Performance Modeling and Prediction of Big Data Workflows: An Exploratory Analysis

Wuji Liu*, Chase Q. Wu*, Qianwen Ye*, Aiqin Hou†, and Wei Shen‡
*Department of Computer Science, New Jersey Institute of Technology, Newark, NJ 07102, USA
†School of Information Science and Technology, Northwest University, Xi'an, Shaanxi 710127, China
‡School of Informatics Science and Technology, Zhejiang Sci-Tech University, Hangzhou, Zhejiang, 310018, China
Email: {wl87,qy57,chase.wu}@njit.edu, houaiqin@nwu.edu.cn, shenwei@zstu.edu.cn

***Abstract*—Many next-generation scientific and business applications feature large-scale data-intensive workflows, which require massive computing resources for execution on high-performance clusters in cloud environments. Such computing resources (e.g., VCores and virtual memory) requested through parameter setting in big data systems, if not fully utilized by workloads, are simply wasted due to the nature of exclusive access made possible by containerization. This necessitates accurate modeling and prediction of workflow performance to make an effective recommendation of appropriate parameter settings to end users. However, it is challenging to determine optimal workflow and system configurations due to the large parameter space and the interaction between various technology layers of big data systems. Towards this goal, we propose a machine learning-based feature selection method to identify influential parameters based on historical performance measurements of Spark-based computing workloads executed in big data systems with YARN. We first identify a comprehensive set of parameters across multiple layers in the big data technology stack including workflow input structure, Spark computing engine, and YARN resource management. We then conduct an in-depth exploratory analysis of their individual and coupled impact on workflow performance, and develop a performance-influence model using random forest for prediction. Experimental results show that the proposed approach identifies important features for performance modeling and achieves high accuracy in performance prediction.**

***Index Terms*—Spark; Big data workflows; machine learning; representation learning; performance modeling and prediction.**

I. Introduction

Next-generation scientific and business applications typically involve the processing and analysis of large-scale simulation, observation, or experimental datasets, which are typically structured and orchestrated as computing workflows. Such big data workflows generally require massive computing resources 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 [1]–[4]. 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.

We consider a general problem of optimizing the execution performance of big data workflows on widely used parallel computing platforms such as Hadoop [5] built on top of Spark [6] and YARN [7]. The life circle of computing workflows in such big data systems spans across a number of stages including submission with workflow input, assignment of executors in Spark, coordination by YARN for container allocation, and data access to HDFS [8] at runtime. Note that the computing resources distributed on such platforms are typically time-shared by multiple workflows submitted by different users, hence causing a high level of topological and temporal complexity in resource sharing and contention. Therefore, efficient scheduling is needed to deal with such complexity for resource reservation and allocation to improve both resource utilization and user satisfaction.

For any given workflow executed in big data systems, it is critical to determine which subset of parameters are more influential than others and what parameter values to set with respect to computation and resource efficiency [9]. Note that executors created in the Spark layer for workflow execution run in virtual containers provisioned by the YARN layer from the underlying computing resources. Such virtualized resources (e.g., VCores and virtual memory), once allocated and granted, are used exclusively by the requesting user during the execution process of queuing tasks. Due to this nature of exclusive access, any allocated and granted resource, if not fully utilized, is simply wasted. Therefore, making an effective recommendation of appropriate parameter settings is not only useful for end users to understand and optimize their strategies for workflow execution, e.g., by determining the degree of parallelism, the number of cores for executors, and the ratio of CPU to memory, but also important for the system's resource manager to wisely schedule workflow execution requests, e.g., by rejecting requests with "over-claimed" memory requirements or granting resources in appropriate amounts that can be actually utilized.

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 [10], [11]. 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.

In this work, we tackle the problem of workflow performance modeling and prediction by strategically selecting a subset of hyper parameters and setting their parameter values using machine learning. Since the importance of a parameter is reflected by the level of its impact (either direct or indirect) on the performance, our goal is to develop an accurate performance-influence model that considers the most influential independent and dependent parameters as input features. Towards this goal, we first collect extensive performance measurements from a large number of workflow executions with different parameter settings across multiple layers using different types of workflows. These performance measurements carry informative knowledge about the pattern and behavior of workflow execution under different configurations in big data systems, which shed light on performance optimization and configuration recommendation. Based on such performance measurements, we identify a comprehensive set of independent features (parameters) in multiple layers of big data systems including workflow input structure (e.g., input data size), Spark computing engine, and YARN resource management. We then conduct an in-depth qualitative and comparative exploratory analysis to investigate the impact of these 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 [12] to identify the most influential parameters, and conduct experiments to evaluate the performance of our method in identifying the best parameter setting for workflow execution.

We summarize our contributions in this work as follows.

- **Exploratory Analysis**. We conduct an in-depth analysis of a comprehensive set of execution-related parameters to qualitatively explain their impact on workflow performance in big data systems. Such analysis helps construct dependent features and also provides a valuable insight into feature selection to build an accurate performance-influence model.
- **Functional and Coupling Analysis**. We show that individual parameters may interact with each other as dependent features and collectively affect workflow performance. Such dependent features reflect complex coupled impact on the performance of big data workflows, which cannot be explicitly modeled by an analytical form. We design a functional mapping strategy to represent such intractable features as explainable variations in workflow performance.
- **Information Theory-based Feature Selection**. Since high-order coupled features carry information related to workflow performance, we propose a feature selection method based on information theory to further filter out the most influential features. This feature selection method quickly identifies critical features that achieve the maximum explainable variation.
- **Parameter Recommendation Using Performance-Influence Model**. We develop a robust performance-influence model by incorporating high-order feature construction for coupled effects and information theory-based feature selection in model training. We evaluate our performance-influence model in terms of various performance metrics based on extensive experiments to choose the best parameter setting for recommendation to end users.

The rest of the paper is organized as follows. In Section II, we conduct a brief survey of existing work in related fields. Section III describes the problem of workflow configuration in big data systems. An exploratory analysis of workflow performance is conducted in Section IV. In Section V, we approximate high-order features using functional mapping. In Section VI, we design an information theory-based feature selection method and predict workflow performance based on a performance-influence model. We evaluate the performance of our approach with discussion of its robustness in Section VII. We conclude our work and sketch a research plan in Section VIII.

II. 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.

A. Computational Steering

The main goal of computational steering [13], [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 [15], a widely used WMS in the high-performance computing (HPC) community, allows users to customize framework configuration to meet various computing needs. Fireworks [16], 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.* [17]

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.

B. 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 [18]–[20].

Miu *et al.* in [21] 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.* [22] 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.

III. 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.

IV. 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.

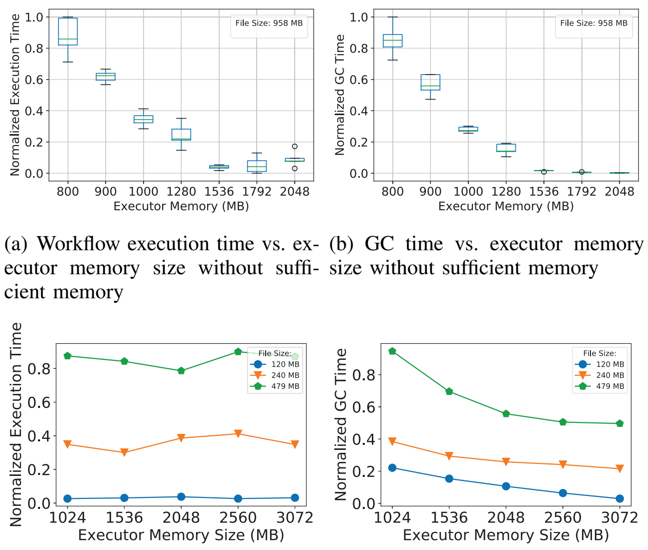
The goal of this empirical study with performance analysis is to understand and explore the individual and coupled effects of various parameters across different layers. Such an exploratory analysis motivates the use of feature selection in performance-influence model development and inspires the design 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.

A. *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 [23]. 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 Fig. 1(a) and Fig. 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 Fig. 1(c) and Fig. 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



(a) Workflow execution time vs. executor memory size without sufficient memory

(b) GC time vs. executor memory size without sufficient memory

(c) Workflow execution time vs. executor memory size when processing small files with sufficient memory

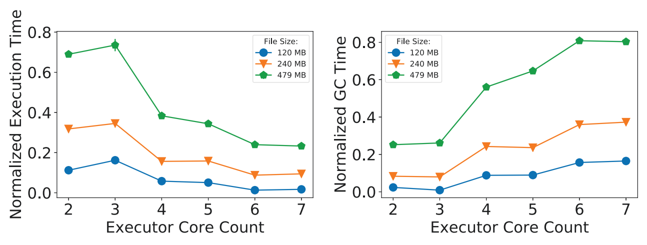
(d) GC time vs. executor memory size when processing small files with sufficient memory

Fig. 1. Illustration of the effect of executor memory size on the performance of the linear regression 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 Fig. 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 Fig. 1(d).

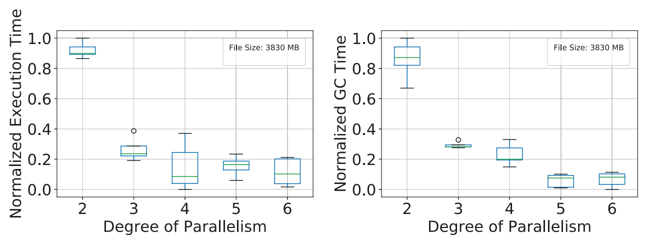
B. *Executor Core Count*

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 pattern is qualitatively consistent. As the core count increases, the workflow execution time decreases as shown in Fig. 2(a), while the GC time increases as shown in Fig. 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.



(a) Workflow execution time vs. executor core count with different file sizes (b) GC time vs. executor core count with different file sizes

Fig. 2. Illustration of the effect of executor core count on the performance of the linear regression workflow.



(a) Workflow execution time vs. parallelism (b) GC time vs. parallelism

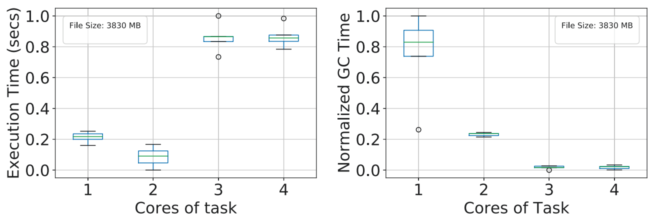
Fig. 3. Illustration of effects of parallelism on the performance of the linear regression workflow.

C. 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 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. Fig. 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 Fig. 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 Fig. 3(b), which is consistent with the total execution time in Fig. 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.

D. 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 Fig. 4(a) and the decrease in the GC time as shown in Fig. 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

Fig. 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.

V. 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 [24]. 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

TABLE I
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

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.

A. *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 I.

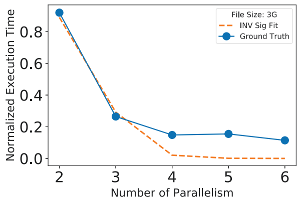


Fig. 5. Performance profile fitting with an inverse sigmoid-based regression function in response to the number of parallelism.

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

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

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

where α and p_0 are hyper parameters. As shown in Fig. 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.

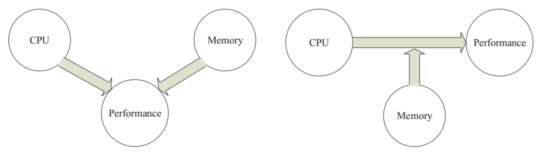
B. *Coupled Features*

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 Fig. 1. Such interactive impact is also termed as k-interact [25]. As illustrated in Fig. 6, individual effect (such as Memory or CPU in Fig. 6(a)) is generally observable and measurable, while coupled effect (such as the one between Memory and CPU in Fig. 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.

VI. 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.



(a) Independent influence (b) Interactive influence

Fig. 6. Independent and interactive influence of Memory and CPU on workflow performance.

A. 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.

1) 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 [25]. We further illustrate this property in Fig. 7, where we use the three-way interaction between CPU, memory, and performance as an example. 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 Sec. V-B, 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 [12]:

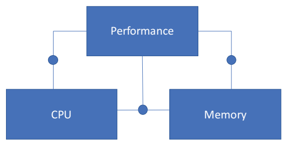


Fig. 7. Three-way positive interaction between CPU, memory, and performance, where blue dots represent mutual information.

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

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 Sec. IV, 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:

$$\operatorname*{argmax}_{\mathcal{S}} I(\mathcal{S} : y), \;\; s.t. \; C(S) < \delta, \tag{2}$$

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 [26] 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 Eq. 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 [26].

Similar to the work in [26], 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 Alg. 1.

Algorithm 1 Greedy Feature Selection

```
Input: candidate feature set $\mathcal{P}$, mutual information threshold $\tau$
Output: selected feature set $X$
1: $X = \phi$;
2: for each $t_i$ in $\mathcal{P}$ do
3:     $t_i = \arg\max_{t_i} I(t_i : \mathcal{Y})$;
4:     if $\arg\max_{t_i} I(t_i : \mathcal{Y}) > \tau$ then
5:         $X = X \cup t_i$;
6: return $X$;
```

B. 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 [27]: 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 [28] to solve the following optimization problem:

$$\underset{\mathcal{M}^*, \theta_m^*}{\operatorname{argmin}} \mathcal{L}(\mathcal{M}(X, \theta_m), y), \tag{3}$$

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 Eq. 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.

VII. 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.

A. 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 Sec. IV, and the second workflow employs random forest. Both workflows feature a pipeline structure that consists of three computing modules as shown in Fig. 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 [29]. 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.

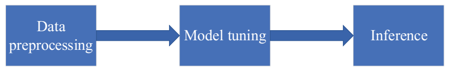


Fig. 8. The pipeline structure of the test workflows for regression.

B. Configuration Space Sampling

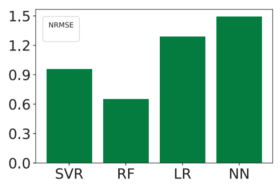
We deploy and run these workflows on the same local Hadoop cluster with Spark and YARN as in the empirical study conducted in Sec. IV. Although Spark and YARN provide a large configuration space, most of the settings are irrelevant to the execution time, e.g., port number, log location, etc. Hence, instead of investigating the entire configuration space, we focus on tunable parameters related to executors and observable runtime features as shown in Tab. I. 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.

C. Performance Prediction Results

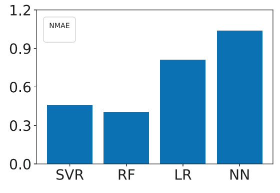
We implement a performance-influence model in Python for workflow execution prediction with different regression algorithms using the `scikit-learn` library [30]. The performance of this prediction model is evaluated in two steps: i) we compare the prediction results of various regression algorithms 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.

1) Performance Comparison of Regression Models

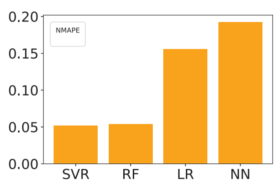
We first split the performance measurement data collected from the execution of two test workflows into two parts for training and testing, respectively, and then perform 10-fold cross validation [28] using the training data to fine tune four representative regression models, i.e., LR, SVR, RFR, and MLP. We measure the prediction accuracy of these models 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 Fig. 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



(a) Normalized Root Mean Square Error (NRMSE)



(b) Normalized Mean Absolute Error (NMAE)



(c) Normalized Mean Absolute Percetage Error (NMAPE)

Fig. 9. Performance comparison of various models in terms of different metrics.

TABLE II
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

baseline model for further investigation with feature selection for performance improvement.

2) *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 II. 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 the corresponding prediction performance. As shown in Fig. 10, the prediction accuracy improves as more features are considered and the top four are considered critical features.

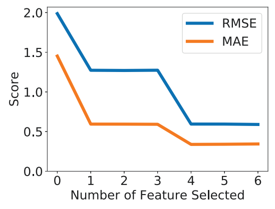


Fig. 10. The performance of the RFR-based predictor with an increasing number of selected features.

VIII. CONCLUSION

In this work, we studied the problem of workflow performance modeling and prediction in big data systems. We conducted an exploratory analysis of the impact of both individual and interacting parameters on workflow performance, and proposed an information-theoretic method to quantify such influence for feature selection. We developed a regression-based performance-influence model that incorporates both individual and constructed features for workflow prediction. Experimental results showed that our predictor achieves high accuracy with respect to actual execution time. We plan to use new sampling methods to accelerate the exploration of the configuration space and apply stochastic algorithms to identify the optimal configuration for workflow execution.

ACKNOWLEDGMENTS

This research is sponsored by National Science Foundation under Grant No. CNS-1828123 with New Jersey Institute of Technology.

REFERENCES

[1] 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.
[2] D. Abramson textitet al., “Nimrod/k: Towards massively parallel dynamic grid workflows,” in *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, 2008.
[3] I. Pouya *et al.*, “Copernicus, a hybrid dataflow and peer-to-peer scientific computing platform for efficient large-scale ensemble sampling,” *Future Generation Computer Systems*, vol. 71, pp. 18 – 31, 2017.
[4] R. Reuillon *et al.*, “Openmole, a workflow engine specifically tailored for the distributed exploration of simulation models,” *Future Gener. Comput. Syst.*, vol. 29, no. 8, p. 19811990, Oct. 2013.
[5] Apache, 2016, Hadoop. http://hadoop.apache.org.
[6] ——, 2016, Spark. http://spark.apache.org.
[7] ——, 2020, YARN. https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html.
[8] ——, 2020, HDFS. https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html.
[9] M. Li *et al.*, “Sparkbench: a spark benchmarking suite characterizing large-scale in-memory data analytics,” *Cluster Computing*, vol. 20, no. 3, pp. 2575–2589, Sep 2017.
[10] J. Ossyra *et al.*, “Highly interactive, steered scientific workflows on hpc systems: Optimizing design solutions,” in *International Conference on High Performance Computing*. Springer International Publishing, 2019.
[11] R. Souza *et al.*, “Keeping track of user steering actions in dynamic workflows,” *Future Generation Computer Systems*, vol. 99, pp. 624 – 643, 2019.
[12] T. Cover and J. Thomas, *Elements of Information Theory*. John Wiley, 2012.
[13] M. Zhu *et al.*, “Computational monitoring and steering using network-optimized visualization and ajax web server,” in *IEEE International Symposium on Parallel and Distributed Processing*, 2008.
[14] C. Wu *et al.*, “System design and algorithmic development for computational steering in distributed environments,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 4, pp. 438–451, 2010.
[15] E. DeelMan *et al.*, “Pegasus: A framework for mapping complex scientific workflows onto distributed systems,” *Scientific Programming*, 2015.
[16] A. Jain *et al.*, “Fireworks: a dynamic workflow system designed for high-throughput applications,” *Concurrency and Computation: Practice and Experience*, vol. 27, no. 17, pp. 5037–5059, 2015.
[17] K, Lee *et al.*, “Utility functions for adaptively executing concurrent workflows,” *Concurrency and Computation: Practice and Experience*, vol. 23, no. 6, pp. 646–666, 2011.
[18] M. Amiri and L. Khanli, “Survey on prediction models of applications for resource provisioning in cloud,” *Journal of Network and Computer Applications*, 2017.
[19] F. Nadeem and T. Fahringer, “Optimizing execution time predictions of scientific workflows applications in the grid through evolutionary programming,” *Future Generation Computer Systems*, vol. 29, no. 4, pp. 926–935, 2013.

[20] 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.
[21] 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.
[22] T. Li, J. Tang and J. Xu, “Performance modeling and prepredict scheduling for distributed stream data processing,” *IEEE Transactions on Big Data*, 2016.
[23] K. Wang and M. Khan, “Performance prediction for apache spark platform,” in *IEEE 17th International Conference on High Performance Computing and Communications*, 2015.
[24] N. Siegmund, A. Grebhahn and S. Apel and C. Kastner, “Performance-influence models for highly configurable systems,” in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, 2015.
[25] A. Jakulin and B. Ivan, “Quantifying and visualizing attribute interactions:an approach based on entropy,” *arXiv:cs.AI/0308002*, 2004.
[26] A. Krause, D. Golovin, “Submodular function maximization,” in *Tractability*, 2014.
[27] M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of Machine Learning*, 2nd ed. The MIT Press, 2018.
[28] C. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.
[29] X. Meng *et al.*, “Mllib: Machine learning in apache spark,” *The Journal of Machine Learning Research*, 2016.
[30] F. Pedregosa *et al.*, “Scikit-learn: Machine learning in python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.