ABSTRACT

ON RESOURCE-EFFICIENCY AND PERFORMANCE OPTIMIZATION IN BIG DATA COMPUTING AND NETWORKING USING MACHINE LEARNING

by
Wuji Liu

Due to the rapid transition from traditional experiment-based approaches to large-scale, computational intensive simulations, next-generation scientific applications typically involve complex numerical modeling and extreme-scale simulations. Such model-based simulations oftentimes generate colossal amounts of data, which must be transferred over high-performance network (HPN) infrastructures to remote sites and analyzed against experimental or observation data on high-performance computing (HPC) facility. Optimizing the performance of both data transfer in HPN and simulation-based model development on HPC is critical to enabling and accelerating knowledge discovery and scientific innovation. However, such processes generally involve an enormous set of attributes including domain-specific model parameters, network transport properties, and computing system configurations. The vast space of model parameters, the sheer volume of generated data, the limited amount of allocatable bandwidths, and the complex settings of computing systems make it practically infeasible for domain experts to manually deploy and optimize big data transfer and computing solutions in next-generation scientific applications.

The research in this dissertation identifies such attributes in networks, systems, and models, conducts in-depth exploratory analysis of their impacts on data transfer throughput, computing efficiency, and modeling accuracy, and designs and customizes various machine learning techniques to optimize the performance of big data transfer in HPN, big data computing on HPC, and model development through large-scale simulations. Particularly, unobservable latent factors such as competing loads on end hosts are investigated and an algorithm named Density-Based Spatial Clustering of

Applications with Noise (DBSCAN) is employed to eliminate their negative impacts on performance prediction using machine learning models such as Support Vector Regression (SVR). Based on such analysis results, a customized, domain-specific loss function is employed within machine learning models such as Stochastic Gradient Descent Regression for throughput prediction to advise bandwidth allocation in HPN. A Bayesian Optimization (BO)-based online computational steering framework is also designed to facilitate the process of scientific simulations and improve the accuracy of model development. The solution proposed in this dissertation provides an additional layer of intelligence in big data transfer and computing, and the resulted machine learning techniques help guide strategic provisioning of high-performance networking and computing resources to maximize the performance of next-generation scientific applications.

ON RESOURCE-EFFICIENCY AND PERFORMANCE OPTIMIZATION IN BIG DATA COMPUTING AND NETWORKING USING MACHINE LEARNING

by
Wuji Liu

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy in Computer Science

Department of Computer Science

December 2021

APPROVAL PAGE

ON RESOURCE-EFFICIENCY AND PERFORMANCE OPTIMIZATION IN BIG DATA COMPUTING AND NETWORKING USING MACHINE LEARNING

Wuji Liu

Prof. Chase. Q. Wu, Dissertation Advisor — Date
Professor of Computer Science, NJIT

Prof. Jason. T. L. Wang, Committee Member — Date
Professor of Computer Science, NJIT

Prof. Senjuti Basu Roy, Committee Member — Date
Associate Professor of Computer Science, NJIT

Prof. Qing. G. Liu, Committee Member — Date
Assistant Professor of Electrical and Computer Engineering, NJIT

Prof. Hui Zhao, Committee Member — Date
Assistant Professor of Computer Science and Engineering, University of North Texas, Denton, Texas

BIOGRAPHICAL SKETCH

Author: Wuji Liu

Degree: Doctor of Philosophy

Date: December 2021

Undergraduate and Graduate Education:

- Doctor of Philosophy in Computer Science,
  New Jersey Institute of Technology, Newark, NJ, 2021
- Master of Science in Electrical Engineering,
  New Jersey Institute of Technology, Newark, NJ, 2014
- Bachelor of Electronic Information Science and Technology,
  Hunan Normal University, Changsha Hunan, 2011

Major: Computer Science

Presentations and Publications:

W. Liu, Q. Ye, C. Wu, Y. Liu, X. Zhou and Y. Shan, "Machine Learning-assisted Computational Steering of Large-scale Scientific Simulations," *IEEE Int'l Conf. on Parallel and Distributed Processing with Applications*, 2021.

D. Yun, W. Liu, C. Q. Wu, N. S. V. Rao and R. Kettimuthu, "Exploratory analysis and performance prediction of big data transfer in High-performance Networks," *Engineering Applications of Artificial Intelligence*, vol. 102, pp. 104-285, Jun. 2021.

Q. Ye, W. Liu and C. Q. Wu, "NoStop: A Novel Configuration Optimization Scheme for Spark Streaming," *50-th Int'l Conf.on Parallel Processing, pp. 1-10*, 2021.

G. Ye, W. Liu, C. Q. Wu, W. Shen and X. Lyu, "On Machine Learning-based Stage-aware Performance Prediction of Spark Applications," *IEEE 39-th Int'l Performance Computing and Communications Conference*, pp. 1-8, 2020.

Q. Ye, C. Q. Wu, W. Liu, A. Hou and W. Shen, "Profiling-Based Big Data Workflow Optimization in a Cross-layer Coupled Design Framework," *Int'l Conf. on Algorithms and Architectures for Parallel Processing*, pp. 197-217, Sep. 2020.

W. Liu, C. Q. Wu, Q. Ye, A. Hou and W. Shen, "Performance Modeling and Prediction of Big Data Workflows: An Exploratory Analysis," *IEEE Int'l Conf. on Computer Communications and Network*, pp. 1-10, Aug. 2020.

D. Yun, W. Liu, C. Q. Wu, N. S. Rao and R. Kettimuthu, "Performance Prediction of Big Data Transfer Through Experimental Analysis and Machine Learning," *IFIP Networking Conference (Networking)*, pp. 181-189, 2020.

W. Liu, D. Yun, C. Q. Wu, and N. S. V. Rao, "On Performance Prediction of Big Data Transfer in High-performance Networks," *IEEE Int'l Conf. on Communications (ICC)*, pp. 1-6, Jun. 2020.

C. Q. Wu, W. Liu, S. Sen, N. S. V. Rao, R. R. Brooks and G. Cordone, "Two-level Clustering-based Target Detection Through Sensor Deployment and Data Fusion," *Int'l Conf.on Information Fusion,* pp 2376-2383, 2018.

To my beloved family

ACKNOWLEDGMENT

I would like to deliver my sincere thanks to my research advisor, Dr. Chase. Wu, who provided me invaluable technical guidance and encouraged me to pursue remarkable achievements during my Ph.D career. Special thanks to Dr. Daqing Yun for collaborating on part of this dissertation.

I would also want to express sincere thanks to my committee members including Dr. Jason Wang, Dr. Senjuti Basu Roy, Dr. Qing Liu, and Dr. Hui Zhao, who provide professional advices and devote large efforts to my dissertation to help me stay on the right research track.

I wish to show my gratitude to the Department of Computer Science for the support. I thank the U.S National Science Foundation (CNS-1828123) for funding.

I extend my thanks to Huiyan Cao, Songlin He, Dong Wei and Qianwen Ye for the days we worked together. I wish to thank Weiqiang Dong, Xiang Sun and Jie Zhang for encouraging me to pursue a PhD.

I would also like to thank my family (Shiwei Liu, Lijuan Sun, Ying Chen and Daniel Liu) for their support and love.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

Chapter **Page**

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

LIST OF FIGURES
(Continued)

Figure **Page**

CHAPTER 1

INTRODUCTION

1.1 Motivation

The advance in supercomputing technology is expediting the transition in various basic and applied sciences from traditional laboratory-controlled experimental methodologies to modern computational paradigms involving complex numerical modeling and extreme-scale simulations of physical phenomena, chemical reactions, climatic changes, and biological processes.

One typical scientific application of such types is the research on the Earth's weather and climate system, which involves multiple physical processes acting over a wide range of scales spanning from microphysics at the level of individual cloud droplets to cloud systems at regional and global scales as shown in Figure. 1.1 [51]. Limited by computational resources and incomplete physical understanding, most of these models contain approximate representations of processes that occur at the spatiotemporal scales smaller than model grid spacing. Such subgrid parameterizations often contain empirical parameters that need to be validated or tuned against measurements. Depending on the subgrid processes in question, the number of tunable parameters can range from several up to hundreds, and the specific values of these parameters likely vary with weather and cloud regimes. Thus, the process of "objective tuning" poses a great challenge to the computational communities as well as the Earth Science community including forecasting of climate, weather, and renewable energies such as wind and solar.

To facilitate the parameter tuning process, the simulation data generated by such models needs to be transferred over high-performance network (HPN) infrastructures to remote sites and analyzed against experimental or observation data

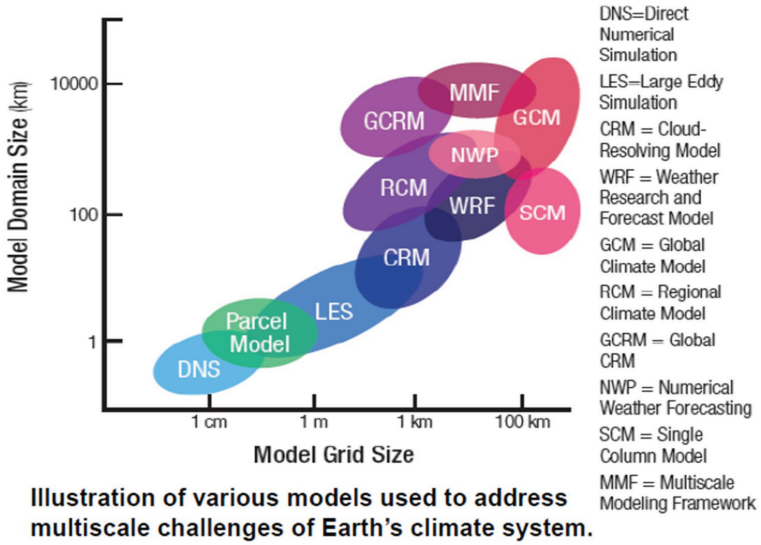


Figure 1.1 Widely-used multi-scale, multi-physics models for the Earth's weather and climate system.

on high-performance computing (HPC) facility. In recent years, high-performance networks (HPNs) featuring high-speed dedicated connections and advance bandwidth reservation have been developed and deployed in a rapidly expanding scope to provide such networking services. For example, OSCARS [2] provides advance reservation of secure virtual circuit with guaranteed bandwidth within ESnet [1], and AL2S enables similar services within Internet2 [4] and across other networks. Google's B4 [46] is a private software-defined application-friendly wide-area network (WAN) platform that could be leveraged for big data sciences and industrial applications at the planet scale. Computer systems such as Data Transfer Nodes (DTNs) in Science DMZ [5] have also been deployed to support geographically distributed science due to the great benefit brought by dedicated connections of HPNs.

However, data transfer is a complex process whose throughput performance is affected by various factors, including not only the hardware specifications of both network segments and end hosts, but also software configurations of operating systems

and data transfer applications. Although the exclusive use of HPN connections minimizes the impact of complex dynamics caused by some factors such as cross traffic, many other elements involved in a typical big data transfer process still affect the performance to a great extent, including i) configurations of end host systems, ii) properties of network connections, and iii) control parameters of data transfer methods and their underlying transport protocols. It is generally very difficult to apply an analytical approach to big data transfer performance prediction, due to i) the lack of accurate throughput performance models for high-performance transport protocols such as UDT [24], ii) the complex composition of end-to-end HPN connections, iii) the complexities of end host configurations; iv) the time-varying workloads in end host systems; and v) other latent variables that may not even be accessible or measurable. Consequently, HPN technologies and services have not been fully utilized for big data transfer regardless of the continuous bandwidth upgrades in backbones.

Moreover, the processing and analysis of such large-scale simulation, observation, or experimental datasets, are typically structured and orchestrated as computing workflows. Such big data workflows generally require massive computing resources for execution on high-performance clusters in cloud environments. Many research efforts have been made to achieve both computation and energy efficiency in workflow execution and most of them adopt a top-down design methodology that takes into consideration both program codes and hardware systems for workflow performance optimization [27, 17, 26, 44]. The technology stack of such computing platforms designed for big data workflows involves a large number of configurable parameters and end users need to request computing resources as needed in advance through parameter setting.

However, finding a satisfactory configuration for workflow execution in such complex systems is challenging to end users, who are primarily domain experts.

Most existing big data systems provide default values for parameter setting, which, unfortunately, do not always yield the best performance. Moreover, the complexity in a workflow execution process makes it very difficult to choose and configure the right set of parameters from different layers in the technology stack as they oftentimes exhibit complex interactive effects within and across layers. Most of the existing work for workflow parameter setting is carried out in the context of computational steering, which enables end users to interact with the computing workflow and system during execution [28, 45]. Although having achieved remarkable success in their intended environments, these methods often place an undue burden on end users to spend a significant amount of time in sifting through the large parameter space based on a try-and-error process. Therefore, it is still an important yet largely unsolved problem to decide the best parameter setting for optimal workflow performance in big data systems, even with the aid of certain domain knowledge in systems and workflows.

1.2 An Integrated Solution

In view of the aforementioned challenges and limitations, we propose an integrated solution to big data movement in HPNs and big data computing in HPC, which is comprised of three major components: i) performance prediction of big data transfer to support bandwidth reservation, ii) performance modeling and prediction of big data workflows, and iii) collaborative computational steering as a service, in a unified framework. The overarching goal of our research is to develop a class of machine learning-assisted big data transfer and big data computing solutions for collaborative computational steering in support of large-scale, simulation-based computational sciences. The proposed solutions are expected to streamline, automate, and optimize the lifecycle of big data workflows and revolutionize the traditional offline and manual steering approaches.

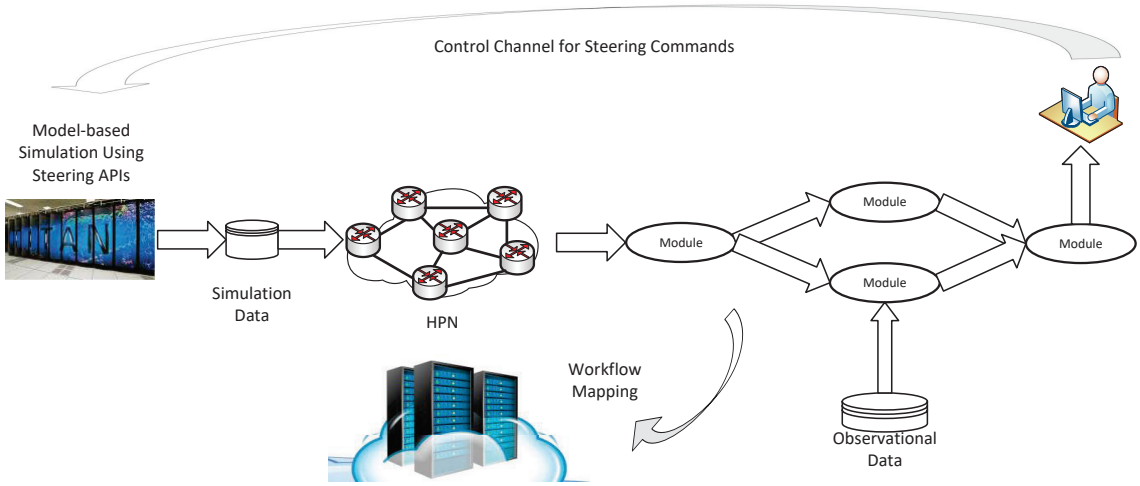


Figure 1.2 Framework of the proposed integrated solution.

The overall design of the proposed framework is illustrated in Figure. 1.2. As shown on the left side of Figure. 1.2, a typical model-based research process in eScience involves model construction, numerical simulation, and parameter tuning, which generates large simulation datasets. These datasets are then transferred to remote facilities using machine learning-assisted transport methods in HPNs. They are further processed by various computing programs against real-life experimental datasets or measurements for model validation or calibration, which are mapped as big data workflow modules and executed in HPC or clusters deployed in clouds. The interactions between the steering engine and the scientific workflow are carried out over a unified communication channel that enables the delivery of various steering commands from multiple users to different simulation processes being executed concurrently.

1.3 Contributions

This dissertation tackles the problem of optimizing the performance of large-scale scientific simulations through strategic recommendations of hyper parameter settings

using machine learning. We summarize the main contributions of this dissertation as follows:

- We conduct an exploratory analysis on the effects of latent factors based on extensive performance measurements collected in the past several years from data transfer tests using different protocols and applications between various end sites in several real-life HPN testbeds. We also propose a clustering-based method to eliminate the negative impact of latent factors on performance prediction. We further develop a robust performance predictor by incorporating the proposed elimination method into data preprocessing and customizing domain-specific loss functions.

- We conduct an in-depth qualitative and comparative exploratory analysis to investigate the impact of hyper parameters on workflow performance. With the findings from the exploratory analysis and domain knowledge, we construct dependent features by mapping subsets of parameters with a number of candidate functions to model the corresponding workflow performance. We further propose a feature selection method based on information theory [16] to identify the most influential parameters.

- We design a framework of Co^2Steer for steering as a service with generic models of simulations and extensible Application Programming Interfaces (APIs) to interact with and guide model-based simulations towards user-preferred directions. This computational steering framework can be applied to a wide spectrum of large-scale scientific applications with similar computation-computing workflows.

CHAPTER 2

PERFORMANCE PREDICTION OF BIG DATA TRANSFER

2.1 Related Work

The significance of high-speed dedicated connections provisioned by HPNs has been widely recognized in both research and industrial communities due to the rapidly growing big data transfer needs of data- and network-intensive applications. In the past decade, a great deal of research efforts have been made to predict data transfer performance using different methods.

2.1.1 Profiling-Based Performance Prediction

Transport performance profiling employs an empirical approach to study the behaviors of different data transfer applications and their underlying transport protocols. A profile of transport performance in response to control parameters of transport methods and network environments is obtained by running data transfer tests with a sweep of the parameter space and collecting corresponding performance measurements. Such profiles can help us understand the network behaviors, facilitate the design of an effective performance predictor, and also be used as benchmarks.

Rao *et al* . in [39] provided large-scale TCP measurements over a set of 10 Gbps dedicated connections with emulated delays ranging from 0 ms to 366 ms, and further in [40] showed that TCP throughput is very sensitive to the connection delay and behaves in a combination of concave and convex functions. Performance profiling of UDT [24], another widely-used data transfer protocol in HPN community [7], is conducted in [18], where UDT behaviors with respect to various application settings and protocol socket options are measured and analyzed. These measurements and analyses show that control parameter settings also significantly affect throughput performance of big data transfer in HPNs. Unfortunately, such effects are not

taken into consideration in conventional performance prediction methods. Liu *et al.* conducted similar research on performance profiling of data transfer methods in [42]. While profiling-based approaches offer better interpretability and explainability as they provide a deeper insight into the behaviors of data transfer methods under various circumstances, they typically incur high overhead, sometimes making them practically infeasible. For example, to obtain a fully-covered transport profile of a given protocol over a given connection, an exhaustive sweeping of the entire parameter space may take hours or even days to complete.

2.1.2 Learning-Based Performance Prediction

Along with the emergence of HPN technologies and the accumulation of performance measurements of big data transfer, machine learning has been increasingly used to investigate and reveal the behavioral patterns of data transfer protocols and the underlying host and network infrastructures.

Mirza *et al* . in [35] considered a set of properties of historical data transfer over network paths as features to train machine learning models, and then used various combinations of subsets of these features for evaluation. Although this work was focused on predicting TCP performance in shared networks, some important features in such environments such as cross traffic were not directly considered when building the model. Liu *et al* . employed regression models to explain the observed performance patterns extracted from the log files of disk-to-disk wide-area file transfer powered by GridFTP [52]. They further in [53] expanded the feature set and developed a model selection strategy for performance prediction of file transfer in wide-area networks. Based on a retraining process, their approach showed promising prediction accuracy, which is verified by a comparative evaluation using Globus logs [3].

2.2 Problem Statement

The throughput performance y of a data transfer over a dedicated connection is considered as a function f of a vector of feature variables $\mathbf{x}$ involving different segments including end hosts, network connections, and applications, i.e., $y = f(\mathbf{x})$. The analytical form of f is typically unknown, and thus we propose to employ machine learning to build a model to approximate f based on historical performance measurements of big data transfer.

More formally, we collect a set of measurements used as the training dataset $\mathcal{T} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$, where $\mathbf{x}_i$ $(i = 1, 2, \ldots, n)$ is a specific set of values of the feature vector $\mathbf{x}$ that collectively determine the corresponding throughput y_i. We aim to estimate f based on $\mathcal{T}$, i.e., $\hat{f}(\mathbf{x}_i) \approx f(\mathbf{x}_i)$ such that $\hat{f}(\mathbf{x}_i)$ is close enough to the "true" value y_i for all training examples in $\mathcal{T}$ and can be used to predict y_i given a future arbitrary $\mathbf{x}_i$ with high accuracy.

The feature vector $\mathbf{x}$ in this context is in the form of a list of observable variables in the three segments of an end-to-end data transfer path: i) end host configurations such as CPU speed, RAM size, etc.; ii) connection properties such as round-trip time (RTT), connection bandwidth, etc.; and iii) control parameters of data transfer applications such as socket buffer size, number of data streams, etc. However, there exist certain unforeseeable and unobservable latent factors including competing loads (since the end hosts are usually shared by multiple users), system dynamics on end hosts, and instabilities along the network connection, all of which may also significantly affect the end-to-end data transfer performance. This is mainly because when the network speed reaches a certain high rate, as in the case of HPNs, the speed of (mainly incoming) traffic may keep the end hosts (mainly the receiver) constantly busy and any perturbation under such conditions caused by any latent factor may overwhelm the end host, leading to unpredictable performance.

From the perspective of performance prediction, the above "abnormal" behaviors may result in large noise in the training dataset $\mathcal{T}$. Considering the number of observable factors in $\mathbf{x}$ and the complexity and randomness caused by the latent factors and other unknowns, it is extremely difficult to build a robust performance predictor for HPN resource management, which is critical to satisfying bandwidth requirements of user requests and minimizing resource waste.

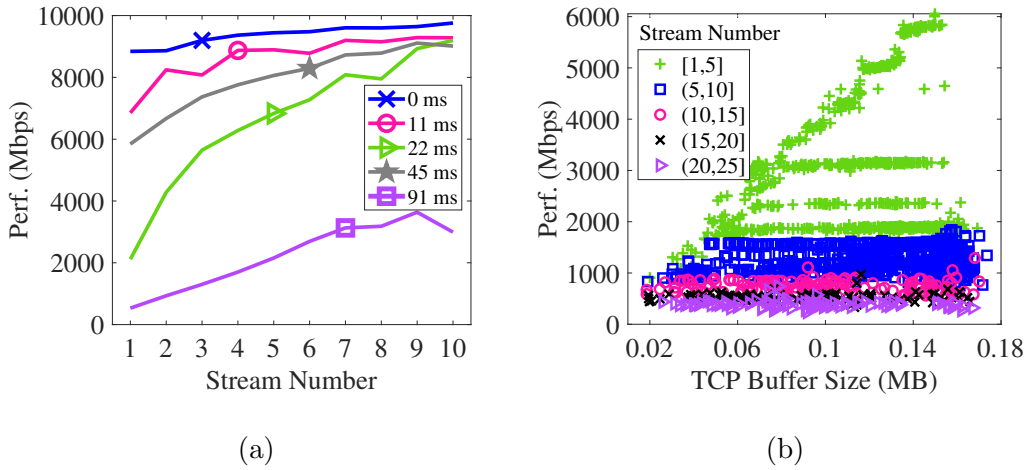
Hence, in addition to normal (well-behaving) performance measurements y, the training dataset $\mathcal{T}$ typically also contains some "corrupted" performance measurements y' under the effects of both feature vector $\mathbf{x}$ and unobservable factors α, i.e., $y' = f(\mathbf{x}) + f'(\alpha)$, where $f'(\alpha)$ represents the collective (negative) effects imposed by the latent variables and other unknowns. In other words, the performance measurements in $\mathcal{T}$ are sampled from a combined set of $\{y\}$ and $\{y'\}$.

Our work has two technical components: i) use a generic clustering-based method in data preprocessing to eliminate the "latent-variable-corrupted" data points from the training set; and ii) employ machine learning methods to build an accurate performance predictor based on the cleaned training set.

2.3 Analysis of Feature Variables and Latent Factors

Effects of Application-Accessible Parameters Here, we focus on three representative control parameters, i.e., buffer size, stream count, and round trip time (RTT), and show their impact on throughput performance. More comprehensive profiling results are provided in [42, 39] for both TCP and UDT [24].

The throughput performance measurements with respect to number of streams and RTT are plotted in Figure. 2.1(a), which shows that, over a 10 Gbps dedicated connection, using multiple streams help achieve better transport performance, especially for a long connection delay. This observation actually has motivated the design of many data transfer toolkits and services such as Globus GridFTP [3] that are



(a) (b)

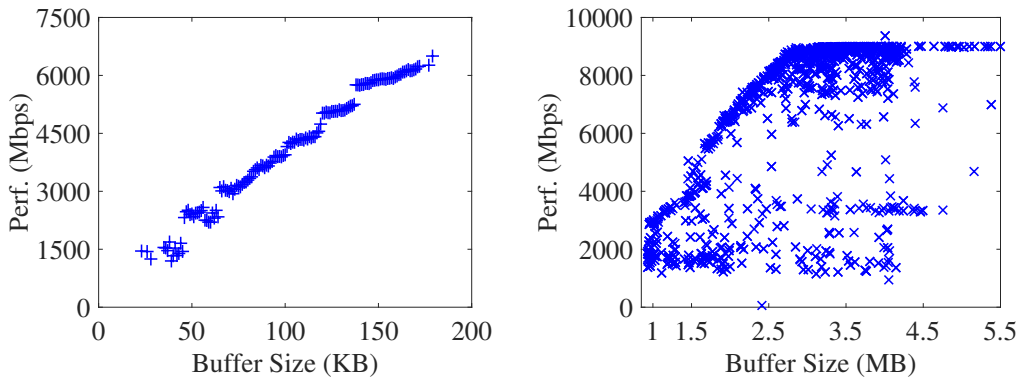
Figure 2.1 Illustration of the impact of application-accessible parameters on TCP performance: (a) performance vs. stream count and RTT; and (b) performance vs. stream count and buffer size.

being widely used for big data transfer. The behavior under different RTTs indicates that achieving satisfactory performance over a long-haul connection is difficult even for a dedicated channel with sufficient bandwidth.

The performance measurements in response to buffer size and number of streams are plotted in Figure. 2.1(b), which shows that, over short connections, using a large number of parallel streams only brings a limited performance gain in comparison with an appropriately set buffer size. In such cases, they jointly dominate the throughput performance of TCP: the performance generally increases as the buffer size increases; however, as the number of streams increases, the performance gains from increasing the buffer size are diminishing, which is probably due to the resource demand of a high stream count overwhelming the end host system.

2.3.1 Effects of Unknowns

The results presented in Subsection. 2.3 suggest the use of machine learning methods for performance predication of big data transfer in HPNs. This is because: i) the performance patterns are qualitatively consistent and stable across different



(a) Without latent effects

(b) With latent effects

Figure 2.2 TCP performance vs. buffer size without and with latent effects.

connections, e.g., the throughput increases as the buffer size and number of streams increase and the achievable throughput decreases as the connection delay increases; and ii) such patterns cannot be modeled analytically, e.g., the slope of performance increase with respect to buffer size increase may vary across different connections, and the optimal number of data streams may depend on not only the network environments but also the end host system configurations. Such multi-dimensional accessible control parameters or features make it difficult, if not at all possible, to derive an analytical form to describe their relationship with the throughput performance.

However, during our extensive experimentation, we found that there may exist certain latent variables that also significantly affect the throughput performance. These latent variables are not easily observable while the data transfer is being performed due to the data transfer application's limited access to the end host system and other unpredictable factors such as competing loads and system dynamics. Such latent effects, if not excluded, could make machine learning-based prediction biased or overfitted.

To show such latent effects, we compare the performance measurements of the same set of data transfer tests conducted on two different testbeds: i) a production HPN testbed where the end hosts of data transfer are computing servers shared by many users (thus competing loads are significant), and ii) our local testbed where the experimental conditions are strictly controlled and competing loads from other users are forbidden.

Figure. 2.2 plots the difference in TCP performance with respect to buffer size with and without the effects of latent variables. As shown in Figure. 2.2(a), TCP throughput almost linearly increases with the increase of buffer size before reaching the peak. In Figure. 2.2(b), the maximal achievable TCP performance follows a similar pattern, i.e., increases linearly as the buffer size increases till reaching the peak. In addition, there also appear a non-negligible number of performance measurements below the maximal ones, which may cause inaccuracy in performance prediction for bandwidth scheduling in HPNs.

Figures. 2.2(a) and 2.2(b) show that in addition to the impact of data transfer applications and network environments, there are a certain number of unobservable factors such as system dynamics and competing loads from "hidden" users that could undermine the performance. Such latent factors originate from unknown features and are very difficult to estimate due to their unpredictable nature and randomness.

In this work, we propose to use machine learning methods to eliminate the (negative) effects of such latent variables during data preprocessing and further build a robust machine learning model for big data transfer performance prediction in HPNs.

2.3.2 Elimination of Latent Effects Using Clustering

We first describe our approach to eliminate latent effects using clustering-based methods, and then compare different clustering algorithms.

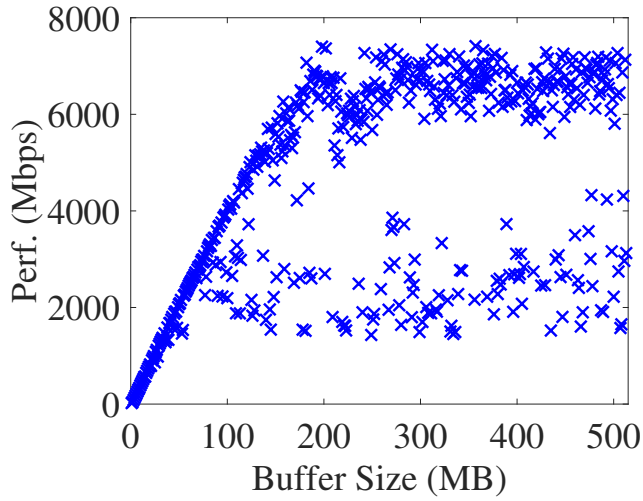
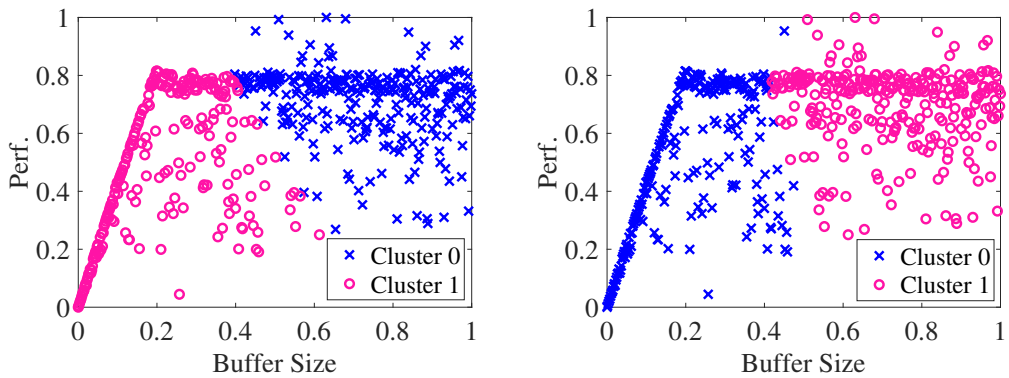


Figure 2.3 The case of UDT performance corresponding to buffer size diverges due to latent effects.

These effects are also illustrated by UDT performance in Figure. 2.3, where we fix all other parameters and only vary the buffer size. It shows that the UDT throughput with respect to buffer size diverges to two different patterns with and without latent effects. These latent factors would seriously impair the quality of a prediction model. This phenomenon motivates us to use a clustering-based method to eliminate the measurements that are observed under the conditions with significant latent effects. Other research (e.g., [53]) also pointed out the negative effects of such latent factors and a threshold-based method is adopted in [53] to eliminate the effects, which, although simple, may introduce an unexpected bias into the performance prediction model.

In addition, due to the nature of the problem, $f(\mathbf{x})$ is considered to be smooth. In other words, with a slight change to any parameter, e.g., buffer size, the change in the throughput performance should be bounded. If we have a sufficient number of data points for different values of control parameters, we are able to see a smooth pattern of throughput performance, as shown in Figure. 2.3.

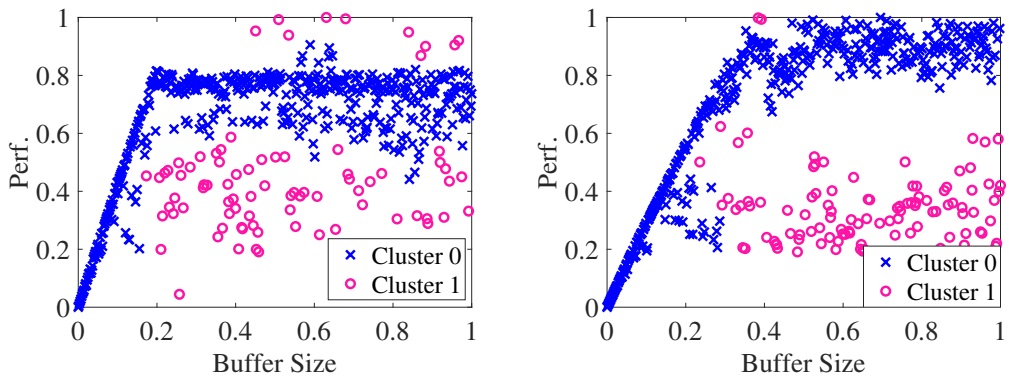


(a) K-means (b) GMM

Figure 2.4 Comparison of different clustering algorithms (values are normalized).

As stated previously, our dataset is a combination of performance observations including both y and y', which are subject to different mapping functions. Our goal is to differentiate the divergence of different performance patterns and rule out the one that is manifested by the "abnormal" data points and is thus less frequently observed, as the regular pattern (exhibited by the normal data points) would appear more frequently in real-life bandwidth scheduling. This can be achieved by using clustering-based methods to categorize y and y' into different clusters with a certain distance measure such that the data points within the same cluster are closer to each other and thus are more likely to be measured under similar conditions with similar latent effects.

Comparison of Different Clustering Algorithms To choose an appropriate clustering algorithm to separate "abnormal" data points from normal ones, we compare several well-known and commonly-used clustering algorithms, as shown in Figure. 2.4. Conventional clustering methods based on Expectation Maximization (EM) such as K-means and Gaussian Mixture Model (GMM) aim to maximize the log-likelihood derived from previous estimates. As shown in Figure. 2.4, K-means



(a) Block size: 8955 bytes

(b) Block size: 26867 bytes

Figure 2.5 Clustering results of DBSCAN (values are normalized).

and GMM perform poorly in differentiating the data points under different levels of latent effects, since they simply divide the data points into two groups with a roughly equal radius. Therefore, we propose to use the Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm [34] to eliminate the data points with latent effects and hence facilitate accurate performance prediction. DBSCAN categorizes the data points into different clusters based on their densities, where tightly-packed points are grouped together and those in low-density regions are classified as outliers. The clustering results of DBSCAN on the same datasets as used in Figure. 2.3 and Figure. 2.4 are presented in Figure. 2.5(a) and Figure. 2.5(b), respectively, which show the effectiveness of DBSCAN in differentiating data points with latent effects.

2.4 Prediction of Big Data Transfer Performance

In this section, we first describe a customized loss function used for building a performance predictor and then present prediction results using various machine learning models. The performance predictors are all implemented in Python based on the `scikit-learn` library [22].

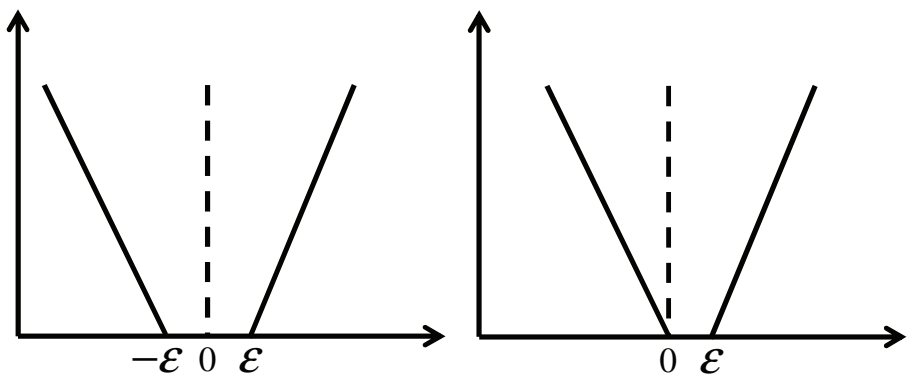
Customized Loss Function Different from traditional supervised learning methods that seek an optimal label for a given feature vector [35, 53], for bandwidth scheduling, we aim to build a model that provides a loosened prediction with a reasonable range.

Together with DBSCAN-based data preprocessing, which eliminates negative latent effects, we build our performance predictor based on a customized loss function, as motivated by the domain knowledge of HPN management that requires the reserved bandwidth over a dedicated connection to match the bandwidth requirement of a data transfer request with minimal waste. Therefore, the optimal predicted transfer performance $\hat{y} = \hat{f}(\mathbf{x}_i)$ should lie within the range $[y_i, \epsilon y_i]$, where $\epsilon \geq 1$ is a small tunable positive number and y_i is the ground truth of the achievable performance with feature vector $\mathbf{x}_i$. The predicted value should be slightly higher than what a data transfer can utilize to satisfy the request and also minimize the waste. Inspired by the ϵ-insensitive loss used by Support Vector Regression (SVR) and other work [41], we customize the ϵ-insensitive loss function in Figure. 2.6(a) by restricting the tolerable errors to be positive only. As shown in Figure. 2.6(b), the optimal value is parameterized by an error tolerance ϵ and our objective is to minimize the following loss

$$\mathcal{L}(\theta, \epsilon) = \sum_{i=1}^{n} \Big\{ \max(y_i - \hat{f}_\theta(\mathbf{x}_i), 0) + \max(\hat{f}_\theta(\mathbf{x}_i) - \epsilon \cdot y_i, 0) \Big\}, \tag{2.1}$$

where the loss $\mathcal{L}(\theta, \epsilon)$ is 0 if the prediction $\hat{f}_\theta(\mathbf{x}_i)$ is larger than the observed value y_i but is within the tolerable range bounded by ϵ; otherwise, $\mathcal{L}(\theta, \epsilon)$ is the absolute error.

Evaluation Metric We define a domain-oriented performance evaluation metric, denoted by γ, similar to the Mean Absolute Percentage Error (MAPE), which is a commonly-used accuracy metric in statistics. Unlike MAPE that counts the absolute error, γ counts only the positive errors that fall out of the range governed by ϵ as



(a) ϵ-intensive loss [41] (b) The customized loss

Figure 2.6 Loss functions.

in Equation. 2.1. This Customized Mean Absolute Percentage Error (CMAPE) is defined as $\gamma = \frac{1}{n}\mathcal{L}(\theta, \epsilon) = \frac{1}{n}\sum_{i=1}^{n}\left\{\max(y_i - \hat{f}_\theta(\mathbf{x}_i), 0) + \max(\hat{f}_\theta(\mathbf{x}_i) - \epsilon \cdot y_i, 0)\right\}, \epsilon \geq 1$, where $\hat{f}_\theta(\mathbf{x}_i)$ is the predicted value given feature $\mathbf{x}_i$, y_i is the corresponding ground truth, n is the total number of test cases. A proper bandwidth allocation should satisfy the user requirement with only an inevitable (as governed by ϵ) amount of waste. In addition to γ, we also count the Effective Prediction Percentage (EPP, denoted by β) among all test cases, i.e., $\beta = \frac{1}{n}\sum_{i=1}^{n}\mathcal{I}\{y_i \leq \hat{f}_\theta(\mathbf{x}_i) \leq \epsilon \cdot y_i\}$, where $\mathcal{I}(\psi)$ is an indicator function that is equal to 1 if ψ is true, and 0, otherwise.

2.4.1 Models in Comparison

We compare four models [37] with the customized loss function defined in Equation. 2.1: i) linear models as represented by Ridge Regression (RR); ii) non-linear models as represented by Support Vector Regression (SVR); iii) Neural Networks (NN), where we use a standard three-layer neural network with ReLU as the activation function; and iv) ensemble models as represented by Random Forest Regression (RFR).

2.4.2 Dataset

Our dataset contains about 100,000 records of throughput performance measurements that are collected from big data transfer tests conducted over local back-to-back connections and in several other HPNs managed by different institutions.

2.4.3 Results

Our performance evaluation includes two parts: i) compare the prediction results of SVR on the original dataset with and without the DBSCAN-based preprocessing as introduced in Subsection. 2.3.2; and ii) compare the performance of the four models in Subsection. 2.4.1 and select the best one with data preprocessing.

SVR Prediction Accuracy With and Without Preprocessing We first run the DBSCAN-based preprocessing to filter out data points that are heavily affected by latent factors and then perform prediction using SVR. This process is repeated without such preprocessing for comparison. As shown in Figure. 2.7, the accuracy of SVR-based performance prediction based on the cleaned dataset consistently outperforms the prediction accuracy based on the original dataset. Note that the dataset used in these tests is collected from a production HPN, where high-end servers used as the sender/receiver of data transfer tests are concurrently used by many other scientists to run their scientific computing jobs.

Prediction Accuracy of Various Models We use the filtered dataset to compare the prediction accuracy of various models as mentioned in Subsection. 2.4.1 in terms of different performance criteria including Root Mean Square Error (RMSE), Mean Absolute Error (MAE), EPP (i.e., β), and CMAPE (i.e., γ). As shown in Figure. 2.8, the linear RR model performs poorly for all four criteria due to the limited richness of its hypothesis function set. The NN model performs even worse than RR for RMSE and MAE, which indicates extensive tuning or more network layers (thus

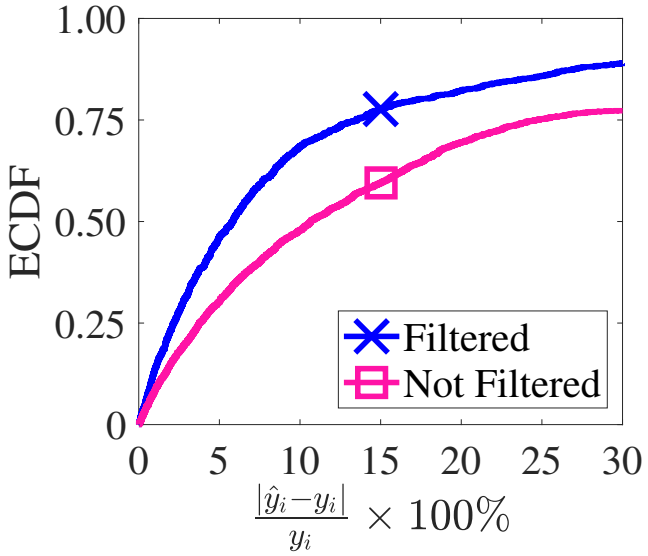
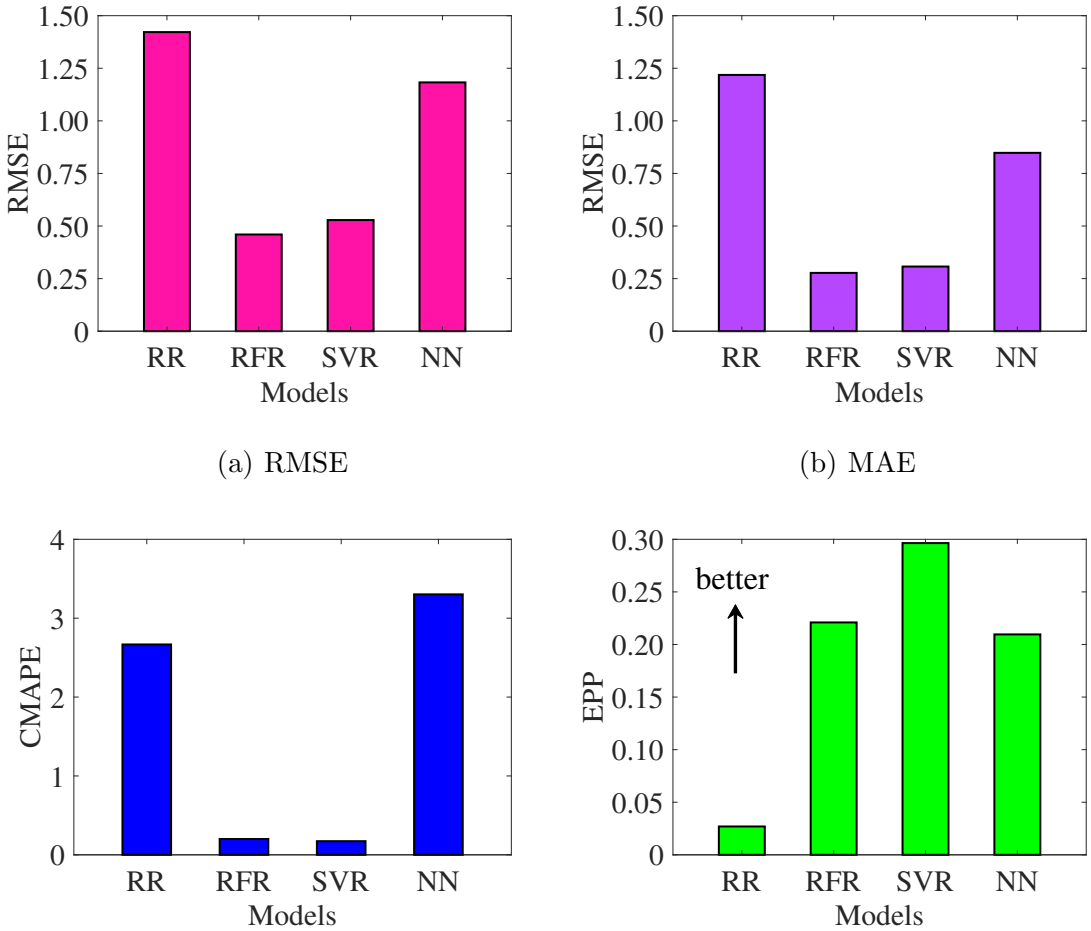


Figure 2.7 Comparison of SVR prediction accuracy with and without DBSCAN-based preprocessing.

higher overhead) are needed. RFR and SVR perform almost equally well for RMSE and MAE. SVR has the best overall performance since it outperforms all other models for both of the metrics defined for bandwidth scheduling, i.e., CMAPE and EPP. Therefore, we choose SVR with the customized loss function for performance prediction in HPNs.



(a) RMSE

(b) MAE

(c) CMAPE (γ)

(d) EPP (β)

Figure 2.8 Performance comparison of various models in different metrics.

CHAPTER 3

PERFORMANCE PREDICTION OF BIG DATA WORKFLOWS

3.1 Related Work

The importance of user interaction with model-based simulations or computing workflows has been well recognized in the broad science community. In the past decade, a large number of research efforts have been made to help end users identify appropriate parameter settings for computational steering or performance modeling. We conduct a brief survey of such efforts in this section.

3.1.1 Computational Steering

The main goal of computational steering [36, 14] for scientific workflows is to identify and recommend the best parameter setting to end users for the simulation or computing procedures. To facilitate real-time steering, some workflow management systems (WMS) adopt bottom-up redesign to provide the capability and flexibility of customization. Pegasus [19], a widely used WMS in the high-performance computing (HPC) community, allows users to customize framework configuration to meet various computing needs. Fireworks [9], yet another powerful workflow system designed for high-throughput performance, achieves high concurrency and efficiency for workflow execution. Such customizable WMS motivate the exploration of hyper parameter settings to optimize workflow performance. For example, Lee *et al* . [29] proposed an adaptive scheduling method for workflow execution by analyzing historical workflow execution data collected in Pegasus. Their analysis shows that the hyper parameter setting of WMS significantly affects the performance of workflow execution with different computing requirements. Unfortunately, such analysis often introduces high complexity in interpreting the impact of parameters and hence provides a limited amount of information to assist in the selection of hyper parameters in WMS.

3.1.2 Performance Modeling

With the pervasive use of workflow technology and the rapid accumulation of performance measurements and provenance data, many research efforts have been made to model workflow execution and predict workflow performance in various WMS in support of computational steering [32, 21, 12].

Miu *et al* . in [48] considered a set of properties of historical workflow execution in WMS as input features to train a decision tree-based model, and then used various combinations of subsets of these features for evaluation. Although this work met with some success in predicting workflow performance, some important features such as execution configuration are not explicitly considered for model construction. Also, the learning process for performance prediction is black-boxed and provides limited information for identifying important hyper parameters. Li *et al* . [47] employed Support Vector Regression (SVR) to model the observed performance pattern and achieved efficient scheduling with a candidate assignment strategy based on performance prediction.

Our research differs from the aforementioned work in two main aspects: i) We focus on the performance of computing workflows executed in modern big data systems, as instantiated by Spark-based computing with YARN resource management. ii) We explore the coupling effects of parameters across various layers in the big data technology stack and incorporate machine learning-based feature selection into the construction of a performance-influence model. We would also like to point out that the proposed exploratory analysis and machine learning-based methods for workflow performance modeling and prediction are generalizable to other big data systems with a customizable framework.

3.2 Problem Statement

The performance (mainly, execution time or makespan) y of a computing workflow executed in big data systems is typically modeled as a function f of a vector $\mathbf{x}$ of features x across various layers including workflow input, WMS, and resource management, i.e., $y = f(\mathbf{x})$. Constructing an accurate model function and identifying the most important components in feature vector $\mathbf{x}$ not only help end users understand how these hyper parameters affect workflow performance, but also provide practical guidance for end users to set parameters for optimal performance. However, due to the complexity of the workflow execution process and the large number of involved control parameters, it is very difficult, if not impossible, to find an analytical form of f, which is typically intractable. Some learning-based approaches such as a black-boxed machine learning model may work in some context, but generally lack interpretability. Thus, we aim to identify and construct a subset of interpretable features $\hat{\mathbf{x}}$, which could provide certain guidance to workflow and system configuration (e.g., the ratio of input data size to memory size), such that a performance-influence model built upon such interpretable features can achieve higher accuracy in performance prediction compared to the original feature vector.

More formally, given a training dataset of historical performance measurements

$$\mathcal{D} = \{(\mathbf{x_1}, \mathbf{y_1}), (\mathbf{x_2}, \mathbf{y_2}), \ldots, (\mathbf{x_n}, \mathbf{y_n})\},$$

where $\mathbf{x}_i$ $(i = 1, 2, \ldots, n)$ is a set of specific values for the feature vector $\mathbf{x}$ that yield the corresponding performance y_i, we aim to construct $\hat{\mathbf{x}}$ based on $\mathbf{x}$, i.e., $\hat{f}(\hat{x}_i) \approx y_i = f(\mathbf{x}_i)$ such that $\hat{f}(\hat{x}_i)$ is close enough to the ground truth y_i for all training examples in $\mathcal{D}$ and could be used to predict y_i with high accuracy for future arbitrary $\mathbf{x}_i$.

The feature vector $\mathbf{x}$ in this context is assembled by a set of parameters across different stages during the life circle of a workflow execution process, including i)

workflow submission stage such as input data size, module functionality, etc.; ii) Spark scheduling such as the number of executors, executor cores, executor memory, etc.; and iii) YARN resource management such as maximum allocated VCores, memory, etc. However, without domain knowledge, it is difficult to identify the high-order representation terms $\hat{x}$ within $\hat{f}$ and build an accurate prediction model.

Hence, in this work, we conduct a comprehensive exploratory analysis to construct candidate representation features, and design an information theory-based learning method to select important dependent features and develop an accurate performance predictor based on such features.

3.3 Exploratory Analysis

We first conduct an empirical study of the impact of various parameters on workflow performance in big data systems through repeatedly testing a workflow-based linear regression experiment. This workflow consists of three pipelined computing modules performing i) data preprocessing to split the input data into two files for training and testing, respectively, ii) model training for linear regression with the training data, and iii) model-based prediction with the testing data. The workflow is implemented in Spark and executed on a local PC cluster consisting of three virtual machine (VM) instances (one master node and two slave nodes), each of which is equipped with eight virtual cores and 24GB memory. By default, each slave node provisions one executor with one virtual core and 1GB of virtual memory.

The goal of this empirical study with performance analysis is to understand and explore the individual and coupled effects of various parameters across different layers. Such an exploratory analysis motivates the use of feature selection in performance-influence model development and inspires the design and incorporation of an information theory-based method in the development of a learning model to achieve high prediction accuracy. Particularly, we focus on investigating the impact

of a set S of parameters that are commonly accessible and tunable by end users in the Spark layer, including executor memory size, executor core count, degree of parallelism, and task core count. To better illustrate the individual impact of each parameter on the execution performance of computing workloads, we fix all other parameters with system default values or customized values within a reasonable range while examining one parameter at a time.

3.3.1 Executor Memory Size

In HDFS, a file is partitioned into data blocks of equal size (except for the last block), which are then replicated and distributed across the cluster. In Spark, multiple tasks are launched to process file splits in parallel, each of which corresponds to a data block by default, in executors on different data nodes. The number of tasks is generally determined by the input file size, the split size, and the submitted program [49]. Spark-based WMS performs in-memory processing for compute-intensive workloads. Executors in Spark are memory-demanding Java Virtual Machine (JVM) processes that provide execution environments for executable units and are executed in containers provisioned according to user requests. After receiving a Spark job with a specific parameter setting, Spark further divides the job into multiple sequential execution stages, each of which contains a set of tasks. In general, a large executor memory size is conducive to the successful completion of a task without being halted for more resources or killed by the system.

To understand the impact of executor memory size on workflow performance, we conduct two sets of experiments where the Spark-based linear regression workflow is executed in both of two executors, each with four virtual cores and different sizes of memory. In the first set of experiments, we process an input file whose size is comparable to the smallest executor memory size (i.e., 800 MB), and repeat each experiment five times. The workflow execution time and garbage collection (GC)

time are measured and normalized as plotted in Figure. 3.1(a) and Figure. 3.1(b), respectively. Note that GC is part of the execution process and the GC time is included in the workflow execution time. We observe that with a relatively small executor memory size (e.g., less than 1.5 $\times$ file size), increasing the executor memory size improves the workflow performance. This is because there is a lack of memory for performing parallel Resilient Distributed Dataset (RDD) operations of four concurrent tasks in each executor. However, further increasing memory size beyond what is needed for the input data size does not bring a corresponding performance gain. For the same reason, the GC time exhibits a similar pattern.

In the second set of experiments, we run the same workflow to process smaller files, and measure the corresponding performance in response to various executor memory sizes, as plotted in Figure. 3.1(c) and Figure. 3.1(d). As the input file size increases from 120MB, 240MB, to 479MB, more tasks are created and executed, hence causing an increase in both the workflow execution time and the GC time. These measurements also show that the impact of executor memory size on the workflow execution time is limited, when processing input data that is relatively smaller than the executor memory size. This is because the executor provides an execution environment with sufficient memory to store and process the entire RDD in Spark. Similar to Figure. 3.1(b), we observe that increasing the executor memory size reduces the GC time because the GC process is less frequently triggered in the presence of sufficient memory, as shown in Figure. 3.1(d).

3.3.2 Executor Core Count

In general, the number of cores determines the computing power of an executor as more cores would be able to run more tasks in parallel and hence achieve faster execution of heavy iterative workloads. While the specific execution dynamics and time for processing different file sizes may be different in scale, the performance

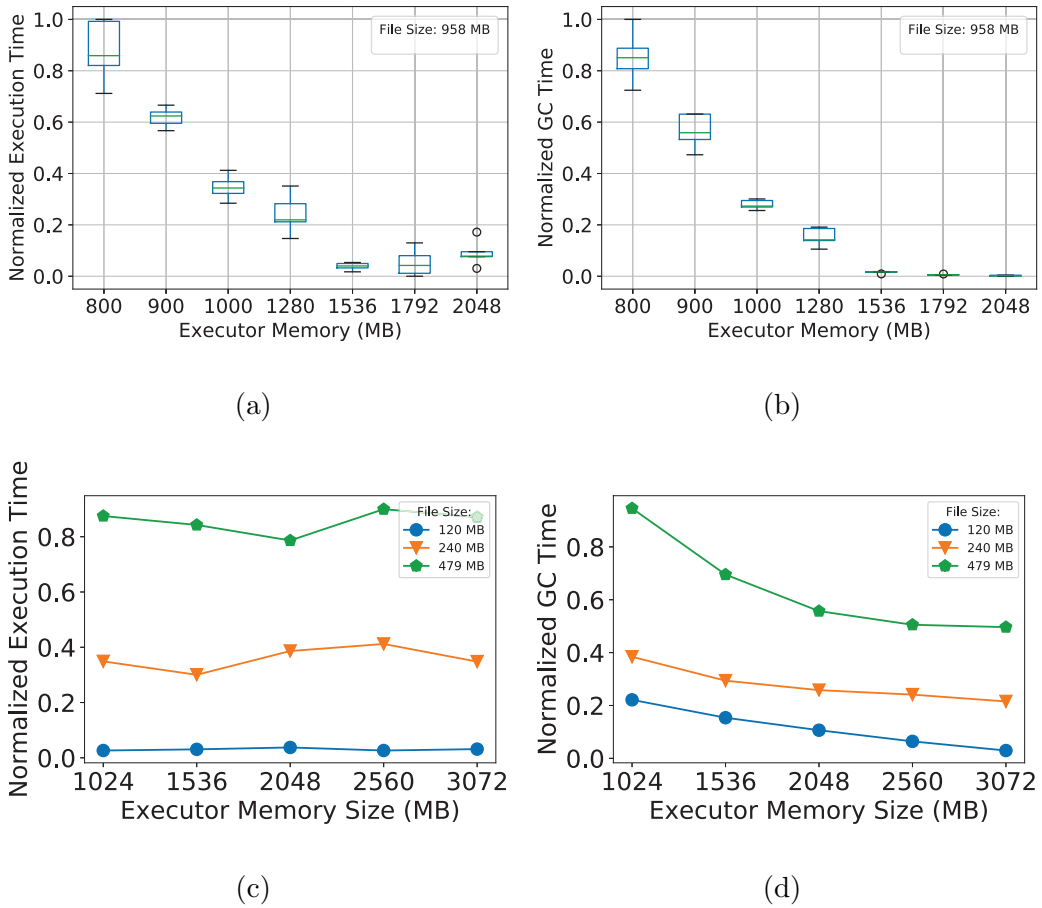


Figure 3.1 Illustration of the effect of executor memory size on the performance of the linear regression workflow: (a) execution time vs. executor memory; (b) GC time vs. executor memory size; (c) workflow execution time vs. executor memory size when processing small files with sufficient memory; and (d) GC time vs. executor memory size when processing small files with sufficient memory.

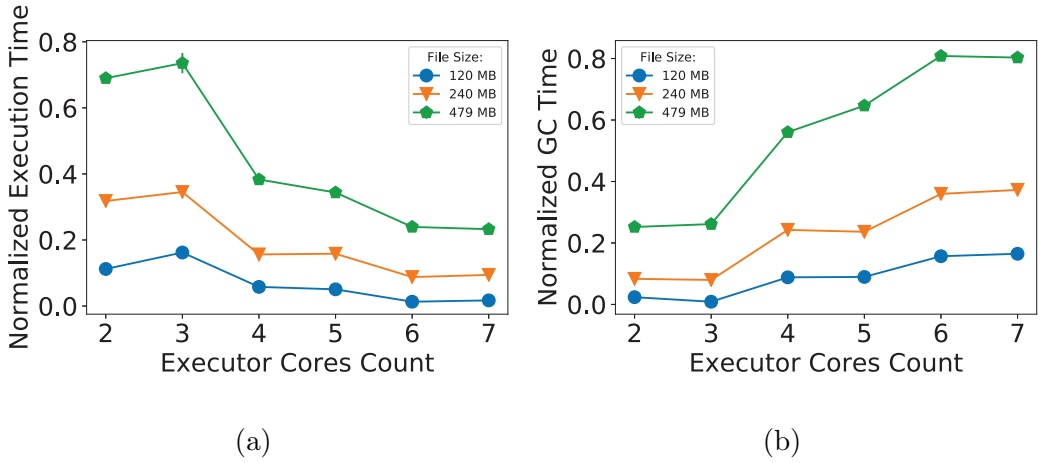
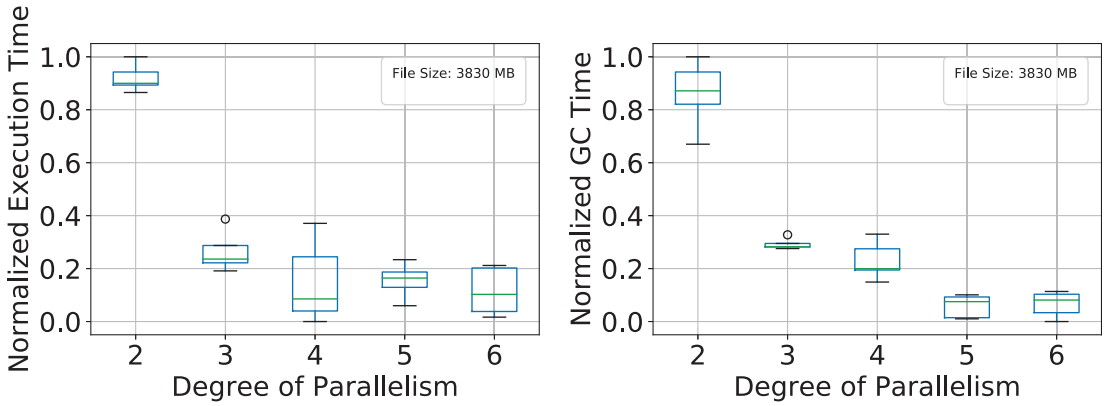


Figure 3.2 Illustration of the effect of executor core count on the performance of the linear regression workflow: (a) workflow execution time vs. executor core count with different file size; and (b) GC time vs. executor core count with different file sizes.

pattern is qualitatively consistent. As the core count increases, the workflow execution time decreases as shown in Figure. 3.2(a), while the GC time increases as shown in Figure. 3.2(b). More executor cores, which mean more computing power to run more concurrent tasks, finish workflow execution faster, but meanwhile requiring more memory to store intermediate results and hence triggering the GC process more frequently. The decrease trend in workflow execution time and the increase trend in GC time reach a plateau after a certain point, indicating that for a given input data size, adding an excessive number of cores to the executor would not bring a significant benefit to the workflow performance.

3.3.3 Degree of Parallelism for RDD

The degree of parallelism is critical to the performance of parallel computing, which is a viable solution to big data processing. It is generally beneficial to increase the degree of parallelism, but the performance gain from parallel processing may be offset by the increased communication overhead for intermediate data collection and exchange. Spark achieves high-level parallel processing by introducing the concept of Resilient



(a) Workflow execution time vs. parallelism

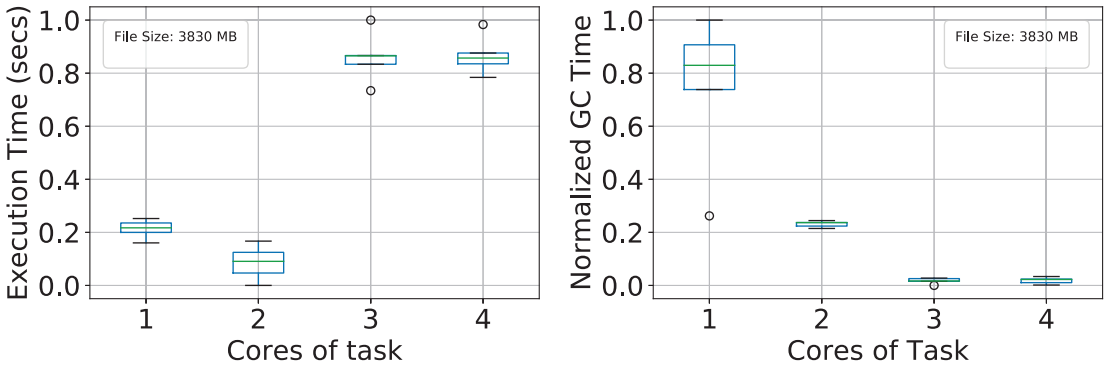
(b) GC time vs. parallelism

Figure 3.3 Illustration of effects of parallelism on the performance of the linear regression workflow.

Distributed Datasets (RDDs), which are transformed from the data source (e.g., files in HDFS) and then partitioned and processed in parallel on different data nodes across the cluster. RDDs could be further divided into smaller partitions to increase the degree of parallelism. The parameter "spark.default.parallelism" defines the largest number of partitions in a parent RDD for distributed RDD operations. Figure. 3.3 plots the performance of the same linear regression workflow to process an input file of about 4GB in response to different settings of "spark.default-parallelism". As shown in Figure. 3.3(a), the performance increases significantly as the degree of parallelism increases until reaching the "optimal" number of parallelism, and remains stable afterwards. The total amount of GC time decreases as the degree of parallelism increases, as shown in Figure. 3.3(b), which is consistent with the total execution time in Figure. 3.3(a). With a higher degree of parallelism, the RDD is divided into smaller partitions, which require less memory for processing, and hence trigger the GC process less frequently.

3.3.4 Task Core Count

A task is an atomic executable unit that can be executed in an executor on a partition of a RDD. The parameter "spark.task.cpus" defines the number of cores allocated to each task. Since Spark tasks are executed in serial and Spark performs parallel computing at the task level, theoretically, increasing the task core count does not affect workflow performance. However, since the executor has a fixed number of cores, increasing "spark.task.cpus" would reduce the number of concurrent task executions (i.e., the degree of parallelism) with less memory needs. This explains the increase in the workflow execution time as shown in Figure. 3.4(a) and the decrease in the GC time as shown in Figure. 3.4(b), which are measured after allocating eight virtual cores to each executor.



(a) Execution time vs. task core count (b) GC time vs. task core count

Figure 3.4 Illustration of effects of task core count on the performance of the linear regression workflow.

We also run several other types of workloads such as random forest regression to examine the impact of different parameters on workflow execution performance. The performance measurements are qualitatively similar to those measured from the linear regression workflow. We would like to point out that the impact of these parameters is complex, especially when there exist coupled effects between

different parameters, which strongly suggest the use of machine learning algorithms for performance modeling and prediction.

3.4 Functional and Coupled features

Big data systems encompass a large parameter space constituted by multiple layers including application (workflow), middleware (e.g., Spark/YARN), and hardware system (VM provisioning). The performance optimization of big data workflow execution in such computing systems requires an exploration of the configuration space, as well as the interacting terms and other high-order mutated terms [38]. Functional and interactive effects are typically hidden to end users. Identifying the most influential hidden terms, e.g, the ratio of two parameters such as memory size and input file size, which largely determine the performance of workflow execution, is of great importance to helping end users with parameter setting for workflow submission to WMS. To achieve this goal, we probe a comprehensive list of terms including the parameters sampled in the configuration space and other terms constructed using heuristic approaches and domain knowledge.

Building an accurate performance-influence model with parameters across multiple layers in big data systems requires an investigation into both independent and interactive parameters. We probe both configurable parameters of big data systems and constructed terms derived from specially designed mapping functions. However, it is theoretically impossible to consider all parameters in constructing the feature pool, as the number of possible combinations is exponentially large with respect to the number of parameters. Hence, we employ different heuristic strategies to sample candidate features that affect workflow execution time.

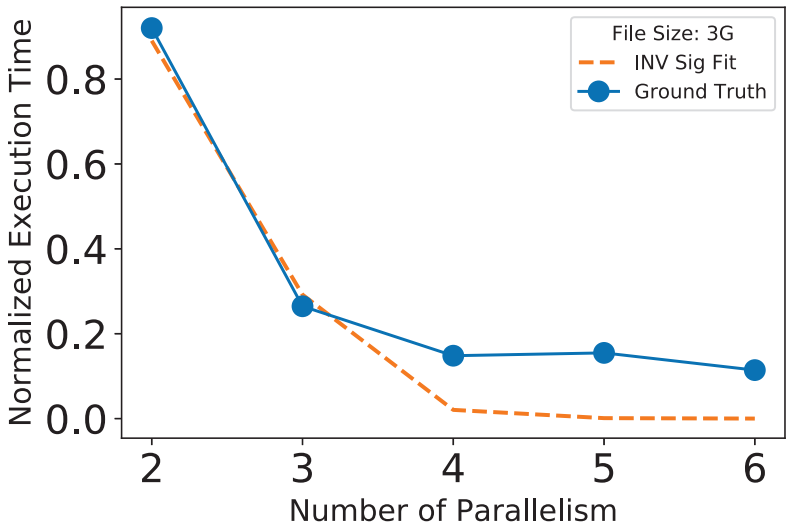


Figure 3.5 Performance profile fitting with an inverse sigmoid-based regression function in response to the number of parallelism.

3.4.1 Domain Knowledge-based Feature Selection

Many existing big data systems such as Hadoop provide a large number of interfaces for end users to specify parameter settings according to the needs of their computing. For example, Spark provides over 160 properties for end users to tune and YARN specifies over 100 properties in the XML configuration files. A black-box optimization approach through an exhaustive profiling strategy is practically infeasible, as the number of required profiling experiments grows exponentially with the number of parameters. Hence, we adopt the human-in-the-loop (HITL) strategy to perform configuration space sampling. Based on the domain knowledge, we consider a list of parameters that are related to the setting of executors and containers as well as some observable intermediate parameters as shown in Table 3.1.

Functional Features The individual impact of any continuous parameter p in the configuration space on the workflow performance could be approximated by a function $f(p)$, which is an important mapping that depicts its independent effect on the performance. Such representation not only facilitates the interpretability of a

Table 3.1 List of Parameters Across Different Layers for Workflow Execution.

Layers	**Parameters**	**Remarks**
Workflow	input file size	integer, MB
	machine learning model	string
WMS	executor memory	integer, MB
	executor CPU	integer
	driver memory	integer
	number of executor	integer
	maximum allocate memory	integer, MB
	shuffle compress	boolean
	locality wait	integer, secs
	number of parallelism	integer
	memory storage fraction	float
Intermediate	CPU consumption	integer
	memory consumption	integer
	total GC time	integer
	total input bytes	integer
	total shuffle read	integer
	total shuffle write	integer

performance-influence model based on machine learning, but also provides a valuable insight into parameter setting.

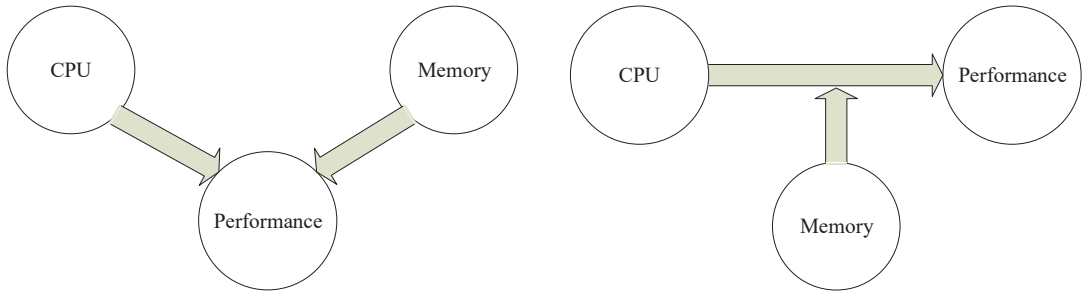
The exploratory analysis in Section. 3.3 suggests that the regression of the influence of executor core count and degree of parallelism can be approximated by a scaled inverse sigmoid function:

$$f(p) = 1 - 1/(1 + e^{-\alpha(p-p_0)}), \tag{3.1}$$

where α and p_0 are hyper parameters. As shown in Figure. 3.5, the performance profile in response to the number of parallelism falls in the convex region of an inverse sigmoid function. In order to expand functional representation and enrich the candidate feature pool, we further construct a set of functional mappings including tanh, sigmoid, inverse sigmoid and exponential functions.

3.4.2 Coupled Features

In big data systems built on Spark and YARN, the setting Y in the YARN layer often serves as a "threshold", as it defines important properties of the container, e.g., maximum memory/CPUs allocated to containers. Different threshold settings in YARN may have a significantly different impact on the workflow execution time, as it controls the total number of containers simultaneously provisioned in the system. The setting S in the Spark layer configures the runtime environment for task execution by allocating computing resources at the executor and task level. Since Y and S are controllable parameters in different layers of the system and hence are independent of each other in parameter setting, the probability of a certain parameter setting $P(Y, S) = P(Y) \cdot P(S)$. However, they may affect each other and have complex coupled effects on the workflow performance, as demonstrated by the interactive impact between executor memory size (Memory) and executor core count (CPU) shown in Figure. 3.1. Such interactive impact is also termed as k-interact [10]. As



(a) Independent influence (b) Interactive influence

Figure 3.6 Independent and interactive influence of Memory and CPU on workflow performance.

illustrated in Figure. 3.6, individual effect (such as Memory or CPU in Figure. 3.6(a)) is generally observable and measurable, while coupled effect (such as the one between Memory and CPU in Figure. 3.6(b)) is typically complex and requires extra efforts to measure.

In order for the predictor to learn the corresponding knowledge from interactive impact across layers, we construct new features to approximate such coupled effects by combining various parameters across different layers and fitting the corresponding performance profile with a certain mapping function. Such parameter combinations include the ratio of the input file size to the executor memory size, and the ratio of the executor memory size to the maximum memory size of a container specified in the YARN layer.

3.5 Workflow Performance Prediction in Big Data Systems

To build an accurate performance-influence model, we propose to use a machine learning-based algorithm to select critical features $\hat{\mathbf{x}}$ from the candidate feature pool. We first present the information-theoretic feature selection method and then discuss the prediction performance of our proposed method.

3.5.1 Machine Learning-based Feature Selection

Feature selection, which is an important problem in machine learning, produces a subset of features with minimum irrelevant and redundant information to help reduce data size and build an effective model without sacrificing prediction accuracy. Traditional methods as exemplified by learning model-based algorithms iterate through all possible combinations of subsets and return one that yields the highest accuracy. However, such an exhaustive search-based strategy is very computationally expensive and practically infeasible when dealing with a large number of features, as the total number of possible subsets grows exponentially. Hence, we propose to employ heuristic algorithms to compute such subsets, and recognize performance patterns with machine leaning models.

Rational on the Use of Mutual Information-based Algorithms In our prediction problem, instead of directly modeling in the original feature space, we conjecture that the performance of big data workflows in big data systems could be approximated by independent features U, functional features D and interactive features H, which are hidden in the candidate feature pool, e.g., $y = f(\hat{\mathbf{x}}) + \epsilon$, where $\hat{\mathbf{x}} = \{U, D, H\}$, and ϵ represents the error caused by system dynamics. Such an approximation strategy not only improves the explainability of the performance-influence model, but also provides end users with valuable insights to parameter setting, e.g., by setting an appropriate ratio of input file size to executor memory size to avoid resource waste.

However, due to the large candidate feature pool, it is extremely challenging to perform feature selection, especially considering the complex interactive impact between individual features, which is commonly recognized as k-way positive interaction [10]. We further illustrate this property in Figure. 3.7, where we use the three-way interaction between CPU, memory, and performance as an example.

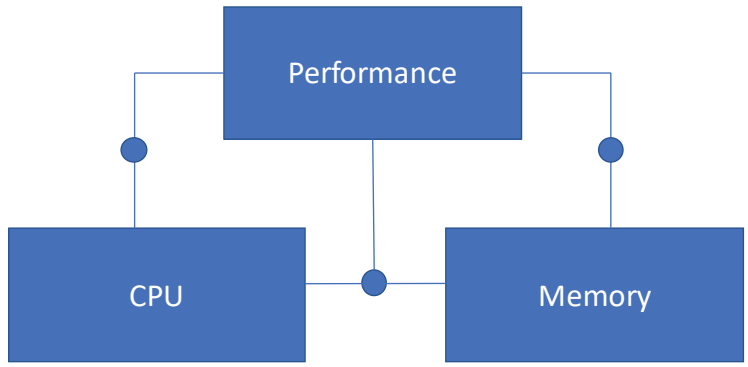


Figure 3.7 Three-way positive interaction between CPU, memory, and performance, where blue dots represent mutual information.

Memory (m) and CPU (c) are independent of each other as they can be specified separately by end users. Hence, the mutual information between m and c is zero without any knowledge from performance y. However, given the response value of performance, the conditional mutual information $MI(c, m|y)$ could be non-zero and measured from historical data. Such analysis further motivates us to use information-theoretic feature selection to validate our conjecture and decide a proper subset of $\hat{\mathbf{x}}$. Based on the new features constructed to represent interactive impact as discussed in Section. 3.4.2, we propose to measure pair-wise mutual information to rank the contribution of interactive features to the performance. The mutual information between two random variables A and B is defined as [16]:

$$D_{KL}(J_{A,B}||M_A \otimes M_B),$$

where $M_A \otimes M_B$ denotes the product of two marginal distributions, $J_{A,B}$ denotes their joint distribution, $||$ denotes the distance between two distributions, and D_{KL} is the Kullback Leibler divergence between two distributions .

As stated previously, the performance y is largely affected by U, D and H. As U could be represented by a weighted value of individual parameters identified in Section. 3.3, our work is focused on quickly identifying the most important subset

of constructed features that affect the workflow execution performance to the largest extent. More formally, we aim to find a subset $\{\mathcal{S}\}$ from candidate features $\{\mathcal{O}\}$, which could maximize $F(\cdot)$ under a constraint such that the total cost $\mathcal{C}$ is limited, where F is an evaluation metric that could be used to evaluate the correlation between $\{\mathcal{S}\}$ and y (e.g., in terms of mutual information, accuracy, etc.), and $\mathcal{C}$ is the iteration constraint. To rank the importance of interactions between features, we propose to use mutual information as a score function to quantify the correlation between S and y. Therefore, our objective is to solve the following optimization problem:

$$\underset{\mathcal{S}}{\operatorname{argmax}}\, I(\mathcal{S} : y),\ \ s.t.\ C(S) < \delta, \tag{3.2}$$

where $I(\cdot)$ denotes the mutual information.

Note that a set function that maps from N-dimensional feature space to a real value, i.e., $f : 2^N \rightarrow \mathbb{R}$, is submodular [11] if for every $A, B \subseteq N$,

$$f(A \cup B) + f(A \cap B) \leq f(A) + f(B),$$

where N denotes the set of all available features, and A and B are two subsets of N. Furthermore, as mutual information belongs to the family of submodular function, maximizing Equation. 3.2 is equivalent to optimizing k-constraint submodular function, which has been proved to be NP-hard and solved by a greedy heuristic approach in [11].

Similar to the work in [11], we choose candidate features that affect y in a greedy manner by thresholding the mutual information between the features and the response vector. More specifically, in each step of feature selection, we evaluate the mutual information between candidate features and y, and then choose the one with the highest value if it is greater than the pre-specified threshold. The pseudocode of this feature selection process is provided in Algorithm. 1.

Algorithm 1: Greedy Feature Selection

Input: candidate feature set $\mathcal{P}$, mutual information threshold τ

Output: selected feature set X

```
X = φ;
for each t_i in P do
    t_i = arg max_{t_i} I(X ∪ t_i : Y);
    if arg max_{t_i} I(t_i : Y) > τ then
        X = X ∪ t_i;
return X;
```

3.5.2 Performance Prediction and Configuration Recommendation

With a subset of critical features selected using the proposed information-theoretic method, we now need to select an appropriate machine learning model that can effectively draw information from both individual and dependent features. Towards this goal, we consider and compare the performance of a set $\{\mathcal{M}\}$ of machine learning models commonly used for regression, including [37]: i) Linear Regression (LR) as a linear model, ii) Support Vector Regressor (SVR) as a kernel-based model, iii) Random Forest Regressor (RFR) as an ensemble model, and iv) Multiple Layer Perceptron (MLP) as a Neural Network model.

We use the experiment-based cross validation method [13] to solve the following optimization problem:

$$\underset{\mathcal{M}^*, \theta_m^*}{\operatorname{argmin}} \mathcal{L}(\mathcal{M}(X, \theta_m), y), \tag{3.3}$$

where $\mathcal{M}$ denotes a machine learning model with hyper parameter θ_m, and $\mathcal{L}$ is the loss function. The best model $\mathcal{M}^*$ obtained by optimizing Equation. 3.3 can be used to predict workflow performance with new parameter settings. Based on such

predictions, we are able to make performance comparison and then select the optimal system configuration that results in the minimum execution time for recommendation.

3.6 Performance Evaluation

In this section, we first describe the experimental settings for executing two test workflows, and then present the prediction results of a performance-influence model based on our feature selection method.

3.6.1 Test Workflows

To evaluate the performance of workflow execution time prediction, we consider two test workflows that perform regression on a set of input files. The first workflow employs linear regression as in the empirical study conducted in Section. 3.3, and the second workflow employs random forest. Both workflows feature a pipeline structure that consists of three computing modules as shown in Figure. 3.8. Specifically, the first module is to split a given input file into two parts with ratio 9:1 for training and testing, respectively, the second module is to train a model using linear regression or random forest, and the third module is to test the trained model. Both of the regression models are implemented in Spark using the `MLlib` library [50]. These two workflows are tested with input files of different sizes within the range $\{120, 240, 479, 958, 1915, 3830\}(MB)$ on a local PC cluster consisting of 3 VM instances, each of which is assigned with eight virtual cores and 24GB memory.

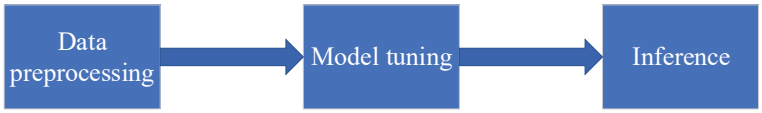


Figure 3.8 The pipeline structure of the test workflows for regression.

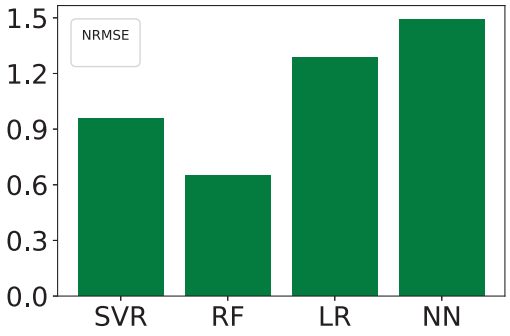
3.6.2 Configuration Space Sampling

We deploy and run these workflows on the same local Hadoop cluster with Spark and YARN as in the empirical study conducted in Section. 3.3. Although Spark and YARN provide a large configuration space, most of the settings are irrelevant to the execution time, e.g., port number, log location, etc. Hence, instead of investigating the entire configuration space, we focus on tunable parameters related to executors and observable runtime features as shown in Tab. 3.1. For numerical parameters, we take sample values incrementally within a valid range, and for non-numerical parameters such as boolean type, we exhaust all possible values. The test workflows are executed with such combinatorial settings and the corresponding workflow performance measurements are used as the data source for performance prediction.

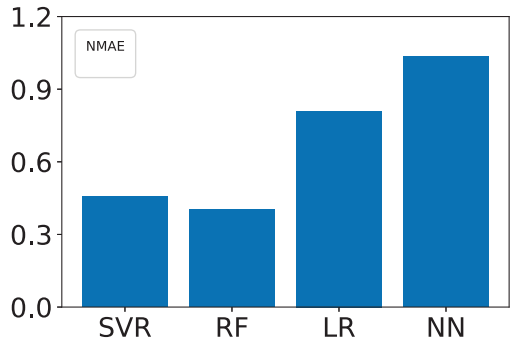
3.6.3 Performance Prediction Results

We implement a performance-influence model in Python for workflow execution prediction with different regression algorithms using the `scikit-learn` library [22]. The performance of this prediction model is evaluated in two steps: i) we compare the prediction results of various regression algorithms based on the original set of individual parameters in terms of different performance metrics and select the best model as the baseline model; and ii) we show the performance improvement of the baseline model based on both individual and interactive features selected by the proposed information-theoretic feature selection method.

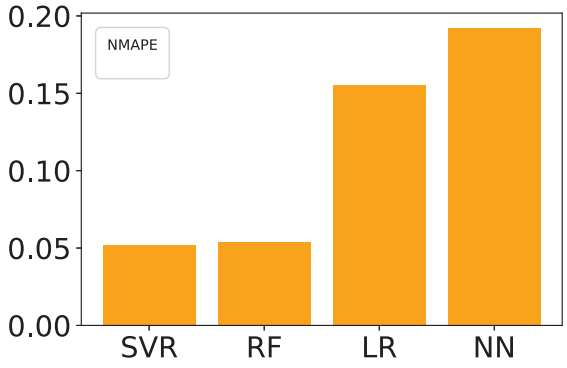
Performance Comparison of Regression Models We first split the performance measurement data collected from the execution of two test workflows into two parts for training and testing, respectively, and then perform 10-fold cross validation [13] using the training data to fine tune four representative regression models, i.e., LR, SVR, RFR, and MLP. We measure the prediction accuracy of these models



(a) Normalized Root Mean Square Error (NRMSE)



(b) Normalized Mean Absolute Error (NMAE)



(c) Normalized Mean Absolute Percetage Error (NMAPE)

Figure 3.9 Performance comparison of various models in terms of different metrics.

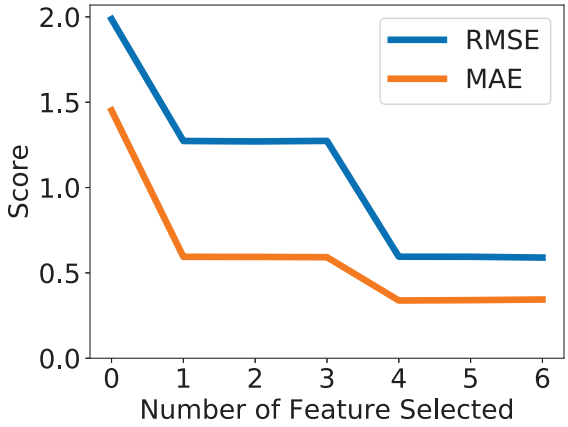


Figure 3.10 The performance of the RFR-based predictor with an increasing number of selected features.

in terms of various performance metrics including Normalized Root Mean Square Error (NRMSE), Normalized Mean Absolute Error (NMAE), and Normalized Mean Absolute Percentage Error (NMAPE), as plotted in Figure. 3.9. The LR model performs poorly as it fails to capture the non-linear nature in the performance-influence relationship. The MLP model exhibits a worse performance because the training data is insufficient to train a neural network architecture with multiple layers. RFR and SVR perform almost equally well in terms of NMAPE. However, RFR has the best overall performance for all metrics, and hence is selected as the baseline model for further investigation with feature selection for performance improvement.

Performance Improvement with Feature Selection We use the proposed information-theoretic feature selection method to identify a subset of critical individual features and construct interactive ones that have the most significant impact on workflow execution time, as tabulated in Table 3.2. These selected features are ranked according to the amount of mutual information between each feature and the response vector, i.e., the workflow execution performance. We rerun the RFR-based performance predictor with an increasing number of selected features and measure

Table 3.2 List of Ranked Critical Features

Name	**Source**	**Description**
CPURatio	Constructed	File size/executor core count
MemoryRatio	Constructed	File size/executor memory size
FileSize	Original	Input file size
InvSigPara	Constructed	Inv-sigmoid mapping of parallelism

the corresponding prediction performance. As shown in Figure. 3.10, the prediction accuracy improves as more features are considered and the top four are considered critical features.

CHAPTER 4

MACHINE LEARNING-ASSISTED COMPUTATIONAL STEERING

4.1 Steering Objectives and Effects of Hyperparameters

In scientific simulations, there are three main goals of computational steering: performance optimization, algorithm experimentation, and model exploration. In performance optimization, steering is used to improve an application's performance, e.g., by balancing workload in parallel applications. In algorithm experimentation, it allows the user to adapt program algorithms at run time, e.g., to experiment with different numerical solving methods. In this work, we focus on model exploration, where computational steering is used to explore parameter spaces and simulation processes to gain additional insights into the model behaviors.

Without loss of generality, we define the steering objective of model exploration as

$$\underset{\vec{x}}{\operatorname{argmin}}\, f(\mathcal{S}(\vec{x}), obs), \tag{4.1}$$

where $\mathcal{S}$ is a model of steering with tunable parameters $\vec{x}$ of interest and f is a user-specified objective function, e.g., the Mean Square Error (MSE) between the output (simulation data) of the steering model and the observation data obs.

To illustrate the effects of hyperparameters on the steering objective f, we simulate the WRF-Solar model with various hyperparameter settings, and then fit the simulation trails and plot the response surface of MSE using Gaussian Regression Model.

As shown in Fig 4.1, the response surface of f is significantly affected by the intricate interplay of tunable parameters such as relative dispersion and condensation rate. Although one ideal approach would be to check all possible parameter settings, it is practically infeasible to do so because the number of possible parameter

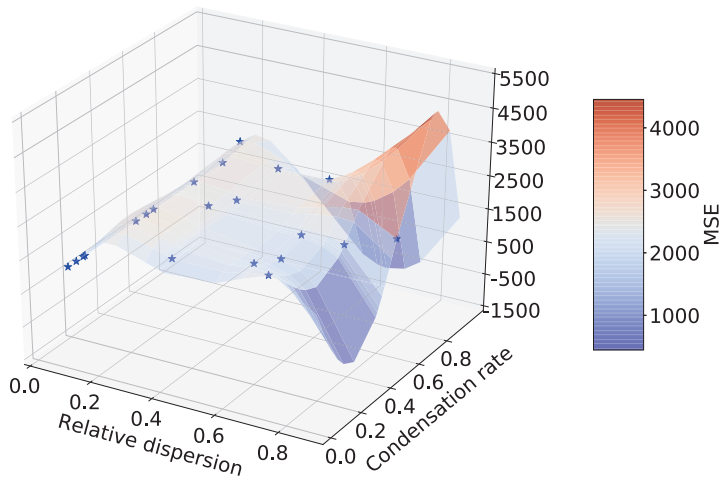


Figure 4.1 Response surface of Mean Squared Error fitted by Gaussian regression.

settings in a given model-based simulation typically grows exponentially with the number of parameters. For example, if a simulation involves the interaction of n binary parameters (that is, each has one of the two states), this leads to checking and understanding 2^n possible parameter settings with respect to the underlying goal [43, 33, 15]. An additional level of complexity arises from the underlying model that dictates how these parameters contribute to the goal. In reality, most parameters are continuous and represent an even much larger number of states or values than binary ones.

4.2 A Machine Learning-Assisted Framework

To bring optimization-guided *autonomy* to the generic steering process in various scientific applications, we propose a framework of Bayesian Optimization-assisted Steering as a Service for collaborative computational steering, Co^2Steer, in support of large-scale simulation-based computational sciences.

As shown in Figure. 4.2, the overarching framework of Co^2Steer integrates the following technical components: i) transport method for steering, ii) machine learning-

based automatic parameter tuning, iii) front-end dashboard, and vi) provenance tracking. The web-based dashboard provides a user interface through a web browser for users to access Co^2Steer, which is hosted as a service. A scientific workflow that constitutes a model-based simulation process and various post-data processing jobs are executed in a specific computing system designated by the user. The interactions between the steering engine and the scientific workflow are carried out over a unified communication channel that enables the delivery of various steering commands from multiple users to different simulation processes being executed concurrently. The output data are analyzed for model validation with visual feedback provided to the user through the dashboard. The entire steering process, and model validation are managed, tracked and recorded in a provenance database.

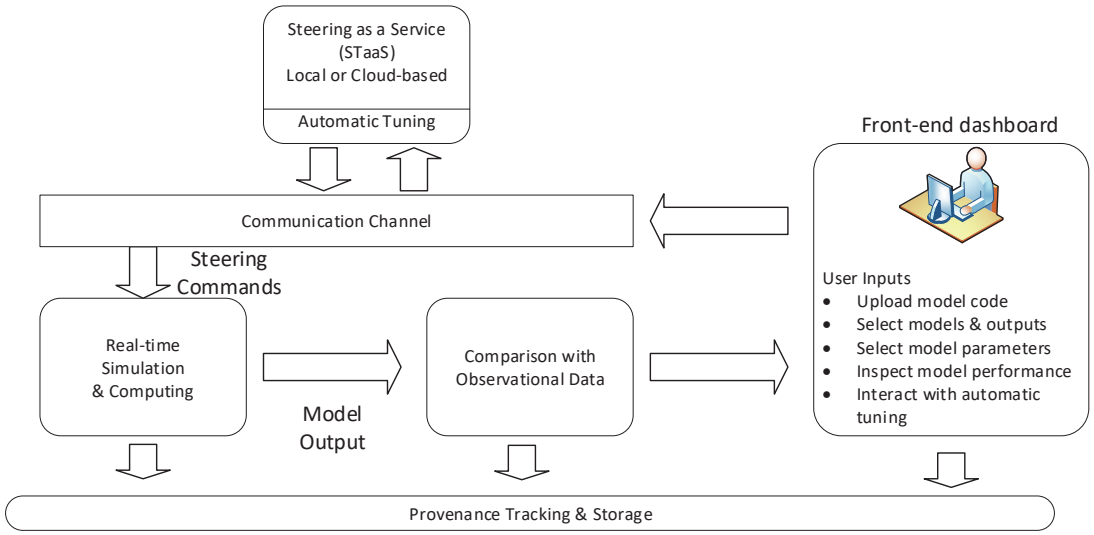


Figure 4.2 Architecture of the Co^2Steer framework.

4.2.1 Steering with Bayesian Optimization

Scientific simulations often encompass a large parameter space that constitutes various of computation-intensive processes. The computational steering problem is to

determine a specific configuration of model parameters to produce a desired output with a limited number of trails.

Our approach consists of two steps: i) build a prior belief model (i.e., a Gaussian regression model $\mathcal{M}$) to describe the relationship between the parameters $\vec{x}$ and the steering objective y using existing historical trails, and ii) automate the steering process based on the underlying model $\mathcal{M}$ and expert guidance. The second step continues in an iterative manner until the goal is achieved with a certain error tolerance. Our approach could be broadly classified as fully automated steering with supervised algorithms and automated steering with hybrid algorithms. These solutions are objective-driven and follow the aforementioned two-step approach. We design the tuning process as an iterative stationary process and propose to use Gaussian process to explore the unknown mapping f that captures the interplay between the parameters and the objective.

Fully Automated Steering with Supervised Algorithms The steering objective of interest is determined by the configuration of a set of control parameters $\vec{x}$ (e.g., relative dispersion and condensation rate in a weather forecast model) and the observation $f(\cdot)$ is corrupted with independent, identically distributed noise $\epsilon = N(0, \sigma^2)$, i.e., $y = f(\cdot) + \epsilon$. Due to the high computation overhead for simulating a large number of complex scientific processes, the evaluation of each configuration could be prohibitively time-consuming. Even with common model complexity and parameter space, it may oftentimes take half a day to complete.

The second step of our iterative approach selects the next configuration of the parameters for validation. Hence, it is important to exploit the historical validations and explore the unknown configurations. BO offers desired properties to balance exploration and exploitation. In each iteration, BO fits the existing dataset using a machine learning model $\mathcal{M}$ (e.g., Gaussian Process Regression), selects the next

query point by maximizing an acquisition function, and updates the dataset. The implementation details are shown in Algorithm. 2.

Algorithm 2: Bayesian Optimization

Input: surrogate model $\mathcal{M}$, simulation model $\mathbb{S}$, historical dataset $\mathcal{D}$ and acquisition function $\mathcal{L}$

Output: The best configuration $\vec{x}^*$

1: **for** $t_i = 1, 2, \ldots, T$ **do**

2: fit $\mathcal{M}$ with $\mathcal{D}$;

3: obtain the next query point $\vec{x}^{'} = \text{argmax}_x \mathcal{L}$;

4: simulate with a new configuration: $y^{'} = \mathcal{S}(\vec{x}^{'})$;

5: consolidate the data: $D = D \cup \{\vec{x}^{'}, y^{'}\}$;

6: query $\mathcal{D}$ to obtain the best configuration $\vec{x}^*$ that results in the minimum y;

7: **return** $\vec{x}^*$;

Semi Automated Steering The use of automated methods does not, however, obviate the need for subjective judgment concerning the priorities and targets of the steering process. Moreover, BO is shown to be overconfident in searching some unexplored boundary region and may lead to unnecessary cost.

To address the overconfidence problem, we propose a hybrid approach to involve humans in the steering loop, but only in a strategic manner. We present the dynamic steering procedure to domain experts, who could provide a wide range of feedbacks - from simple binary feedbacks (e.g., labeling a parameter as important vs. unimportant) to non-binary feedbacks (e.g., iteratively shrinking the search boundary), or a combination of both. Based on such feedbacks, we update model M and repeat the overall computational loop until certain accuracy is achieved.

4.2.2 API Design and Provenance Tracking

One essential function of Co^2Steer is to enable computational steering by a group of remote collaborative users who wish to keep track of the simulation process SIM and communicate and share simulation/analysis results DS_{out} or DS_{final} with peers while the simulation is being steered in batch mode. To make this possible, the simulation steering can no longer remain closed and needs to open up channels to intercept and distinguish steering commands from different users at runtime. Towards this goal, we define the syntax and semantics of a set of generic core APIs, and provide steering capability through automatic mode: Users upload their codes through the dashboard and Co^2Steer identifies the locations in the codes for taking appropriate steering actions at the entry or exit of a loop that implements the body of a simulation process.

Figure. 4.3 illustrates the code skeleton of typical model-based simulation programs that call a set of essential API functions for computational steering by multiple users simultaneously. Among them, $Co2Steer\_init()$ initializes the steering process by subscribing to the communication channel and registering with the steering service and provenance database. A steering action SA is captured by $Co2Steer\_recvMsg()$ in the beginning of each iteration to update the parameter setting used in the simulation process SIM. All changes and corresponding results are recorded in the provenance database and also sent back to a group of participating users for analysis and examination.

Provenance is a key part of the architecture and service in the common computing infrastructure for tracking processes and analyzing results. Particularly, in a model-based simulation process, it is critical to keep track of all configuration options, model versions, parameter choices, etc., all of which have an impact on the outcome of the model simulation. Such provenance information provides a complete view of the derivation process from original sources to final results, and

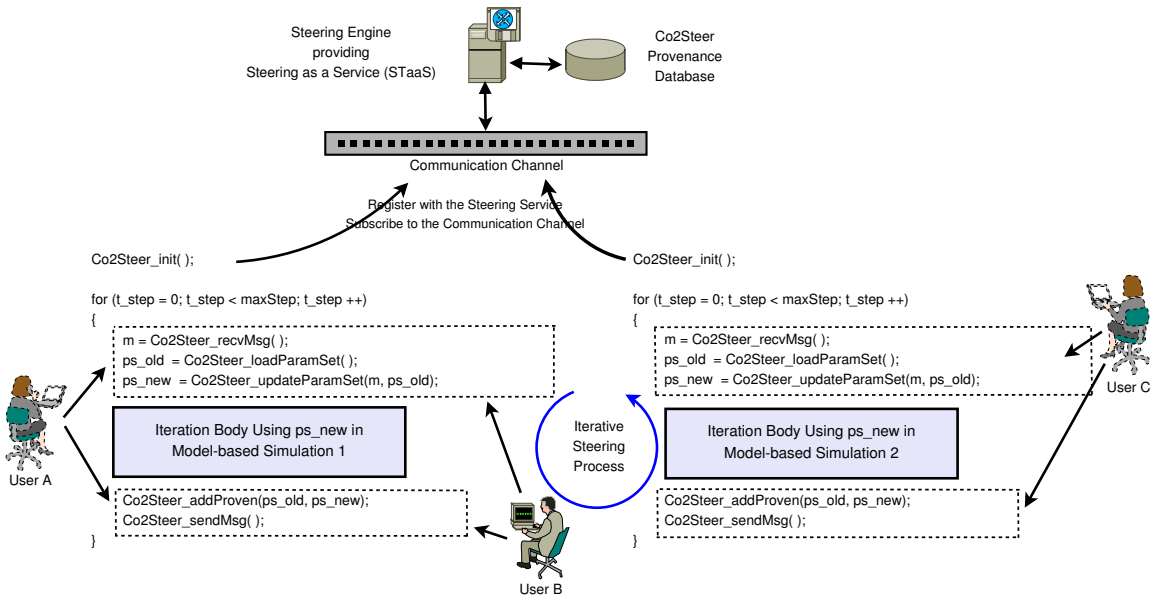


Figure 4.3 A code skeleton of typical model-based simulations that make steering API calls for computational steering engine and communication channel.

enables scientists to verify the correctness of their simulations and reproduce them if necessary. We design a provenance component using script and integrate it into the Co^2Steer steering engine to provide complete provenance information related to the execution of the simulation for post-data processing and analysis. The provenance component automatically records all the information about the simulation process such as the execution time and parameter settings, and tracks the history and evolution of all trials.

4.3 Convergence Rate Analysis

The steering objective of our Bayesian Optimization-based framework for large-scale scientific simulations is to find the optimal hyperparameter setting $\vec{x}'$ with the least number of iterations such that the error between simulation result $\mathcal{S}(\vec{x}')$ is arbitrarily close to the observation obs, i.e.,

$$f(\mathcal{S}(\vec{x}'), obs) - f(\mathcal{S}(\vec{x}^*), obs) < \delta, \tag{4.2}$$

where $\vec{x}'$ is the best guess of our optimization method, $\vec{x}^*$ is the unknown global optimal setting of $\mathcal{S}$, and δ is a positive constant.

To determine the convergence speed of computational steering, we need to estimate the total number of iterations required to achieves δ accuracy. Moreover, the general global optimization problem is theoretically intractable without making any assumptions to function f [30]. Without loss of generality, we consider f to be Lipschitz continuous, i.e., the simulation error $f(\cdot)$ cannot vary arbitrarily fast as we change $\vec{x}$. The convergence rate of our approach is dominated by the theoretical bound of the Bayesian Optimization technique, which contains two intertwined components: the surrogate component for exploitation and the acquisition component for exploration. Our approach follows the standard setting, where the Gaussian process regression model is selected as the surrogate model and the expected improvement (EI) function is used as the acquisition model [25]. The convergence speed of BO has been widely investigated [8, 20]. As stated by Bull in [8], under certain hypothesis, the expected improvement-based BO is shown to converge to the minimum of any function on its Reproducing-Kernel Hilbert Space (RKHS) with rate $\mathcal{O}(\frac{1}{\delta^{v/d}})$, where v is a smooth measure of f and d is the number of parameters to optimize over. Note that the smooth measure v of f is assumed to be greater than one to make BO better than random guesses. The time complexity of BO is $\mathcal{O}(n^3)$, where n is the number of historical trails, because BO solves Cholesky decomposition problem for each iteration.

4.4 Validation with Real-World Applications

We develop a prototyped Co^2Steer service that enables, optimizes, and tracks steering-driven simulation-oriented scientific workflows for model exploration and evaluation. This prototype steering service is deployed on a virtual machine (VM) instance equipped with eight processors and 20GB memory. We test Co^2Steer for

model-based simulation in a real-life large-scale scientific application, i.e., a WRF model for climate research.

4.4.1 Case Study: Weather Research and Forecasting

The physical forecasting model of WRF-Solar is built on the widely used community Weather Research and Forecast (WRF) model [6] with an emphasis on forecasting solar and wind energies, and contains numerous parameterized subgrid processes that affect model performance, including cloud and aerosol microphysics, radiative transfer, planetary boundary layer, turbulence, convection, and land surface. The tunable parameters involved in the model are shown in Table 4.1. We first introduce the interested parameters, and then illustrate the tuning effects of such parameters on the simulation quality. We further perform a point-wise comparison between our work and default settings and verify the convergence speed of our solution.

Description of Physical Parameterizations and Important Tunable Parameters

The WRF-Solar model is implemented based on Thomas scheme [23], which contains various sub-processes that simulate aerosol process, cloud droplet, liquid water, etc,.

Particularly, in this case study, we investigate two parameters, namely, relative dispersion and condensation rate, which are accessible and tunable in representative processes that simulate the cloud-to-rain autoconversion and effective radius in liquid water clouds. The final output of this WRF model emulates the evolution of solar irradiance volume in 90 days.

The effective radius r_e is simulated based on a Gamma distribution, which contains two degrees of freedom, i.e., shape parameter and slope parameter.

The relative dispersion ϵ affects r_e through the product of a dimensionless parameter β and the mean volume radius v, i.e.,

$$r = \beta \cdot v. \tag{4.3}$$

Table 4.1 List of Model Parameters

Symbol	Description	Default Value	Range
ρ_s	Density of snow	100 kg/m^3	50-200 kg/m^3
ρ_g	Density of graupel	500 kg/m^3	[450, 700] kg/m^3
ρ_i	Density of ice	890 kg/m^3	[700, 900] kg/m^3
a	Mass power-law constant	0.069	0.0185-0.176
b	Fall speed power-law constant	2	[1.9, 2.2]
α	Fall speed power-law constant	Rain: 4854	0.15
		Ice:1847.5	336-1847.5
		Snow:40	129.6-40
		Graupel:442	351.2-442
β	Fall speed power-law constant	Rain: 1	Fixed
		Ice:1	0.6635 - 1
		Snow:0.55	0.42- 0.55
		Graupel:0.89	0.37 - 0.89
f	Fall speed power-law constant	Rain: 195	Fixed
		Ice: 0	Fixed
		Snow: 125	100-125
		Graupel: 0	Fixed
C	Capacitance of hydrometeor	Sphere: 0.5	15%
		Plates/aggregates: 0.15	15%
E_{yx}	Collection efficiencies	si: 0.05	15% within 0 - 1
		rs: 0.95	1
		rg: 0.75	15% within 0 - 1
		ri: 0.95	15% within 0 - 1
D_0	Lower limit of hydrometeor diameter	Cloud:$1E^{-6}$	C: [$0.5E^{-6}$, $2E^{-6}$]
		Rain: $50E^{-6}$	R: [$50E^{-6}$, $100E^{-6}$]
		Snow: $200E^{-6}$, m	S: [$150E^{-6}$, $250E^{-6}$]
		Graupel:$250E^{-6}$, m	[$200E^{-6}$, $300E^{-6}$]
β_{con}	Condensation rate constant	$1.15E^{23}$	[$1.02E^{20}$, $1.67E^{24}$]
ϵ	Relative dispersion of cloud droplet spectrum	0.1	0.01 to 1.4

For cloud droplets, rain, cloud ice and snow, β is calculated as:

$$\beta = \frac{(1 + 2 \times \epsilon^2)^{2/3}}{(1 + \epsilon^2)^{1/3}}, \tag{4.4}$$

where ϵ is the relative dispersion. In consideration of all unknown effects (e.g., turbulence-related processes), an empirical condensation rate constant β_{con} is defined to emulate the turbulence effect.

Tuning Effects on Simulation Quality The simulation result of the WRF model is a time-series sequence that represents solar irradiance change in 90 steps. The tuning effects of the relative dispersion on the volume of solar irradiance are plotted in Fig 4.4. We observe that the simulation result approaches the observation data as the value of dispersion decreases. However, further decreasing the value of dispersion does not bring performance improvement. This is because some of the simulation processes are skipped due to a negligible value of relative dispersion. We would like to point out that the impact of relative dispersion is complicated, which strongly suggests the use of machine learning algorithms for parameter tuning.

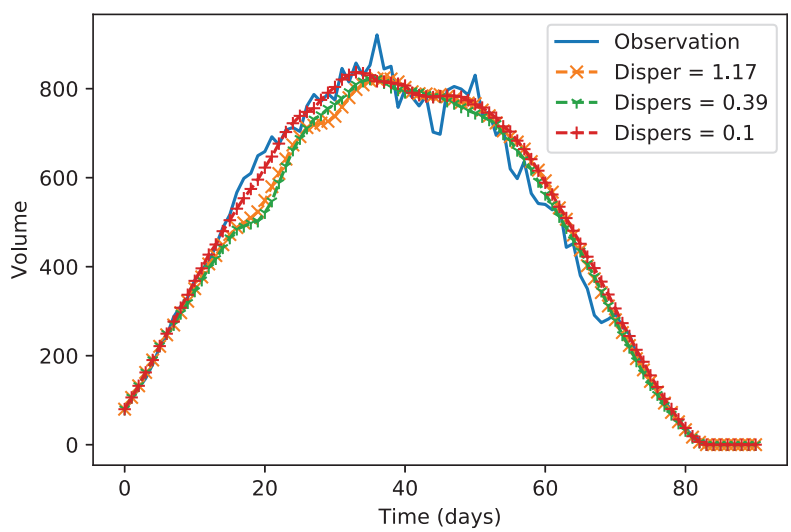


Figure 4.4 Illustration of tuning effects of dispersion on simulation quality.

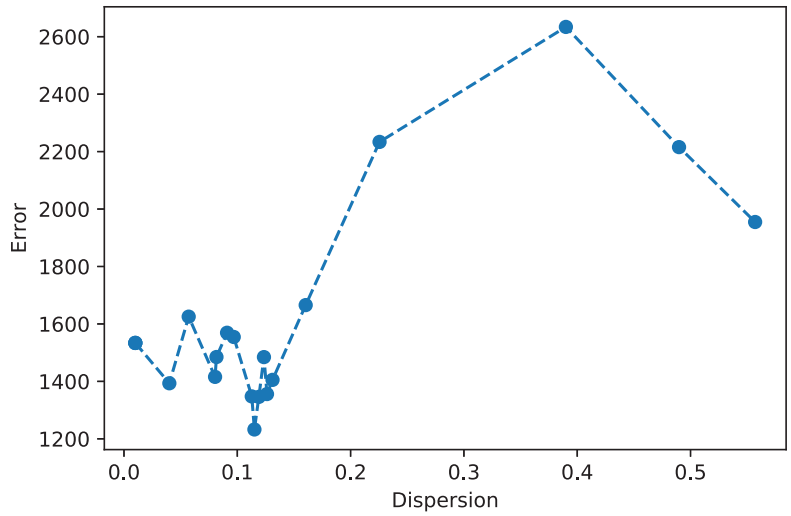


Figure 4.5 Illustration of effects of dispersion on the Mean Squared Error.

Methods in Comparison We compare BO with another heuristic method using random walk [31]. Instead of exploring the search space as a compact real realm, random walk first transforms the searching space into grids, and then iteratively explores the search space by moving a fixed length at a randomly generated angle. We also compare BO and random walk with the default setting to show the performance improvement over manual tuning recommended by domain experts.

Evaluation Metrics The execution of Co^2Steer in climate research involves both model-based simulations and real-world observations. To evaluate the effectiveness of tuning, we calculate Mean Squared Error (MSE), and consider average cumulative regret that measures the convergence rate of steering. In each iteration of online steering, the instantaneous "regret" at the i-th iteration is defined as the distance from the current evaluated value $f(\mathcal{S}(\vec{x}_i), obs)$ to the optima $f(\mathcal{S}(\vec{x}^*), obs)$, i.e.,

$$r = f(\mathcal{S}(\vec{x}_i), obs) - f(\mathcal{S}(\vec{x}^*), obs). \tag{4.5}$$

The average accumulative regret over t time-steps is calculated as $\frac{1}{t}\sum_i r_i$.

Experimental Setting As BO-based tuning relies on historical trails, we randomly generate two data points to initialize the tuning process. For random walk, we select the center point of the search area as the start point and set the step size to be $\pm$ 0.014 for relative dispersion and $\pm$ $1.67E^{22}$ for condensation rate, respectively. In each iteration, the random walk algorithm moves a single mosquito step for each dimension with equal probability.

Results The performance evaluation includes two parts: i) evaluate the effectiveness of our approach in terms of MSE, and ii) evaluate the efficiency of our approach in terms of average accumulative regret.

We first plot the smallest MSE among historical tuning trails in terms of iterations, as shown in Figure. 4.6. The performance of the default setting forms a straight line and can be used as the baseline. The random walk algorithm performs poorly due to the limitation of iterations. The simulation error of our approach drastically decreases with more iterations of steering, and achieves a plateau much lower than random walk and the default setting recommended by domain experts.

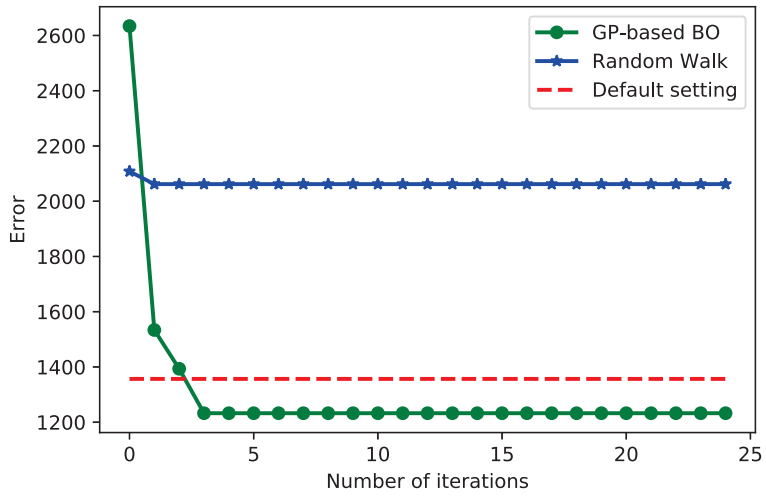


Figure 4.6 Performance comparison in terms of MSE.

We further compare the convergence rate of our approach with that of random walk in terms of average accumulative regret. Figure. 4.7 shows that our approach converges faster than random walk.

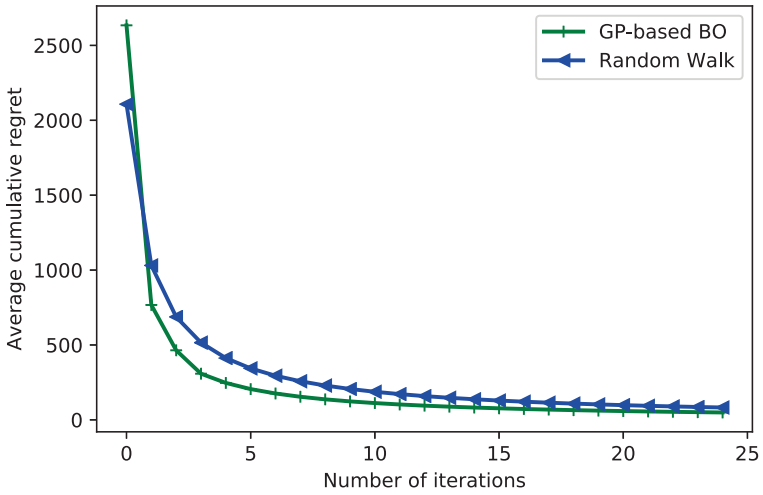


Figure 4.7 Convergence rate comparison in terms of average accumulative regret.

CHAPTER 5

CONCLUSION

This dissertation focus on the development of machine learning-assisted big data workflow solutions and big data transfer solutions for collaborative computational steering to support large-scale simulation-based computational sciences.

This dissertation first identifies a comprehensive list of attributes involved in a typical big data transfer process and then conducts in-depth exploratory analysis of their impacts on application-level throughput, which provides insights into big data transfer performance and motivates the use of machine learning. It further investigates unobservable latent factors such as competing loads on end hosts and use Density-based spatial clustering of applications with noise (DBSCAN) to eliminate their negative impact on performance prediction using various machine learning models such as Support Vector Regression (SVR) and Gradient Boosting Regression. It also design and implement a customized domain-specific loss function within machine learning models such as Stochastic Gradient Descent Regression. Extensive experimental results show that the proposed predictor achieves higher accuracy than several state-of-the-art methods.

On the other hand, the performance of big data computing on HPC platforms is also largely determined by workflow parameter settings and the configurations of underlying computing systems, we propose a codesign framework to help speeding online data reduction and optimizing performance. We developed a prototype steering as a service auto-tuning framework, which consists of a set of APIs for domain experts to execute, monitor, and interact with model-based simulations. We conducted extensive experiments to evaluate the efficiency and effectiveness of our parameter tuning method with a real-life WRF model.

REFERENCES

[1] Energy Sciences Network. http://www.es.net, accessed on November 22, 2021.

[2] ESnet OSCARS Service. http://www.es.net/oscars, accessed on November 22, 2021.

[3] Globus Data Transfer. https://bit.ly/2m10qLI, accessed on November 22, 2021.

[4] Internet2 Advanced Layer 2 Service. https://goo.gl/4iAbQn, accessed on November 22, 2021.

[5] Science DMZ DTNs. https://bit.ly/2k3qBQM, accessed on November 22, 2021.

[6] The Weather Research & Forecasting Model. http://www.wrf-model.org/index.php, accessed on November 22, 2021.

[7] UDT-Powered Projects. https://bit.ly/2JZtA7n, accessed on November 22, 2021.

[8] A. D. Bull. Convergence rates of efficient global optimization algorithms. *Journal of Machine Learning Research*, 12(88):2879–2904, 2011.

[9] A. Jain, S. P. Ong, W. Chen, B. Medasani, X. Qu, M. Kocher, M. Brafman, G. Petretto, G. Rignanese, G. Hautier, D. Gunter and K. A. Persson. Fireworks: a dynamic workflow system designed for high-throughput applications. *Concurrency and Computation: Practice and Experience*, 27(17):5037–5059, 2015.

[10] A. Jakulin and B. Ivan. Quantifying and visualizing attribute interactions:an approach based on entropy. *arXiv:cs.AI/0308002*, 2004.

[11] A. Krause and D. Golovin. Submodular function maximization. In *Tractability*, 2014.

[12] A. Matsunaga and J. Fortes. On the use of machine learning to predict the time and resource consumed by applications. In *IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, 2010.

[13] C. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[14] C. Wu, M. Zhu, Y. Gu and N. Rao. System design and algorithmic development for computational steering in distributed environments. *IEEE Transactions on Parallel and Distributed Systems*, 21(4):438–451, 2010.

[15] C. Yilmaz, M. B. Cohen and A. A. Porter. Covering arrays for efficient fault characterization in complex configuration spaces. *IEEE Transactions on Software Engineering*, 32(1):20–34, 2006.

[16] T. Cover and J. Thomas. *Elements of Information Theory*. John Wiley, 2012.

[17] D. Abramsonand, C. Enticott and I. Altinas. Nimrod/k: Towards massively parallel dynamic grid workflows. In *Proc. of the 2008 ACM/IEEE Conf. on Supercomputing*, 2008.

[18] D. Yun, C. Q. Wu, N. S. V. Rao, Q. Liu, R. Kettimuthu and E. Jung. Profiling Optimization for Big Data Transfer Over Dedicated Channels. In *Proc. of the 25th Int'l Conf. on Computer Communication and Networks*, 2016.

[19] E. DeelMan, G. Singh, M. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob, and D. S. Katz. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming*, 2015.

[20] F. Berkenkamp, A. P. Schoellig and A. Krause. No-regret bayesian optimization with unknown hyperparameters. *Journal of Machine Learning Research*, 20(50):1–24, 2019.

[21] F. Nadeem and T. Fahringer. Optimizing execution time predictions of scientific workflows applications in the grid through evolutionary programming. *Future Generation Computer Systems*, 29(4):926–935, 2013.

[22] F. Pedregosa *et al.* Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[23] G. Thompson, R. M. Rasmussen and K. Manning. Explicit forecasts of winter precipitation using an improved bulk microphysics scheme. part i: Description and sensitivity analysis. *Monthly Weather Review*, 132(2):519 – 542, 01 Feb. 2004.

[24] Y. Gu and R. Grossman. UDT: UDP-based Data Transfer for High-speed Wide Area Networks. *Computer Networks*, 51(7):1777–1799, 2007.

[25] I. Frazier. A tutorial on Bayesian optimization. Preprint 1807.02811, arXiv, 2018.

[26] I. Pouya, S. Pronk, M. Lundborg and E. Lindahl. Copernicus, a hybrid dataflow and peer-to-peer scientific computing platform for efficient large-scale ensemble sampling. *Future Generation Computer Systems*, 71:18 – 31, 2017.

[27] J. Kohl, T. Wilde and D. Bernholdt. Cumulvs: Interacting with high-performance scientific simulations, for visualization, steering and fault tolerance. *The International Journal of High Performance Computing Steering and Fault Tolerance*, 2006.

[28] J. Ossyra, A. Sedova, M. Baker and J. C. Smith. Highly interactive, steered scientific workflows on hpc systems: Optimizing design solutions. In *Int'l Conf. on High Performance Computing*. Springer International Publishing, 2019.

[29] K. Lee, N. W. Paton, R. Sakellariou and A. Fernandes. Utility functions for adaptively executing concurrent workflows. *Concurrency and Computation: Practice and Experience*, 23(6):646–666, 2011.

[30] L. C. W. Dixon. Global optima without convexity. Technical report, Numerical Optimisation Centre, 1978.

[31] G. F. Lawler and V. Limic. *Random Walk: A Modern Introduction*. Cambridge University Press, 2010.

[32] M. Amiri and L. Khanli. Survey on prediction models of applications for resource provisioning in cloud. *Journal of Network and Computer Applications*, 2017.

[33] M. B. Cohen, P. B. Gibbons, W. B. Mugridge and C. J. Colbourn. Constructing test suites for interaction testing. In *Proc. of the 25th Int'l Conf. on Software Engineering*, pages 38–48. IEEE, 2003.

[34] M. Ester, H. Kriegel, J. Sander and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. of the Second ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Dining*, pages 226–231.

[35] M. Mirza, J. Sommers, P. Barford and X. Zhu. A Machine Learning Approach to TCP Throughput Prediction. *IEEE/ACM Transactions on Networking*, 18(4):1026–1039, August 2010.

[36] M. Zhu, Q. Wu and N. S. V. Rao. Computational monitoring and steering using network-optimized visualization and ajax web server. In *IEEE International Symposium on Parallel and Distributed Processing*, 2008.

[37] M. Mohri, A. Rostamizadeh, and A. Talwalkar. *Foundations of Machine Learning*. The MIT Press, 2nd edition, 2018.

[38] N. Siegmund, A. Grebhahn, S. Apel and C. Kastner. Performance-influence models for highly configurable systems. In *Proc. of the 2015 10th Joint Meeting on Foundations of Software Engineering*, 2015.

[39] N.S.V. Rao, Q. Liu, S. Sen, D. Towlsey, G. Vardoyan, R. Kettimuthu and I. Foster. TCP Throughput Profiles Using Measurements over Dedicated Connections. In *Proc. of the 26th Int'l Symp. on High-Performance Parallel and Distributed Computing*, pages 193–204, 2017.

[40] N.S.V. Rao, S. Sen, Z. Liu, R. Kettimuthu and I. Foster. Learning concave-convex profiles of data transport over dedicated connections. In *Proc. of the 1st Int'l Conf. on Machine Learning for Netw.*, 2018.

[41] P. Ye, J. Qian, J. Chen, C. Wu, Y. Zhou, S. D. Mars, F. Yang and L. Zhang. Customized regression model for Airbnb dynamic pricing. In *Proc. of the 24th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Dining*, pages 932–940.

[42] Q. Liu, N. S. V. Rao, C. Wu, D. Yun, R. Kettimuthu and I. Foster. Measurement-Based Performance Profiles and Dynamics of UDT Over Dedicated Connections. In *Proc. of the IEEE 24th Int'l Conf. on Network Protocols*, 2016.

[43] R. Brownlie and J. Prowse and M. S. Phadke. Robust testing of at&t pmx/starmail using oats. *Bell Labs Technical Journal*, 71(3):41–47, 1992.

[44] R. Reuillon, M. Leclaire and S. Rey-Coyrehourcq. Openmole, a workflow engine specifically tailored for the distributed exploration of simulation models. *Future Gener. Comput. Syst.*, 29(8):1981–1990, October 2013.

[45] R. Souza, V. Silva, J. J. Camata, A. Coutinho, P. Valduriez and M. Mattoso. Keeping track of user steering actions in dynamic workflows. *Future Generation Computer Systems*, 99:624 – 643, 2019.

[46] S. Jain *et al* . B4: Experience with a Globally-Deployed Software Defined WAN. *ACM SIGCOMM Comput. Commun. Rev.*, 43(4):3–14, 2013.

[47] T. Li, J. Tang and J. Xu. Performance modeling and prepredict scheduling for distributed stream data processing. *IEEE Transactions on Big Data*, 2016.

[48] T. Miu and P. Missier. Predicting the execution time of workflow activities based on their input features. In *SC Companion: High Performance Computing, Networking Storage and Analysi*, 2012.

[49] K. Wang and M. Khan. Performance prediction for apache spark platform. In *IEEE 17th Int'l Conf. on High Performance Computing and Communications*, 2015.

[50] X. Meng *et al.* Mllib: Machine learning in apache spark. *The Journal of Machine Learning Research*, 2016.

[51] Y. Liu. Introduction to the special section on fast physics in climate models: Parameterization, evaluation, and observation. *Journal of Geophysical Research: Atmospheres*, 124(15):8631–8644, 2019.

[52] Z. Liu, P. Balaprakash, R. Kettimuthu and I. Foster. Explaining Wide Area Data Transfer Performance. In *Proc. of the 26th Int'l Symp. on High-Performance Parallel and Distributed Computing*, pages 167–178, 2017.

[53] Z. Liu, R. Kettimuthu, P. Balaprakash, N. S. V. Rao and I. Foster. Building a Wide-Area Data Transfer Performance Predictor: An Empirical Study. In *Proc. of the 1st Int'l Conf. on Machine Learning for Networking*, 2018.