2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems

Modeling HPC Storage Performance Using Long Short-Term Memory Networks

Youbiao He
Department of Computer Science
Iowa State University
Ames, IA, United States
yh54@iastate.edu

Dong Dai
Department of Computer Science
University of North Carolina at Charlotte
Charlotte, NC, United States
dong.dai@uncc.edu

Forrest Sheng Bao
Department of Computer Science
Iowa State University
Ames, IA, United States
fsb@iastate.edu

***Abstract*—As scientific applications become increasingly data-intensive, the I/O and storage components can easily become the bottlenecks of high performance computing (HPC) systems. Such a trend drives recent development of complex, layered HPC I/O software stack, which contains a huge number of tunable parameters, to achieve good I/O performance. It then becomes necessary for the application developers to fully understand the performance projection of their applications given a set of chosen parameters, and also their potential effects on other concurrent applications. Although there are a number of recent studies aiming to model the relationship between I/O performance of applications and their parameters to provide such insights, the complexity of HPC systems and the interactions between concurrent applications make existing approaches over-simplified and far from practical. In this paper, we try to solve this problem using a powerful deep learning tool, the long short-term memory (LSTM), together with the full history of I/O settings of all concurrent applications in the system. To prove the concept, we carry out extensive experiments using the IOR benchmark and train the model using real system logs. Empirical results show that our LSTM-based model can predict the write speed more accurately than the state-of-the-art approaches. We hope this work can initiate the study toward more accurate and practical I/O performance estimation in HPC systems.**

***Index Terms*—High Performance Computing, Parallel I/O, Deep Learning, LSTM**

I. Introduction

In recent years, scientific applications running on high performance computing (HPC) platforms are becoming increasingly data-intensive. Such a trend makes applications' I/O performance more and more critical to the overall efficiency and performance of HPC platforms. To achieve the best I/O performance, scientific application developers often leverage the existing, layered I/O software stack to facilitate their development. Hence, a typical data access in an HPC application can easily go through multiple I/O libraries such as HDF5 [1] or netCDF [2], ROMIO [3], I/O scheduler [4], burst buffer [5], and parallel file system library [6], etc. The key challenge of utilizing such a layered I/O stack is that each of these software layers has many configurable options/parameters, affecting the overall I/O performance [7]. In addition, many of these parameters affect not only their own layers but also closely interact with other layers and change their performance. This complexity lays significant challenges on the application developers as it is hard for them to choose the appropriate parameters for each storage stack layer to achieve expected, optimized I/O performance.

Previous studies have shown that the I/O-relevant parameters of different layers in the I/O stack could largely determine the overall I/O performance of the applications; and it is feasible to predict the expected I/O performance based on the known parameters of an application [8]–[11]. For example, Behzad *et al.* develop a non-linear regression model to predict the data write time given the parameters of the Lustre stripe settings [8]. The HPC application developers, then, can utilize such prediction to choose their parameters wisely. These studies provide important insights for the application developers as well as the HPC system administrators to fine-tune their applications and systems.

However, these studies have two major issues that significantly limit their usages in real-world scenarios. First, because of the large number of cross-layer tunable parameters and their wide range of values, the exploration space could be huge. The interactions among parameters from different layers could also be complex and hard to model by using simple regression methods. So, most of the existing studies often only focus on parameters from a specific layer to achieve accurate prediction. On the other hand, it is known that an optimal setting in one layer does not necessarily lead to optimal performance across layers. For example, an optimal *chunk size* in HDF5 [12] can lead to poor performance just because the number of I/O processes (configured in the MPI I/O library) does not match it [13].

Second, in the real-world scenarios, it is typical to have many applications running concurrently, competing for I/O resources and affecting each other. Many existing studies [8], [10], [11] often make a simple assumption that the single application will take the full system and build models based on this unrealistic assumption. Hence, the built models can easily become inaccurate when the applications run in a real system. In addition, the constantly changing I/O workloads from concurrent applications also complicate the problem of parameter-based I/O performance prediction, which limits the accuracy of simple machine learning methods such as linear or

This work is partially funded by NSF grants MCB-1821828 and CNS-1817089.

978-1-7281-2058-4/19/$31.00 ©2019 IEEE
DOI 10.1109/HPCC/SmartCity/DSS.2019.00157

nonlinear regression. Note that, in current HPC storage stack, most of the I/O-relevant parameters cannot change when the applications are running. Hence, without knowing the possible concurrently running applications in advance, developers may find it not particularly useful to predict the dynamic I/O performance of their applications under different workloads. However, we believe that the I/O parameters can become changeable by revising the libraries; then developers can leverage the accurate dynamic prediction to set rules for parameter changing during runtime to significantly improve the performance. In addition, the accurate dynamic performance prediction with the consideration of concurrent workloads will be useful to HPC system administrators and job schedulers to optimize the whole system. Thus, we focus on this aspect in this study.

The interactions among parameters of multiple storage software layers and among concurrent applications make the whole system very complex to model or predict. Existing studies fall short of handling these scenarios. Fortunately, the recent progress on deep learning (DL) shows that the deep neural networks have the ability to model very complex but deterministic systems [14]. Hence, this study boldly hypothesizes that the I/O performance of each application in a concurrent environment can be modeled as a function of the I/O-relevant parameter settings of all running applications and the history of them by using deep learning techniques. Through this idea, we formulate the problem into predicting HPC applications' I/O performance from a stream of I/O settings, and solve it with the latest advances of sequence modeling in deep learning. To the best of our knowledge, this is the first try in this direction and we believe that this new holistic approach is promising in the future to enable dynamic, on-line parameter tuning for production HPC systems.

To achieve this, in this study, we first design a series of experiments based on the representative I/O benchmark IOR [15] to collect training data, which are I/O parameters of concurrent applications and their I/O performance in each sample interval. Then, based on the training data from a large number of experiments, we propose a long short-term memory (LSTM) network to model and predict the I/O performance of each application. LSTMs [16] are one special kind of recurrent neural networks (RNNs), which are efficient to extract dependency information in the sequential data in classification or regression tasks. In practical HPC applications, the I/O performance at each timestamp is not only determined by the current system state and workloads but also by those of previous time. Therefore, we leverage an LSTM network to fully take these dependencies into account in this study. Our experimental results also confirm the advantages of LSTM. Our contributions are as follows:

- We show the feasibility of modeling concurrent I/O performance as a function of the history of and current I/O parameters of concurrent applications.
- We formulate the problem into a deep learning task and design an LSTM model to tackle the task.
- We design a strategy to generate real data to train the LSTM-based model and show its effectiveness through intensive experiments.

The rest of the paper is organized as follows. The background of HPC storage stack and the LSTM are introduced in sections II and III, respectively. In section IV, we present our HPC experiment design and LSTM-based prediction model. The experimental results are shown in section V. The related work and conclusions are given in sections VI and VII, respectively.

II. Background on HPC Storage System

HPC platforms are designed for performance-critical tasks, which require a highly concurrent and rapid storage system that is achieved using a complex storage stack [17]. They usually consist of a deep storage stack including layers like I/O libraries, caching and buffering, parallel file system, and high-end storage devices (usually RAID). A demonstrative example of these core layers is given in Fig. 1, where I/O libraries cover both the high-level I/O libraries like HDF5 [1], netCDF [2], or ADIOS [18], and the middleware layer MPI I/O library like ROMIO [3]. Their requested data might be stored in an optional buffering layer like Burst Buffer [5], or in parallel file system that normally runs on a standalone storage cluster, connected through high-speed network. RAID is then built upon persistent devices like hard disks to improve data integrity and reliability.

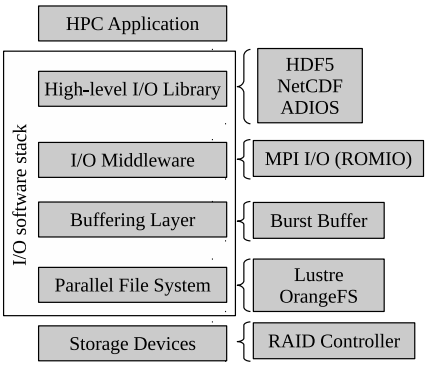


Fig. 1: Typical HPC storage stack.

For each storage stack layer, numerous parameters affect its performance. Because of the complexity of the storage system, these parameters might have a significant impact on the performance in other layers too. For example, HDF5 I/O library supports *chunked* data layout, where datasets are split into multiple chunks stored separately in the file [12]. Since chunks are accessed individually, the I/O performance could be better if each process operates on their own chunks without contention, which is relevant to the number of parallel I/O processes [13]. Interestingly, the number of I/O processes is not controlled by HDF5 itself; instead, by the number of aggregators defined by collective I/O optimization in MPI I/O library [19]. This simple example shows that one single parameter (e.g., *chunk size*) can impact, and also can be

impacted by, parameters from other layers (e.g., *aggregator number*). It is clear that complex correlations among layers of storage stack make the performance prediction a challenging task.

Many existing works leverage machine learning techniques such as linear regression, decision tree, to predict the I/O performance [8]–[11]. However, they only work with simplified problems, where either only a small set of parameters from a single layer is considered; or build the system based on unrealistic assumptions such as the whole HPC storage system is taken by a single application. In the real world, there are many cross-layer parameters to tune and there are often more than one applications running on the HPC system, contending the I/O resources and affecting each other. In these cases, the parameter setting of one application will affect the I/O performance of both itself and all the other concurrent applications. To accurately predict I/O performance of applications, the parameter settings of all applications should be used to represent the system state and as the input of the prediction model. The traditional machine learning techniques such as linear regression and decision tree, can not extract the dependency in the time-sequential data, hence are limited for our use case.

In this study, we propose to utilize a long short-term memory (LSTM) model to solve the aforementioned problem. More details are introduced in the following sections.

III. Deep Learning: Long Short-Term Memory

Deep learning is a rapidly emerging machine learning technique that uses extremely multi-layer, thus *deep*, networks to mimic very complicated multi-variate functions [14]. The complexity of I/O performance hence makes it a perfect application of deep learning. The typical deep learning architectures include convolutional neural networks (CNNs) and recurrent neural networks (RNNs).

In this study, we leverage a long short-term memory (LSTM) network, one of special kind of RNNs, to predict the real-time I/O performance of multiple concurrent applications. First, unlike the traditional machine learning techniques that all the inputs and outputs are treated as independent of each other, LSTMs are capable of modeling input and/or output comprised of sequences of elements that are not independent [16]. In our case, the I/O performance at different time steps are highly related. Thus, to accurately predict the I/O performance using a history of I/O settings, the dependency in the input and/or output data must be considered. Second, unlike the standard/simple RNNs that can only deal with the short-term dependency, LSTMs are able to extract both long and short-term dependency in the sequential data. In the short-term dependency problem, we only need to take the short history into account while performing the present task. In this work, the I/O performance at the current time step is not only affected by the recent system states such as the previous one or two steps of system states but also impacted by those of a longer history, which should be considered as a long short-term dependency problem.

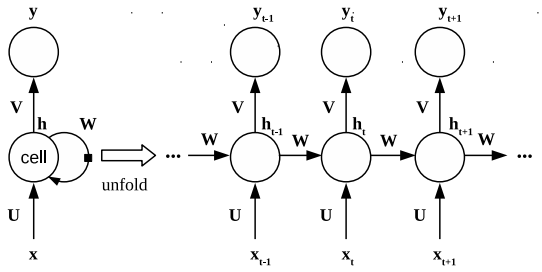


Fig. 2: A recurrent neural network and its unfolding structure. A RNN unit dealing with the dependencies in the sequential data is called a *cell*. It gets inputs from other cells at previous time steps, which is represented with a black square (representing a delay of one time step) on the left.

Fig. 2 shows the basic structure of RNN and the unfolding in time of the computation. The output of each RNN cell $\boldsymbol{h}_t$ and the output of the RNN for each state $\boldsymbol{y}_t$ are computed as follows:

$$\begin{cases} \boldsymbol{h}_t = \tanh(\boldsymbol{a} + \boldsymbol{W}\boldsymbol{h}_{t-1} + \boldsymbol{U}\boldsymbol{x}_t) \\ \boldsymbol{y}_t = \boldsymbol{b} + \boldsymbol{V}\boldsymbol{h}_t \end{cases} \tag{1}$$

where $\boldsymbol{W}$, $\boldsymbol{U}$ and $\boldsymbol{V}$ are the weight matrices, $\boldsymbol{a}$ and $\boldsymbol{b}$ are the bias. It is clear that the output $\boldsymbol{y}_t$ at any step t is jointly determined by the current input x_t and the output of the cell of last step $\boldsymbol{h}_{t-1}$, which represents the dependency on the history of the input.

The LSTMs have the same structure as this standard/simple RNN. The difference is that each cell of an LSTM network has a more complex structure. It consists of input gate, forget gate and output gate, thus, a more complicated function to map the current input $\boldsymbol{x}_t$ and last cell output $\boldsymbol{h}_{t-1}$ to the current cell output $\boldsymbol{h}_t$. This also makes LSTM able to remember the input for a long time. In our case, $\boldsymbol{x}_t$ is the I/O setting of the current time step t, and a history of I/O settings of all the previous steps are represented by $[..., \boldsymbol{x}_{t-2}, \boldsymbol{x}_{t-1}]$. They jointly determine the I/O performance y_t at the time step t. The dependency on the history of I/O settings up to step $t-1$ is represented as the output of the last cell, $\boldsymbol{h}_{t-1}$. Therefore, by using this "memory" mechanism of LSTM networks, the long and short-term dependencies of the I/O settings and I/O performance are extracted and effectively utilized in our prediction task.

IV. Methodology and System Design

In this study, we model this problem as a sequence prediction problem. Suppose that we periodically sample the system. At every time step, we have current I/O settings and the corresponding performance of each application. Given a stream of such I/O settings and I/O performance, we intend to approximate a function that, for each application, maps a history of I/O settings up to step t to the I/O performance at the same step t. Such idea is illustrated in Fig.3. More details will be discussed in the following subsections.

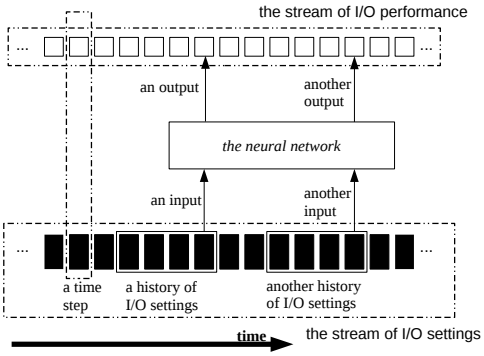


Fig. 3: The problem formulated as the mapping from the stream of I/O settings to the stream of I/O performance, for all the concurrent applications.

A. I/O Parameters and Synthetic I/O Benchmark

The tunable part in Table I lists all the parameters we choose to represent the I/O settings. These cross-layer parameters interact with each other, and determine the I/O performance of each application as discussed in section II. Most of the existing studies such as [8]–[11] either use part of these parameters or do not consider the interactions of the parameters between different storage stack layers. As the first step of our work, we choose these typical cross-layer parameters from three different I/O stack layers to predict the I/O performance of concurrent applications.

The constant parameters in Table I, such as *transfer size*, *block size* and *segment count*, are selected to get the best I/O performance while running multiple applications. In addition, due to the limit of the resource of the HPC cluster, the maximum number of concurrently running IORs are set to 5 and the number of processes of each IOR is set to 64.

To effectively train our LSTM network, we need to collect enough data reflecting the relationships between I/O settings and their corresponding I/O performance of all the concurrent applications. In this study, instead of using real applications to generate such training data, we utilize the I/O benchmark IOR [15] to generate needed training data. First, the previous study [15] has shown that IOR is good at representing and simulating the I/O behaviors of real applications. It is able to reflect most of HPC I/O requirements in the broader HPC workload, and the real full application can be effectively replaced by it. Second, real applications are complex and do not expose their I/O parameters, making it difficult to effectively collect the data. IOR exercises configurations across different storage stack layers such as MPI-IO, HDF5, and POSIX, also making it a natural choice for our research.

B. Experiment Setup

All experiments are conducted on the HPC platform CloudLab [20]. CloudLab is a large-scale and flexible distributed scientific infrastructure designed to support fundamental advances in cloud architectures and applications. We create an HPC cluster by instantiating an XML-based RSpec profile on the CloudLab. We use a Lustre file system with 5 Lustre server nodes (1 node for MGS/MDT and 4 nodes for OSTS) and 16 Lustre computing nodes. Each node has 16 cores and 16GB of memory.

We define one experiment as a concurrent run of multiple (5 in this work) IORs of distinct settings. Each IOR reads from or writes to one file on the same drive. As illustrated in Fig. 4, the IORs are started sequentially. Two consecutive IORs start apart by a random period between 10 and 30 seconds.

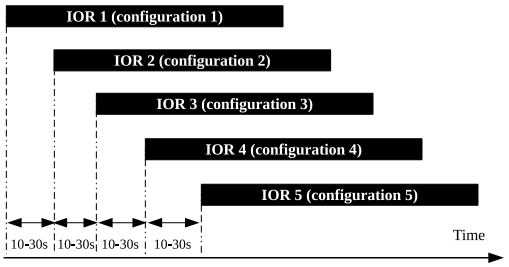


Fig. 4: One experiment consists of 5 IORs, referred to as IOR 1 to IOR 5 based on their starting times in the rest of the paper. They start sequentially with a delay randomly picked from 10 to 30 seconds between two consecutive ones.

In the design of IOR, each IOR includes multiple (64 in this work) processes. They write to the same file, but to different parts of it. Each written part for one process consists of a fixed amount of blocks, which is defined by the parameter *segment count* (20 in this work as shown in Table I). The block size is the same for all processes in one IOR application. For example, to write a file of 64MB, each of the 64 processes writes a part of 1MB. The processes of the same IOR do not start simultaneously but sequentially with a short-time (within one millisecond) delay between two consecutive ones as set in IOR source code. An IOR process does not write continuously but has a random-length cool-down period after writing a transfer unit. The size of the transfer unit is the same for all processes in one IOR. For example, if the transfer size is 64KB, it will take 16 times for a process to write its 1MB block of a file. In our experiment, the transfer size is equal to block size. i.e., one block has one transfer unit.

Each IOR has 7 tunable parameters listed in Table I. Parameters are randomly chosen from the parameter space per-IOR and per-experiment. Real-time write speed of each IOR is used to measure the I/O performance. Since there are 5 IORs in our experiments, 5 write speeds are collected at each time step.

1) Data Collection: In this subsection, we discuss how to prepare the data for training the LSTM-based model. As mentioned earlier, when training a neural network, training samples, each of which is a tuple of an input and an output, are fed into the neural network for it to update itself.

In this paper, we want to predict per-application, real-time I/O performance for concurrent programs, given a history of

TABLE I: The Parameter Settings for HPC Experiment

Parameter Type	Parallel I/O Stack Layer	Parameter Name	Tuning Range	Tuning Step
Tunable	Lustre	stripe count (strp_fac)	1-4	1
		stripe size (strp_unt)	16MB-128MB	16MB
	MPIIO	collective buffer nodes (cb_nds)	8-64	8
		collective buffer size (cb_buf_size)	16MB-128MB	16MB
	HDF5	HDF5 alignment (align(thresh, bndry))	(1KB, 2MB)-(16KB, 32MB)	(1KB, 2MB)
		HDF5 chunk size	8MB-64MB	8MB
Constant	number of running IORs		5	
	transfer size		8MB	
	block size		8MB	
	segment count		20	
	number of processes		64	

I/O settings of them. Therefore, in a training sample, the input is a history (20 time steps in this work) of I/O settings of all IORs and the output is the I/O performance of them. The structure of each pair of input and output is illustrated in Fig. 5. Because there are 8 I/O-setting parameters (7 tunable ones from Table I plus whether the IOR is reading or writing) per application, 5 applications concurrently, and 20 time steps, the input is a vector of 800 numbers. If at a time step, less than 5 IORs are running concurrently, the elements in the input vectors corresponding to the non-running IORs are padded with zeros. This is a common technique used in machine learning to ensure the fixed length of the input vector. The output is relatively simple, just 5 numbers, which are the per-application write speeds of 5 IORs.

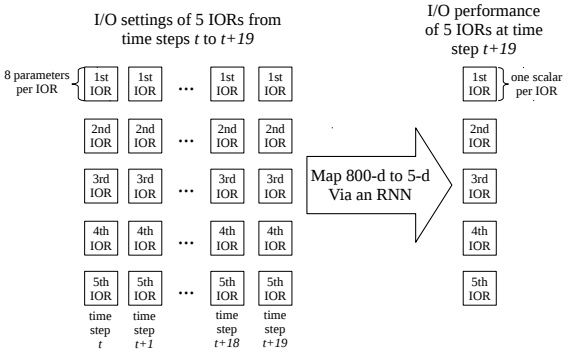


Fig. 5: The format of collected data from training an RNN.

The write speed of an IOR is estimated as follows. Because each IOR consists of 64 processes, the write speed of an IOR is the sum of the write speed of all its 64 processes. Because it is very hard to precisely measure the write speed of a single process, we use an approximation. If at the time of the time step i (note that i is just an integer not a timestamp) an IOR process is writing a transfer unit, then the write speed of the IOR process at time step i is approximated by $\frac{\text{transfer size}}{t_e - t_s}$ where t_s and t_e are the start and end timestamps of the written transfer unit. If at the time of the time step i, the IOR process is cool-down, then its write speed at time step i is zero. The start and end times of each transfer unit of each IOR process can be logged by modifying the source code of the IOR program.

2) Data Cleaning: As described in the last subsection, we assume that the write speed is constant during a transfer-size write, and we add all these write speeds up to compute the total speed of all processes. As a result, some collected data are redundant and some are inaccurate for measuring I/O performance. Therefore, we clean the data by discarding those redundant and inaccurate data.

For the redundant data, assume we have two samples at time steps i and j (i and j are not necessarily consecutive), at which all the processes write the same transfer unit, then the write speed of every process at these two time steps will be the same, thus, the total write speed of the IOR at these two time steps are the same. If this occurs at time steps i and j for every running IOR, we will get two exactly same samples, which makes one of them redundant and discarded.

The inaccurate data is caused by the following fact: The write speed of a process while writing a transfer unit can be impacted by the starting or ending of the other IOR. Suppose some process p of IOR 1 is writing a transfer unit from time t_s to t_e, and IOR 2 starts to write its file in this time range. Then the write speed of IOR 1 on process p will be impacted and can no longer be viewed as constant. In practice, there are many processes in IOR 1 to be impacted. In this case, all the data collected during t_s and t_e are inaccurate and discarded in this work.

C. LSTM Network Design

Our network uses a typical LSTM network architecture given in Fig. 6, which consists of an input layer, hidden layers (includes 2 LSTM layers), and a fully-connected layer. The input layer takes the 800d input vector. In each state[1] of the LSTM, 40 elements of the input vectors are fed into the first LSTM layer of the hidden layers. The 40 elements correspond to the I/O settings of 5 IORs ($5 \times 8 = 40$) at a time step. The LSTM layer has 20 hidden states, corresponding to 20 such 40-element batches, thus 20 time steps. Finally, the output from the LSTM layers is fed into a fully-connected layer and outputs a 5d vector, corresponding to the write speeds from IOR 1 to IOR 5.

[1]not to be confused with the steps when sampling an HPC system

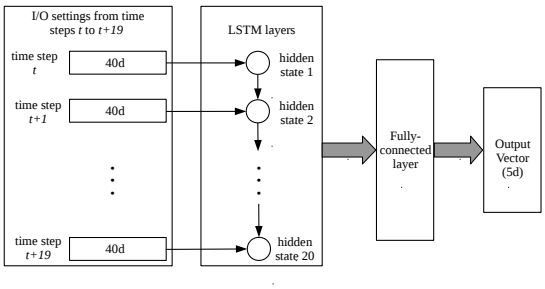


Fig. 6: Our LSTM network architecture.

V. Experiments

In this section, we verify our hypothesis that it is possible to predict real-time per-application I/O performance (write speed) based on a history of I/O settings with proposed LSTM model. To demonstrate the advantage of the proposed LSTM model on this prediction problem, 4 other machine learning techniques such as linear regression, decision tree, dense network, and simple RNN, are also implemented as the baselines.

A. LSTM network setup

We prepare training data from 1200 experiments as described in Section IV-B1. After data cleaning, there are 57735 samples. We use 10-fold cross-validation [21], thus 90-to-10 split between training and test sets, to train and test the neural networks. Cross validation is a model evaluation technique such that all samples in the dataset can be used while there is no overlap between the training and test sets in each round.

Our LSTM network is developed based on Keras [22], a high-level neural networks API, using Tensorflow backend [23]. The learning rate is set to 0.01 and the batch size is 120 while training. The loss function is mean absolute error (MAE), which is defined as follows:

$$\frac{1}{n_s}\sum_i |y_i - \hat{y}_i| \tag{2}$$

where y_i and $\hat{y}_i$ are the ground-truth and predicted values, respectively, of one training sample, and n_s denotes the total number of the training samples. All samples are normalized by min-max scaling [24] before fed into the LSTM network.

B. Evaluation metrics

We evaluate the performance of our model using mean absolute percentage error (MAPE) [25]. Let $\hat{z}_i$ be the predicted value of one test sample, z_i be the corresponding ground-truth value, and n_t denote the number of test samples, then the MAPE is computed as follows:

$$\frac{1}{n_t}\sum_i |\frac{z_i - \hat{z}_i}{z_i}| \tag{3}$$

It tells the absolute difference between the predicted and ground-truth write speed as a percentage.

C. Results and discussions

TABLE II: MAPE (%) of 10-fold Cross Validation (mean)

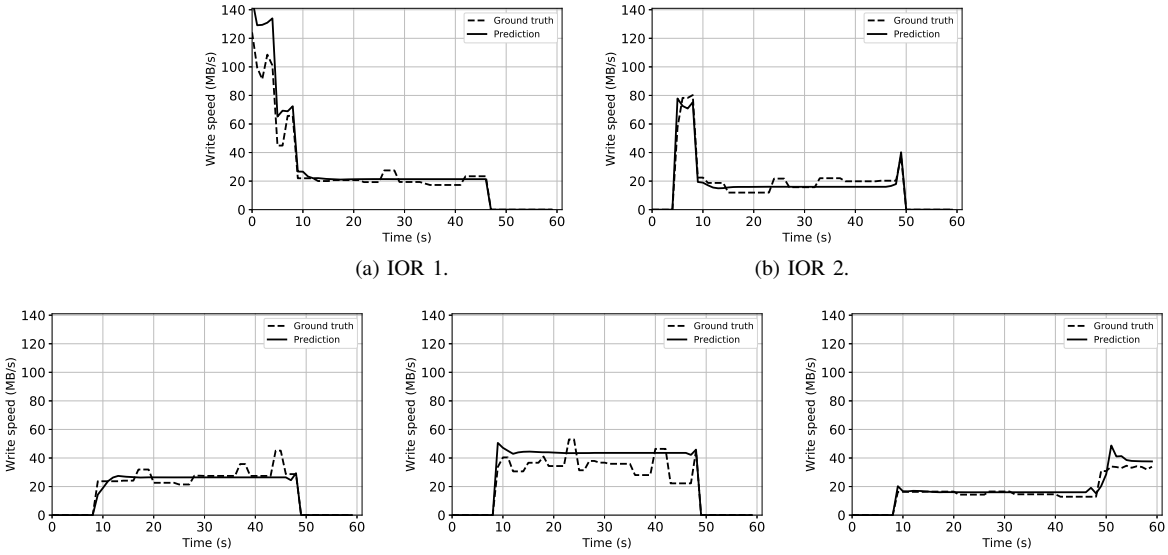
	IOR 1	IOR 2	IOR 3	IOR 4	IOR 5
Linear Regression	31.34	27.56	28.64	28.55	28.57
Decision Tree	27.94	27.59	28.38	29.29	28.62
Dense Network	38.55	31.39	28.17	25.53	28.49
Simple RNN	28.87	25.89	26.97	26.47	25.99
LSTM	**17.35**	**16.42**	**15.00**	**14.87**	**15.73**

To validate our model, we implement 4 other classic machine learning models as the comparison baselines, i.e. linear regression, decision tree, dense network, and simple RNN. The linear regression and decision tree based performance prediction models are implemented with Python package scikit-learn, and the dense network and simple RNN models are implemented with Python in Keras. To find the optimal hyper-parameters of these baselines, the grid search is performed on each model. For the decision tree, the maximum depth is set to 7 after searching in the range of $[1, 19]$ with step 2. The dense network has 2 fully-connected layers, chosen from 2 to 4. The number of neurons for each layer is finally set to 100 after searching in the range of $[50, 150]$ with the step 25, and the hyperbolic tangent (tanh) is chosen as the activation function for each layer after comparing with rectifier (relu) and sigmoid function. The simple RNN has the same structure as our LSTM network. The only difference is it uses simple RNN as its cell instead of LSTM.

The prediction performance of these baselines and our LSTM network on each of the 5 IORs are reported in Table II. For any of the IORs, the LSTM model gives the much lower MAPE compared with other machine learning methods, which shows that our initial results are most accurate and encouraging. In addition, the simple RNN also gets a better overall prediction performance compared with 3 other baselines. The MAPE of simple RNN is just higher than that of the decision tree when predicting IOR 1 and dense network when predicting IOR 4. This is because that the RNN is able to extract the dependency information in the sequential data. But the simple RNN is not good enough by only dealing with the short-term dependency to predict the I/O performance in our problem as discussed in section III. Therefore, this result demonstrates that the long-term dependency indeed exists in the sequential I/O performance of multiple concurrent applications and LSTM is a suitable model to predict the I/O performance by using a history of I/O settings.

In Fig. 7, we visualize the write speeds, from the ground truth and from our prediction model, from one randomly selected experiment. The overlap between the two curves echos the promising results reported above in MAPE.

As another way of looking into the results, on one of the IORs, for all time steps, we make a correlation plot of actual and predicted write speed in Fig. 8. The linear regression between actual and predicted write speed is represented in the solid line. Such a line for perfect prediction, i.e., write speed is always exactly predicted, is given in a dashed line. The close proximity of the two lines reveals the great success



(a) IOR 1. (b) IOR 2.

(c) IOR 3. (d) IOR 4. (e) IOR 5.

Fig. 7: The ground truth and prediction of I/O performance for all 5 IORs from one selected experiment.

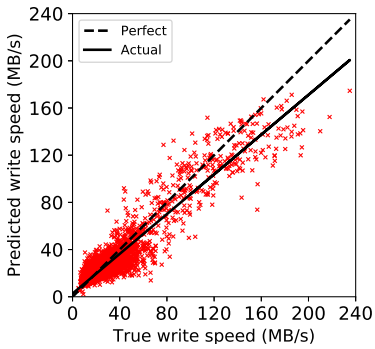


Fig. 8: Correlation analysis of the predicted and ground-truth write speed for IOR 1

of our approach, with points distributed well along either of them.

VI. Related Work

In the modern application of high performance computing (HPC) systems, the large-scale scientific simulations often produce a large amount of I/O traffic due to unprecedented growth of the volumes of data. Therefore, in recent years, there are a significant amount of studies focusing on improving parallel I/O performance. One of the most popular approaches for the I/O performance improvement is analyzing parallel I/O behavior. Among these studies, some of them implement the existing I/O performance analyzing tools. Darshan is an I/O characterization tool, designed to capture the I/O behavior suitable for the long-term deployment for applications [26]. Uselton *et al.* [27] explore the performance ensembles to understand their I/O behaviors by using an extended version of an existing performance tool called IPM (Integrated Performance Monitoring). Some other works of I/O behavior analyzing such as [28]–[30], are devoted to characterizing the workload or I/O behavior of some specific file systems of supercomputers by statistically analyzing a large amount of data from a variety of applications.

Another popular technique of I/O performance improvement is tuning parallel I/O parameters by modeling I/O performance or searching the parameter space of HPC system. In [8], [10], [11], the machine learning techniques are leveraged to model I/O performance and automatically select the appropriate parameter values. Behzad *et al.* [7] develop an auto-tuning system that search a large configurable parameter space by exploring genetic algorithms. They also propose another auto-tuning framework that select the optimal configurations by matching the data write patterns, which is extracted by tracing high-level I/O accesses, with previously tuned I/O kernels [31].

The proposed model in this work is capable of predicting the I/O performance with more details. Therefore, our model can be applied in both analyzing I/O behavior and modeling I/O performance.

VII. Conclusion and Future Work

In this work, we propose a long short-term memory (LSTM) based approach to model and predict the real-time I/O performance of the parallel applications based on their key I/O-relevant parameter selections in HPC systems. Specifically, we introduce and discuss the design of HPC experiments and the LSTM-based prediction model in detail, and also evaluate

our LSTM based model by comparing with other machine learning techniques. The results confirm that LSTM is able to extract the long short-term dependency in the sequential I/O performance and gives the accurate prediction, thus, we believe that our approach is a more effective way to accurately predict I/O performance of HPC applications. In the future, we plan to enlarge the tunable parameter space by increasing both the granularity and the dimension of parameters, and further apply proposed approach combined with other sequence prediction algorithms to make more accurate prediction.

REFERENCES

[1] Mike Folk, Gerd Heber, Quincey Koziol, Elena Pourmal, and Dana Robinson. An Overview of the HDF5 Technology Suite and Its Applications. In *Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases*, pages 36–47. ACM, 2011.

[2] Jianwei Li, Wei-keng Liao, Alok Choudhary, Robert Ross, Rajeev Thakur, William Gropp, Robert Latham, Andrew Siegel, Brad Gallagher, and Michael Zingale. Parallel netCDF: A High-Performance Scientific I/O Interface. In *Supercomputing, 2003 ACM/IEEE Conference*, pages 39–39. IEEE, 2003.

[3] ROMIO website. http://www.mcs.anl.gov/research/projects/romio/index-archived.html.

[4] Dong Dai, Yong Chen, Dries Kimpe, and Robert Ross. Two-choice Randomized Dynamic I/O Scheduler for Object Storage Systems. In *SC14: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 635–646. IEEE, 2014.

[5] Nawab Ali, Philip Carns, Kamil Iskra, Dries Kimpe, Samuel Lang, Robert Latham, Robert Ross, Lee Ward, and P Sadayappan. Scalable I/O Forwarding Framework for High-Performance Computing Systems. In *2009 IEEE International Conference on Cluster Computing and Workshops*, pages 1–10. IEEE, 2009.

[6] Lustre Software Release. http://lustre.org/.

[7] Babak Behzad, Huong Vu Thanh Luu, Joseph Huchette, Surendra Byna, Ruth Aydt, Quincey Koziol, Marc Snir, et al. Taming Parallel I/O Complexity with Auto-Tuning. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 68. ACM, 2013.

[8] B. Behzad, S. Byna, S. M. Wild, Prabhat, and M. Snir. Dynamic model-driven parallel I/O performance tuning. In *2015 IEEE International Conference on Cluster Computing*, pages 184–193, Sept 2015.

[9] Bing Xie, Yezhou Huang, Jeffrey S. Chase, Jong Youl Choi, Scott Klasky, Jay Lofstead, and Sarp Oral. Predicting output performance of a petascale supercomputer. In *Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '17, pages 181–192, New York, NY, USA, 2017. ACM.

[10] F. Isaila, P. Balaprakash, S. M. Wild, D. Kimpe, R. Latham, R. Ross, and P. Hovland. Collective I/O tuning using analytical and machine learning models. In *2015 IEEE International Conference on Cluster Computing*, pages 128–137, Sept 2015.

[11] S. Kumar, A. Saha, V. Vishwanath, P. Carns, J. A. Schmidt, G. Scorzelli, H. Kolla, R. Grout, R. Latham, R. Ross, M. E. Papka, J. Chen, and V. Pascucci. Characterization and modeling of PIDX parallel I/O for performance optimization. In *SC '13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 1–12, Nov 2013.

[12] Mark Howison. Tuning HDF5 for Lustre File Systems. In *Workshop on Interfaces and Abstractions for Scientific Data Storage (IASDS10),, Heraklion, Crete, Greece, September 24, 2010*, 2012.

[13] Babak Behzad, Joseph Huchette, Huong Vu Thanh Luu, Ruth Aydt, Surendra Byna, Yushu Yao, Quincey Koziol, et al. A Framework for Auto-Tuning HDF5 Applications. In *Proceedings of the 22nd international symposium on High-performance parallel and distributed computing*, pages 127–128. ACM, 2013.

[14] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[15] Hongzhang Shan, Katie Antypas, and John Shalf. Characterizing and predicting the I/O performance of HPC applications using a parameterized synthetic benchmark. In *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, SC '08, pages 42:1–42:12, Piscataway, NJ, USA, 2008. IEEE Press.

[16] Zachary Chase Lipton. A critical review of recurrent neural networks for sequence learning. *CoRR*, abs/1506.00019, 2015.

[17] J Chen, A Choudhary, S Feldman, B Hendrickson, C Johnson, R Mount, V Sarkar, V White, and D Williams. Synergistic challenges in data-intensive science and exascale computing. *DOE ASCAC Data Subcommittee Report, Department of Energy Office of Science*, 2013.

[18] Jay F Lofstead, Scott Klasky, Karsten Schwan, Norbert Podhorszki, and Chen Jin. Flexible IO and Integration for Scientific Codes Through the Adaptable IO System (ADIOS). In *Proceedings of the 6th international workshop on Challenges of large applications in distributed environments*, pages 15–24. ACM, 2008.

[19] Rajeev Thakur, William Gropp, and Ewing Lusk. Data Sieving and Collective I/O in ROMIO. In *Frontiers of Massively Parallel Computation, 1999. Frontiers' 99. The Seventh Symposium on the*, pages 182–189. IEEE, 1999.

[20] Robert Ricci, Eric Eide, and CloudLab Team. Introducing cloudlab: Scientific infrastructure for advancing cloud architectures and applications. *; login:: the magazine of USENIX & SAGE*, 39(6):36–38, 2014.

[21] Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'95, pages 1137–1143, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[22] Keras: The python deep learning library. https://keras.io/#why-this-name-keras.

[23] Keras: Deep learning library for theano and tensorflow. https://faroit.github.io/keras-docs/0.3.0/.

[24] Sebastian Raschka. About feature scaling and normalization, 2014.

[25] Arnaud de Myttenaere, Boris Golden, Bndicte Le Grand, and Fabrice Rossi. Mean absolute percentage error for regression models. *Neurocomputing*, 192:38 – 48, 2016. Advances in artificial neural networks, machine learning and computational intelligence.

[26] P. Carns, R. Latham, R. Ross, K. Iskra, S. Lang, and K. Riley. 24/7 characterization of petascale I/O workloads. In *2009 IEEE International Conference on Cluster Computing and Workshops*, pages 1–10, Aug 2009.

[27] A. Uselton, M. Howison, N. J. Wright, D. Skinner, N. Keen, J. Shalf, K. L. Karavanic, and L. Oliker. Parallel I/O performance: From events to ensembles. In *2010 IEEE International Symposium on Parallel Distributed Processing (IPDPS)*, pages 1–11, April 2010.

[28] Y. Kim, R. Gunasekaran, G. M. Shipman, D. A. Dillow, Zhe Zhang, and B. W. Settlemyer. Workload characterization of a leadership class storage cluster. In *2010 5th Petascale Data Storage Workshop (PDSW '10)*, pages 1–5, Nov 2010.

[29] Huong Luu, Marianne Winslett, William Gropp, Robert Ross, Philip Carns, Kevin Harms, Mr Prabhat, Suren Byna, and Yushu Yao. A multiplatform study of I/O behavior on petascale supercomputers. In *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '15, pages 33–44, New York, NY, USA, 2015. ACM.

[30] Bing Xie, Jeffrey Chase, David Dillow, Oleg Drokin, Scott Klasky, Sarp Oral, and Norbert Podhorszki. Characterizing output bottlenecks in a supercomputer. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, pages 8:1–8:11, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press.

[31] Babak Behzad, Surendra Byna, Prabhat, and Marc Snir. Pattern-driven parallel I/O tuning. In *Proceedings of the 10th Parallel Data Storage Workshop*, PDSW '15, pages 43–48, New York, NY, USA, 2015. ACM.