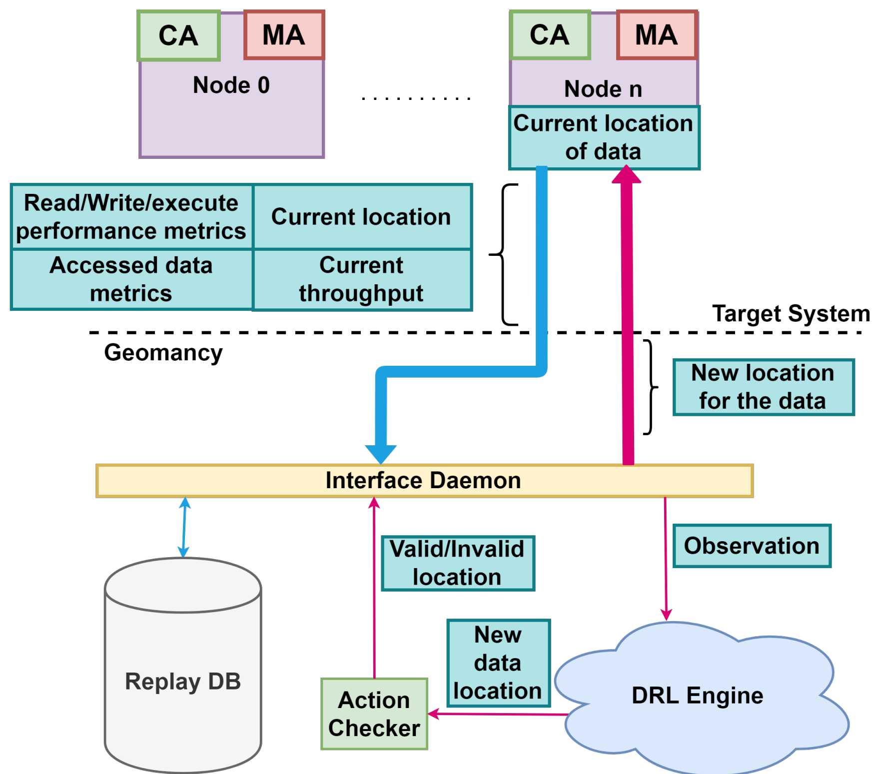


Figure 4.1: Geomancy's architecture. The control agents and the monitoring agents are located on each available location on the target system. Thick arrows represent the communication between the target system and Geomancy. Thin arrows represent the communication between the agents of Geomancy. Red arrows represent the data flow during the decision process. Blue arrows represent the data flow during the performance data collection.

After access features are collected, the Interface Daemon stores the raw performance data into the ReplayDB, a SQLite database located outside the target system. The Interface Daemon is a networking middleware that allows parallel requests to be sent between the target system, Geomancy, and internally within it. Our system captures groups of accesses as one access to lower the overhead of transferring the performance data from the target system to Geomancy's database. Overall transferring data from the target system to the database takes around 3ms on average. The ReplayDB stores new

performance data at each action taken by Geomancy, and each action is indexed by a timestamp representing the time when Geomancy changed the data layout to show an evolution of the data layout and corresponding performance.

Geomancy has an observation phase where it places the files randomly in the system to collect through information on the nodes it can accesses. That performance data is then stored in the ReplayDB. Using the data in the ReplayDB, the Deep Reinforcement Learning [67] (DRL) engine determines any updates needed to be done to the target system's data layout. For this system, we don't utilize traditional reinforcement learning but rather we use Online Learning as a way to update the system. For our data we found that 12000 timesteps for each location allowed for our model to train correctly. The ReplayDB is used for experience replay during the training for the DRL engine. The DRL engine re-trains a neural network using the most recent performance values for each location a file was stored at from the ReplayDB to calculate future values of the throughput. By Replaying past experience, the DRL engine gain insight as to how each node's performance changes with regards to the file that is stored on it. Geomancy selects the location with the highest predicted performance.

4.2.2 Online Learning

We approach the layout problem as a unsupervised online learning problem where the throughput of the system is the reward. Our neural network predicts the throughput of accessing a piece of data at every potential location it can exist. To calculate the future throughput of an access at a certain location, we model how each input feature (file location, file size, or any feature describing the action executed on the file such number of bytes read or written) interacts with other input features. Additionally, to avoid future bottlenecks, Geomancy needs to know when to change the data layout to preempt potential accesses that could cause a bottleneck. Given a large trace of throughput measurements, file locations, and transfer overheads, we use these features from the traces to train a neural network. This neural network evaluates all of the places a file can be placed, and returns each location's future performance if the

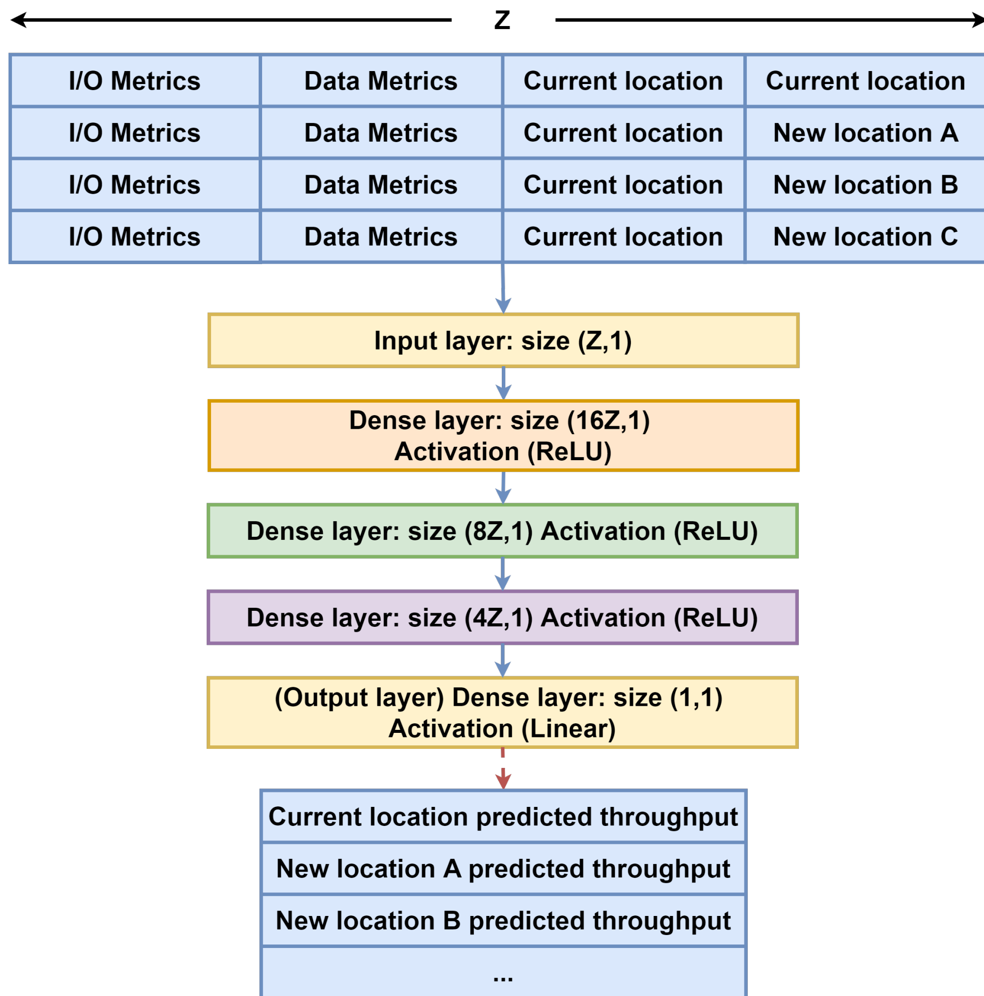


Figure 4.2: Neural network architecture. Z is the number of performance metrics used to describe an access. In the BELLE II experiment, we used 6 performance metrics. In the experiment provided by CERN, we used 13 performance metrics. Discussion of the layer sizes can be found in Evaluation.

file were to be placed there. As seen in Figure 4.2, the width of our neural network will increase as the number of performance metrics are given to it, thus allowing the ability to handle additional performance metrics.

We have first chosen to experiment with fully connected *dense* neural networks to determine relationships between input features. We use the Rectified Linear Unit (ReLU) activation function [68],

which limits outputs to be positive. This is useful when predicting throughput since throughput is greater than or equal to zero, and our predictions should be as close as possible to the target values. Because we are focusing on modeling contention, trends become important to model, which means that a linear activation function may produce comparable results when combined with ReLU.

Because any training feature can influence the behavior of another, dense networks are a useful model type that can discover such interactions. When input features vary between training cycles, the new weights that are calculated by taking the dot product of the input values and the previous weights influence the activation function. The output of the dense neural network is what the network believes is the next pattern it should expect in the next cycle. This is what we consider a *prediction* generated from a hypothesis function.

4.2.3 Modeling target filesystem throughput

Geomancy will model the change in throughput of the system during the run of the workload. This value will allow it to measure if the changes to the data layout is actually increasing the performance of the target system. Since there exist workloads that are more latency sensitive, we will explore modeling latency of the system in the future. The throughput of access number i is calculated using the following equation, with rb representing the number of bytes read, wb being the number of bytes written, ots and otms being the open timestamp (the second and millisecond parts), and the cts and ctms (the second and millisecond parts).

$$Tp_i = \left(\frac{rb_i + wb_i}{(cts_i + \frac{ctms_i}{1000}) - (ots_i + \frac{otms_i}{1000})}\right)$$

Geomancy calculates the throughput for the location chosen in the training data. This means that a batch of data contains the information of the file with every row only having the location varying between each locations the file can be located on. Using that information, the DRL engine predicts the

throughput for each row. This will then allow Geomancy to select the location with the highest value and move the data at that location. One of the rows uses the current location of the data since we wanted to include the possibility that moving the data will not improve the performance of the system.

4.2.4 Discovering Features

To model the change in throughput, we identified performance values that are correlated with the average I/O throughput of workloads running on the system. Correlated values (referred to as *features*) will directly influence or change another aspect of the system when the feature changes, and we measure correlation using the Pearson's correlation coefficient.

The EOS access logs tracks an enormous variety and amount of storage system features, and from these records we were able to narrow down types of features that are prevalent across many other systems and determine how they affect the throughput of the system. Doing so, we can identify potential features that can be used to model the throughput of the workload. Every entry in the EOS access logs corresponds to one file interaction, from open to close. Each access is described by 32 values, such as EOS file ID (*fid*) and file open time as a UNIX timestamp (*ots*). The closer the average correlation of a feature over all available inputs is to one or negative one demonstrates how positively or negatively correlated that feature is to throughput, respectively. Choosing the features with largest absolute correlation values (positive or negative) usually improves model accuracy [69]. Many features in the EOS access logs [60] are uncorrelated with changes in throughput, and training the neural network with these features may prevent the neural network from converging quickly, increasing training time and decreasing accuracy. Some features that are less correlated, such as which day the access happened, might bring valuable information to understanding how varying demand affects performance.

We identified six features from the workload traces in the EOS system using the correlation between the input features and the target feature (the throughput of the access), as seen in Figure 4.3. We also focused on features that did not require a special type of permission to access so as to allow

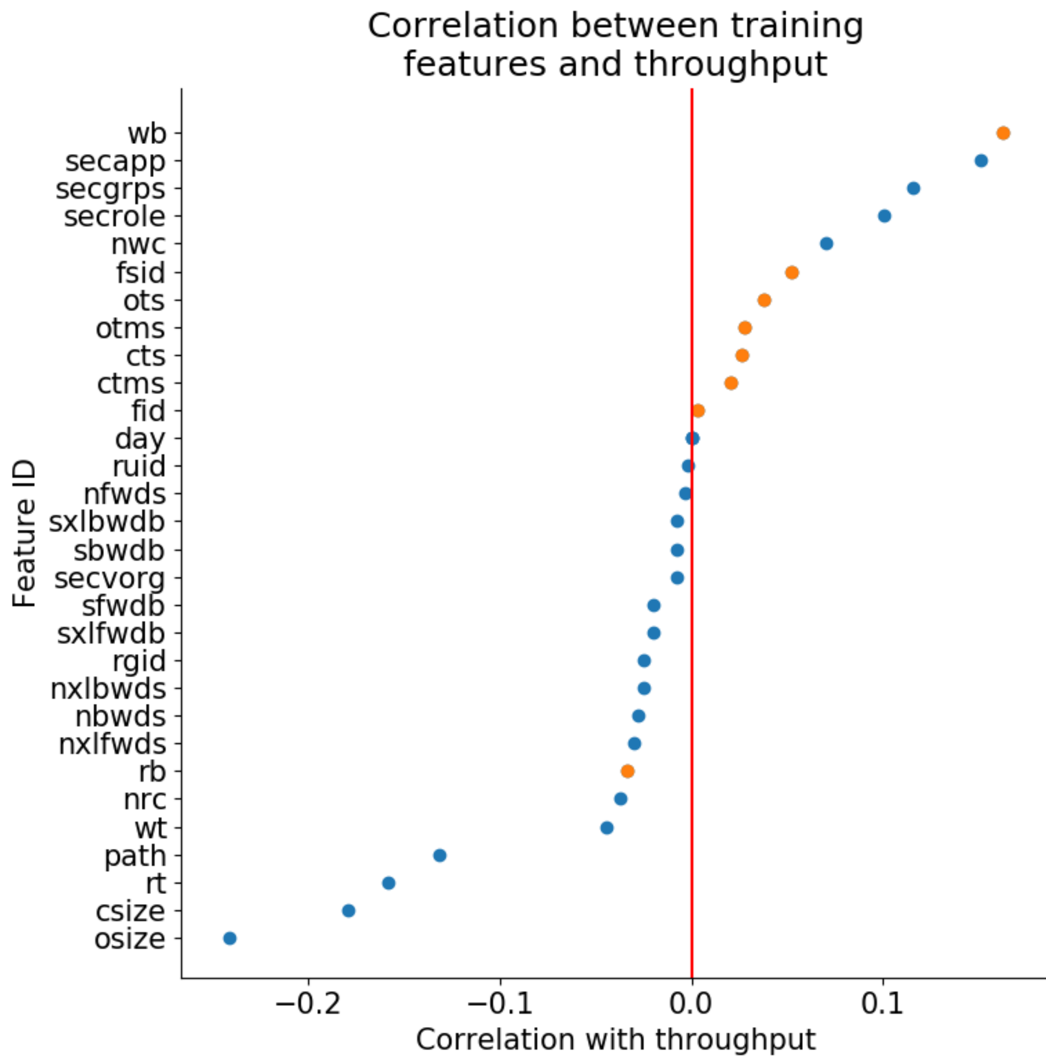


Figure 4.3: Correlation between the raw access features found in the EOS logs and the throughput. We choose features (orange) that are commonly found in scientific systems that also happen to be positively correlated.

Geomancy to run without heavily impacting the security of the target system. These features range in correlation since we wanted to experiment with features that have different non-zero correlation strength.

- Bytes read (*rb*) and Bytes written (*wb*)

- Open time stamp in seconds (*ots*) and milliseconds (*otms*)
- Close time stamp in seconds (*cts*) and milliseconds (*ctms*)
- File ID (*fid*)
- Current location (File System ID (*fsid*))
- Future location (File System ID that the file is moved to)

By using such features, we gain an approximation of how the measured throughput will impact a throughput sensitive workload. To represent the number of bytes read and written, we use the size of the data since it will be the amount of data moved by Geomancy. We also compare these selected features to those selected by WinnowML, and verified that they were the same, ensuring that the features we selected were frequently selected by WinnowML.

First, the amount of data written to a file does affect the throughput since larger pieces of data need more time to transfer than smaller files. Second, the timestamp when a file is open or closed indicates how many processes are accessing the system at a certain time, which may cause drops in the throughput. Third, the file ID, or data ID, is not heavily correlated to the throughput, but it shows that not all files are accessed in the same manner. Fourth, the location of the data on the system also affects the system since accesses vary at each node. Strongly negative correlated features such as read time (*rt*) and write time (*wt*) were not used for our live experiment. We wanted to model the access to the file independently of the action, which means that we want to capture the entire time the file is opened from the start, with open time stamp in seconds (*ots*) and milliseconds (*otms*), to the end with closed time stamp in seconds (*cts*) and milliseconds (*ctms*).

The client group (*secgrps*), client role (*secrole*), the application identifier (secapp) and the number of write calls (*nwc*) are all features that we will look into more in the future. However those identifiers are not all readily available on all systems and therefore we chose to forgo them.

The analysis of the EOS traces demonstrated that certain features common across many systems were valuable towards modeling I/O performance. By doing such modeling, we show that Geomancy was able to adequately model access patterns before we attempted to tune a real system. From there, we translated the success of I/O modeling to practical tuning of I/O.

4.2.5 Smoothing/Normalizing Features

When the DRL engine wants to update the neural network, the DRL engine requests training data from the ReplayDB via the Interface Daemon. The Interface Daemon will select the most recent X accesses which are used as values to predict the next Y values. Before the data from the ReplayDB can be used by the DRL engine for training, the numerical data is normalized by the Interface Daemon to decimal values between zero and one, and the categorical data into numerical parameters in the same range. One item of categorical data that is converted to a numerical value is the file path. To convert a file path, we assign a unique numerical index to each level of the path. Each index is combined together to form a unique number that describes one path. We considered using inodes, however having the same inode for two different files could cause problems for creation and deletion of files. Additionally, we did not use hashes since we want files located in similar locations to have close IDs to maintain a sense of locality. For example, a unique path and filename `foo/bar/bat.root` can be translated into `123` if `foo` is assigned to 1, `bar` is assigned to 2, and `bat.root` is assigned to 3.

We remove smaller variations from data in the ReplayDB by applying a moving average [70]. Because the neural network is trained, validated and tested using 12,000 entries, we need to apply some smoothing technique to mitigate outliers. In total, it takes around 6ms $\pm$ 1ms to train, validate and test the neural network using 12,000 values of 6 features on our experimental platform. Other smoothing methods such as cumulative average can be used, however they lose short term fluctuations that can indicate a rapid decrease in performance.

All requests for data contain the X most recent accesses for each of the storage devices from

the ReplayDB, thereby creating a batch. This enables Geomancy to retrain the DRL Engine with the most recent data, learning from the changing workloads and actions it took on the system. Currently, we run Geomancy at the workload level, where the workload has its own data and other workloads don't access that data. This limits the negative impact of laying out out the data in a way that heavily slows down the system to just that workload. Additionally, Geomancy has a phase where it only observes the data being moved around the system to learn how each location fluctuates in performance. During that phase, it does not control how the data is being moved in the system. As part of future work we will extend Geomancy to handle larger systems, which means we will need to expand the action checker to mitigate any negative affects of the feedback loop. The data is batched by data ID, and each batch contains performance information for the data over all available storage devices. The target values are the performance of the most recent accesses for each data ID at each location.

4.2.6 Predicting effects of file remapping

Before any predictions are made, any potential storage points that the file can be put on are refreshed and saved as a configuration file. Thus, whatever prediction a neural network makes is constrained by where the file can go. When the neural network makes a prediction, it creates a data structure that represents what the throughput I/O of a storage mount will be if a file of a certain type is moved there. For example, if a "root" file is moved to storage location A, it will have a performance X, and if the root file were to be moved to storage location B, it would have a performance Y. This prediction also encompasses if the file is not moved from its current location, with an accompanying predicted I/O throughput. Since Geomancy knows that the file is currently at location A (for example), a prediction that indicates that moving the file from A to B implies that there will be a performance gain if the file is moved.

Geomancy does not sample from an exhaustive map of where all file types can go on the system. Like a board game, it can only take a limited amount of actions from a space of actions, and

cannot move the files wholesale. Specifically, a situation where all root files from all unique storage points are crammed into another storage point cannot happen immediately, but if predicted throughput improvements indicates that this situation is most beneficial for performance, Geomancy will direct the system to rearrange itself into this configuration over time. To tackle larger storage systems of millions of files and dozens of mount points, we will need a data movement scheduler (implemented either as a second neural network or algorithm) that determines a cooldown between file movement. We have left this as future work, and we intend on implementing this as further development in improving larger and multi-user workloads.

4.2.7 Hyperparameter Tuning

Table 4.1: Model architectures

Model number	Components
Model 1	**16Z (Dense) ReLU, 8Z (Dense) ReLU, 4Z (Dense) ReLU, 1 (Dense) Linear**
Model 2	16Z (Dense) ReLU, 8Z (Dense) ReLU, 1 (Dense) ReLU
Model 3	16Z (Dense) ReLU, 8Z (Dense) ReLU, 4Z (Dense) ReLU, 1 (Dense) ReLU
Model 4	16Z (Dense) ReLU, 8Z (Dense) ReLU, 1 (Dense) Linear
Model 5	16Z (Dense) Linear, 8Z (Dense) Linear, 4Z (Dense) Linear, Z (Dense) Linear, 1 (Dense) ReLU
Model 6	(X4)16Z (Dense) ReLU, 1 (Dense) ReLU
Model 7	(X5)16Z (Dense) ReLU, 1 (Dense) ReLU
Model 8	(X5)Z (Dense) ReLU, 1 (Dense) ReLU
Model 9	(X4)Z (Dense) ReLU, 1 (Dense) ReLU
Model 10	(X4)Z (Dense) ReLU, 1 (Dense) Linear
Model 11	Z (Dense) ReLU, 1 (Dense) Linear
Model 12	Z (LSTM) ReLU, 1 (Dense) Linear
Model 13	Z (GRU) ReLU, 1 (Dense) Linear
Model 14	Z (SimpleRNN) ReLU, 1 (Dense) Linear
Model 15	Z (GRU) ReLU, Z (Dense) ReLU, 1 (Dense) Linear
Model 16	Z (GRU) ReLU, Z (Dense) ReLU, Z (Dense) ReLU, 1 (Dense) Linear
Model 17	Z (GRU) ReLU, 4Z (Dense) ReLU, Z (Dense) ReLU, 1 (Dense) Linear
Model 18	**Z (SimpleRNN) ReLU, 4Z (Dense) ReLU, Z (Dense) ReLU, 1 (Dense) Linear**
Model 19	Z (SimpleRNN) ReLU, Z (Dense) ReLU, Z (Dense) ReLU, 1 (Dense) Linear
Model 20	Z (SimpleRNN) ReLU, Z (Dense) ReLU, 1 (Dense) Linear
Model 21	Z (LSTM) ReLU, Z (Dense) ReLU, 1 (Dense) Linear
Model 22	Z (LSTM) ReLU, Z (Dense) ReLU,Z (Dense) ReLU, 1 (Dense) Linear
Model 23	Z (LSTM) ReLU, 4Z (Dense) ReLU,Z (Dense) ReLU, 1 (Dense) Linear

To determine a useful model, we compared 23 neural network architectures and report their performance in Table 4.1. We used $Z = 6$ features that we selected from the PNNL server. Each layer in the neural networks is represented using the following format: number of neurons (type of layer)

selected activation function. Architectures in bold are the two networks that performed the best out of the 23 networks in terms of accuracy when predicting future throughput of each storage point on the PNNL system. A thorough model search can reveal other architectures with better accuracy, however for the scope of the paper we limit our search to these 23 architectures. This gives us a wide range of networks to experiment with from fully dense networks to common recurrent networks.

For all models, the training set of data is represented by 60% of the available data. The next 20% (not used in training) is used in validation. The final 20% of the data is used as a test set. All three of these sets are separate sets of data that never appear in another set. Additionally all models ran for 200 epochs, and use standard gradient descent as an optimization function. As part of future work, we will validate our model selection by rerunning the model search while using early stopping to prevent any overfitting. We tested out the Adam optimizer but it ended up giving us a higher mean and standard deviation of the absolute relative error. Keeping these hyperparameters consistent across all the models ensured that we fairly compared all 23 architectures.

Since we do not assume any pre-defined relationship between the selected features, we test dense layers which enabled us to model relationships between all features. Additionally, since modeling throughput is a time series problem, we also experiment with recurrent networks. We targeted three commonly used layers: Long Short Term Memory (LSTM [71]), Gated Recurrent Units (GRU [72]) and Simple RNN (the base RNN structure).

Table 4.2: Model comparisons on predicting performance over all mounts. The time was measured using the time package in python. We started it before the model started training and ended it when the model finished training. We repeated the process for the prediction time. Additionally here we represent the standard deviation of each value to show how it fluctuates over time.

Model number	Mean of Absolute relative Error (%)	Training time (s)	Prediction time (ms)
1	**18.88** ± **16.92**	**25.66** ± **0.80**	**55.4** ± **3.6**
2	*Diverged*	24.04 ± 1.01	49.2 ± 3.6
3	44.30 ± 21.48	25.21 ± 0.39	54.4 ± 2.6
4	20.07 ± 17.77	22.75 ± 0.50	46.5 ± 2.5
5	*Diverged*	26.29 ± 0.48	60.2 ± 3.0
6	17.63 ± 15.95	40.27 ± 0.34	70.0 ± 3.3
7	17.72 ± 16.02	47.96 ± 0.45	80.6 ± 3.7
8	18.50 ± 16.42	23.82 ± 0.50	61.0 ± 2.8
9	44.30 ± 21.48	21.93 ± 0.42	54.4 ± 2.4
10	42.67 ± 22.70	21.93 ± 0.44	54.3 ± 2.5
11	42.68 ± 22.72	15.76 ± 0.66	35.6 ± 1.6
12	29.77 ± 21.64	42.61 ± 0.55	111.5 ± 3.6
13	28.67 ± 20.83	36.86 ± 0.80	96.5 ± 3.1
14	28.96 ± 22.02	23.19 ± 0.93	63.3 ± 3.5
15	24.66 ± 19.51	38.60 ± 1.25	107.2 ± 6.5
16	25.57 ± 20.33	39.01 ± 0.63	111.2 ± 3.7
17	21.72 ± 18.80	39.66 ± 1.14	113.9 ± 6.1
18	**18.77** ± **16.83**	**27.10** ± **0.81**	**78.0** ± **3.3**
19	42.70 ± 22.74	26.37 ± 0.71	77.1 ± 3.1
20	28.00 ± 22.78	25.38 ± 1.39	73.3 ± 4.9
21	42.70 ± 22.74	44.14 ± 1.48	121.5 ± 6.9
22	21.47 ± 19.40	43.76 ± 0.70	125.0 ± 4.1
23	23.81 ± 20.25	44.07 ± 1.46	126.7 ± 4.5

Diverged: A model that diverged is a model that completely failed to capture the mean and variation of the target value. Usually resulting in the same prediction happening over and over again.

In Table 4.2, we report the accuracy of all 23 models when modeling throughput on the people mount. Models 6 and 7 have some of the lowest absolute error between the prediction and target values, however they also have higher than average prediction time, 70.0 ms $\pm$ 3.3 ms and 80.6 ms $\pm$ 3.7 ms. Although the accuracy is acceptable, a high prediction time makes the latency between prediction and application of file movement unacceptably high. We can also see that model 8 and 1 have similar accuracy values and training time. For this system, we chose to go with model 1 because of a slightly lower training and prediction time. Models 1 and 18 also have some of the lowest absolute error and prediction times between all the models, plus the variance in absolute error is lower than those of the rest of the models. Model 18 has the highest training and prediction time between both models, however it also has the lowest mean and standard deviation of the absolute error between the target and predicted throughput over all the models. Models 6 and 18 have similar mean and standard deviation, however model 18 saves around 13s when training, which is beneficial we increase the number of training features. As seen, there is a delicate balance that needs to be struck between prediction accuracy and training/prediction time.

We chose model 1 since many other models diverged on one or more other storage points other than the people mount. Model 1 is the only model that correctly captures the rise and fall in throughput for all storage points, while other model architectures diverged (failed to produce useful predictions). Model 18 did similarly to model 1 on the other mount however on the USBtmp Model 18 had a Mean Absolute Relative Error (44.96% $\pm$ 21.78%) about 2 $\times$ higher then Model 1 (23.91% $\pm$ 21.66%). Since we wanted a model that accurately modeled all the available ports, we decided to go with model 1.

Table 4.3 lists the prediction errors for model 1 using each available storage point on the BlueSky system. We observe that model 1 has no worse than 43.15% mean absolute error for the files0 mount, despite many users bombarding the system with requests, with an average mean absolute error of about 17.93% over all the mounts. This means that the model can correctly capture the normal rise and fall in I/O throughput on individual devices with reasonably high accuracy. With this knowledge, we argue that model 1 is sufficient for modeling our experimental workload on the BlueSky system. We

Table 4.3: Prediction accuracy of model 1 on each individual BlueSky's storage points. Model 1 has no worse than 43.15% mean absolute error (23.62 + 19.53 max deviation from mean).

Storage point	Absolute Relative Error (%)
USBtmp	14.73 ± 12.70
pic	16.78 ± 15.14
tmp	15.68 ± 14.61
file0	23.62 ± 19.53
var	16.03 ± 12.82
people	20.76 ± 18.99

can increase this accuracy using other features such as number of write and read calls, etc. Additionally, when running on a larger system we may find that recurring networks may increase the accuracy more then using a dense network. We will be investigating this as part of future work.

In the future we will experiment with other models that can enable us to get a higher accuracy on mounts like the files0 mount, for which our model has at worst a 43.15% mean absolute error. Additionally, we expect that a user of Geomancy should tune the neural network to the system Geomancy is applied to. Here, we demonstrate how we tune the neural network to work on the performance data collected from PNNL system.

The low standard deviation of model 1 means that we will be able to readjust the prediction using the mean absolute error. To determine if we have to add or subtract $\mu_{AE} \times prediction$ to $prediction$, we can take the sign of the average relative error to indicate if most of our current predictions are under or over the target values. If the sign is positive, we are underpredicting by some amount, and vice versa. For model 1, the relative error was 2% when applied to the people mount, meaning that all predictions are adjusted by the following formula:

$$AdjustedPrediction = prediction_i \pm \mu_{AE} \times prediction_i$$

A potential barrier against scalability is the model search and hyperparameter adjustments needed if workloads become more diverse or a large number of new metrics are used for training. In this situation, a new model search, similar to the one conducted on the BlueSky system, would be required to account for the new computing environment since the environment drastically changed. We see this happening in the model search conducted for the EOS dataset and the live system metrics. We were constrained by what metrics were available to Geomancy, and thus had to restrain the metrics taken from the EOS dataset to those also available on BlueSky such that the model chosen would effectively model a target given what features were available.

4.2.8 Checking Actions

The Action Checker is a separate module that acts as the last sanity check for file movements in case permissions or availability changes in the system. The DRL engine sends a list of storage devices with their corresponding predicted throughput to the Action Checker. The Action Checker removes any invalid storage devices. Using the remaining storage devices, the model predicts the throughput of the location where each file can be moved to, including not moving the file at all. Once the prediction is done, we will move the file to the location with the highest predicted throughput.

In case all storage devices are invalid, a random movement is performed. The random movement allows Geomancy to learn more about the relationship between file movements and the changes in performance in the system. If we were to not move the files, we would not know whether or not moving it would help the performance or not. Additionally, moving the files allows for Geomancy to discover more of the target system such as new mounts.

Overall, random decisions are used by Geomancy 10% of the runs to keep an updated list of storage availability on the system and performance changes in the system. This means that as hardware, data and workloads change in the system, our system is able to adapt and change the data layout

accordingly.

4.3 Experimental Setup

Our experiments with Geomancy utilizes one computation node with six file systems based on different storage hardware located on PNNL's BlueSky system [66]. On this node, we have access to a NFS mounted home directory (people), two temporary RAID 1 mounts (var, tmp), a RAID 5 mount (file0), a Lustre file system (pic), and an externally mounted hard disk drive (USBtmp). The architecture of this system is illustrated in Figure 4.4, and we refer to those mounts interchangeably as *storage devices*. The RAID 5 storage device has the highest I/O throughput performance while the externally mounted HDD has the lowest. The NFS home directory is connected via 10 Gbit Ethernet to a shared storage server used by multiple users who conduct work that stresses the system at all hours. In particular, the home NFS storage server can have long latencies of several hours if other users run I/O heavy workloads. This interference affects the performance of the workload, and is a obstacle that a model must detect and overcome.

On this system, Geomancy is tasked to model how the placement of each file affects overall I/O throughput to a targeted workload. Maximizing I/O bandwidth to the workload lowers the time this experiment needs to run, and Geomancy must learn how workload demand, individual storage I/O, and external user demand is balanced to deliver the most I/O bandwidth.

4.3.1 Experiment 1: performance improvements

As there are many potential policies to spread files, we use a basic spread policy (evenly across all available mounts) as a baseline for this paper. The policies we compare Geomancy against are inspired by common caching algorithms, and we refer to them by the name of the original algorithm. In the algorithms below, we evenly spread the files across all available storage devices, however it is possible

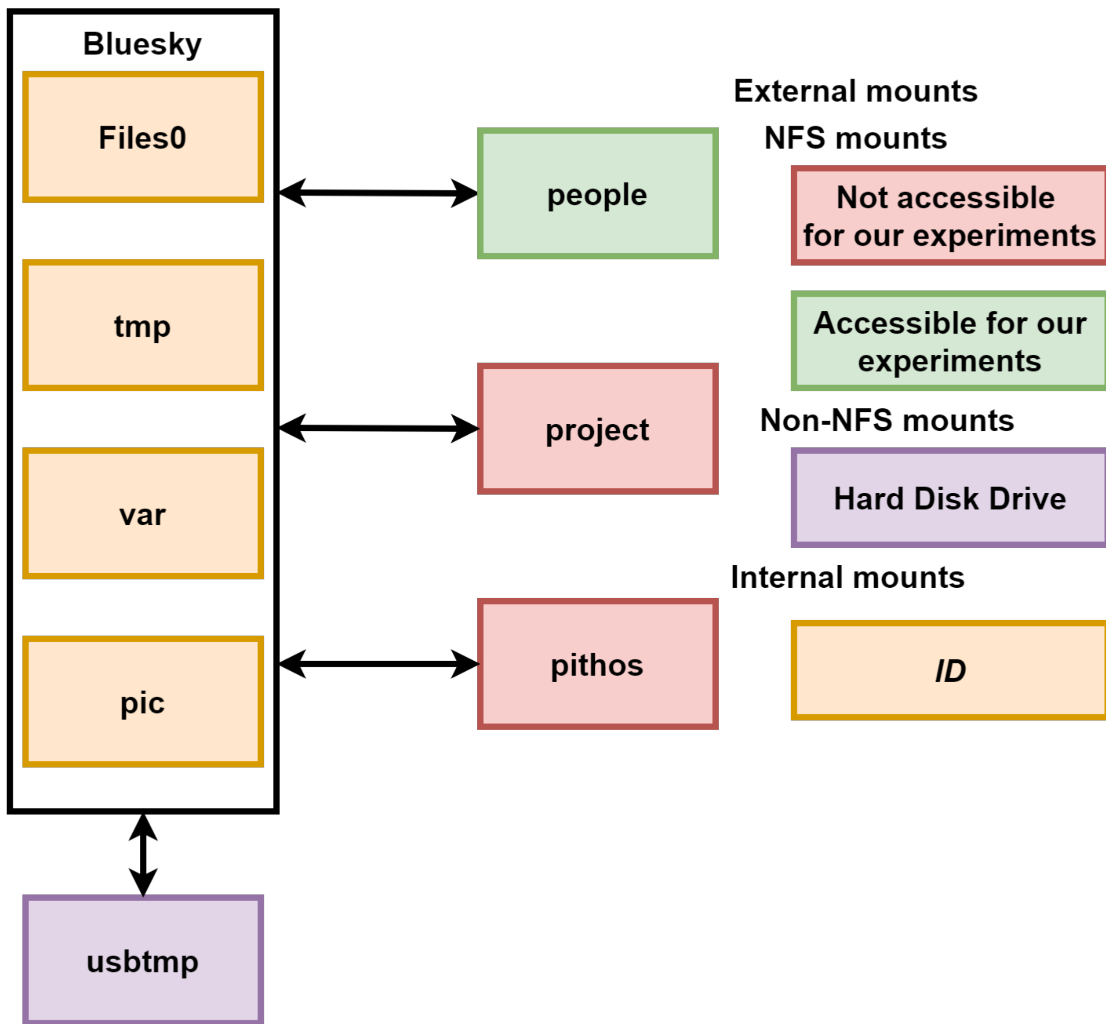


Figure 4.4: A visual representation of the storage devices on the BlueSky system. This system is shared among many other users.

to spread files based upon the capacities of the storage devices. Before any experiments are executed, BELLE II is run until Geomancy's monitoring agents can capture 10000 accesses for each file used by the workload. The data collected gives Geomancy and the other basecases enough information to start modeling the performance over time.

Our base cases are the performance of the BELLE II workload when it uses heuristics to

determine the placement of data or uses a static layout of data calculated from Geomancy or random placement. Our base case with a static layout of data is a simulation of manually tuning data layouts. In the base case, the data is stored once and does not change to account for system performance fluctuations. Hence, our experimentation is aimed to demonstrate that learning file access patterns to forecast slowdowns and respond to them is an effective strategy to improve performance.

4.3.1.1 Least Recently Used (LRU)

The effect of a LRU policy causes the least recently used files to move to the slowest storage device, and the most recently used files move to the fastest storage devices available. This experiment starts by taking the current total average throughput at each storage device using data collected in the ReplayDB. Next, all 24 files, described in Section 4.1, are divided evenly across the available six storage devices in groups of four. The group containing the most recently accessed files is placed into the fastest storage device, the second group is placed on the second fastest storage device, and so forth. In case a file was not used or the files cannot be evenly divided, the remaining files are put on the slowest node. All storage device performance information is updated between every run.

4.3.1.2 Most Recently Used (MRU)

The Most Recently Used (MRU) algorithm, as described by Chou *et al.* [73], places the most recently used files on the slowest storage devices. This algorithm has benefits for files that are scanned in a looping sequential access pattern, similar to how the BELLE II workload accesses files.

4.3.1.3 Least Frequently Used (LFU)

The LFU policy, as described by Gupta *et al.* [74], places heavily accessed files on fast nodes and lower accessed files on slower nodes. Like the previous heuristics, we start by ordering the available storage devices by throughput. Then we sort the files from most to least accessed, and the sorted files

are divided equally into groups. The group containing the most accessed files are placed into the fastest storage device, the second group is placed into the second fastest storage device, and so forth. If a file was not used or the files cannot be evenly divided, the remaining files are placed on the slowest node.

4.3.1.4 Random static and dynamic random

In random static, we randomly shuffle the locations of every file requested by the workload. The files are never moved again once they are moved the first time. We additionally compared Geomancy to random dynamic which shuffles the locations of the data between several runs of the workload.

4.3.1.5 Geomancy static placement

Geomancy static uses one prediction of Geomancy when trained with a database of past performance metrics. This prediction assigns files to their storage points, and never moves them again. This component of the experiment uses approximately 10,000 performance metrics from the dynamic random experiment, and this data is used to train the DRL engine's neural network to produce the placement.

4.3.1.6 Geomancy dynamic placement

Finally in Geomancy dynamic, Geomancy moves data every five runs of the workload. We only run Geomancy every five runs of the workload since we observed that adding a cool down period after file movement increased performance benefits. Moving files less frequently caused new placements to be less relevant, and lowered performance benefits. Also, if Geomancy moves files too often on BlueSky, the additional overhead from moving the files diminishes the performance increase achieved by Geomancy. Hence, we run Geomancy every five workloads over 9,000–16,000 accesses.

4.3.1.7 All experiments

All the base cases described above are executed individually to gather their effect on the workload, with no input from Geomancy or other algorithms. Dynamic base cases (LRU, LFU, MRU and random dynamic) are repeatedly run during the execution of the workload to rearrange data as those algorithms deem best. By updating the layout, dynamic base cases become more accurate when calculating future performance value since they can access the updated performance values from the ReplayDB.

In all experiments, we represent the progression of time using access number since the file access time window is not constant. At the beginning of each run, the workload requests the current locations of the files from a configuration file that Geomancy configures after any data movement. Currently Geomancy moves whole files in one movement; however, in the future, we will incrementally move a file to address parallel accesses. Whenever the experiment needs to read files, it will look up within this configuration file the latest locations of the files and read them from those storage devices. Throughput of the workload is measured after every I/O access. Each node has a monitoring agent that observes the file accesses on that node. Using the node's clock, the monitoring agent records when the access starts and ends. It also uses the difference between these values and the amount of bytes accessed to calculate the throughput using the following formula described in Section 4.2.4. On average, Geomancy moves between 1–14 files in one movement.

A file movement is determined if its location has changed in the entries in the ReplayDB. Using that information in addition to the timestamp of the data movement, we create clusters of data movement that happen every five runs of the workload. These clusters are used to identify how many files were moved by Geomancy during the creation of a new layout. We represent these values under the performance graph as bars that align with the dotted vertical lines on the performance graph representing the timestamp where Geomancy applied one of the data layouts it created, as seen in Figure 4.5.

Each file accessed by our workload can be placed on every location available to the workloads.

In our case, our workload uses 24 files (at most) over 6 potential locations, creating a search space of potential layouts of 24^6. This gives us 191,102,976 potential ways to distribute the data. Because our search space, we found that a dense model produced the most accurate predictions. As part of future work, we will also explore other heuristics used to model time series and compare them to our list of neural networks. Currently we have opted for the use of neural networks over heuristics since they are quicker to adapt to changes when compared to most heuristics.

At worst, with the selected model, Geomancy takes 26.5 seconds to train and predict a new layout. In total, Geomancy runs at least 4.5 times per run of the BELLEII workload (the BELLEII workload takes around 2 minutes to run). Over the 300 runs of the workload, Geomancy created at least 1350 potential layouts, of which 60 are ever applied to the BlueSky system. This is not an exhaustive search, it only applies layouts that the NN predicts will increase throughput performance. In the future, we will increase the frequency of file movements when we add another model that allows us to know when best to move individual files.

4.3.2 Experiment 2: drawbacks of placing all data on a single storage point

In experiment 2, we measure the I/O performance of each storage point if all files are placed and read solely on those points. We compare those performance metrics against a data layout proposed by Geomancy. The evolution of the performance over time allows us to compare the variation of each approach over time. In Figure 6, the data points of Geomancy and the mount points represented the average accesses throughput done by the workloads over 500 accesses. When Geomancy does not access a mount point, we repeat the past observed value for that mount in the graph. We are looking to demonstrate the learning pattern of Geomancy over time.

Although it would seem that Geomancy's data layout throughput should be the sum of all the throughputs at each storage point, this is not the case. The workload does not access all of the files at once at every single data access. For example, if we had 4 files, two storage points (A being less contended

than B), Geomancy may choose to put more files on point A. But, this does not mean that all of the files being used will come from A or B solely. It may be that one file from A is being used, and one file from B is being used, causing the average to appear to be lower than the total potential throughput of B and A for all files.

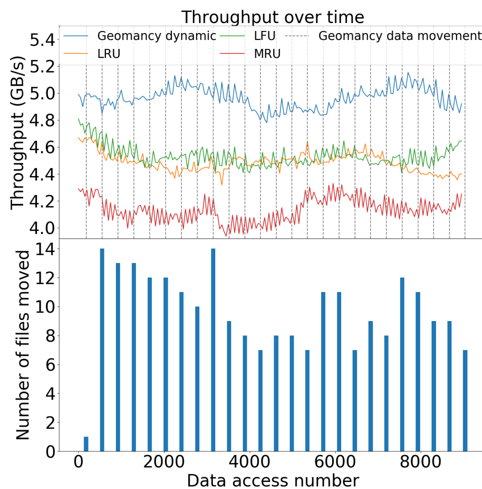
4.3.3 Experiment 3: impact on other workloads

To demonstrate Geomancy's data movement overhead on other workloads in the system, we measure Geomancy's throughput when it moves data at specific data movement milestones. The blue line indicates a duplicate workload (not tuned by Geomancy) accessing a different set of data. In this situation, the contention of storage devices has changed, and Geomancy must now respond to the changed environment. We ran a version of the workload without Geomancy tuning it alongside a version of the workload that the data layout was tuned by Geomancy, shown by the orange line. The common part of both workloads is the fact that they access common mounts, but they do not use the same data. Here again we show the average performance change over time for both the tuned and non-tuned workloads.

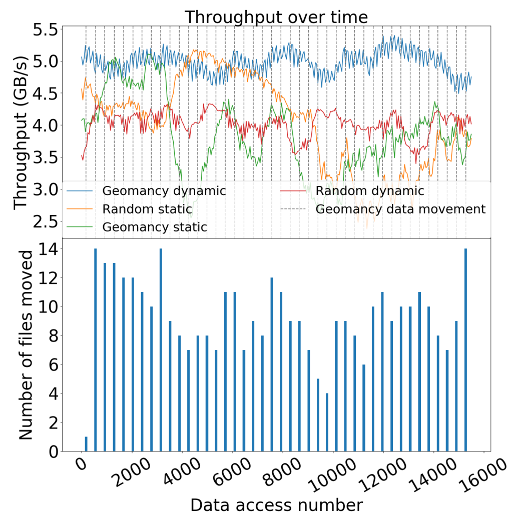
4.4 Evaluation

Geomancy outperforms both static and dynamic data placement algorithms by at least 11%, as shown in Figure 4.5. The bar chart corresponds to how many files are moved by Geomancy at that access number. We can see that most of the time Geomancy only moves a small subsets of the file to other nodes. Most static placement methods have lower performance, and only a few randomly chosen data placements challenges the performance gain made by Geomancy.

In our first experiment, as seen in Figure 4.5a, only the LFU experiment, with an average throughput of 4.46 GB/s, somewhat approaches Geomancy's overall average performance of 4.98 GB/s. We also observed that placement policies like LRU have difficulty dealing with nodes—such as the



(a) Geomancy's performance compared to LRU and its variations



(b) Geomancy's performance compared to static approaches and random dynamic

Figure 4.5: Geomancy's performance compared to dynamic and static solutions on the live system. Gray dotted lines represent data movements done by Geomancy dynamic. Size of the data that Geomancy moves ranges from 583 KB to 1.1 GB. We focused on showing the number of files moved by Geomancy to show that it does not move a significant amount of data at once, and it only moves the data when needed.

RAID-5 node—that have large imbalance between read- and write-speeds. The effect that Geomancy had upon this system was a predictive move of data to counter the fluctuations in performance before it occurred, demonstrating that learned access patterns were able to predict and prevent the slowdown.

In our second experiment, as seen in Figure 4.5b, Geomancy outperforms all naive policies. Compared to *random static* which has an average throughput of 4.01 GB/s, Geomancy has a 24% increase over the 16,000 accesses. Similarly, *Geomancy static* has an average throughput of 3.81 GB/s while Geomancy has a 30% increase over it for 16,000 accesses. Hence, even if an optimized layout is created

for a single period of time, the optimized layout is better than randomly shuffling the data.

In our third experiment, as seen in Table 4.4, we compared Geomancy's performance to the individual storage device's performance. We can see that on average the *USBtmp* storage device has a throughput of 0.63 GB/s, the *pic* storage device has a throughput of 2.05 GB/s, the *var* storage device has a throughput of 1.26 GB/s, the *people* storage device has a throughput of 1.69 GB/s, the *file0* storage device has a throughput of 7.61 GB/s and the *tmp* storage device has a throughput of 1.65 GB/s. The *files0* storage device saw the least amount of external traffic during the experiment and the *pic* and *people* storage devices received the heaviest. The *files0* storage device has the highest average throughput; however, if we were to move all files onto *files0*, its performance would suffer greatly. Geomancy was able to not only move files to *files0* to best utilize it, but also use it without deteriorating its performance into uselessness.

What we observe from these comparisons are threefold. One, by allowing Geomancy to forecast bottlenecks, we can prevent the large peaks and valleys in performance seen in static placements of data. Because contention on each storage storage device changes, we can see that the fluctuations in contention negatively impact performance on data that is stuck on contended storage devices. Hence, when data is moved before the drops in performance, overall performance fluctuates little. Two, there is significant performance improvement over never moving the data, or only manually placing the data once. As we have hypothesized, an ideal placement of data at a certain period of time will not be ideal later during a workload's execution. Thus in all static placements, they perform worse overall, and they fluctuate in performance. Three, the mechanism to improve performance does not require moving all of the workload's files at once. At most, 14 files were moved every five runs of the workload, which is a minimal load upon a network that can transfer thousands of such files every second. Therefore, there is no need to decide with heuristics which files should be moved.

Unlike the other approaches that strictly use the accesses to the data as information on where to put the data, Geomancy uses time, location, and per storage device performance to gain a more complete

view on performance. Doing so, Geomancy is able to adapt the layout as the performance of the system changes and other workloads are started. Approaches such as LRU, LFU or MRU that only use whole-file frequency for placement suffers from bottlenecks or creates placements that work for only brief periods of time.

4.5 Overhead Study

Table 4.4: Performance and utilization of storage points available to Geomancy. In this table, we demonstrate the variation of the throughput at every mount and their respective usage observed by Geomancy. In some cases, the standard deviation, showing the variation of the performance, does surpass the mean, indicating that the performance varies significantly over time. Additionally, it shows that in many cases the mounts are contended, and when they become less contended the throughput shoots up. This creates an extreme difference in performance.

Storage point	Average throughput (GB/s)	Average usage (%)
USBtmp	0.63 ± 0.47	0.1
pic	2.05 ± 3.85	0.3
tmp	1.65 ± 3.44	21.2
file0	7.61 ± 13.73	64.8
var	1.26 ± 2.81	6.3
people	1.69 ± 3.46	7.4

To demonstrate Geomancy's data movement overhead, we measure Geomancy's throughput when it moves data at specific data movement milestones. In Table 4.4 we can see that even though file0 has higher potential throughput, it also has higher deviation in its performance. Geomancy was able

to lower this inconsistent performance by spreading files across other devices. The standard deviation shown in the table seems to go below zero, but that only indicates that at some times the transfer may halt due to contention.

In Table 4.2, we showed that the prediction overhead of our selected neural network was on average 55.4ms and the training overhead was on average 25.6s when the neural network was trained using six features. When training our neural network architecture, on a dedicated machine using a TITAN Xp GPU, with 13 input performance metrics selected from the CERN EOS logs, our neural network takes 23.1s to train and 48.2ms to predict future placement on average. These values were calculated the same way as the timing values reported in Table 4.2. The predicting overhead can be mitigated by having the prediction be done in parallel with the target system which means that the perceivable overhead of Geomancy is only visible when the prediction is sent to the target system.

Figure 4.6 shows the effect of Geomancy when the system receives a sudden change. The blue line indicates a duplicate workload (not tuned by Geomancy) accessing a different set of data. In this situation, the contention of storage devices has changed, and Geomancy must now respond to the changed environment. As seen, Geomancy was able to respond to the changes, and is working to restore performance. In some occasions, such as around timestep 20000, Geomancy had the added benefit of increasing the performance of other workloads running in parallel by lowering the performance of the workload it was tuning for a brief period. This was not the initial goal of the online learning but it does show a larger impact of running Geomancy for other workloads running on the system.

4.6 Related work

Geomancy positions itself as an unsupervised and generalizable tool among a growing number of data placement modeling techniques. In production situations, standard approaches such as algorithms and heuristics begin to show brittleness when accesses suddenly change. Our tool reacts faster than

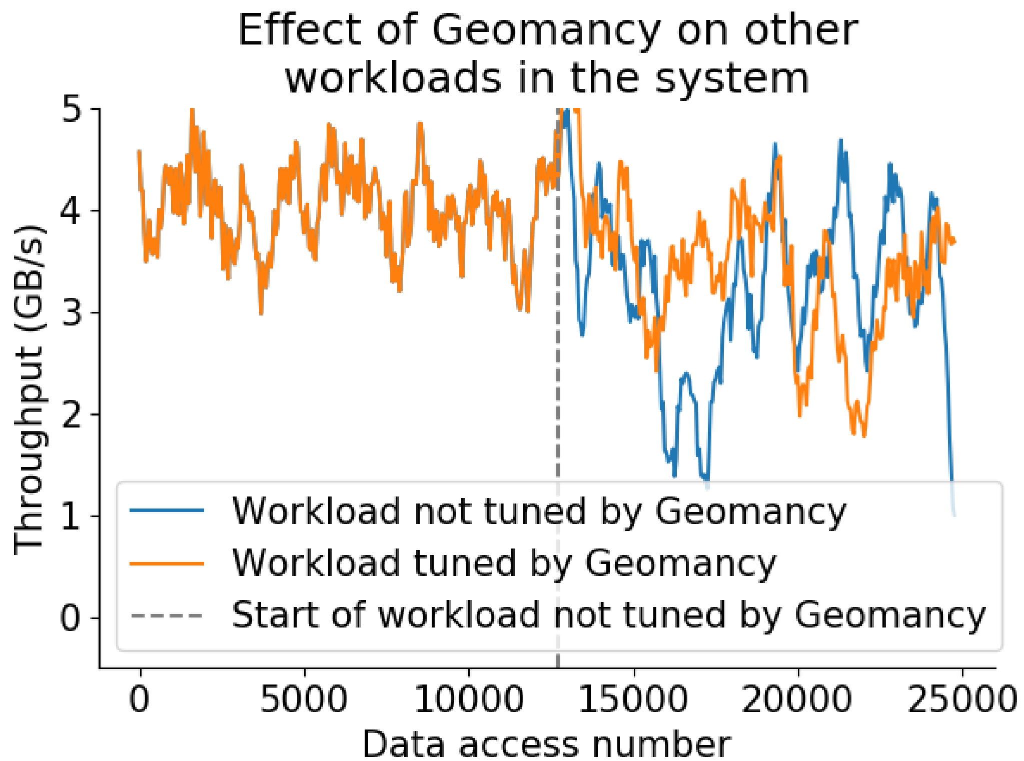


Figure 4.6: If a new workload is started in parallel to running workloads, Geomancy is able to actively learn how to place data according to this new change. Doing so, Geomancy is able to respond to the changes and attempt to push performance back to what it once was.

standard heuristic approaches to drastic changes in performance. Thus stabilizing the performance much more then other approaches.

4.6.1 Static approaches

Early work in data layouts involves distributing data evenly across a majority of hardware pieces present in systems. MMPacking [75] utilizes replication and a weighted scheduling algorithm to distribute video files across N servers, achieving fairness by making every server have at most $N-1$

copies of the same video. Randles *et al.* compared three different load balancing techniques on experiments set up on Repast.net [76]. Their experimental *honeybee foraging allocator* outperformed random walk and active clustering allocators if the system contains a diverse set of nodes. They concluded that finding the best trade-off between several of their discussed algorithms is needed, and finding a way to switch between algorithms will benefit the system as a whole. Nuaimi *et al.* surveyed several other load balancing algorithms [77], and concluded similarly about the trade-off situation of different algorithms.

4.6.2 Applying dynamic solutions to tuning system performance

Model-less, general purpose approaches usually treat the target system as a black box with knobs and adopt a specific search algorithm, such as hill climbing or evolutionary algorithms [78–80]. ASCAR [81] directly tunes the target system and can automatically find traffic control policies to improve peak hour performance. These search-based solutions are often designed as a one-time process to find the efficient parameter values for a certain workload running on a certain system. The search process usually requires a simulator, a small test system, or the target system to be in a controlled environment where the user workload can be repeated again and again, testing different parameter values.

Li *et al.* designed CAPES [82] (Computer Automated Performance Enhancement System), which demonstrated how neural networks can be used to enhance system performance. Our approach resembles the one used in that work, in that it uses system performance metrics to update a target system and re-trains a neural network to produce a prediction. Unlike CAPES, however, it observes and learns from the executing workload, and uses the observations to propose data layouts that improve the target system's performance.

4.6.3 Automated data placement techniques

Other approaches have been taken to create dynamic caching for large scientific systems such as the one described by Wang *et al.* [83]. They created a data management service, Univistor, that pro-

vides a unified view of the available nodes in the system. In cases where a fast NVRAM cache is available, existing software may not be aware of how to best utilize it. Like Geomancy, Univistor analyzes the workloads running, and finds chances to cache data on a fast burst buffer to increase performance. In contrast, however, is the requirement of a tiered storage cluster with performance strictly going up as storage densities decrease. Subedi *et al.* [84] proposes another approach. They created Stacker, a framework that achieves similar data management between components of a workload in a scientific system using n-gram Markov models to predict when a data movement should occur. Like Univistor, it too utilizes a fast cache to increase performance by staging and unstaging hot data.

Geomancy presents itself as an analogy to such approaches, yet does not require the existence of a fast burst buffer to increase performance. In our experimentation, we have varying levels of performance, but no one storage layer dedicated to caching. We also do not interact with striping, such as the approach done in Rush *et al.*'s work [85]. Although smart striping techniques do increase storage performance, it has moderate gains in performance and comparable dampening effects on performance variation, yet requires modification of the file system. LWPtool [86] provides a similar service, and also adds tools to change a workloads code to point it to a new data's location. Like that of Geomancy, workloads are instructed to use the new data's location, however this approach requires rewriting running source code.

4.7 Conclusion

With the results of our studies, we show that re-distributing data among many storage devices of differing capabilities can increase the performance of the workloads that run on the system. Experimentally, we demonstrate inter-workload congestion reduction and increases in overall throughput from 11% to 30%. With this knowledge in mind, we can further improve the benefits of re-laying out data by reducing the overhead of applying new data layouts.

Chapter 5

Identifying modeling techniques to improve network routing at the switch level

Now that we can determine where to place the data, we turn our attention to controlling how the data is routed to the new location. Scientific networks, such as the Caltech Tier 2 network, receive consistent access from researchers all around the world. For these systems it becomes difficult to have a global controller that observes the entire system, since using such a controller may create additional overhead to the system. The added overhead of such a controller comes from the additional dataflow requested between the controller and the switches. We develop Diolkos, a system that can dynamically reroute data flows through new ports at the switch level. Using this system we explore how different model types affect the prediction accuracy and the resulting performance improvements when applied on a scientific network. Not every type of performance modeling technique will have the same effect when applied to performance data. We want to tune the network routes to preemptively reroute the data before network bottlenecks happen. In this thesis, we evaluated multiple modeling approaches to predicting changes in performance on network switch ports.

5.1 Design

Diolkos, shown in Figure 5.1, is a system that dynamically changes the ports taken by data flows in the system depending on forecasted network throughput. Diolkos runs on the network switches to reduce the overhead of gathering relevant training data, focusing only on what is visible within the switch. To prevent switches from working against each other, each switch has a map of where the hosts are in the system. At the site level, flows are tagged with their source and destination host address, and the switches use this tagged information and their own map to make sure that the selected port will not cause an infinite loop in the system.

Every r_c seconds, Diolkos collects the number of bytes that has flowed through each port of the switch, referred to as throughput. These measurements are stored in a database for future usage to train the prediction model. Using the dataset, our model can learn how the traffic changes over time. The model is trained in parallel to throughput metric gathering, and generates an updated port selection for each switch every $(N + O) \times r_c$ seconds, where r_c is the collection rate that changes depending on the network. N and O are part of a set of hyperparameters detailing prediction output, as described later in Section 5.2. The model generates a new routing policy that is applied if the current port has a lower predicted performance compared to the port selected by Diolkos. Doing so, data flows only through the ports that the model calculates to have the highest forecasted performance value.

Figure 5.1: High level design of the Diolkos system and target system. The outlined arrows correspond to data exchanged between the network of switches and Diolkos. The thick dotted arrow is the flow taken by the new calculated port by the model. The thin arrows are data paths between the components of Diolkos. The thin dotted arrow is the path taken by data during the training round. Undotted thick and thin arrows are the paths taken by data collected from the target system.

5.1.1 Training data

The training data used to train the Diolkos model is the throughput at each port of the switches and the time this data was taken. The throughput of port j of switch i at time t is calculated using Equations 5.1 and 5.2, where BR_i and BT_i are the number of bytes received and transmitted respectively at port j of switch i, and T is the measurement period in seconds. Each throughput value is referred

to as a *snapshot* of the switch's performance, measurements of the throughput at each port taken at approximately five-second intervals. We sample every five seconds to stay in line with other related work about network sampling [87, 88]. We limit the rate to avoid overburdening the switch while gathering enough data.

$$NB_{i,j,t} = BR_{i,j,t} + BT_{i,j,t} \tag{5.1}$$

$$TP_{i,j,t} = \frac{NB_{i,j,t} - NB_{i,j,t-T}}{T} \tag{5.2}$$

Diolkos stores the training data in a database located on the switch. This database is used to replay the data to the model during training. If the model is trained using only recent observations, it would not be able to take advantage of the trends visible over time, decreasing prediction accuracy. Training data, collected from the Diolkos's database, is formatted as seen in Figure 5.2.

For all models created, we specify a set of hyperparameters: M, N, and O. M is the amount of data per batch of training data, N is how far into the future the model predicts, and O is the number of timesteps the model predicts. The choice of M,N, and O is crucial to have a functioning system. Predictions created for timesteps too close to the training data may not be relevant once applied to the system because predictions need to be far enough in the future to allow for the prediction to be sent to the network. Further, Diolkos should calculate predictions using a reasonable amount of training information which is determined by the M value. We address the effect of M, N, and O on the accuracy of the prediction in Section 5.2.

All training data is split by time into three partitions. The first 20% of the data is used for testing the model, the second 60% of the data is used for training, and the third 20% of the data is used for validation. This data is also normalized to prevent noise from adversely affecting training. In addition, a moving average of size 75 is applied to all training data to reduce the affect of noise. The moving

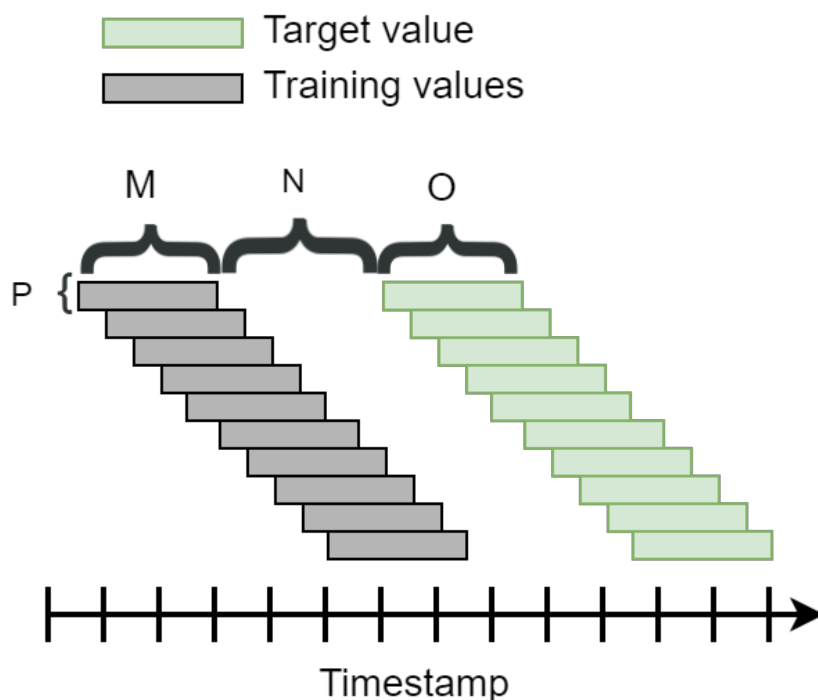


Figure 5.2: We define M, N and O to identify how much training data is used, how far into the future we predict, and how many timesteps should be predicted. P represents the port number on the switch.

average window represents approximately six minutes and 25 seconds of real-time data collection.

All training data is stored in the *ReplayDB*, a database that stores the historical throughput data. It serves as a mechanism to replay past performance fluctuations as training data for the prediction engine. We cap the ReplayDB at approximately 20,000 entries (2.3 days of real life performance values), which allows the models and heuristics to have a relatively low error rate. Any older entries are not considered during training.

5.1.2 Prediction Engine

The *Prediction Engine* uses either a model or a heuristic to predict future throughput fluctuations. We use throughput at each port as the target of the regression for training the models and

heuristics. We compared techniques that are commonly used for time series modeling such as models like Dense models, Simple Recurrent Neural Networks (SRNNs), Gated Recurrent Units [72] (GRU) and Long Short Term Memory [71] (LSTM) models. We have chosen these neural networks for their ability to model interactions between older trends and current trends, producing high accuracy predictions. Although SRNNs have fallen out of favor for GRUs and LSTMs, we include them as a base case for recurrent model experimentation. The output layer of the models contains the same number of neurons as there are ports. All the neural network models we experiment with use the Adam optimizer to update the weights of the model during back-propagation.

We also implement three heuristics that can also make throughput predictions: Linear Fit, Exponentially Weighted Moving Average [89] (EWMA), and Holt-Winter [90]. Linear fit provides is a simple solution that has low computational overhead. Our next approach is EWMA, which uses a sliding window to record the change of average over time. Our final approach is Holt-Winter which uses a combination of the sliding window of EWMA with seasonality data present in the training data. Both our models and heuristics are online models since they apply changes to the system in real-time.

Diolkos predicts the I/O throughput value at every port on a switch for $N + O$ timesteps in the future. The selected information lets Diolkos select the port with the highest average throughput value over all $N+O$ timesteps to route data through. It ensures that the selected port has the highest throughput on average over the next few timesteps while keeping data flows from thrashing between different routes. We can also set $N + O$ to be how long a flow routed through a port should persist before it is able to change. In practice, rerouting flows must be done in regard to when the flows can safely changed while limiting packet loss.

5.1.3 Port selection

Once the neural network predicts the throughput of each port, Diolkos selects the port with the highest throughput. The selected port is used to reroute the current flow for the next $(N + O)$ seconds.

Diolkos ensures that the ports selected are chosen based upon historic data about how traffic flows. We use a hyperparameter β, which is the threshold for how low throughput at a port can be before it is considered to not have any traffic. Before training is done, certain ports may have never been utilized. Hence, when Diolkos tries to flow data, it will never predict that port to have any throughput because it was never included as training data. To prevent this, if a port falls below β it is considered over the high throughput ports to enable Diolkos to start testing the ports that have never had data flow. Hence, the usage of β differentiates ports from being bottlenecked and from being unused. The value of β changes depending on the network and the current load of the network. In future work, we will develop a way to dynamically change this value based upon the workload.

5.1.4 Site-level coordination

The path selection module uses a network map that contains the location of the host and switches. During port selection, we ensure that Diolkos produces a viable prediction that does not cause a loop, even if the first prediction may be predicted to have higher throughput. For example, in a mesh network as described in Figure 5.6, if a flow goes from host one to host two, ports 3 and 2 are always prioritized over port 1 since routing the flow through port 1 causes the flow to go back to the source host.

All flows are tagged with information about where it originated from and its destination. In case a prediction tries to cause a loop, Diolkos looks through any other available port and chooses the one that has the next highest predicted performance that does not cause a loop. We assume that Diolkos eventually finds a port that data can flow through without causing a loop. In future work, these predictions can also consider the destination switch value as a training parameter to lower packet loss.

5.1.5 Security challenges

In the scope of our paper we focus on performance, however we acknowledge potential security implications of our technique. Changing the ports taken by flows increases the attack surface of the

network. Distributed attacks, such as distributed denial of service attacks and internet worms, are at increased effectiveness since data is flowed with maximum parallel usage of switches. We have identified several techniques that could mitigate this weakness: Chanho *et al.* [91] developed a technique that filters out harmful packets through the use of a packet buffer that compares the information from incoming packets to valid preset values. This buffer can be added to the Tofino switches to identify harmful packets. Zhang *et al.* [92] designed a framework to defend networks from distributed attacks, and their approach [93] is a decentralized framework that can be applied at the switch level. We envision the application of these techniques to lower the issue of the increased attack surface. Singh *et al.* [94] developed a sifting method to limit the propagation worms and viruses in a network. If applied at the switch level, we can secure individual switches while limiting the performance impact on the network.

5.2 Model and hyperparameter search

To experimentally demonstrate the Diolkos project, we begin with a model accuracy study and a hyperparameter study to determine which technique accurately predicts future performance changes. We first select one model and one heuristic based upon each algorithm's predictive accuracy when modeling port throughput on a real switch. Then we determine a set of hyperparameters M, N and O that allow these models to perform well.

5.2.1 Selecting the Neural Network Model

Prior to simulation, we determine the heuristic and neural network model with the lowest predicting error when calculating throughput on a network switch with static M, N and O values. The selected static values of M, N and O are $M = 5$, $N = 1$, and $O = 1$. If each port of a switch has the possibility of affecting another, then our approach should use those interactions to accurately model the network throughput through that switch. To measure a model's effectiveness, we measure the Mean of

the Absolute relative Error (μ_{AE}) and the Standard Deviation of the Absolute relative Error (σ_{AE}). If the predictions ever grow in error, it indicates that the model is impacted by noise. We calculate Absolute relative Error (AE) between predicted value P and target value T and at timestep i as: $AE_i = \frac{|T_i - P_i|}{P_i}$

Prior to training, data is collected from ports 0, 4, 8, and 12 from the Caltech Tofino switches in the format as seen in Figure 5.2. During data collection, we monitor the traffic flowing through each one of these ports as the number of bytes transmitted. We sample every five seconds until we obtain 20,000 snapshots of information. We attempt to predict the throughput of a singular port (e.g. 0, 4, 8, or 12) using data from the same port or data from all ports. For example, one particular experiment uses data from only port 4 to train a single dense hidden layered model to predict throughput at port 4. Next, we use data from all ports to predict port 4's throughput again. We do this to quantitatively prove that port interaction exists: if there is increased prediction accuracy when using data from all ports instead of one port, then we must use all port data as training data.

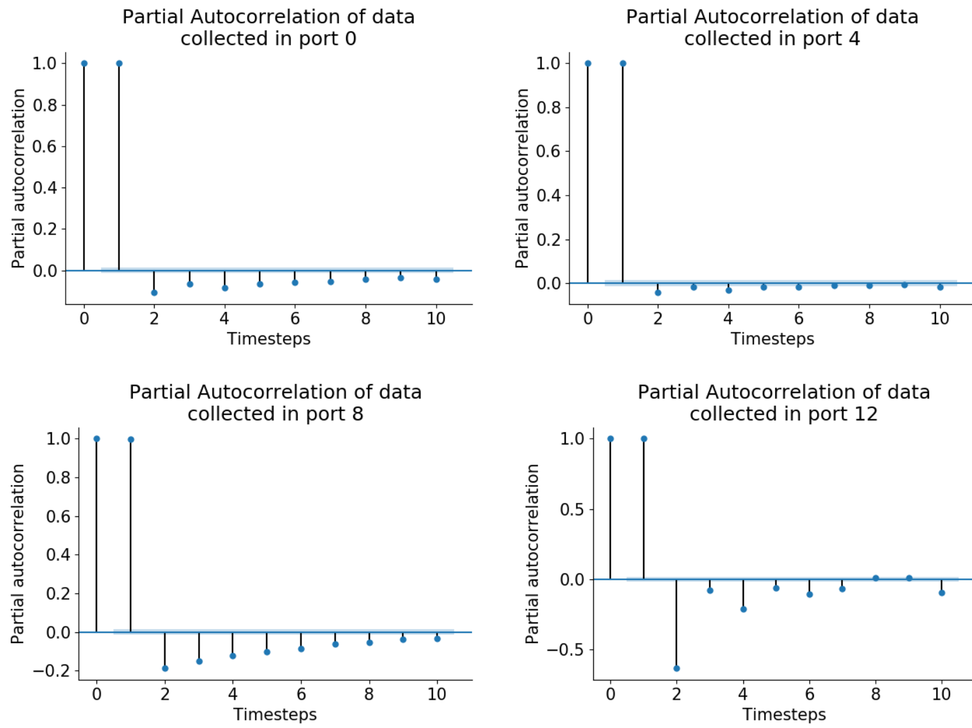


Figure 5.3: Auto-correlation for ports 0, 4, 8 and 12, values close to 0 and in the blue region indicate that data at that lag has no correlation.

5.2.2 Structured and unstructured data flows

To determine if neural networks are required to model scientific network data flows, we calculated the partial autocorrelation for the throughput of each port on the Tofino switches in the Caltech Tier 2 network. We calculate many autocorrelation values with a varying amount of lag, a summary of the relationship between data at one time step and previous time steps. Doing so, we can determine if at any point the autocorrelation becomes zero, an indicator that the data flowing through the port has no noticeable patterns and cannot be predicted using a neural network.

When graphing out the autocorrelation of each port, we observe that ports 0, 8, and 12 have high autocorrelation values, while port 4 has near zero autocorrelation, as seen in Figure 5.3. This

indicates that the type of data flowing through ports 0, 8 and 12 have structure, and thus can be modeled with neural networks. Port 12 has lags in the data that are uncorrelated: lags values of 8 and 9 are completely uncorrelated. However, port 0 and port 8 never reach an autocorrelation of 0, indicating that any lag less than or equal to 10 will have some structure for those ports. With this finding, we argue that neural network models can effectively model the data flows on ports 0, 8, and 12. For the sake of brevity, we choose to focus on port 8 for experiments 1 and 2.

5.2.3 Neural network model and heuristic selection

Modeling technique	μ_{AE} (%)
Dense 1HL A	$\mathbf{0.45 \pm 0.30}$
GRU 1HL A	0.45 ± 0.32
Dense 0HL A	0.47 ± 0.34
SimpleRNN 1HL A	0.47 ± 0.35
GRU 2HL A	0.55 ± 0.40
EWMA	$\mathbf{0.54 \pm 0.41}$
Holt-Winter	0.63 ± 0.45

Table 5.1: μ_{AE} for port 8 for the 7 models with the lowest prediction error out of the 19 tested. Our selected model is in bold. EWMA and Holt-Winter perform similarly, however we have chosen to compare our Dense network against EWMA since EWMA had the highest accuracy.

To determine the best model to apply, we take a look at accuracy measurements in Table 5.1. For this table and all following graphs, *HL* represents the number of hidden layers, *I* represents models trained with individual port data, and *A* represents models trained with all port data. Heuristics such as linear fit often mispredicts by a significant margin, resulting in a high μ_{AE} and σ_{AE}. Among all models,

the Gated Recurrent Unit model with one hidden layer trained with all port data and the dense model with one hidden layer trained with all port data performed the best with similarly low μ_{AE}. σ_{AE} for both of these models are also equally low, however the dense model has slightly lower μ_{AE}. Hence, we use this model, henceforth referred to as the *Dense* model, to determine optimal hyperparameters for it prior to applying it to a simulated network. As for the heuristics, we use the EWMA heuristic since it is the highest scoring heuristic.

5.2.4 Hyperparameter search, determining M, N and O

We determine a suitable combination of M, N, and O because a practical usage of Diolkos requires that the prediction be accurate and timely enough to implement. By increasing M, we increase the size of the training data batch, potentially increasing accuracy at the expense of training time. Increasing N gives more time to implement the result of a prediction at the expense of prediction accuracy. Increasing O increases the number of predictions generated in one cycle, allowing the system to cache predictions, also at the cost of prediction accuracy. A network cannot conveniently wait for a prediction that could maybe (at best) increase throughput. We can make predictions about throughput more useful if we predict far enough into the future, or generate enough predictions so the network can cache predictions.

We vary M, N, and O to discover each model's predictive performance (μ_{AE}) as the number of inputs change, how far into the future the model predicts, and the number of predictions the model creates. This prediction is checked against real values, and an error is returned for each predictive model. The higher the μ_{AE}, the lower we consider each model's accuracy for that M, N, and O combination. Doing so, we can optimize M, N, and O per model to suggest a hyperparameter setting per modeling technique that maximizes accuracy. We expect each model to have a different optimized M, N and O.

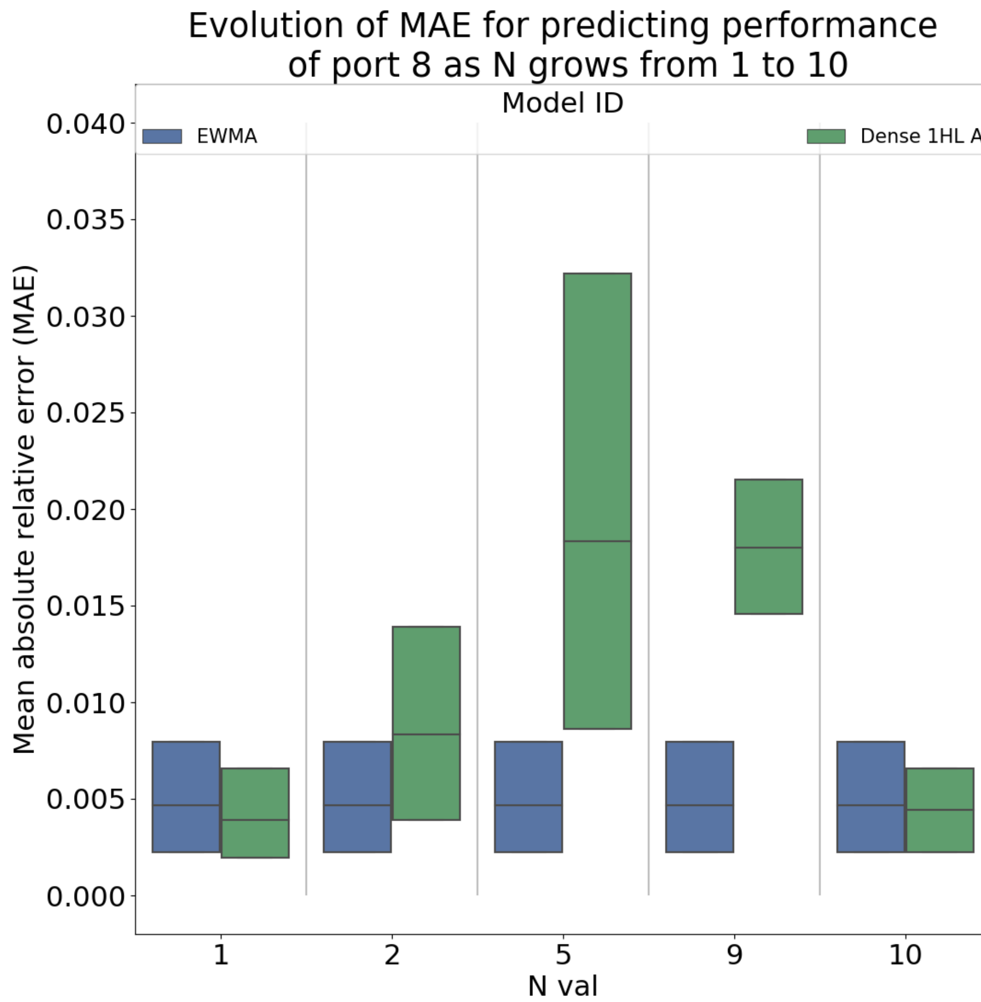


Figure 5.4: Average median and lower to upper quartile values of the absolute error across all the models as N increases from 1 to 10, M=5 and O=1 for port 8 (structured data). As N increases, EWMA does not change since the amount of data for the average stays consistent. In contrast, the Dense model focuses too much on the additional variations present in the data, hence as N increases the noise present in the data causes error to increase for certain values of N.

As N increases, as seen in Figure 5.4, we observed that for EWMA the median of the absolute

error does not change at all. We can explain that by the fact that the overall traffic does not vary much on port 8. Thus EWMA was able to capture the mean which was sufficient. Since the Dense network was trained over all the ports, it was impacted by the noise present on the other ports. Additionally, we can observe that reaching $N = 10$ and $N = 1$, the Dense model's accuracy surpasses the accuracy of EWMA. When looking at the corresponding lag for port 0 and 12, we can see that at lag 10 and 1 they both have high negative or positive correlations. For simulation, we choose $N = 1$; however, the result would be similar whether we chose $N = 1$ or $N = 10$. With every timestep representing 5 seconds of realtime, a value of 1 tells us that Diolkos has 5 seconds to apply a routing decision on the Caltech network, plenty of time to propagate a routing change on our target network.

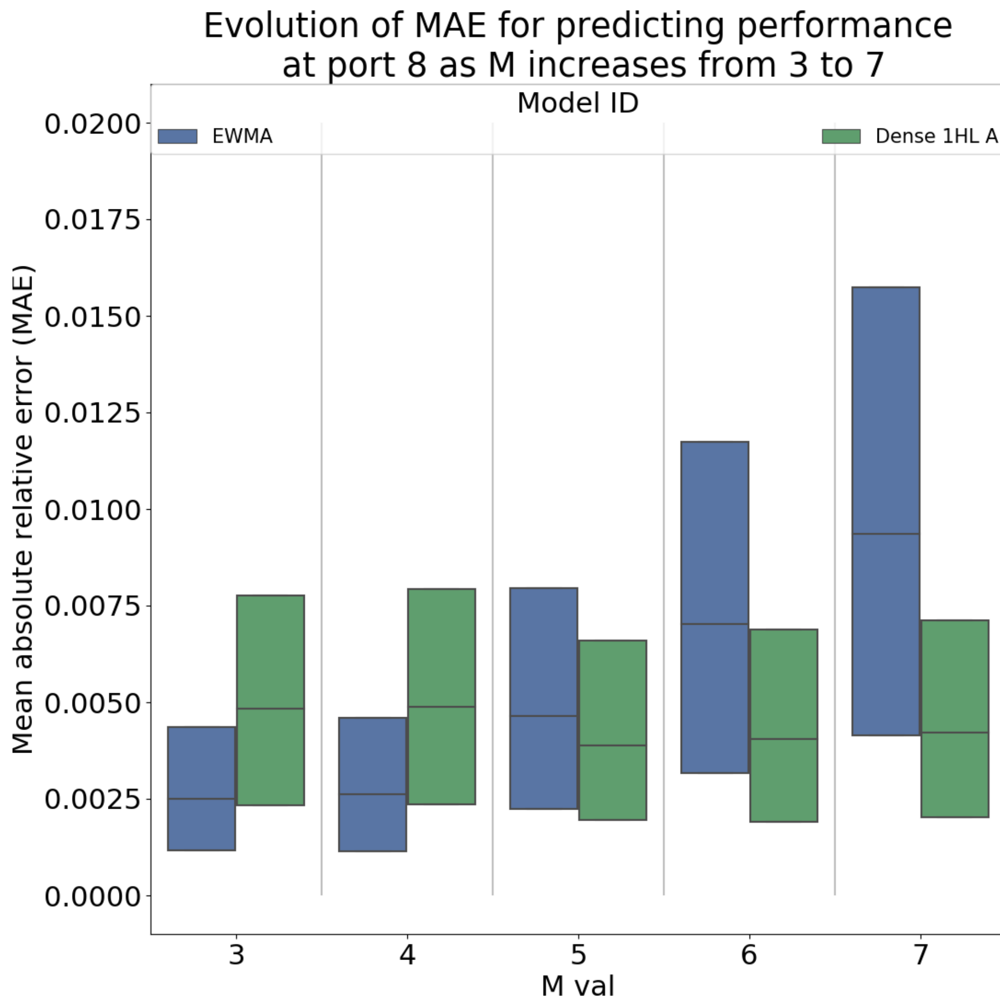


Figure 5.5: Average median and lower to upper quartile values of the absolute error across all selected models as M increases from 3 to 7, N=1 and O=1 for port 8 (structured data). As M increases, EWMA's median of the absolute error increases exponentially and the Dense model's median of the absolute error decreases slightly.

From the values $M = 3$ to $M = 7$, as seen in Figure 5.5, prediction error for EWMA increases exponentially since the more data it uses for training the more noise gets added. The extra noise causes

the mean calculated by the EWMA approach to misrepresent the true future value. At $M = 5$, the Dense model has the lowest median of the absolute error value, and any values less or more then five increases the median of the absolute error value. Another benefit of $M = 5$ is that both EWMA and the Dense model have similar error rates. Since we are looking to evenly compare both approaches, we choose to use $M = 5$ as the number of timesteps per batch of training data.

O value	μ_{AE} (%) of EWMA	μ_{AE} (%) of Dense
1	0.54 $\pm$ 0.41	0.44 $\pm$ 0.30
2	78.94 $\pm$ 97.56	76.90 $\pm$ 40.94
3	90.16 $\pm$ 89.10	92.56 $\pm$ 69.34
4	101.92 $\pm$ 95.99	106.23 $\pm$ 64.92

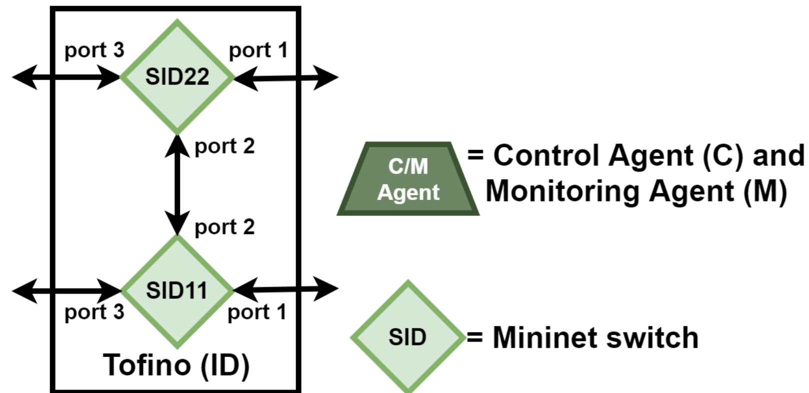
Table 5.2: Beyond values of O=1, neither approach produces predictions that are accurate. Thus, every prediction cycle should produce at most one prediction.

Unlike M and N, there is only one useful value of O ($O = 1$), as seen in Table 5.2. Beyond $O = 1$, the μ_{AE} grows too high to allow for useful predictions, and the σ_{AE} renders any model unreliable. Worse, both the Dense and EWMA models begin to occasionally predict values outside of what the switches are capable of. Although it would have been ideal to have multiple predictions per prediction cycle (to cache routing changes, for example), no model effectively predicted several beneficial routing changes in one cycle. Fortunately, as caching predictions is only a potential computation time saving measure, Diolkos can function normally with one prediction made per cycle.

The strongest values of M, N, and O are $M = 5$, $N = 1$, and $O = 1$ for the Dense and EWMA models. The selected hyperparameters maximizes the accuracy for each model given an adequate amount of training data per batch, the distance into the future we predict, and only one prediction made per cycle. With the aforementioned numbers, we set all models to have the same M, N, and O values for

our simulation.

5.3 Experimentation: contention reduction on a simulated network



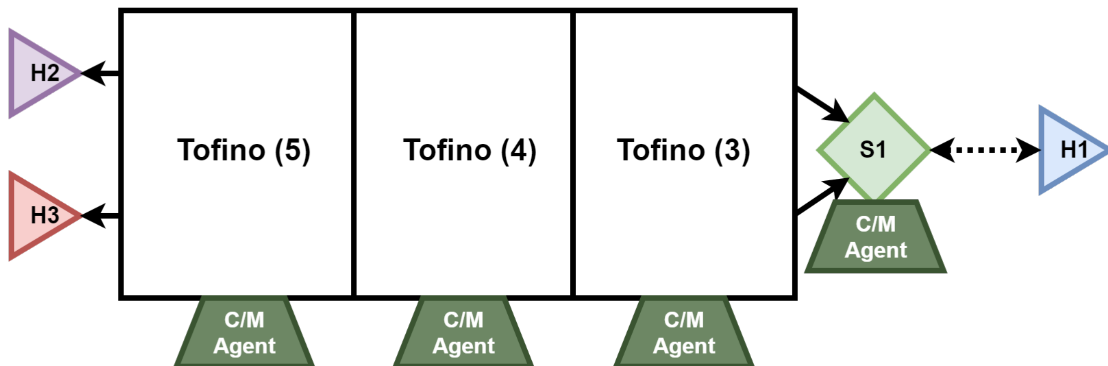


Figure 5.6: *ID* is the number of the Tofino. *22* and *11* are IDs for the available routes. Incoming links into switch 1 (s1) are 100Mbit links. The rest of the links are all 40Mbit links. We are using a mesh network just to represent the a single path on the Caltech Tier 2, which uses a star topology, system within the Caltech campus.

To better understand the end performance gain of using any technique to route data at the switch

level, we simulated a network with a similar topology to the one in the Caltech Tier 2 network and routed data using either a heuristic or a neural network. To simulate this network, we use Mininet [95], a network simulator that can simulate many types of networks, including ones with with loops in it. Mininet is a proven, reliable testing platform for network experiments when hardware is not a viable solution [96]. Researchers [97] have been able to reproduce performance benefits from simulated routing algorithms developed on a Mininet simulated network onto real hardware. Because of its viability for testing experimental algorithms with the goal of relevancy on real hardware, we have chosen this simulator to conduct experiments.

To control the ports that data may flow through, we use a custom Ryu controller [98], which allows user-configurable data flows in a network topology where loops can exist. Ryu reconfigures a data flow by updating a routing table in any simulated switch it wishes to redirect flows. This update happens every $N + O$ seconds when a data flow passes through a switch, with the values for N and O determined from the hyperparameter search. We do so to avoid thrashing ports unnecessarily.

In Figure 5.6, we illustrate the network topology used in this experiment. We choose to only simulate the path of the Caltech Tier 2 network that was passing through these switches since that was the path we used to collect performance values for our model and hyperparameter search. Switch 1 represents the MLxe8-Core switch, Host 2 represents the XRootD-cache-1 and Host 3 represents XRootD-cache-2 in Tofino setup on the Caltech Tier 2 system architecture. Host 1 is used to simulate the inbound traffic that would traverse the 100Gbit link. On these paths there is underutilized port parallelism through the Tofino switches, and therefore has the potential to have higher performance if the flows are dynamically rerouted for parallel data transfer. Boxed pairs of simulated switches represents one Tofino switch (e.g. switches 311 and 322 represent one Tofino switch with 4 ports similar to a Tofino switch in the Caltech Tier 2 network). Each simulated Tofino switch contains an instance of Diolkos. Mininet sets a hard upper limit for the bandwidth of the links in a simulated network to 1000 Mbits per second (125 MBytes per second), which prevents us from simulating 100 GBit and 40 GBit links. Thus, all links in our simulated

network run at 40 MBits, except the dotted link between S1 and H1, which runs at 100 MBits. We do this to preserve the ratio of bandwidth available in the Tier 2 network. We also experiment with a ring topology, as seen in Figure 5.7, where all the links are 100 Mbits. The ring topology shows that Diolkos can bring performance benefits to other topologies.

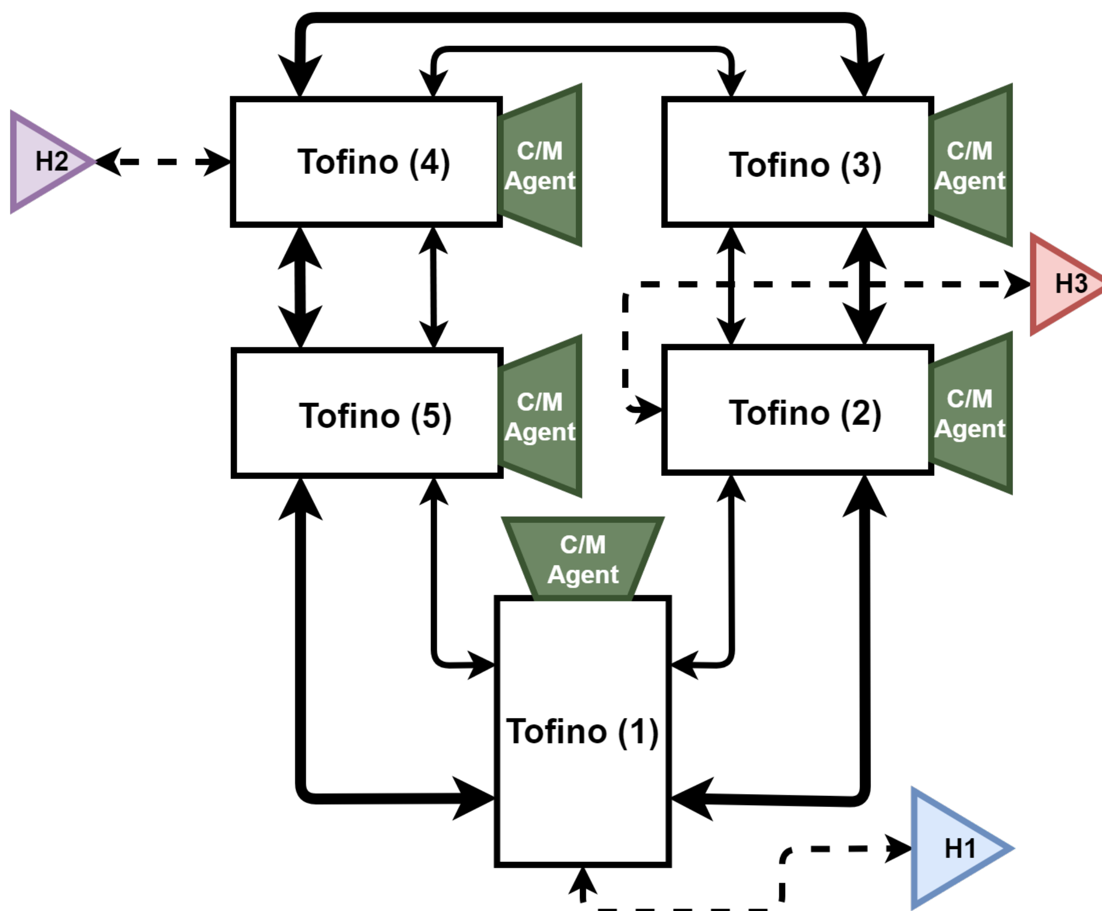


Figure 5.7: The Tofino switches are similar to that described in Figure 5.6. Each link in the network are 100 Mbit links. The thicker solid links are the outer paths numbered 00 and the thinner solid links are the inner path numbered 11. The dotted link is the link between the hosts and switches

Our simulated network resembles the Caltech Tier 2 network because it has similar paths where data could flow in the network while maintaining the link size ratio in the original network. The goal

of our simulated network is to deliver data produced from H2 and H3 to H1, and vice versa. The data is allowed to flow through any path it wishes through the network so as long as it reaches its target destination. Our example network can contain loops. Diolkos prevent infinite packet transit times on these loops.

Within this experiment, we conduct four case studies as described in Table 5.3. The workload placed upon this network creates TCP flows between H2 and H1, and H3 and H1. Any host can initiate a flow, and each flow send packets of 128 KB in bursts of 10 seconds that are transmitted to the destination host. This amount of data is continuously transmitted for a varying period of time ranging from 50 seconds to 380 seconds, repeating over seven hours. The types of flows generated and redirected are IPv4 flows and ARP flows, common flows seen in most networks. Other flows do occur in our simulation, such as healthchecks, and we do not redirect these flows because they happen rarely and do not react well to delays in arrival. All flows are generated with iPerf [99], a data flow generation program often used to benchmark network performance. While ten flows in our network easily causes some congestion if the flows are not routed efficiently, in several studies we increase the number of flows to 20 to demonstrate a surge and fall in demand. We focus on switches that simulate Tofinos to represent the original setup on the Caltech Tier 2 network.

Study 1 represents the most basic network routing situation that Diolkos may face: given a moderate number of switches and control of only the system's Tofino switches, route data to maximize throughput. Studies 2a and 2b changes the network to a ring topology to demonstrate Diolkos's effectiveness on a different network topology. This study also required for the path selector in Diolkos to have slight modifications: flows are allowed to loop, but the act of looping will make the predicted path slower than a direct path. Hence, we rely on the predictions to create the most direct, and fastest, paths to destination. Studies 3a and 3b adds more hops in the network with more Tofinos and 10x more network traffic. We also allow Diolkos to control all of the switches in the network to demonstrate scaling and potential improvements if more of the network is controlled by Diolkos. We have chosen our high traffic

flow count to be 100 since Benson *et al.* [88] demonstrated that some education and private datacenters can receive between 100 to 10,000 flow arrivals per switch in one second. In addition, Roy *et al.* [100] measured that Facebook datacenters received between 100 and 500 flows per second. Hence, we believe that 100 flows per second during these case studies is a valid representation of the load a real network can expect to encounter.

In study 4a, the hosts generate 10 flows, increasing to 20 flows after 33% of the experiment has run, then returns to 10 flows after 66% of the experiment has elapsed. These studies demonstrate Diolkos's ability to run on a system that has a surge in demand that then decreases back to normal. Study 4b starts with 100 flows, increases to 200 flows after 33% of the experiment is run, and returns back to 100 flows after 66% of the experiment.

The goal of these case studies is to show that a highly accurate predictive model can increase network throughput, more specifically the throughput at the port level, and it is resilient to scale when the optimization problem becomes more difficult. Additionally, we demonstrate how prediction accuracy plays a important role when determining the paths in the network by letting Diolkos control all the switches in some studies. This is important since performance is not only determined by throughput, but also by successful packet arrival. Not only should throughput increase, but packet loss must also be low.

To measure success, we measure throughput of the network in bytes per second at each switch, and calculate the average throughput of the network over all switches' individual ports and the standard error of the average throughput. Using the standard error in this case instead of the standard deviation allows us to compare the average benefit that rerouting had on each port of the network. `iperf` is not used for measuring data flows, we instead implemented our own throughput monitoring software in the Ryu controller that reports the speed at each port of the network.

Each of these case studies run for approximately 7 hours, during which data flows through the network. Each instance of Diolkos on the simulated Tofinos will attempt to maximize throughput of the data flows flowing through that switch. Because we will only compare the best heuristic against the

Study number	Number of Tofinos	Number of flows	Control type	β	Topo
1	3	10	partial/full	0.01	mesh
2a	5	10	partial	0.01	star
2b	5	100	partial	0.02	star
3a	4	20	partial	0.01	mesh
3b	4	100	partial	0.005	mesh
4a	4	10-20-10	partial	0.01	mesh
4b	4	100-200-100	full	0.005	mesh

Table 5.3: *partial* means that Diolkos will only control the Tofino switches, while *full* means that Diolkos controls all switches. For mesh networks, Diolkos will only control the Tofino switches. In ring toplogies, Diolkos will only partially control the Tofino switches (the side that does not contain a host). When all the switches are controlled, there is a version of Diolkos on each switch, including non-Tofino switches.

best neural network model, the results of the simulation experiment only focus on those two models. As a base case, we also run each case study with the default Ryu controller, which attempts to route data regardless of its performance. All switches gather performance data for approximately 25,000 timesteps before training is initialized and Diolkos begins to suggest routing changes.

We simulate the traffic observed on the Tofino located on Caltech Tier 2 network since we want to demonstrate the performance benefit of applying our selected neural network when compared to the heuristic with the highest accuracy. To do so, we use the Ryu controller to create traffic that has similar correlation structure as port 8 over time. We focus on port 8 since the other ports have little to no structure which would cause the model and heuristic to have similar benefits when applied. We use the created

workload as a simulated environment to measure the performance benefits of applying our different approaches. From this experiment, we determine which method produces the highest performance gain.

5.4 Results

Using the M, N and O values we have determined from the hyperparameter search, we apply the Dense model, EWMA heuristic and the base Ryu controller to different simulated variations of the network path topology observed from Tier 2. From these applications, we measure average throughput at each port, and compare the end performance that each technique achieved. All graphs reported here represent the mean at 95% confidence since we aim to compare mean performance.

5.4.1 Study 1 results

EWMA and the base case both suffer from bottlenecks, and the source of the bottlenecks for the base comes from it hardly utilizing switches 311, 411 and 511. Worse, with hosts on switches 500 and 511, the base case failed to route data from switch 1 to 500. Because the data that the Dense and EWMA models were trained on did not contain data from all paths, we observe that it led EWMA to under-utilize some switches. Although EWMA was able to detect that switch 311 was underused, and sent data through it; it also under-utilized switch 322. Further, it was not able to correctly send flows to switch 511.

Contrary to the base case and EWMA, the Dense model was able to evenly spread the flows across the network. Dense achieved an average throughput of 2.36 MB/s $\pm$ 0.01 MB/s, 14% greater than the average throughput of the network when EWMA is used (2.07 MB/s $\pm$ 0.01 MB/s) or the base controller was used (1.92 MB/s $\pm$ 0.02 MB/s). Additionally, when the network routes the flows using the Dense model's prediction, it is able to get a higher number of flows to reach the server, usually around 90% to 100% of the flows after the first round of training. From this, we can conclude that Diolkos is able

to apply a Dense neural network model to successfully route data between hosts and achieve throughput gains.

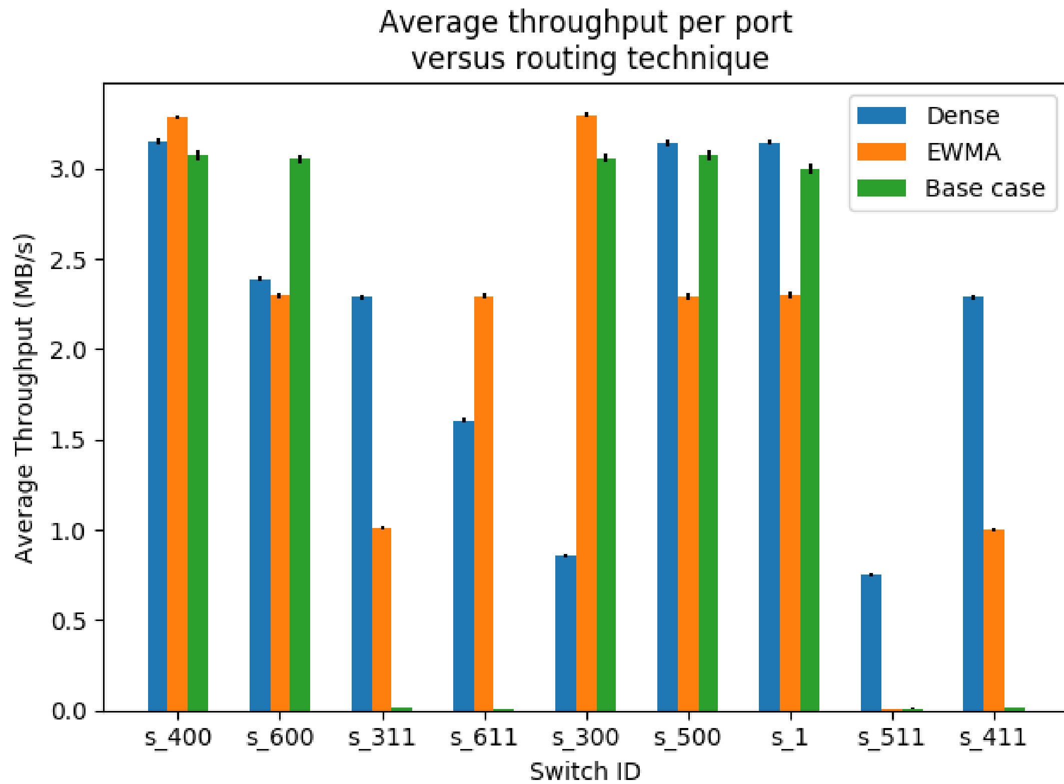


Figure 5.8: Average throughput per port over each switch, depending on the modeling technique used when we added an additional Tofino switch between Tofino 5 and hosts 2 (H2) and 3 (H3).

In Figure 5.8, Dense was again able to dynamically disperse the flows across the system to make full use of the available hardware, including the new switches. The Dense model's predictions allows the network to gain a throughput of 2.18 MB $\pm$ 0.01 MB/s on average, an 11% average increase in throughput over that of EWMA (1.97 MB/s $\pm$ 0.01 MB/s). In contrast, the base case controller only achieved an average throughput of 1.70 MB/s $\pm$ 0.02 MB/s. Additionally, the Dense network predictions are able to avoid under-utilizing switches, unlike EWMA and the base case. In this study, EWMA and

the base case had many data flows that did not reach their target hosts.

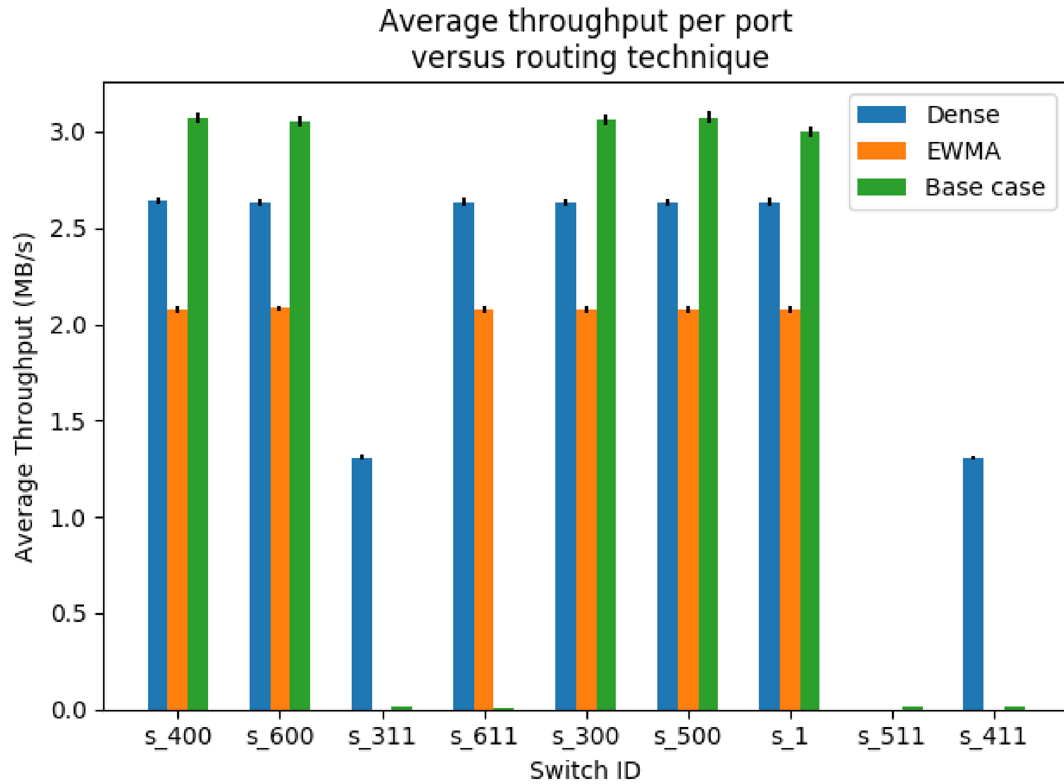


Figure 5.9: When Diolkos controls all the switches, including the non-Tofino switches, all approaches underutilized half of Tofino 5. This reduces the overall throughput of the network. However, the Dense model allowed the network to use the rest of the path.

In Figure 5.9, we observe that the EWMA model failed to use three switches, producing an average throughput of 1.38 MB/s $\pm$ 0.01 MB/s. This under-performs compared the base case controller, which had a throughput of 1.70 MB/s $\pm$ 0.02 MB/s. The Dense model's predictions also failed to use one switch, switch 522; however, it was able to effectively use the rest of the switches to maintain an average throughput of 2.05 MB/s $\pm$ 0.02 MB/s. The drop observed when using EWMA and Dense predictions can also be explained by the fact that having an instance of Diolkos on all switches required the models

to discover the available paths in the network. Hence, Diolkos can control all of the switches in the network to gain performance, though it is not strictly necessary.

5.4.2 Study 2 results

In the ring topology experiment, the network had a average throughput of 3.73 MB/s $\pm$ 0.04 MB/s when the Dense model made predictions, which is 37% higher than that of the base controller (2.72 MB/s $\pm$ 0.03 MB/s) and 13% greater than that of the EWMA heuristic (3.31 MB/s $\pm$ 0.04 MB/s). The base controller struggled using the network to its full extent, unlike the Dense model and EWMA. The Dense model manages to use more of the network than other techniques, and achieved the highest total throughput over all ports.

Increasing the number of flows in the network from 10 to 100 flows demonstrated the Dense model's superiority. The Dense model's predictions achieved an average throughput per port of 7.88 MB/s $\pm$ 0.05 MB/s, 31% greater than the throughput of the EWMA model (6.02 MB/s $\pm$ 0.04 MB/s) and 60% greater than that of the base case (4.92 MB/s $\pm$ 0.04 MB/s). We can also see, in Figure 5.10, that the Dense model's predictions allows the network to use most of the switches available in the network, granting it the highest throughput compared to the other two approaches. Additionally, unlike the mesh network, both models and our base controller opted to not use the entire network, leave room for other flows to utilize the network.

5.4.3 Study 3 results

In a larger network with increasing demand, we observe that the Dense model maintains an average throughput value of 2.41 MB/s $\pm$ 0.01 MB/s. In contrast, EWMA has a throughput value of 2.25 MB/s $\pm$ 0.01 MB/s, only 4% higher than the base case of 2.17 MB/s $\pm$ 0.02 MB/s. Although it may appear that more flows in the network will produce a higher measured throughput, the base case and EWMA will continue to underutilize switches, forcing more data to be bottlenecked through whatever

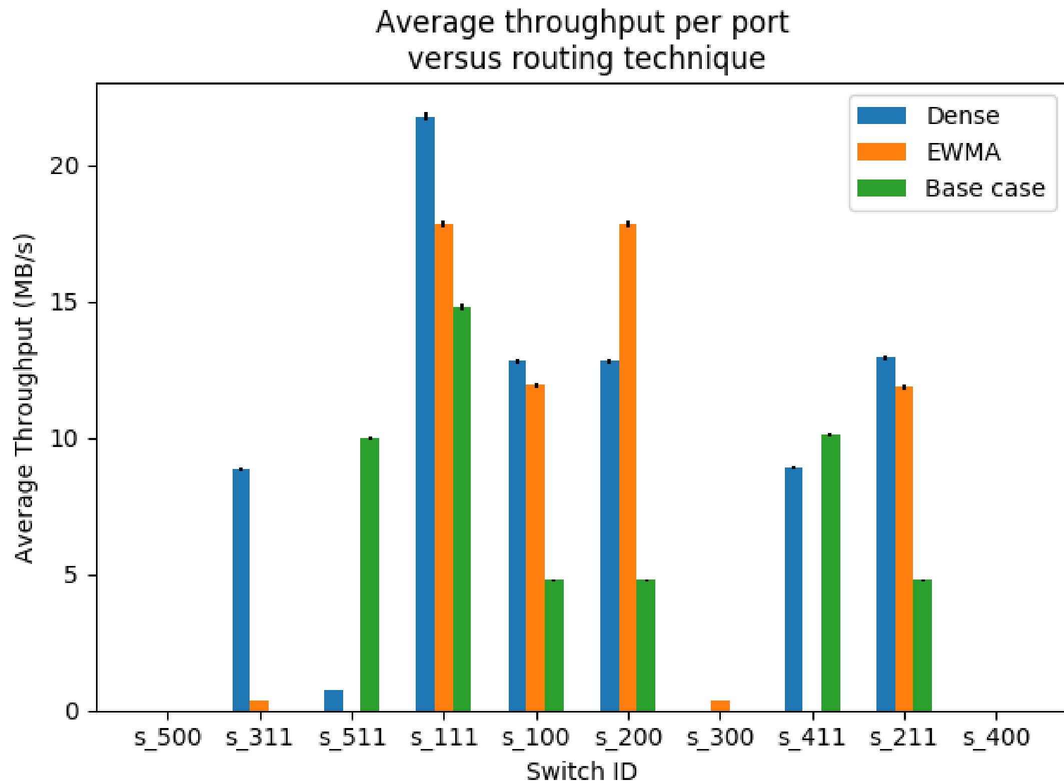


Figure 5.10: A ring topology still demonstrates the Dense model's superiority over other modeling techniques, demonstrating that Diolkos is network topology agnostic.

switches were utilized. The better the predictions are, the more switches are used, thus demonstrating that better predictions allows the network to use underutilized switches.

When the network is hit with 100 flows, as seen in Figure 5.11, the Dense model attains a 3.48 MB/s $\pm$ 0.01 MB/s average throughput per port, 27% higher than the base controller (2.73 MB/s $\pm$ 0.02 MB/s) and 11% higher than the EWMA model (3.14 MB/s $\pm$ 0.01 MB/s). Even when the network's demand surges beyond its intended capacity, the Dense model was much more successful in reducing contention than that of the other techniques. Although its performance gain was much more modest than that of smaller surges in network usage, the Dense model was able to demonstrate its ability

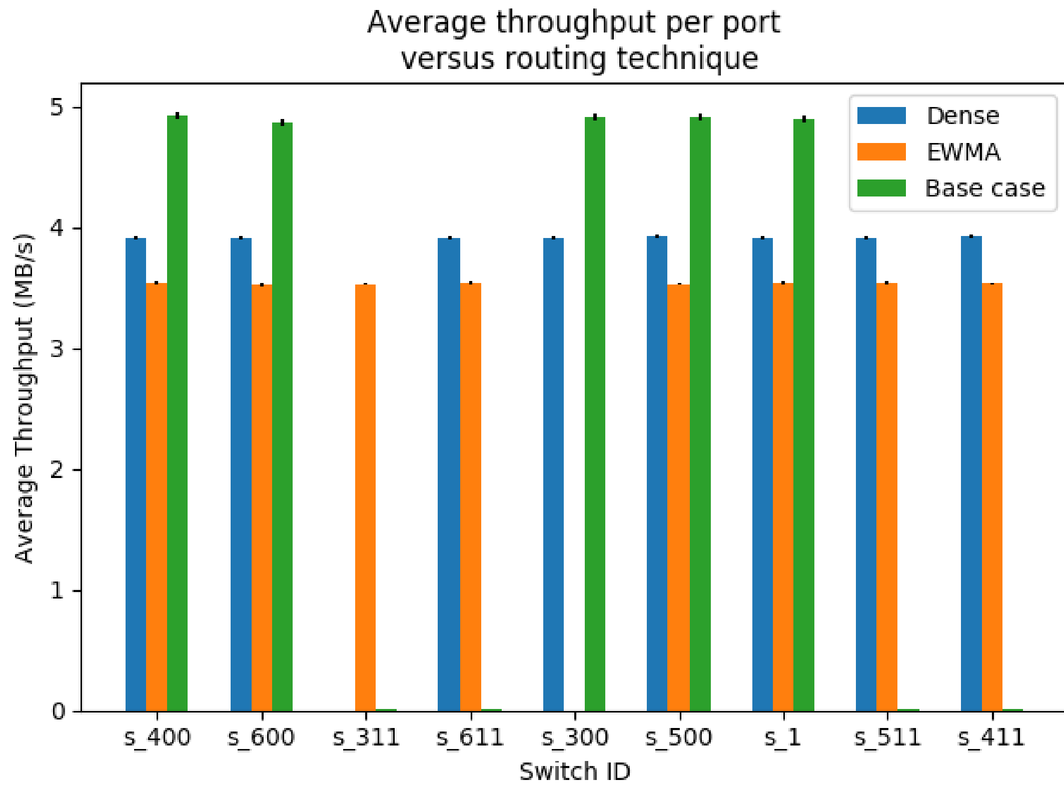


Figure 5.11: Increasing the number of flows from 10 to 100 and letting Diolkos only control the Tofino switches demonstrates its ability to react to surges in demand.

to re-flow data flows in response to changes in demand, preventing the drops in performance that other techniques suffered from.

5.4.4 Study 4 results

In a situation where demand surges and lowers, the Dense model has an average throughput of 2.86 MB/s ± 0.02 MB/s per port, which is 28% higher than the throughput per port achieved using EWMA's predictions (2.23 MB/s ± 0.02 MB/s). The Dense model's performance is also 49% higher than that of the base controller (1.92 MB/s ± 0.02 MB/s). The drawback observed in this experiment is

that there was a drop of flows reaching the target host, down to an average of 60%. In future work we will include the flow's target IP as part of the training features to ensure that we not only get a performance improvement, but also arrival guarantees to their target destination.

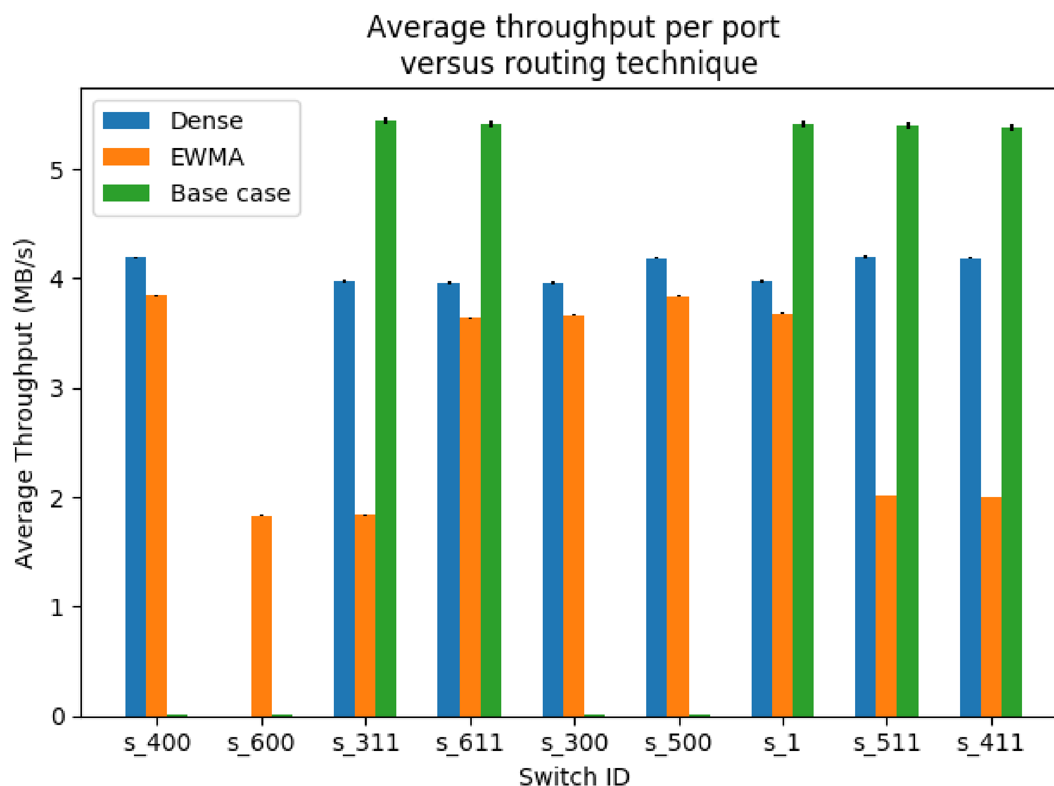


Figure 5.12: Rises and falls in demand does not cause the Dense model to bottleneck in performance. Rather, it responds to it to maintain high average throughput.

Increasing the severity of the surge, while letting Diolkos control all the switches of the network, does not prevent Diolkos from reducing contention, as seen in Figure 5.12. The Dense model was able to get a average throughput per port of 3.63 MB/s $\pm$ 0.01 MB/s, 21% higher than that of the base case controller (3.01 MB/s $\pm$ 0.02 MB/s) and 24% higher than that of EWMA (2.93 MB/s $\pm$ 0.01 MB/s). This demonstrates that a lower accuracy approach, even if it is able to update prediction more frequently,

does negatively impact the resulting performance. We also observe that both the base case and the Dense model did not get packets to the host on switch 611. This shows that if the base case does not go through all the switches, the model may be not able to detect a switch. However when we made our path selector aware of that missing data, it was able to start evenly spreading the flows across the system.

5.5 Related Work

Broch *et al.* [101] and Cano *et al.* [102] compared different routing protocol such as DSDV [103], TORA [104, 105], DSR [106, 107] and AODV [108]. They found that these approaches have heavy routing overheads, and they identified a need to intelligently route network data based on past traffic data. They claim that a network should be able to predict bottlenecks that are building in the system and reroute data preemptively. Diolkos identifies specifically what type of model will accurately predict the future performance fluctuation and give the best performance benefit at the switch level.

Wang *et al.* [109] outlines several challenges in regards to network tuning based on machine learning. The authors agree that modeling computer networks is a difficult task where past data can be leveraged for forecasting network changes. Specifically, the authors state that machine learning has higher benefits when modeled with an adequate neural network. We demonstrated that at the switch level the structure of the flow has a higher impact on the predictability of the performance of the flow.

Voellmy *et al.* [110] describes how Software Defined Networks (SDN) allows for simpler controls over the network; however, the simplicity comes at the cost of scalability. People have used machine learning to classify flows in SDNs [111]. Gholami *et al.* [112] used heuristics to lower the congestion of SDNs. They demonstrated their techniques using mininet to quantify the benefit of applying their approach, however they did not describe the difference in performance benefits that different modeling approaches may have when selecting routes on the network. Additionally, they mention that applying neural networks could be an upcoming technique for lowering congestion on networks.

Corson *et al.* [113] used “link reversal” algorithms which is structured as a sequence of diffusing computations where each computation consists of a sequence of directed link reversals. They used a “physical” or “logical” clock to establish a temporal order of topological change events. Doing so, they can dynamically reallocate flows. Chen *et al.* [114] explores the use of deep Q networks for elastic optical networks. Using these neural networks, they developed a human-level network provisioning agent that balances pathing requests over the available network. Such approaches are similar to those we use in Diolkos. Rexford *et al.* [115] discusses the computation of routes in an IP network. The authors describe the challenges of finding features that describe routing metrics, and modeling them for useful applications to routing. They state that optimizations based on small area searches for paths have good effect on load balancing and network optimization. This work is one of the reasons we focus on switch level controllers instead of whole network controllers.

Mak *et al.* [116], Hong *et al.* [117] and Jain *et al.* [118] have developed centralized systems that controls the amount of traffic flows. The benefit is near 100% utilization on the switches of their network, but it requires a central controller, risking a bottleneck in the system since the data needs to be transferred externally over to their system. Diolkos uses a switch level controller, thus reducing the data-flow needed to come out of the network.

5.6 Conclusion

We were able to simulate performance increases from applying Diolkos to switches. We found that different models have different performance benefits, and a Dense neural network prediction model balanced data flows for up to a 28% increase in throughput over baselines, and up to a 49% increase over the EWMA prediction heuristic. Overall, our dense model was able to discover use for underutilized switches, and effectively route data flows through them to increase performance. Additionally, a high accuracy model allows the model to find available paths in the system even at the switch level.

Chapter 6

Conclusion and Future Work

This thesis describes my work on two approaches, Diolkos and Geomancy, that manage contention and bottlenecks in large scientific systems. We have built prototype systems to validate these ideas and demonstrated their performance. Geomancy can effectively place data within a system of many disparate storage points, and we can leverage Diolkos to execute efficient end-to-end transmission of the data being moved. These improvements come with no need to change either hardware or server software. With these capabilties, this thesis outlines a point to point efficient data restructuring system that can respond to storage and network contention, and improve the performance of the system without manual intervention. Further, we have made strides at winnowing down a feature list to select only features that were relevant to the performance aspect that we wanted to model. With this reduced feature list, we can better utilize online learning techniques for applications of Diolkos and Geomancy that are faster, more efficient, and relevant over a long time.

6.1 Future work in optimizing system modeling via feature selection

We would like to expand WinnowML to also handle clustering. To do so we can examine the mutual information score between the predictions and target to identify potential clusters in the data. Mutual information is the number of common values between a target cluster and the training values cluster. We can also explore the difference in accuracy when the features are ordered by mutual information score versus recall. For enumerating clusters, we could use the recall score to identify how many actual clusters were predicted by the neural network.

6.2 Future work in applying modeling techniques to data layouts

In the future, we will create a separate model which will be used to predict gaps in accesses for files on the system. Gaps are defined as periods of time, where the individual file is not accessed by any workloads, that is long enough for Geomancy to move the file to the new location. We will not consider moving files that are always accessed and never released since there can be no way to optimally move it. Geomancy will concurrently generate new locations for all the files based on observations. Once a gap is found, the control agents on the system will move the file to the new location determined by Geomancy's algorithm when the predicted gap starts. Using this model, we will be able to get a better idea on how our workload scales when the system and the number of clients increases.

6.3 Future work in identifying modeling techniques for network routing

A future case study should determine other features that can also describe performance. In this paper, we focus on bytes per second at each port of the switch to determine which route to select. Since we will experiment on more complex networks such as dragonfly networks and InfiniBand (IB) networks, our selected features and our site-wide coordination may not be adequate for these networks. When applying Diolkos to these networks, we will determine if any new networking coordination needs to be done to ensure proper flow control. Additionally, we will update Diolkos to dynamically update the β threshold that is used to determine that a path is not being used by the network. β must change as the load on the network changes, and be large enough so that Diolkos does not forgo using a potential link to alleviate pressure on overused links.

Bibliography

[1] H. W. Tyrer, *Advances in Distributed and Parallel Processing: System paradigms and methods*, vol. 1. Intellect Books, 1994.

[2] K. R. Kaplan and R. O. Winder, "Cache-based computer systems," *Computer*, vol. 6, no. 3, pp. 30–36, 1973.

[3] C. Chatfield, "The holt-winters forecasting procedure," *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, vol. 27, no. 3, pp. 264–279, 1978.

[4] S. ur Rahman, G.-H. Kim, Y.-Z. Cho, and A. Khan, "Deployment of an sdn-based uav network: Controller placement and tradeoff between control overhead and delay," in *2017 International Conference on Information and Communication Technology Convergence (ICTC)*, pp. 1290–1292, IEEE, 2017.

[5] H. Li, P. Li, S. Guo, and A. Nayak, "Byzantine-resilient secure software-defined networks with multiple controllers in cloud," *IEEE Transactions on Cloud Computing*, vol. 2, no. 4, pp. 436–447, 2014.

[6] A. Altmann, L. Toloşi, O. Sander, and T. Lengauer, "Permutation importance: a corrected feature importance measure," *Bioinformatics*, vol. 26, no. 10, pp. 1340–1347, 2010.

[7] E. Erwin, K. Obermayer, and K. Schulten, “Self-organizing maps: ordering, convergence properties and energy functions,” *Biological Cybernetics*, vol. 67, no. 1, pp. 47–55, 1992.

[8] R. Bro and A. K. Smilde, “Principal component analysis,” *Analytical Methods*, vol. 6, no. 9, pp. 2812–2831, 2014.

[9] K. Han, Y. Wang, C. Zhang, C. Li, and C. Xu, “Autoencoder inspired unsupervised feature selection,” in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2941–2945, IEEE, 2018.

[10] J. Wang, P. Zhao, S. C. Hoi, and R. Jin, “Online feature selection and its applications,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 3, pp. 698–710, 2013.

[11] H. Huang, S. Yoo, and S. P. Kasiviswanathan, “Unsupervised feature selection on data streams,” in *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pp. 1031–1040, 2015.

[12] A. Peters, E. Sindrilaru, and G. Adde, “EOS as the present and future solution for data storage at CERN,” *Journal of Physics: Conference Series*, vol. 664, no. 4, p. 042042, 2015.

[13] B. T. Zivkov and A. J. Smith, “Disk caching in large database and timeshared systems,” in *Proceedings Fifth International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pp. 184–195, IEEE, 1997.

[14] D. Lee, J. Choi, J.-H. Kim, S. H. Noh, S. L. Min, Y. Cho, and C. S. Kim, “Lrfu: A spectrum of policies that subsumes the least recently used and least frequently used policies,” *IEEE transactions on Computers*, no. 12, pp. 1352–1361, 2001.

[15] S. Wen, D. Qin, T. Lv, L. Ge, and X. Yang, “Traffic identification algorithm based on improved lru,” in *7th IEEE International Conference on Cyber Security and Cloud Computing*

(CSCloud)/6th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom), pp. 157–159, IEEE, 2020.

[16] A. Murugan and S. Ganesan, "Hybrid lru algorithm for enterprise data hub," *9th International Journal of Innovative Technology and Exploring Engineering (IJITEE)*, November 2019.

[17] N. Alzakari, A. B. Dris, and S. Alahmadi, "Randomized least frequently used cache replacement strategy for named data networking," in *2020 3rd International Conference on Computer Applications & Information Security (ICCAIS)*, pp. 1–6, IEEE, 2020.

[18] S. Wold, K. Esbensen, and P. Geladi, "Principal component analysis," *Chemometrics and Intelligent Laboratory Systems*, vol. 2, no. 1-3, pp. 37–52, 1987.

[19] L. McInnes, J. Healy, and J. Melville, "Umap: Uniform manifold approximation and projection for dimension reduction," *arXiv preprint arXiv:1802.03426*, 2018.

[20] L. v. d. Maaten and G. Hinton, "Visualizing data using t-SNE," *Journal of Machine Learning Research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[21] S. Ozawa, S. Pang, and N. Kasabov, "An incremental principal component analysis for chunk data," in *2006 IEEE International Conference on Fuzzy Systems*, pp. 2278–2285, IEEE, 2006.

[22] H. Zou, T. Hastie, and R. Tibshirani, "Sparse principal component analysis," *Journal of Computational and Graphical Statistics*, vol. 15, no. 2, pp. 265–286, 2006.

[23] A. Y. Ng, "Feature selection, L 1 vs. L 2 regularization, and rotational invariance," in *Proceedings of the 21st International Conference on Machine Learning*, p. 78, 2004.

[24] W. Zhao, A. Krishnaswamy, R. Chellappa, D. L. Swets, and J. Weng, "Discriminant analysis of principal components for face recognition," in *Face Recognition*, pp. 73–85, Springer, 1998.

[25] S. Perkins, K. Lacker, and J. Theiler, “Grafting: Fast, incremental feature selection by gradient descent in function space,” *Journal of machine learning research*, vol. 3, no. Mar, pp. 1333–1356, 2003.

[26] J. Zhou, D. Foster, R. Stine, and L. Ungar, “Streaming feature selection using alpha-investing,” in *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pp. 384–393, 2005.

[27] X. Wu, K. Yu, H. Wang, and W. Ding, “Online streaming feature selection,” in *Proceedings of the 27th International Conference on Machine Learning*, 2010.

[28] J. Li, X. Hu, J. Tang, and H. Liu, “Unsupervised streaming feature selection in social media,” in *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pp. 1041–1050, 2015.

[29] Y.-W. Chang and C.-J. Lin, “Feature ranking using linear SVM,” in *Causation and Prediction Challenge*, pp. 53–64, 2008.

[30] P. S. Bradley and O. L. Mangasarian, “Feature selection via concave minimization and support vector machines,” in *Proceedings of the 15th International Conference on Machine Learning*, vol. 98, pp. 82–90, 1998.

[31] S. Maldonado, R. Weber, and F. Famili, “Feature selection for high-dimensional class-imbalanced data sets using support vector machines,” *Information Sciences*, vol. 286, pp. 228–246, 2014.

[32] J. K. Brewer and J. R. Hills, “Univariate selection: The effects of size of correlation, degree of skew, and degree of restriction,” *Psychometrika*, vol. 34, no. 3, pp. 347–361, 1969.

[33] P. M. Granitto, C. Furlanello, F. Biasioli, and F. Gasperi, “Recursive feature elimination with random forest for PTR-MS analysis of agroindustrial products,” *Chemometrics and Intelligent Laboratory Systems*, vol. 83, no. 2, pp. 83–90, 2006.

[34] J. Benesty, J. Chen, Y. Huang, and I. Cohen, "Pearson correlation coefficient," in *Noise reduction in speech processing*, pp. 1–4, Springer, 2009.

[35] A. M. Frieze and M. R. B. Clarke, "Approximation algorithms for the m-dimensional 0-1 knapsack problem: worst-case and probabilistic analyses," *European Journal of Operational Research*, vol. 15, no. 1, pp. 100–109, 1984.

[36] R. Poli, "An analysis of publications on particle swarm optimization applications," *Essex, UK: Department of Computer Science, University of Essex*, 2007.

[37] L. Golubchik, S. Khanna, S. Khuller, R. Thurimella, and A. Zhu, "Approximation algorithms for data placement on parallel disks," *ACM Transactions on Algorithms (TALG)*, vol. 5, no. 4, p. 34, 2009.

[38] A. Chervenak, E. Deelman, M. Livny, M.-H. Su, R. Schuler, S. Bharathi, G. Mehta, and K. Vahi, "Data placement for scientific applications in distributed environments," in *Proceedings of the 8th IEEE/ACM International Conference on Grid Computing*, GRID '07, (Washington, DC, USA), pp. 267–274, IEEE Computer Society, 2007.

[39] B. Junqueira and B. Reed, "Hadoop: The definitive guide," *Journal of Computing in Higher Education*, vol. 12, no. 2, pp. 94–97, 2001.

[40] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," 2004.

[41] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S.-H. Bae, J. Qiu, and G. Fox, "Twister: a runtime for iterative mapreduce," in *Proceedings of the 19th ACM international symposium on high performance distributed computing*, pp. 810–818, ACM, 2010.

[42] T. Dysart, P. Kogge, M. Deneroff, E. Bovell, P. Briggs, J. Brockman, K. Jacobsen, Y. Juan, S. Kuntz, R. Lethin, *et al.*, "Highly scalable near memory processing with migrating threads on

the emu system architecture," in *Proceedings of the Sixth Workshop on Irregular Applications: Architectures and Algorithms*, pp. 2–9, IEEE Press, 2016.

[43] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, A. Vahdat, *et al.*, "Hedera: dynamic flow scheduling for data center networks.," in *Proceedings of the 7th USENIX Symposium on Networked Systems Design and Implementation (NSDI 10)*, pp. 89–92, 2010.

[44] D. Margolin, K. Ognyanoya, M. Huang, Y. Huang, and N. Contractor, "Team formation and performance on nanohub: A network selection challenge in scientific communities," *Networks in social policy problems*, pp. 80–100, 2012.

[45] K. Bloom, "Us cms tier-2 computing," *Journal of Physics: Conference Series*, vol. 119, no. 5, p. 052004, 2008.

[46] F. N. A. Laboratory, "Computing facilities and middleware."

[47] Computing, M. Sciences, and H. E. P. at Caltech, "Cms caltech tier2."

[48] H. Newman, A. Barczyk, A. Mughal, S. Rozsa, R. Voicu, I. Legrand, S. Lo, D. Kcira, R. Sobie, I. Gable, *et al.*, "Efficient lhc data distribution across 100gbps networks," in *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*, pp. 1594–1599, IEEE, 2012.

[49] H. Newman, A. Mughal, D. Kcira, I. Legrand, R. Voicu, and J. Bunn, "High speed scientific data transfers using software defined networking," in *Proceedings of the Second Workshop on Innovating the Network for Data-Intensive Science*, p. 2, ACM, 2015.

[50] H. Newman, M. Spiropulu, J. Balcas, D. Kcira, I. Legrand, A. Mughal, J. Vlimant, and R. Voicu, "Next-generation exascale network integrated architecture for global science," *Journal of Optical Communications and Networking*, vol. 9, no. 2, pp. A162–A169, 2017.

[51] K. Nahm, A. Helmy, and C.-C. Jay Kuo, "Tcp over multihop 802.11 networks: issues and performance enhancement," in *Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing*, pp. 277–287, ACM, 2005.

[52] Y. Li, K. Chang, O. Bel, E. L. Miller, and D. D. E. Long, "CAPES: Unsupervised Storage Performance Tuning Using Neural Network-based Deep Reinforcement Learning," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '17, (New York, NY, USA), pp. 42:1–42:14, ACM, 2017.

[53] M. Jensen, "Traffic analysis and experimentation platform (TAEP)." Published: 14 Nov 2017 at 14:00.

[54] B. Networks, "Tofino programmable switch." `https://www.barefootnetworks.com/technology/`, 2018.

[55] V. Sivaraman, S. Narayana, O. Rottenstreich, S. Muthukrishnan, and J. Rexford, "Heavy-hitter detection entirely in the data plane," in *Proceedings of the Symposium on SDN Research*, pp. 164–176, ACM, 2017.

[56] A. Gulli and S. Pal, *Deep learning with Keras*. Packt Publishing Ltd, 2017.

[57] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, "Scikit-learn: Machine learning in python," *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[58] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Information processing & management*, vol. 45, no. 4, pp. 427–437, 2009.

[59] M. Herbster and M. K. Warmuth, "Tracking the best expert," *Machine learning*, vol. 32, no. 2, pp. 151–178, 1998.

[60] "IT Analytics Working Group: EOS File Transfer Logs." https://twiki.cern.ch/twiki/bin/view/ITAnalyticsWorkingGroup/EosFileAccessLogs, 2019.

[61] G. Adde, B. Chan, D. Duellmann, X. Espinal, A. Fiorot, J. Iven, L. Janyst, M. Lamanna, L. Mascetti, J. M. P. Rocha, *et al.*, "Latest evolution of eos filesystem," in *Journal of Physics: Conference Series*, vol. 608, p. 012009, IOP Publishing, 2015.

[62] A. J. Peters and L. Janyst, "Exabyte scale storage at cern," in *Journal of Physics: Conference Series*, vol. 331, p. 052015, IOP Publishing, 2011.

[63] D. M. Asner, E. Dart, and T. Hara, "Belle II experiment network and computing," *arXiv preprint arXiv:1308.0672*, 2013.

[64] M. Iwasaki, K. Furukawa, T. Nakamura, T. Obina, S. Sasaki, M. Satoh, T. Aoyama, and T. Nakamura, "Design and Status of the SuperKEKB Accelerator Control Network System," in *Proceedings, 5th International Particle Accelerator Conference (IPAC 2014): Dresden, Germany, June 15-20, 2014*, p. THPRO109, 2014.

[65] R. Brun and F. Rademakers, "Root-an object oriented data analysis framework," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 389, no. 1-2, pp. 81–86, 1997.

[66] "Advanced Computing, Mathematics and Data Research Highlights." https://bit.ly/2SU3YNT, dec 2017.

[67] R. S. Sutton and A. G. Barto, "Reinforcement learning: An introduction," 2011.

[68] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814, 2010.

[69] G. M. Weiss and F. Provost, "The effect of class distribution on classifier learning: an empirical study," *Rutgers Univ*, 2001.

[70] P. Praekhaow, "Determination of trading points using the moving average methods," in *GMSTEC 2010: International Conference for a Sustainable Greater Mekong*, vol. 27, 2010.

[71] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[72] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Gated feedback recurrent neural networks," in *International Conference on Machine Learning*, pp. 2067–2075, 2015.

[73] H.-T. Chou and D. J. DeWitt, "An evaluation of buffer management strategies for relational database systems," *Algorithmica*, vol. 1, no. 1-4, pp. 311–336, 1986.

[74] M. Gupta, V. Sridharan, D. Roberts, A. Prodromou, A. Venkat, D. Tullsen, and R. Gupta, "Reliability-aware data placement for heterogeneous memory architecture," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 583–595, IEEE, 2018.

[75] D. N. Serpanos, L. Georgiadis, and T. Bouloutas, "MMPacking: A load and storage balancing algorithm for distributed multimedia servers," in *Proceedings International Conference on Computer Design. VLSI in Computers and Processors*, pp. 170–174, Oct 1996.

[76] M. Randles, D. Lamb, and A. Taleb-Bendiab, "A comparative study into distributed load balancing algorithms for cloud computing," in *Proceedings of the 2010 IEEE 24th International Conference on Advanced Information Networking and Applications Workshops*, pp. 551–556, April 2010.

[77] K. A. Nuaimi, N. Mohamed, M. A. Nuaimi, and J. Al-Jaroodi, "A survey of load balancing in cloud computing: Challenges and algorithms," in *Proceedings of the 2012 Second Symposium on Network Cloud Computing and Applications*, pp. 137–142, Dec 2012.

[78] P. Jamshidi and G. Casale, "An uncertainty-aware approach to optimal configuration of stream processing systems," in *Proceedings of the 24th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS '16)*, Sept. 2016.

[79] A. Saboori, G. Jiang, and H. Chen, "Autotuning configurations in distributed systems for performance improvements using evolutionary strategies," in *Proceedings of the 28th International Conference on Distributed Computing Systems (ICDCS '08)*, pp. 769–776, June 2008.

[80] K. Winstein and H. Balakrishnan, "TCP ex machina: Computer-generated congestion control," *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '13)*, vol. 43, pp. 123–134, Aug. 2013.

[81] Y. Li, X. Lu, E. L. Miller, and D. D. E. Long, "ASCAR: Automating contention management for high-performance storage systems," in *Proceedings of the 31st IEEE Conference on Mass Storage Systems and Technologies*, June 2015.

[82] Y. Li, K. Chang, O. Bel, E. L. Miller, and D. D. E. Long, "CAPES: Unsupervised storage performance tuning using neural network-based deep reinforcement learning," in *Proceedings of the 2017 International Conference for High Performance Computing, Networking, Storage and Analysis (SC17)*, Nov. 2017.

[83] T. Wang, S. Byna, B. Dong, and H. Tang, "Univistor: Integrated hierarchical and distributed storage for hpc," in *2018 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 134–144, IEEE, 2018.

[84] P. Subedi, P. Davis, S. Duan, S. Klasky, H. Kolla, and M. Parashar, "Stacker: an autonomic data movement engine for extreme-scale data staging-based in-situ workflows," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, p. 73, IEEE Press, 2018.

[85] E. N. Rush, B. Harris, N. Altiparmak, and A. Ş. Tosun, "Dynamic data layout optimization for high performance parallel i/o," in *2016 IEEE 23rd International Conference on High Performance Computing (HiPC)*, pp. 132–141, IEEE, 2016.

[86] C. Yu, P. Roy, Y. Bai, H. Yang, and X. Liu, "LWPTool: A Lightweight Profiler to Guide Data Layout Optimization," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 11, pp. 2489–2502, 2018.

[87] C. Zhang, H. Zhang, J. Qiao, D. Yuan, and M. Zhang, "Deep transfer learning for intelligent cellular traffic prediction based on cross-domain big data," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1389–1401, 2019.

[88] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pp. 267–280, 2010.

[89] J. S. Hunter, "The exponentially weighted moving average," *Journal of quality technology*, vol. 18, no. 4, pp. 203–210, 1986.

[90] R. Davies and A. Huitson, "A sales forecasting comparison," *Journal of the Royal Statistical Society. Series D (The Statistician)*, vol. 17, no. 3, pp. 269–278, 1967.

[91] C. Park, S. W. Kim, and S. W. Kim, "Apparatus and method for preventing network attacks, and packet transmission and reception processing apparatus and method using the same," June 2 2011. US Patent App. 12/701,253.

[92] G. Zhang and M. Parashar, "Cooperative detection and protection against network attacks using decentralized information sharing," *Cluster Computing*, vol. 13, no. 1, pp. 67–86, 2010.

[93] S. R. Snapp, J. Brentano, G. Dias, T. L. Goan, L. T. Heberlein, C.-L. Ho, and K. N. Levitt, "Dids (distributed intrusion detection system)-motivation, architecture, and an early prototype," 1991.

[94] S. Singh, G. Varghese, C. Estan, and S. Savage, "Detecting public network attacks using signatures and fast content analysis," Oct. 23 2012. US Patent 8,296,842.

[95] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets-IX, (New York, NY, USA), pp. 19:1–19:6, ACM, 2010.

[96] R. L. S. De Oliveira, C. M. Schweitzer, A. A. Shinoda, and L. R. Prete, "Using mininet for emulation and prototyping software-defined networks," in *2014 IEEE Colombian Conference on Communications and Computing (COLCOM)*, pp. 1–6, IEEE, 2014.

[97] M. Erel, E. Teoman, Y. Özçevik, G. Seçinti, and B. Canberk, "Scalability analysis and flow admission control in mininet-based sdn environment," in *2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*, pp. 18–19, IEEE, 2015.

[98] Ryu, "Ryu." `https://osrg.github.io/ryu/`, 2019.

[99] C.-H. Hsu and U. Kremer, "Iperf: A framework for automatic construction of performance prediction models," in *Workshop on Profile and Feedback-Directed Compilation (PFDC), Paris, France*, Citeseer, 1998.

[100] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pp. 123–137, 2015.

[101] J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu, and J. Jetcheva, "A performance comparison of multi-hop wireless ad hoc network routing protocols," in *Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking*, pp. 85–97, 1998.

[102] J.-C. Cano and P. Manzoni, "A performance comparison of energy consumption for mobile ad hoc network routing protocols," in *Proceedings 8th International Symposium on Modeling, Analysis*

and Simulation of Computer and Telecommunication Systems (Cat. No. PR00728), pp. 57–64, IEEE, 2000.

[103] C. E. Perkins and P. Bhagwat, "Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers," *ACM SIGCOMM computer communication review*, vol. 24, no. 4, pp. 234–244, 1994.

[104] V. D. Park and M. S. Corson, "A highly adaptive distributed routing algorithm for mobile wireless networks," in *Proceedings of INFOCOM'97*, vol. 3, pp. 1405–1413, IEEE, 1997.

[105] V. Park and S. Corson, "Temporally-ordered routing algorithm (tora)," tech. rep., IETF internet draft, 2001.

[106] D. B. Johnson, "Routing in ad hoc networks of mobile hosts," in *1994 First Workshop on Mobile Computing Systems and Applications*, pp. 158–163, IEEE, 1994.

[107] D. B. Johnson and D. A. Maltz, "Dynamic source routing in ad hoc wireless networks," in *Mobile computing*, pp. 153–181, Springer, 1996.

[108] C. Perkins, E. Belding-Royer, and S. Das, "Rfc3561: Ad hoc on-demand distance vector (aodv) routing," 2003.

[109] M. Wang, Y. Cui, X. Wang, S. Xiao, and J. Jiang, "Machine learning for networking: Workflow, advances and opportunities," *Ieee Network*, vol. 32, no. 2, pp. 92–99, 2017.

[110] A. Voellmy and J. Wang, "Scalable software defined network controllers," in *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, pp. 289–290, 2012.

[111] P. Amaral, J. Dinis, P. Pinto, L. Bernardo, J. Tavares, and H. S. Mamede, "Machine learning in

software defined networks: Data collection and traffic classification," in *2016 IEEE 24th International Conference on Network Protocols (ICNP)*, pp. 1–5, IEEE, 2016.

[112] M. Gholami and B. Akbari, "Congestion control in software defined data center networks through flow rerouting," in *2015 23rd Iranian Conference on Electrical Engineering*, pp. 654–657, IEEE, 2015.

[113] M. S. Corson and V. D. Park, "Adaptive routing method for a dynamic network," Dec. 23 2003. US Patent 6,667,957.

[114] X. Chen, J. Guo, Z. Zhu, R. Proietti, A. Castro, and S. Yoo, "Deep-rmsa: A deep-reinforcement-learning routing, modulation and spectrum assignment agent for elastic optical networks," in *Proceedings of the 2018 Optical Fiber Communications Conference and Exposition*, pp. 1–3, IEEE, 2018.

[115] J. Rexford, "Route optimization in ip networks," in *Handbook of Optimization in Telecommunications*, pp. 679–700, Springer, 2006.

[116] T. Mak, P. Y. Cheung, K.-P. Lam, and W. Luk, "Adaptive routing in network-on-chips using a dynamic-programming network," *IEEE Transactions on industrial electronics*, vol. 58, no. 8, pp. 3701–3716, 2010.

[117] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving high utilization with software-driven wan," in *Proceedings of the 13th ACM SIGCOMM conference*, pp. 15–26, 2013.

[118] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, *et al.*, "B4: Experience with a globally-deployed software defined wan," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 3–14, 2013.