Geomancy: Automated Performance Enhancement through Data Layout Optimization

Oceane Bel*, Kenneth Chang*, Nathan R. Tallent‡, Dirk Duellmann§,
Ethan L. Miller*†, Faisal Nawab* and Darrell D. E. Long*
Emails: obel@ucsc.edu, kchang44@ucsc.edu, tallent@pnnl.gov, Dirk.Duellmann@cern.ch,
elm@ucsc.edu, fnawab@ucsc.edu and darrell@ucsc.edu
*University of California, Santa Cruz, †Pure storage, ‡Pacific Northwest National Labs, §CERN

Abstract—**Large distributed storage systems such as high-performance computing (HPC) systems used by national or international laboratories require sufficient performance and scale for demanding scientific workloads and must handle shifting workloads with ease. Ideally, data is placed in locations to optimize performance, but the size and complexity of large storage systems inhibit rapid effective restructuring of data layouts to maintain performance as workloads shift.**

To address these issues, we have developed Geomancy, a tool that models the placement of data within a distributed storage system and reacts to drops in performance. Using a combination of machine learning techniques suitable for temporal modeling, Geomancy determines when and where a bottleneck may happen due to changing workloads and suggests changes in the layout that mitigate or prevent them. Our approach to optimizing throughput offers benefits for storage systems such as avoiding potential bottlenecks and increasing overall I/O throughput from 11% to 30%.

I. 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. 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.

To address this issue, we have developed Geomancy, a tool that improves system performance by finding efficient data layouts using reinforcement learning in real-time. Geomancy targets systems that serve and process petabytes of data, such as particle collision analysis [2]. Workloads on these systems are commonly spread across multiple storage devices which can lead to storage devices becoming contended over time. If a heavily used storage device becomes contended, the delay can be felt across the system. Geomancy only interacts with the system to monitor performance at each storage device of the system and to move data to a new location in the system.

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.

II. Background and Motivation

Dynamic data layout algorithms are algorithms that dynamically change the location of data in response to performance drop [3]. In such systems, there is a mix of storage devices and computation nodes, and storage hardware may not necessarily be local to computation. We will discuss current solutions to solving performance bottlenecks that involve understanding how data layouts relate to performance, and how I/O improvements can be achieved in situations where computation cannot be moved close to data.

A. 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 [4], MapReduce [5], and Twister [6] 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 [7] architecture, a processor-in-memory architecture for data objects in DRAM 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 vs. 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.

B. 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 [8] 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 [9] and the Sliding Window algorithm [10]; however, these solutions are rarely seen in production systems.

Heuristics cannot determine if a change in latency is temporary or long lasting. Some heuristics spread 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.* [11] 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.

All of the previous approaches are limited by their inability to quickly search through enough data layout to identify ones that can increase performance. Neural networks can efficiently search large search spaces and adapt to changing environments. In our data placement problem, the search space is all the potential locations a piece of data can be placed in a system. Using this, we can model how a system reacts to moving data around by observing past accesses and how they affect present ones.

III. Hardware for live testing

Our experiments with Geomancy utilizes one computation node with six mounted storage devices from PNNL's Bluesky system [12]. 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 1, 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.

IV. Motivating workloads

Geomancy was motivated by 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. 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 used 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,

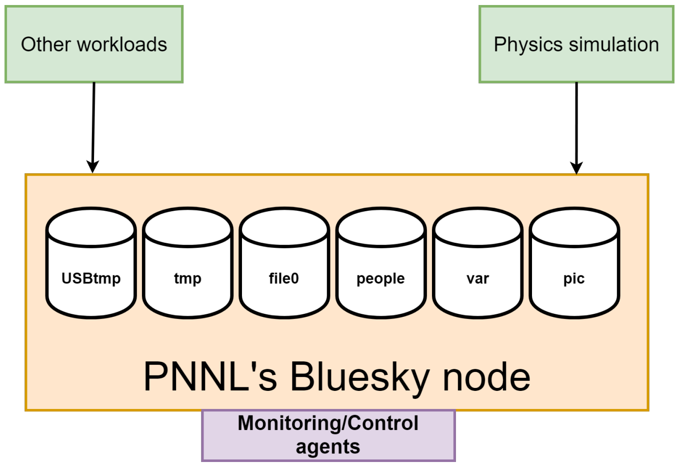


Fig. 1: A visual representation of the storage devices on the Bluesky system. This system is shared among many other users.

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 [13] is a storage cluster with an analysis, archive and tape pool. An analysis pool is a low latency disk base caching storage [14]. 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 systems telemetry.

The PNNL BELLE II [15] workload utilizes data from the specialized particle detector from the SuperKEKB [16] 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 [17] files of size from 583 KB to 1.1 GB on the Bluesky 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 want to show how Geomancy acts in an actual scientific system faced with a common workload for such a system.

V. Geomancy Design

Geomancy uses reinforcement learning to predict performance fluctuations of different storage devices. 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 system'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 2 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.

A. 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

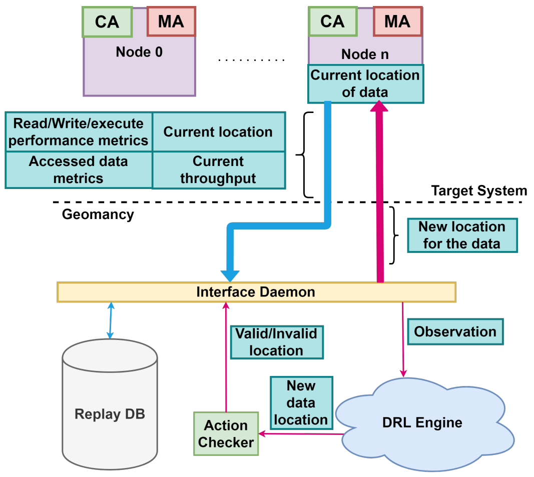


Fig. 2: Geomancy's architecture. The control agents and the monitoring agents are located on each available location on the target system. Thick arrows represent the communication between the target system and Geomancy. Thin arrows represent the communication between the agents of Geomancy. Red arrows represent the data flow during the decision process. Blue arrows represent the data flow during the performance data collection.

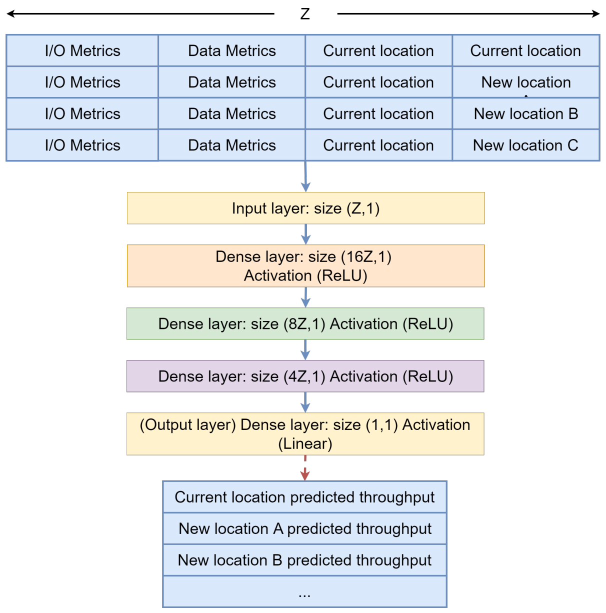

Z
I/O Metrics
Data Metrics
Current location
Current location
I/O Metrics
Data Metrics
Current location
New location
I/O Metrics
Data Metrics
Current location
New location B
I/O Metrics
Data Metrics
Current location
New location C
Input layer: size (Z,1)
Dense layer: size (16Z,1) Activation (ReLU)
Dense layer: size (8Z,1) Activation (ReLU)
Dense layer: size (4Z,1) Activation (ReLU)
(Output layer) Dense layer: size (1,1) Activation (Linear)
Current location predicted throughput
New location A predicted throughput
New location B predicted throughput
...

Fig. 3: 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.

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. Geomancy limits how often and how much data can be transferred at once without creating a bottleneck in the network for other workloads which is caused by the transfer cost outweighing the benefits.

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 Geomancy. Geomancy 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 Geomancy's dataset 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.

Once the data is stored in the ReplayDB, the Deep Reinforcement Learning [18] (DRL) engine determines any updates needed to be done to the target system's data layout. The DRL engine re-trains a neural network using the most recent values stored in the ReplayDB to calculate future values of the throughput.

B. *Unsupervised Deep Reinforcement Learning*

We approach the layout problem as a unsupervised deep reinforcement 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. As seen in Figure 3, 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. As a fully connected neural network, its weights are determined by the backpropagation of all input features. We use the Rectified Linear Unit (ReLU) activation function [19], 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.

Using dense layers enables Geomancy to calculate future values by finding relationships between all input features. 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.

C. Modeling target

Geomancy will model the throughput of the system during the run of the workload. This value will allow Geomancy 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 data with every row only having the location varying between each locations the data can be located on. Using that information, Geomancy calculates 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.

D. 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 Pearsons 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 [20]. Many features in the EOS access logs [21] 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.

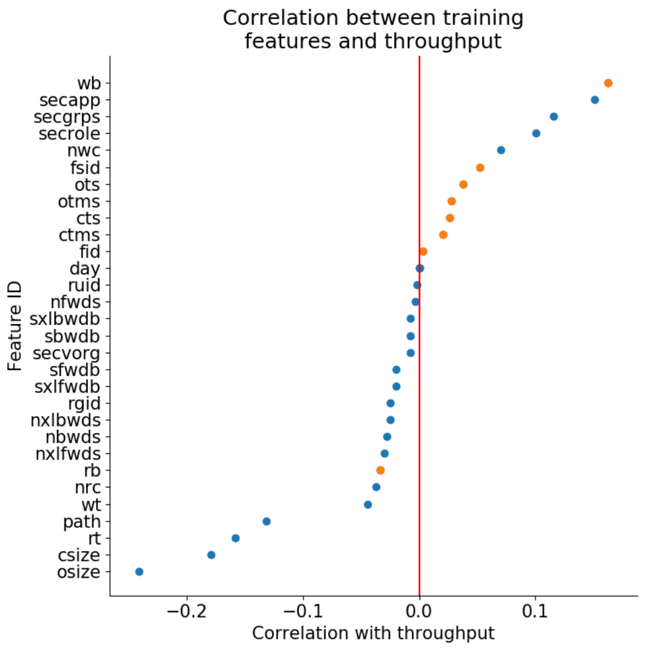


Fig. 4: Correlation between the raw access features found in the EOS logs and the throughput. We choose features (orange) that are commonly found in scientific systems that also happen to be positively correlated.

We identified six features from the workload traces in the EOS system, as seen in Figure 4. These features range in correlation since we did not want to limit the scope of what the features represent.

- Bytes read (*rb*)
- Bytes written (*wb*)
- Open time stamp in seconds (*ots*) and milliseconds (*otms*)
- Close time stamp in seconds (*cts*) and milliseconds (*ctms*)
- File ID (*fid*)
- File System ID (*fsid*)

We use correlation to identify features that are correlated with the measured throughput. By using such features, we gain an approximation of how the measured throughput will impact a throughput sensitive workload.

First, the amount of data written to a file will 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 correlated to the throughput, but it shows that not all files are accessed 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 since we wanted to model the access to the file independantly of the action which means that we want to capture the entire time to file is opened from the start, with open time stamp in seconds (*ots*) and milliseconds (*otms*), to the end, with close 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 identifyers 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.

E. 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` is assigned to 3.

We remove smaller variations from data in the ReplayDB by applying a moving average [22]. 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. 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.

F. Location generation

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, Geomancy can only take a limited amount of actions from a space of actions, and cannot move the files wholesale. Specifically, the situation that Geomancy takes all root files from all unique storage points and crams it 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.

G. Hyperparameter Tuning

To determine a useful model, we compared 23 neural network architectures and report their performance in Table I. We used Z features that we selected from the PNNL server, equalling six. 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 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 through model

TABLE I: 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	16Z (Dense) ReLU, 16Z (Dense) ReLU, 16Z (Dense) ReLU, 16Z (Dense) ReLU, 1 (Dense) ReLU
Model 7	16Z (Dense) ReLU, 16Z (Dense) ReLU, 16Z (Dense) ReLU, 16Z (Dense) ReLU, 16Z (Dense) ReLU, 1 (Dense) ReLU
Model 8	Z (Dense) ReLU, Z (Dense) ReLU, Z (Dense) ReLU, Z (Dense) ReLU, Z (Dense) ReLU, 1 (Dense) ReLU
Model 9	Z (Dense) ReLU, Z (Dense) ReLU, Z (Dense) ReLU, Z (Dense) ReLU, 1 (Dense) ReLU
Model 10	Z (Dense) ReLU, Z (Dense) ReLU, Z (Dense) ReLU, 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

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. 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.

TABLE II: Model comparisons on predicting performance over all mounts

Model number	Mean of Absolute relative Error (%)	Training time (s)	Prediction time (ms)
1	**18.88 ± 16.92**	**25.657 ± 0.801**	**55.4 ± 3.6**
2	*Diverged*	24.043 ± 1.008	49.2 ± 3.6
3	44.30 ± 21.48	25.208 ± 0.388	54.4 ± 2.6
4	20.07 ± 17.77	22.746 ± 0.502	46.5 ± 2.5
5	*Diverged*	26.290 ± 0.478	60.2 ± 3.0
6	17.63 ± 15.95	40.266 ± 0.341	70.0 ± 3.3
7	17.72 ± 16.02	47.956 ± 0.447	80.6 ± 3.7
8	18.50 ± 16.42	23.822 ± 0.498	61.0 ± 2.8
9	44.30 ± 21.48	21.931 ± 0.421	54.4 ± 2.4
10	42.67 ± 22.70	21.925 ± 0.439	54.3 ± 2.5
11	42.68 ± 22.72	15.755 ± 0.661	35.6 ± 1.6
12	29.77 ± 21.64	42.608 ± 0.549	111.5 ± 3.6
13	28.67 ± 20.83	36.860 ± 0.801	96.5 ± 3.1
14	28.96 ± 22.02	23.187 ± 0.930	63.3 ± 3.5
15	24.66 ± 19.51	38.597 ± 1.247	107.2 ± 6.5
16	25.57 ± 20.33	39.008 ± 0.625	111.2 ± 3.7
17	21.72 ± 18.80	39.659 ± 1.136	113.9 ± 6.1
18	**18.77 ± 16.83**	**27.102 ± 0.807**	**78.0 ± 3.3**
19	42.70 ± 22.74	26.371 ± 0.708	77.1 ± 3.1
20	28.00 ± 22.78	25.378 ± 1.386	73.3 ± 4.9
21	42.70 ± 22.74	44.136 ± 1.476	121.5 ± 6.9
22	21.47 ± 19.40	43.760 ± 0.701	125.0 ± 4.1
23	23.81 ± 20.25	44.066 ± 1.462	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.

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 [23]), Gated Recurrent Units (GRU [24]) and Simple RNN (the base RNN structure).

In Table II, 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 ± 3.3 ms and 80.6 ms ± 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 paper, 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

TABLE III: Prediction accuracy of model 1 on each individual Bluesky's storage points

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

between prediction accuracy and training/prediction time.

We chose model 1 since many other models diverged on one or more other storage points other then 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% ± 21.78%) about 2 × higher then Model 1 (23.91% ± 21.66%). Since we wanted a model that accurately modeled all the available ports, we decided to go with model 1.

Table III 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 56.85% prediction accuracy for the files0 mount, despite many users bombarding the system with requests, with an average accuracy of about 81.12% 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 can increase this accuracy using more features such as number of write and read calls, etc.

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 only got a 76.38% accuracy. 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 $MAE \times prediction$ 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:

$$AdjustedPrediction = prediction_i \pm MAE \times prediction_i$$

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.

H. 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, Geomancy 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 decision 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, Geomancy is able to adapt and change the data layout accordingly.

VI. Experimental Setup

a) 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 to spread files based upon the capacities of the storage devices. Before any experiments are executed, BELLE 2 is run until Geomancys 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.

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 IV, 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.

MRU: The Most Recently Used (MRU) algorithm, as described by Chou *et al.* [25], 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.

LFU: The LFU policy, as described by Gupta *et al.* [26], 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.

Random 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.

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.

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.

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 V-D. 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.

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 is not as large as it can get, we found that a polynomial representation, as shown by the dense model, produced the most accurate predictions. The benefit of dense networks over heuristics such as exponentially moving average is that neural networks are able to update their weights over time unlike heuristics which will need human input to update and therefore they might not be able to capture sudden changes in performance. This is why we have opted for the use of neural networks over 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.

b) 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.

c) 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.

VII. Evaluation

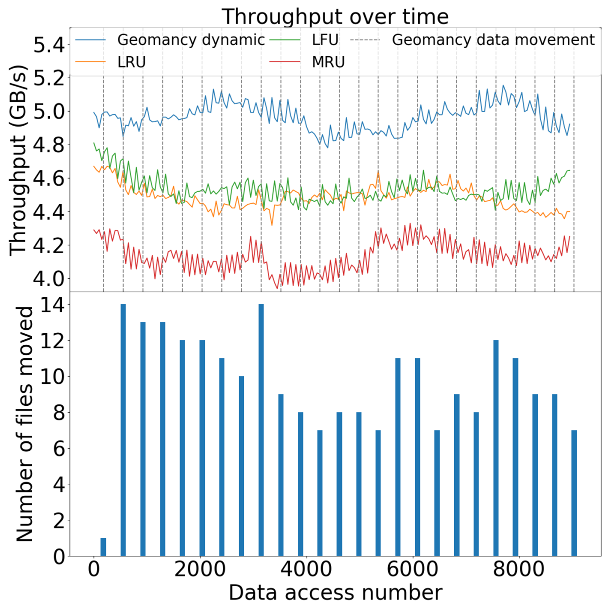
Geomancy outperforms both static and dynamic data placement algorithms by at least 11%, as shown in Figure 5. In all figures, a vertical gray line represents when Geomancy decides to move data. The blue lines below each graph corresponds to how many files are moved by Geomancy at that access number. 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 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 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.

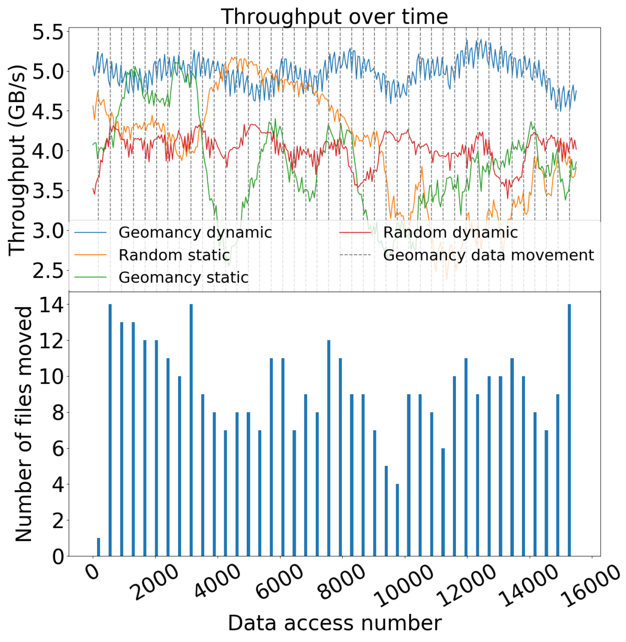
In our second experiment, as seen in Figure 5b, Geomancy outperforms all naive policies. Compared to *random static* which has an average throughput of 4.01 GB/s, Geomancy has a 24% increase over the 16,000 accesses. Similarly, *Geomancy static* has an average throughput of 3.81 GB/s while Geomancy has 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 IV, 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 files every



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



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

Fig. 5: Geomancy's performance compared to dynamic and static solutions on the live system. Gray lines represent data movements done by Geomancy dynamic. Size of the data that Geomancy moves ranges from 583 KB to 1.1 GB.

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.

VIII. Overhead Study

TABLE IV: Performance and utilization of storage points available to Geomancy

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	21.175
file0	7.61 ± 13.73	64.8125
var	1.26 ± 2.81	6.25
people	1.69 ± 3.46	7.3625
Geomancy	5.49 ± 11.59	100

To demonstrate Geomancy's data movement overhead, we measure Geomancy's throughput when it moves data at specific data movement milestones. In Table IV 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 II, we showed that the prediction overhead of our selected neural network was at most 53.7ms and the training overhead was on average 25.3s 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. 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 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

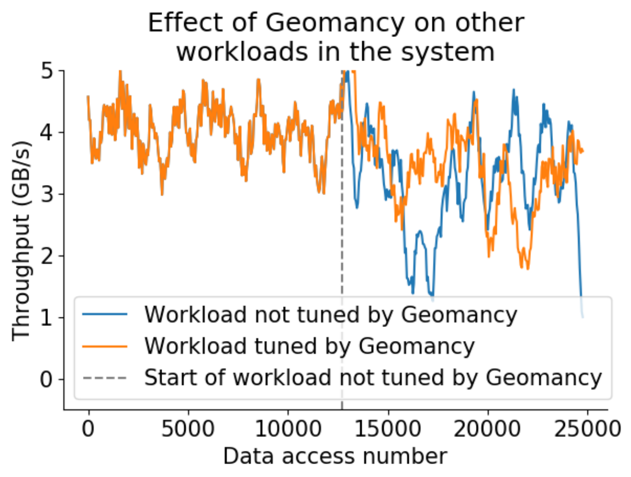


Fig. 6: When Geomancy is faced with a new environment, such as when another workload is started, it will take time to adapt to the new changes. Although the original performance drops, Geomancy is able to respond to the changes and attempt to push performance back to what it once was.

of the workload it was tuning for a brief period. This was not the initial goal of the reinforcement learning but it does show a larger impact of running Geomancy for other workloads running on the system.

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

A. Static approaches

Early work in data layouts involves distributing data evenly across a majority of hardware pieces present in systems. MM-Packing [27] 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 [28]. 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 [29], and concluded similarly about the trade-off situation of different algorithms.

B. 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 [30], [31], [32]. ASCAR [33] 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 [34] (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.

C. 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.* [35]. They created a data management service, Univistor, that provides 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.* [36] 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 [37]. 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 [38] 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.

X. Conclusion and Future Work

Geomancy accurately captures changes in performance as the workloads runs on the target system. Those predictions are then able to be used to change the data layout of the system. Experimentally, we demonstrate inter-workload congestion reduction and increases in overall throughput from 11% to 30%. Compared to algorithmic or manual data placement, Geomancy is superior in that it reduces bottlenecks and can anticipate when performance fluctuations may happen. By predicting when and where performance may drop, moving data before the slowdowns occurs stabilizes performance and prevents fluctuations in throughput. With our modeling technique, we gain adequate accuracy and low error variations between predicted and real values while keeping a low training and prediction time as seen in table II.

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.

Acknowledgements

We would like to thank our collaborator James Byron for their help in reviewing this paper and our collaborators at Pacific Northwest National Laboratories 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." We also thank NVIDIA Corporation for their donation of a TITAN Xp GPU which will be used as part of the development of Geomancy. 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.

References

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

[2] 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.

[3] Y. Hu, R. J. Blake, and D. R. Emerson, "An optimal migration algorithm for dynamic load balancing," *Concurrency: Practice and Experience*, vol. 10, no. 6, pp. 467–483, 1998.

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

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

[6] 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.

[7] 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.

[8] 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.

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

[10] 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.

[11] 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.

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

[13] 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.

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

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

[16] 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.

[17] 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.

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

[19] 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.

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

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

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

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

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

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] 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.

[36] 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.

[37] 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.

[38] 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.