Cost-aware Resource Recommendation for DAG-based Big Data Workflows: An Apache Spark Case Study

Mohammad-Mohsen Aseman-Manzar, Soroush Karimian-Aliabadi, Reza Entezari-Maleki, Bernhard Egger, and Ali Movaghar

Abstract—The era of personal resources being sufficient for enterprise big data computations has passed. As computations are executed in the cloud, small policy changes of cloud operators may cause considerable changes in operational costs. Carefully choosing the amount of resources for a given application is thus of great importance. This, however, requires a priori knowledge of the application's performance under different configurations. Creating a performance prediction model needs to account for the heterogeneity of resources and the diversity in application workflows. Previous approaches for heterogeneous environments consider a black-box representation of the application which results in single-purpose models. This paper addresses the problem with two gray-box prediction models using linear programming (LP) and mixed-integer linear programming (MILP). Given a set of available resources, the models consider Apache Spark applications and their Directed Acyclic Graph (DAG) of workflow running on top of a Hadoop-YARN cluster. We then propose a configuration recommendation algorithm to optimize the cost-performance trade-offs when renting machine instances. The accuracy of the proposed models is evaluated with real-world executions of several representative applications on the Wikipedia dataset and the TPC-DS benchmark. The average error of only 3.28% for the proposed prediction models demonstrates the practicality of the proposed approach in handling cost-performance trade-offs.

Index Terms—Apache Spark, Big Data Frameworks, Performance Evaluation, Resource Recommendation, Cost Model.

1 Introduction

Data is flooding businesses on an ever-increasing scale and demands storing, cleansing, formatting, classification, and mining. These processes are affected by the unusual volume of the data and need special tools, frameworks, and computing paradigms. Despite the challenges, the opportunity for extracting valuable information from the Big Data is encouraging businesses to tolerate the burden for the hope of prominence and is keeping them searching for cost-effective high-performance Big Data processing clusters. In 2020, Big Data market was estimated to be worth $130 billion and is predicted to reach $234 billion by 2026 [1]. Unsurprisingly, a noticeable portion is related to software and tools related to Big Data analytics. In 2021, $82 billion worth of spending was dedicated to reporting, analysis, relational data warehouses, and non-relational analytic data stores [2].

The key to efficient Big Data processing is a computing paradigm that increases parallelism and was first noticed in the MapReduce (MR) scheme. While the realization of the MapReduce paradigm in the Hadoop framework became a well-known and widely used solution, extensions and improvements were introduced to boost its performance and expressiveness further. One of these extensions is the increase in the multiplicity of Map and Reduce stages and the flexibility in their interconnection represented as a Directed Acyclic Graph (DAG). DAG-based workflows, supported in well-implemented frameworks such as Apache Tez [3] and Apache Spark [4], enable complex applications and queries to be programmatically transformed to a graph of stages, each running hundreds of tasks in parallel. A considerable performance improvement in such frameworks is in-memory computations of Apache Spark, where stages refer to the intermediate data without reloading it from the disk. These advantages and other features, have promoted Apache Spark as a popular choice.

Public or private clouds are hosting most of the processes related to Big Data analytics, and data scientists are paying for these resources proportional to the time the analysis takes. In order to reduce costs, it is necessary first to have an accurate prediction of the job runtime for each cluster setup, and second, find an optimal configuration of resources using the prediction. However, there are two main challenges in this way. The heterogeneity of resources and the diversity in the application workflow. Usually, cloud providers present different types of machines with different prices, and according to the experiments, this diversity reflects in different stages of the application too.

On the other hand, the model proposed for runtime prediction should be practical no matter what structure the application is following. The workflow of Big Data applications varies from simple two-stage DAGs to those

- *M. M. Aseman-Manzar, S. Karimian-Aliabadi, and A. Movaghar are with the Department of Computer Engineering, Sharif University of Technology, Tehran, Iran.*
  E-mail: {asemanmanzar,skarimian}@ce.sharif.edu, movaghar@sharif.edu
- *R. Entezari-Maleki is with the School of Computer Engineering, Iran University of Science and Technology, Tehran, Iran, and also with the School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran.*
  E-mail: entezari@iust.ac.ir
- *B. Egger is with the Department of Computer Science and Engineering, Seoul National University, Seoul, Korea.*
  E-mail: bernhard@csap.snu.ac.kr (Corresponding author)

with tens of stages and complex interconnections. While resource recommendation is extensively studied in different environments, from grids to clouds, none have considered the application execution graph and the heterogeneity of resources at the same time.

Realizing the exact Apache Spark's task placement and scheduling for a given application is not possible before the actual execution. On the other hand, mainstream Big Data frameworks including Apache Spark usually place the tasks and allocate resources in a way that minimizes the runtime. Therefore, using an optimization method, the optimized scheduling scheme and, subsequently, the runtime of an application can be predicted. The result of the optimizer is claimed here to be close to what is recorded in real-world executions as both try to minimize the execution time. However, the optimization not only depends on the characteristics of the infrastructure but also on that of the application, i.e., the number of tasks in each stage, the mean execution time of each task, etc. Since these values vary from one application to another, it is necessary to estimate them for each application prior to the actual run. This phase is called profiling and usually requires running the application on a few small samples rather than the original dataset.

Considering the challenges of performance prediction described earlier and assuming an optimized task placement, we propose an Apache Spark application runtime prediction approach using Linear Programming (LP) that considers both the DAG and the heterogeneity of instances. To the best of our knowledge, this is the first gray-box proposition that allows studying workflows with complex DAGs while running on top of a heterogeneous infrastructure. The state-of-the-art methods in this field either consider black-box models with no information of the underlying DAG of workflow or asuume a homogeneous infrastructure. The proposed LP model uses micro-benchmarking, i.e., collecting profile information from sample runs of a scaled-down version of the application, to assign values to the unknown parameters of the system. We validate the accuracy of the predictor against the measurements performed during a real-world execution of the benchmarks. In addition, and to demonstrate the practicality of the proposed model, an instance selection algorithm is proposed that suggests the optimal set of instances to rent from the cloud provider to minimize cost while maintaining performance.

The main contributions of this paper, thus, are as follows:

1) Formulating the problem of predicting Spark application runtime as an LP model while considering the heterogeneity of resources and the DAG of the application. A Mixed-Integer Linear Programming (MILP) model is also presented, which optimizes the same objective function but with different levels of detail.
2) Validating the proposed prediction approach based on the real-world experiments conducted on the TPC-DS benchmark and Wikipedia dataset.
3) Designing a resource selection algorithm based on the prediction approach that searches the state space of possible configurations and finds the optimal one regarding the cost and performance.

The remaining parts of this paper are organized as follows. In Section 2, related proposals available in the literature are introduced. Section 3 is dedicated to the description of the features of the application frameworks. Sections 5 and 4 present our runtime prediction approach and the proposed resource recommendation algorithm, respectively. Results obtained by the proposed models and their validation against the real-world systems are reported in Section 6. Finally, Section 7 concludes the paper and provides directions for future work.

2 Related Work

One of the aspects of every distributed computing system that usually gains attention in the literature is resource allocation and scheduling. This is a crucial part of such systems that affects performance, energy consumption, cost, user satisfaction, etc [5], [6]. Big data frameworks, mainly Apache Spark, are not an exception, and a considerable number of published work are dedicated to exploring different ways of reducing costs while maintaining an acceptable level of service performance. Their concern is primarily minimizing the costs of the provider, and few works have discussed the satisfaction of the users while users are paying for the hours of resource usage. Since methods and goals are often similar, we closely study related work and classify main approaches.

Learning from previous executions to predict the future runs is offered by significant number of research work [7], [8], [9], [10]. The authors of [11] employ micro-benchmarks to empower a learning-based performance model. Apart from predicting the execution time of the whole job, in [12] a similar idea was used to predict the execution time of tasks which is helpful in recognizing straggler ones. A performance model for homogeneous Apache Spark cluster has been proposed in [13] which predicts execution time using historical data and chooses a machine learning algorithm among a list of regression methods according to their historical errors for each stage of the job. The aggregated execution time of the whole job was predicted by another learned model, which takes individual stage prediction as input. Authors in [14] have collected over 40000 traces of past jobs in order to test the applicability of different methods from neural networks to regression tree. Micro-benchmarking, itself, might be inefficient, therefore, [7] has discussed techniques to decrease the number of data points without significant loss of accuracy. Taking another step forward, some articles used achievements in performance modeling to tune the cluster.

Decisions related to resource management are extensively coupled with performance forecasts. As [15] has proposed, investigation of historical data enables novel scheduling approaches. Even an offer to lose the accuracy of the performance model in favor of faster prediction is acceptable if the selectivity of the model is not affected [16]. Some have focused on other measures such as [17] which optimized parallelism parameters, or [18] which predicted the availability of the resources. If a configuration is proved to perform well with a specific application, most likely, it will be a good choice for similar applications as well. This idea was explored in [19] with gradually enriching the database of top-rated configurations. While most of the proposed solutions can be employed at runtime, some have

mainly addressed evolutionary methods to tune the number of VMs for MR applications [20] or to improve utilization [21].

A related research field concerns cost optimization of database queries. One common approach focuses on optimizing the cost of data transfers between nodes of distributed databases such as [22] that applies Integer Linear Programming (ILP) to model and optimize the transmission cost. The authors of [23] employ ILP to find the optimal solution to the index selection problem and use a branch-and-bound technique to identify sub-optimal solutions in case the ILP formulation takes too much time to solve. Recently, several works exploited techniques based on machine learning like RNN, CNN, and Mixture Model for cost-based query estimation and optimization [24], [25], and [26]. The focus of our work is not limited to database queries and covers the optimization of general DAG workflows.

Apart from machine learning, analytical models, such as, Petri nets, queuing theory, process algebra, and computational geometry, are proved helpful in guiding tuning decisions [27], [28], [29], [30]. Based on the stochastic derivation of Petri nets, [31] has proposed a performance model for Apache Spark applications with acceptable accuracy and compositional modeling technique. The same authors of [31] have applied lumping technique to cope with the growing state-space issue in MR model [32] and fixed-point iteration technique for the same issue in the Spark model [33]. The use of Fluid Petri-nets is explored in [34] to resolve state-space explosion when evaluating the performance of Hadoop or Spark applications in a homogeneous environment. They have assumed that the DAG can be replaced with its critical path, while no specific method is given for such transformation. The mean-field analysis proposed in [35] is claimed to be practical for estimating the completion time of MR applications. However, the approach can not be easily applied to DAGs.

Some researchers have suggested to speed up the execution of a Spark job by rewriting the code in different ways and according to specific rules [36]. The different versions are then evaluated by cost/weight functions to find the best code, while the cluster setup remains unchanged. In [37], the authors have used Evolutionary Markov Chain Monte Carlo method to find the best assignment of the parameters. Again, the physical configuration of machines is fixed, and only the framework parameters are tuned. In [38], a closed-form performance model for Spark was used to compute the required resources taking into account the deadline of a job. Their model considers that the estimation for the completion time of the job multiplied by the number of parallel processors is the sum of all RDDs runtime. This assumption is not realistic since, often, the execution time is not evenly distributed. Finally, [39] exploited the use of spot-instances while predicting the effect beforehand. The predictor, however, ignores the waiting time when multiple stages are synchronized.

Another crucial part of the evaluation process is benchmarking. Many researchers confirm that experimental results obtained based on synthetic datasets are commonly reliable for this purpose, specifically in a Big Data environment [40], [10], [16], [17]. Finding real data which truly represents the characteristics of Big Data is a major obstacle, while mature data generators, widely used and available today, have developed to tackle this problem. Among them, the Transaction Processing Performance Council (TPC) benchmarks provide datasets rich in desirable Big Data characteristics and have attracted numerous users [41], [8], [23], [31], [32], [33], [34].

3 DAG-based Workflows and Apache Spark

In MapReduce (MR) paradigm, tasks are grouped into meaningful stages, namely, Map, Shuffle, and Reduce. The flow of data can also be organized through processing stages. Directed Acyclic Graph (DAG) has been extensively used in scientific computational workflow modeling, especially large-scale computing-intensive or data-intensive applications. Workflow modeling and MapReduce paradigm meet in Apache Tez framework, where multiple MR jobs are arranged in a DAG. Tez runs on top of the Hadoop YARN [42] cluster and manages the intermediate output of each MR job. However, a mature combination of DAG-based workflow and task encapsulation and parallelism can be found in Apache Spark framework.

3.1 Apache Spark

When it comes to processing Big Data, Apache Spark is usually the most proper option with its different extensions for SQL queries, machine learning, schema manipulation, etc. More importantly, Apache Spark is fast due to its particular memory usage scheme and it is capable of connecting to other popular frameworks, namely, Hadoop and YARN. Therefore, users can enjoy the performance of Spark along with sophisticated YARN resource management and Hadoop Distributed File System (HDFS) storage methods. The data-flow inside each job is represented by a DAG in which vertices stand for stages and edges represent the flow of the data from one stage to another. Stages in Spark are meaningful encapsulations of tasks that can run in parallel, knowing that each task is the atomic unit of a process.

The orchestration of concepts described above, however, is implemented behind the scene. SparkContext is the main object of the application, which holds the code as well as the processes that the user has defined. It is wrapped by the Driver, which is responsible for collaborating with other nodes and is specific to a single application. The Driver needs worker nodes to do the computations and asks them via the cluster manager. Cluster manager here is assumed to be the YARN but is not limited to. A worker node contains an executer that performs individual tasks. The number of tasks that can run in parallel inside an executer is equivalent to the number of cores configured for the executer. Configuration of each executer, namely its memory and cores, are user-defined.

Some other details of the application are embedded inside the SparkContext. It will create a job whenever it encounters an action inside the user program and forms the execution DAG afterward. According to the explanation mentioned above, this DAG holds stages as well as the tasks residing in them. The SparkContex rationale behind the DAG formation is how the dataset is transformed from one state to another and relies on an abstraction named

TABLE 1
Fixed input parameters of the MILP and LP models

Parameter	Definition	Used in MILP	Used in LP	Input Source
E	Set of executors	✓	✓	Cluster configuration
E_i	Set of executors with the same type of executor i	✓	✓	
S	Set of stages in the application	✓	✓	Single-node analysis phase
P_j	Direct parents of stage j in the DAG	✓	✓	
R_j	All parents of stage j in the DAG	-	✓	
t_{ij}	Sum of the tasks execution time of stage j in executors of E_i	✓	✓	
tm_{ij}	Average of the maximum execution time of tasks in executors of E_i for stage j	✓	✓	
tr_{ij}	$tr_{ij} = t_{ij} - tm_{ij} \times \lvert E_i \rvert$	✓	✓	
L	Leaf stages of the DAG (stages without any children)	-	✓	
B	An upper bound for stage completion time (Eq. 2)	✓	-	

Resilient Distributed Dataset (RDD), which is one of the key characteristics of the Spark framework. The first RDD is the initial dataset and lazily transforms to other RDDs on-demand, in the sense that all the mappings will be applied at once and right when an action is triggered. Tracking and saving changes on RDDs have enabled Spark to meet the fault tolerance through the lineage. If an RDD is lost eventually, this lineage helps Spark build that RDD again from scratch.

3.2 Heterogeneity impact

Although all executors are similar in configuration, in a heterogeneous cluster, executors running on different machines show different characteristics. This diversity is exploited in [7] for matrix multiply and least squares solver applications running on different clusters. Clusters are similar in the aggregated number of cores and amount of memory but different in the type of deployed Amazon EC2 servers. Results show that although the number of executors is the same in each configuration, the difference in the type of the server has led to a considerable fluctuation in execution time. Surprisingly the largest server is not always the best choice regarding the execution time. Assuming that each stage in the DAG is treated like an application with sub-tasks, this evidence is the basis for our claim that different stages demonstrate different performances on a single server. Therefore, the selection of the server type and multiplicity is a critical decision concerning optimization purposes.

In order to come up with an optimized suggestion, the behavior of each stage on each server type should be realized first. Since this is dependent on the nature of the application, which is not always clear, the best approach is to employ micro-benchmarking in which the application is tested against different server types on small subsets of the original dataset. Results of the micro-benchmarking phase are not sufficient to directly decide on the server types because even with a fixed set of resources, different allocation and scheduling schemes, especially among parallel stages, lead to different runtime. In order to address this complexity, two performance prediction approaches are presented next, along with the detail of the micro-benchmarking step.

4 Performance Prediction

In this section, we propose an approach to predict the runtime of a given Big Data application on a heterogeneous cluster containing an arbitrary number of nodes with different computational capabilities. We assume that the primary goal of the scheduler in the mainstream frameworks, such as Apache Spark or YARN, is the minimization of the application runtime. Hence, we employ optimization formalisms, namely Linear Programming (LP) and Mixed Integer Linear Programming (MILP), to find the best placement of tasks and, thereby, the minimum runtime. By doing so, we predict the runtime of the application running on these frameworks.

The two proposed formulations are discussed in Sections 4.2 and 4.3. These formulations use input parameters that describe the nature of the application, data, and computational resources. The assignment of these input parameters, listed in Table 1, is perfomed during a micro-benchmarking phase called single-node analysis and discussed in the following section.

4.1 Single-node Analysis Phase

In order to come up with a precise model, some input parameters are defined and used in the LP and MILP models. However, actualization of these parameters completely depends on the application, as they exhibit different characteristics. Therefore, it is necessary to have an estimate of the behavior of a particular application running on the original dataset. Our approach uses a single-node analysis phase for this purpose, during which micro-benchmarks with different data sizes run on a single instance per each instance type. For example, for three samples of 2%, 6%, and 10% of the original data and an infrastructure with four different instance types, 12 micro-benchmarks are executed. This idea of running the application with different data sizes enables us to obtain the application's profile with respect to each instance type; that is, how performance metrics change with an increase in the input data size on a particular instance type.

During the execution of these so-called micro-benchmarks, we carry out measurements to compute the input parameters of the proposed formulations. These measured parameters are

1) The DAG of the application's workflow including stages and their dependencies.
2) The number of executors of each instance type.
3) The execution time of each stage (sum of tasks execution times without considering the waiting times caused by the dependency graph) for each executor.
4) The average of maximum tasks execution time in each executor for each stage.

The next step is to learn from the measurements for the parameters mentioned earlier, for each micro execution and to estimate the same parameters for the actual data size. A similar technique is implemented in [7] and is employed here to interpolate values of parameters 3 and 4. The main advantage of the method proposed in [7] is using both executor count and sample size for training. It is worth mentioning that parameters 1 and 2 are assumed to be constant and can easily be achieved from the log of an experiment on a sample input.

Finally, input parameters of the LP and MILP model can be assigned with estimations performed during the single-node analysis phase. We propose two models for the performance prediction, each capable of estimating the execution time, but with different levels of detail. These two models are described in Sections 4.2 and 4.3. The main advantage of our approach is that the proposed LP and MILP models can predict the execution time of the application on a large data set with few micro-benchmarks on small samples without the need to rent the entire cluster. Using these prediction models, in Section 5, we propose an algorithm to search the space of cluster configurations and find the best cluster configuration, again without renting the cluster.

4.2 Mixed-integer Linear Programming Model

Our first execution time prediction model uses MILP. MILP problems are mathematical optimizations like LP problems, but they have more modeling power than LP problems because they can restrict some of the variables from real numbers to be integer or binary.

In the optimization formulation figured in Eq. 1, universal quantifiers are used for the sake of brevity and can be simply replaced with multiple flat constraints. The problem presented in this formulation is designed to find the best task scheduling assignment for the Big Data scheduler by minimizing the application runtime as stated in the problem objective 1a. Our approach seeks to find the best starting and ending times of each stage in each executor, with the help of some knowledge from the single-node analysis phase, including the performance of each stage in each machine. The input parameters and variables of this model are summarized in Tables 1 and 2, respectively.

Constraint 1b will be replaced by $|E| \times |S|$ constraints that each one restricts the end time of stage j in executor i to be equal to its start time plus its duration. In constraint 1b, the duration of stage j in executor i is calculated by $tm_{ij} + (tr_{ij} \times p_{ij})$, where p_{ij} is used to model the share of the workload assigned to executor i, and tm_{ij} is an approximation for the executors' deserialization time. The separation between this specific task and others running on an executor is necessary to increase the model's accuracy because, according to experiments, the deserialization time imposed on the initial tasks of an executor results in a considerable deviation from the average task runtime. The timeline of the application execution is demonstrated in Fig. 1 for a representative example, which shows the notable difference in runtime of the first task on each executor and the rest.

$$minimize\ T \tag{1a}$$

$$subject\ to$$

$$\forall i \in E\ \forall j \in S \tag{1b}$$
$$e_{ij} = s_{ij} + \underbrace{tm_{ij} + (tr_{ij} \times p_{ij})}_{duration\ of\ stage_j\ in\ executor_i}$$

$$\forall j \in S \tag{1c}$$
$$1 = \sum_{i \in E} p_{ij}$$

$$\forall j \in S \tag{1d}$$
$$T \geqslant e_j$$

$$\forall i \in E\ \forall j \in S \tag{1e}$$
$$s_j \leqslant s_{ij}$$

$$\forall i \in E\ \forall j \in S \tag{1f}$$
$$e_j \geqslant e_{ij}$$

$$\forall j \in S\ \forall k \in P_j \tag{1g}$$
$$s_j \geqslant e_k$$

$$\forall i \in E\ \forall j \in S\ \forall k \in S \tag{1h}$$
$$max_{ijk} - min_{ijk} \geqslant (e_{ij} - s_{ij}) + (e_{ik} - s_{ik})$$

$$\forall i \in E\ \forall j \in S\ \forall k \in S \tag{1i}$$
$$max_{ijk} \geqslant e_{ij}$$

$$\forall i \in E\ \forall j \in S\ \forall k \in S \tag{1j}$$
$$max_{ijk} \geqslant e_{ik}$$

$$\forall i \in E\ \forall j \in S\ \forall k \in S \tag{1k}$$
$$max_{ijk} \leqslant e_{ij} + (B \times bmax_{ijk})$$

$$\forall i \in E\ \forall j \in S\ \forall k \in S \tag{1l}$$
$$max_{ijk} \leqslant e_{ik} + (B \times (1 - bmax_{ijk}))$$

$$\forall i \in E\ \forall j \in S\ \forall k \in S \tag{1m}$$
$$min_{ijk} \leqslant s_{ij}$$

$$\forall i \in E\ \forall j \in S\ \forall k \in S \tag{1n}$$
$$min_{ijk} \leqslant s_{ik}$$

$$\forall i \in E\ \forall j \in S\ \forall k \in S \tag{1o}$$
$$min_{ijk} \geqslant s_{ij} - (B \times bmin_{ijk})$$

$$\forall i \in E\ \forall j \in S\ \forall k \in S \tag{1p}$$
$$min_{ijk} \geqslant s_{ik} - (B \times (1 - bmin_{ijk}))$$

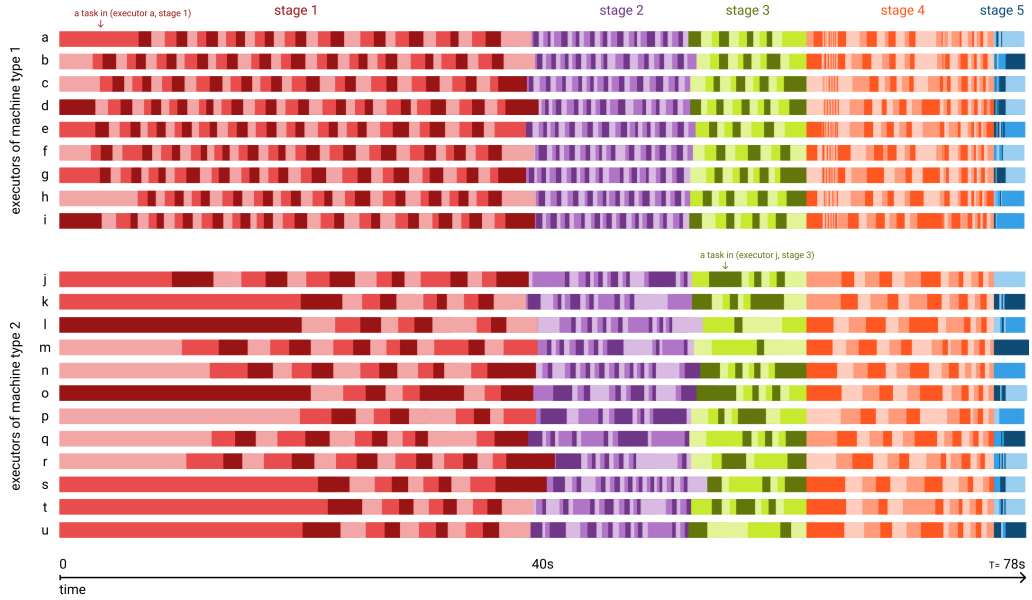
$$\forall i \in E\ \forall j \in S \tag{1q}$$
$$B \geqslant e_{ij}$$

$$All\ Variables \geqslant 0 \tag{1r}$$

Apart from the constraints that are modeling the execution times, we need some others to secure the correct representation of the application. First of all, to make sure that stage j will get all of the executions it needs, constraint 1c is included. Afterward, the workflow dependencies among stages of the DAG are represented by constraints Constraints 1e, 1f, and 1g. Constraint 1d is used alongside

TABLE 2
Variables of the MILP and LP models

Variable	Meaning	Used in MILP	Used in LP
s_j	Stage j start time (start time of the first task of stage j)	✓	✓
e_j	Stage j end time (end time of the last task of stage j)	✓	-
s_{ij}	Start time of the first task of stage j in executor i	✓	-
e_{ij}	End time of the last task of stage j in executor i	✓	-
p_{ij}	Portion of the workload of stage j assigned to executor i	✓	✓
min_{ijk}	$min_{ijk} = min(s_{ij}, s_{ik})$	✓	-
max_{ijk}	$max_{ijk} = max(e_{ij}, e_{ik})$	✓	-
$bmin_{ijk}$	A binary variable to calculate min_{ijk}	✓	-
$bmax_{ijk}$	A binary variable to calculate max_{ijk}	✓	-
d_j	Duration between the first and the last task of stage j, including the duration of these tasks	-	✓
T	Completion time of the application	✓	✓



$S = \{1, 2, 3, 4, 5\}, L = \{5\}$
$P_1 = \{\}, P_2 = \{\}, P_3 = \{\}, P_4 = \{1, 2, 3\}, P_5 = \{4\}$
$R_1 = \{\}, R_2 = \{\}, R_3 = \{\}, R_4 = \{1, 2, 3\}, R_5 = \{1, 2, 3, 4\}$
$E = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u\}$
$E_a = E_b = E_c = E_d = E_e = E_f = E_g = E_h = E_i = \{a, b, c, d, e, f, g, h, i\}$
$E_j = E_k = E_l = E_m = E_n = E_o = E_p = E_q = E_r = E_s = E_t = E_u = \{j, k, l, m, n, o, p, q, r, s, t, u\}$

Fig. 1. Timeline plot of query 33 tasks of the TPC-DS benchmark running on a cluster with two machine types, each with 9 and 12 executors. Horizontal bars are individual executors, while vertical color themes represent stages, and different shades of a color are used just for a visual distinction of tasks. The DAG of query 33 is provided in Fig. 3. Some of the input parameters defined in Table 1 are calculated from the DAG and cluster configuration as well for better understanding.

the minimization objective 1a for valid estimation of the runtime.

Constraint 1h is used to avoid the overlap of two stages in one executor. Without this constraint, nothing will stop parallel stages from being scheduled in one executor on a same duration of time. Herein, we add an assumption that each executor has a single core and can only run one task at a time. However, this assumption is not restricting the generality of the problem because a single executor with multiple cores can be modeled by multiple single-core executors.

Constraints 1i to 1l and 1m to 1p are used to define the max_{ijk} and min_{ijk} variables, respectively. In these constraints, parameter B is a fixed input and is an upper bound for each of the variables s_{ij} and e_{ij}, as declared in constraint 1q.

A possible proposal for computing B is given in Eq. 2, while other definitions are applicable too.

$$B = \sum_{j \in S} \max_{i \in E} t_{ij} \tag{2}$$

In constraints 1i to 1l, the binary variable $bmax_{ijk}$ is defined to enable both outcomes of the assignment $max_{ijk} = max(e_{ij}, e_{ik})$. Constraints 1i to 1l reduce to equation system 3 if $bmax_{ijk} = 0$, and consequently, $max_{ijk} = e_{ij}$ while $bmax_{ijk} = 1$ leads to the equation system 4 and $max_{ijk} = e_{ik}$ assignment. So the MILP problem can set the value of max_{ijk} to $max(e_{ij}, e_{ik})$ by assigning the proper value for $bmax_{ijk}$. The same explanation holds for $min_{ijk} = min(s_{ij}, s_{ik})$.

$$\begin{aligned} & max_{ijk} \geqslant e_{ij} \\ & max_{ijk} \geqslant e_{ik} \\ & max_{ijk} \leqslant e_{ij} \\ & max_{ijk} \leqslant e_{ik} + B \end{aligned} \tag{3}$$

$$\begin{aligned} & max_{ijk} \geqslant e_{ij} \\ & max_{ijk} \geqslant e_{ik} \\ & max_{ijk} \leqslant e_{ij} + B \\ & max_{ijk} \leqslant e_{ik} \end{aligned} \tag{4}$$

The MILP model presented in Eq. 1 is NP-hard and in scenarios with a high multiplicity of $|E_i|$ becomes less practical. Thus, to reduce the effect of this complexity on the solving time, we propose an approximation technique in which we reduce the number of executors from $|E_i|$ (the original number) to an imaginary $|E_i'|$ (a smaller number). This reduces the number of variables and constraints, and subsequently, the solving time. Omitting a portion of the executors in the model is analogous to neglecting the inter-interference among some of them and therefore imposing errors. However, we further investigate the effect of this approximation on the execution time estimation in Section 6 and show that the additional error is acceptable in representative scenarios. The reduction mentioned above is achieved by applying Eq. 5, where the input parameter tr_{ij} in the original problem is replaced with tr'_{ij}. This change is proportional to the reduction in the number of executors, and tm_{ij} remains untouched.

$$tr'_{ij} = tr_{ij} \times \frac{|E_i'|}{|E_i|} \tag{5}$$

4.3 Linear Programming Model

Recalling that in the MILP model of Eq. 1 some constraints are included to detect and avoid overlapping two stages in a single executor. Next, we propose an LP formulation in which these constraints are changed, and the calculation of the sum of stage runtimes is introduced. Similar to the previous model, the LP formulation predicts the execution time of DAG-based applications and uses input parameters and variables described in Tables 1 and 2.

The proposed LP model is presented in Eq. 6 and consists of a set of constraints. The minimization objective 6a is the same as the objective of the previous model, and similarly, the Eq. 6 is designed to find the best task scheduling assignment and minimize the application runtime by using the benefits of heterogeneity in the cluster.

Constraint 6b models the DAG's dependency relations. In addition, constraints 6c and 6d ensure the completeness of the application. For valid estimations, constraint 6g is introduced along with the objective 6a. With the help of the parameter R_j, which represents the rich dependency of stage j (all direct and indirect parents of stage j), we can calculate a lower bound for stage start time. Furthermore, the overlaps can be avoided by considering constraints 6f and 6e for all stages.

$$minimize\ T \tag{6a}$$

$$subject\ to$$

$$\forall j \in S\ \forall k \in P_j \tag{6b}$$
$$s_j \geqslant s_k + d_k$$

$$\forall i \in E\ \forall j \in S \tag{6c}$$
$$d_j \geqslant tm_{ij} + (tr_{ij} \times p_{ij})$$

$$\forall j \in S \tag{6d}$$
$$1 = \sum_{i \in E} p_{ij}$$

$$\forall i \in E\ \forall j \in S \tag{6e}$$
$$s_j \geqslant \sum_{k \in R_j} tm_{ik} + (tr_{ik} \times p_{ik})$$

$$\forall i \in E \tag{6f}$$
$$T \geqslant \sum_{j \in S} tm_{ij} + (tr_{ij} \times p_{ij})$$

$$\forall j \in L \tag{6g}$$
$$T \geqslant s_j + d_j$$

$$All\ Variables \geqslant 0 \tag{6h}$$

5 Cost-aware Resource Recommendation

By predicting the execution time of any given DAG-based application on any cluster, we can run a cost-aware search to find and recommend the best cluster configuration. We propose Algorithm 1 that exhaustively checks each possible cluster configuration and compares their costs, assuming that there is a limited number of instances for each machine type. The terms used in this algorithm are explained in Table 3.

First, for each machine type, Algorithm 1 temporarily allocates an instance from the cloud and runs the single node analyzer. Then, it searches through all cluster configurations and predicts their runtime and cost. The LP and MILP models presented in Section 4 can be used to predict the former, and an analytical model (a cost function provided by the service provider) is usually effective in predicting the latter. After calculating the runtime and cost for all cluster configurations, the proposed algorithm returns two configurations, one that minimizes cost and the other that minimizes the runtime considering a given budget.

Proposed models assist both the provider and the end-user in resolving the cost-runtime trade-offs when it comes to selecting of the proper resource configuration to be allocated or rented. In order to better illustrate such capability, two multi-dimensional plots represented in Fig. 2 are

created using the model of Eq. 1 for two exemplary cost functions illustrated in Eq. 7 and Eq. 8. The input parameters of the model are acquired from experiments on query 76 of the TPC-DS benchmark. Both given cost functions are charging the application proportional to its runtime T and the machine fees but with different coefficients. The representative scenario consists of two machine types for which the number of acquired instances are denoted with a and b. Each point is colored according to the estimated runtime, i.e., dark shades represent smaller ones. The points that represent the minimum cost are marked in the figure for each of the cost functions. The configuration that minimizes the runtime is marked in the figure as well, assuming a budget of 150$. As we can see in this figure, a change in the fee of one of the machines can have a significant impact on the proposed cost-based decision for the cluster configuration. Therefore having a prediction model and a resource recommendation algorithm can lead us to a better cluster configuration.

$$cost_1(T, a, b) = T \times (a + 0.7b) \quad (7)$$

$$cost_2(T, a, b) = T \times (a + 0.14b) \quad (8)$$

6 Experimental Results

In order to evaluate the accuracy of the proposed models, we perform multiple real-world experiments on a cluster running on two dedicated HP servers, empowered by Intel Xeon 2960-V4 and Intel Xeon E5520 processors with 24 and 16 processing cores and 45 and 60 gigabytes of RAM, respectively. We assign 9 and 12 executors to these servers, respectively, and each executor has two gigabytes of RAM and one processing core. Docker 18.3.5 is used to run the cluster and docker swarm to orchestrate the services. Apache Spark 2.4.1 is used on top of Apache Hadoop 3.1.1 to make the Big Data platform.

The experiments are performed with 300 gigabytes of Wikipedia data and the well-known industrial TPC-DS [43] benchmark on 500 gigabytes of input data. TPC-DS is a benchmark from the Transaction Processing Performance Council (TPC) for Decision Support (DS) systems and has similar characteristics to real data [41]. In addition, Wikipedia dump data [44] is used as a real-world dataset to better investigate the practicality of the proposed method.

Due to HDFS limitations, the Wikipedia dataset needs some preprocessing on names and hierarchies which are performed in the Purdue MapReduce Benchmarks Suite (PUMA) [45]. Therefore, PUMA is used here for that purpose. TPC-DS Kit [1] is used to perform some edits and bug fixes on the TPC-DS v2.10 queries, alongside some syntactical edits performed by IBM scripts [2] to run the TPC-DS queries on Spark. The scripts created for running the docker swarm cluster, executing experiments, and parsing the traces are published on Github [3,4] for reproducibility purposes.

1. https://github.com/gregrahn/tpcds-kit
2. https://github.com/IBM/spark-tpc-ds-performance-test
3. https://github.com/mohsenasm/spark-on-yarn-cluster
4. https://github.com/mohsenasm/Python-Spark-Log-Parser

Algorithm 1 Proposed Resource Recommendation Algorithm

Input: app, $idata$, mts, cm, crp , sna, $maxno$, b
Output: $bctime$, $bccost$

for all $mt \in mts$ **do**
  $vm \leftarrow allocatevm(mt)$
  $snanalyser(vm,\ app,\ idata)$
  $deletevm(vm)$
end for

$btime \leftarrow +\infty$
$bcost \leftarrow +\infty$
$bctime \leftarrow null$
$bccost \leftarrow null$

for all $tconf :< n_1, ..., n_{|mts|} >; n_i \in \{0, ..., maxno\}$ **do**
  $ttime \leftarrow crpredictor(tconf)$
  $tcost \leftarrow cmodel(tconf, ttime)$
  if $ttime < btime$ and $tcost \leq b$ **then**
    $btime \leftarrow ttime$
    $bctime \leftarrow tc$
  end if
  if $tcost < bcost$ **then**
    $bcost \leftarrow tcost$
    $bccost \leftarrow tconf$
  end if
end for
return $(bctime,\ bccost)$

TABLE 3
The explanation of terms used in Algorithm 1

Term	Meaning
app	Big Data application
$idata$	Input data
mts	Set of machine types
mt	A machine type
$cmodel$	Cost model
$crpredictor$	Cluster runtime predictor (either the LP or the MILP model)
$snanalyser$	Single node analyser
$maxno$	Max number of instances for each machine type
b	Budget
vm	A virtual machine
$allocatevm$	A function that allocates a VM from the cloud
$deletevm$	A function that deallocates a VM
$btime$	Best runtime
$bcost$	Best cost
$ttime$	Temp runtime
$tcost$	Temp cost
$bctime$	Best cluster configuration for runtime
$bccost$	Best cluster configuration for cost
$tconf$	Temp cluster configuration (a tuple $< n_1, ..., n_{\|mts\|} >$ where n_i represents the number of instances of type i in the cluster)

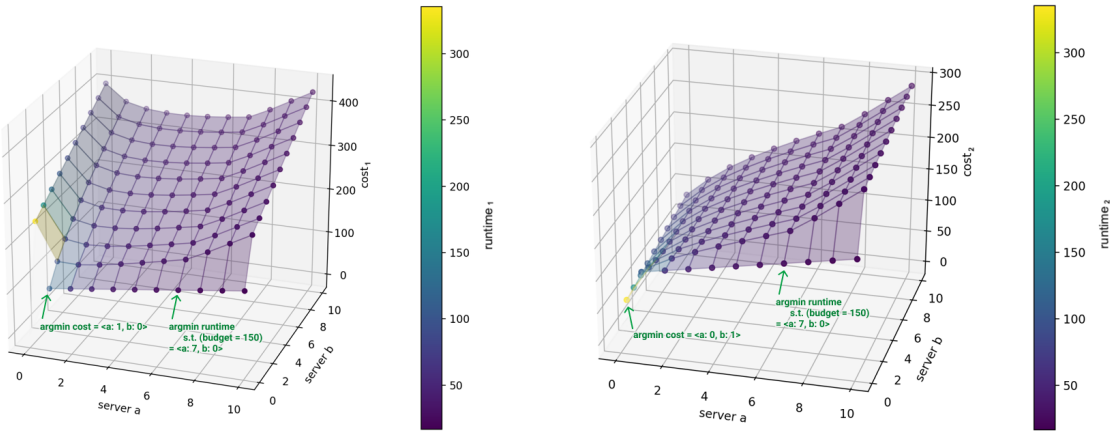


Fig. 2. Cost-runtime-configuration plots for query 76 and cost functions given in Eqs. 7 and 8 based on the number of machines for each type.

For the TPC-DS dataset, five queries 20, 33, 56, 58, and 76 are used in the experiments as they have different behaviors and DAG structures and could be good representatives of different workloads. DAGs of stages for the selected queries are illustrated in Fig. 3. For the Wikipedia dataset, three applications Word Count, Grep, and TF-IDF[5] are used, in which the Grep application uses a long regex from [46], and the TF-IDF application uses Spark MLlib. We use the Gurobi solver and the PuLP Python package to solve the MILP and LP models.

The evaluation results for the MILP model introduced in Eq. 1 are summarized in Table 4. This model uses the outputs of the single-node analysis phase as its inputs, but since we want to evaluate the accuracy of the MILP model, we measure these inputs from the experiments instead of micro-benchmarking. The experiments are performed ten times, and the average of the results is used to feed the model. The MILP model achieves a percent error of 3.28% on average. The percent error is calculated by Eq. 9.

$$Percent\ Error = \frac{|Actual\ value - Predicted\ value|}{Actual\ value} \times 100 \tag{9}$$

The experimental results for the LP model introduced in Eq. 6 are summarized in Table 5. The LP model has an average percent error of 3.28% and requires no approximation due to the polynomial complexity of its solving time, measured as 0.32s on average. Although the constraints of these two models have many differences, according to Tables 4 and 5, their prediction accuracy is identical.

As mentioned earlier, the time complexity of solving the MILP model is NP-hard, and it grows with the number of executors and stages. To reduce the effect of this complexity on the solving time and make it practical, we used approximation defined in Eq. 5 for queries 33, 56, and 58. The approximation parameters are included in Table 4, which are smaller than the original values 9 and 12 for the number of executors. Intuitively, this approximation is not expected to make a considerable difference in the model output for situations where the number of approximated executors is greater than the number of concurrent stages. This claim is examined in Table 6, where the model prediction for the execution time is reported for query 20 and with arbitrary approximation parameters. For two servers with 9 and 12 executors, there can be $9 \times 12 - 1$ combinations for selecting the approximation parameters, but for the sake of brevity, we showed only nine, as their results are almost the same. According to the results, model output remains almost the same for varying combinations.

5. Term Frequency-Inverse Document Frequency

7 Conclusions and Future Work

Presented in this paper, the runtime of Apache Spark applications can be predicted using the proposed MILP and LP models, assuming a heterogeneous Hadoop-YARN cluster as an underlying layer. This is achieved by solving the MILP and LP models and finding the minimum application runtime, as we assume that the primary goal of the mainstream frameworks like Apache Spark is the runtime minimization as well. This paper then proposed a cost-aware resource recommendation algorithm to select the best configuration of different machine types. The workflow of the application represented as a DAG is reflected in the proposed models along with the details of individual tasks running in parallel on different types of machines. Some parameters of the models are application-dependent and require a priori profile. For such parameters to be assigned correctly, we perform a single-node analysis phase on each VM type that uses the micro-benchmarking approach, in which the application runs on micro samples of the data in various sizes to predict the variables for the actual data size. Due to the low scalability of the MILP model, an approximation method was presented to decrease the complexity and thus the solving time. The accuracy of the results obtained from the proposed models is extensively evaluated against real-world executions. Different datasets and query applications were generated by the TPC-DS benchmark, and additionally, three representative applications, namely, Word Count, Grep, and TF-IDF were executed on a Wikipedia dataset. An acceptable average error of only 3.28% was realized when comparing the results of the proposed models with those measured from the real-world runs. Solving the LP model took about $0.32s$ on average, which indicates high

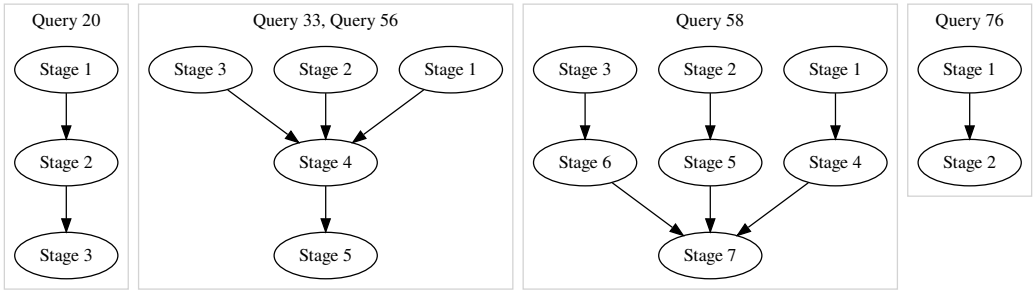


Fig. 3. DAGs of stages for the selected TPC-DS queries

TABLE 4
Actual and predicted execution times for the MILP model of Eq. 1

Application	# Stages	Actual time (ms)	Predicted time (ms)	Percent error (%)	Approximation of executors on server 1	Approximation of executors on server 2	Model solving time (s)
TPC-DS 20	3	29474	27341	7.24	-	-	0.6
TPC-DS 33	5	78794	76701	2.66	3	3	9.3
TPC-DS 56	5	84675	81672	3.55	3	3	15.7
TPC-DS 58	7	77565	77427	0.18	1	2	295
			77427	0.18	1	1	5.3
TPC-DS 76	2	96430	89437	7.25	-	-	0.4
Word Count	2	2052353	2025728	1.30	-	-	0.21
Grep	1	1171413	1160599	0.92	-	-	0.13
TF-IDF	3	7392433	7161435	3.12	-	-	0.48
average	-	-	-	3.28	-	-	4.01

TABLE 5
Actual and predicted execution times for the LP model of Eq. 6

Application	# Stages	Actual time (ms)	Predicted time (ms)	Percent error (%)	Model solving time (s)
TPC-DS 20	3	29474	27341	7.24	0.32
TPC-DS 33	5	78794	76701	2.66	0.32
TPC-DS 56	5	84675	81673	3.55	0.32
TPC-DS 58	7	77565	77428	0.18	0.33
TPC-DS 76	2	96430	89437	7.25	0.30
Word Count	2	2052353	2025728	1.30	0.31
Grep	1	1171413	1160600	0.92	0.30
TF-IDF	3	7392433	7161435	3.12	0.32
average	-	-	-	3.28	0.32

scalability. Representative real-world scenarios of a provider with two machine types were examined as well to demonstrate the practicality of the proposed cost-aware resource recommendation algorithm.

While the proposed algorithm for resource recommendation is searching the state space for the best configuration, the proposed LP/MILP model can be enriched with a proper cost function and create a non-LP formulation (although with an extra overhead) to minimize the cost. Moreover, metaheuristics can be adopted to speed up the search. However, the effectiveness of both the solutions mentioned earlier is yet to be assessed.

In order to improve the accuracy of the proposed models, data transmission among nodes can be considered as well, especially, between stages where more shuffling is performed. Our work lays the ground to integrate such details into the model. Extra detail could also be a feature of the framework, namely, speculative execution, where multiple instances of tasks are running on different nodes. This redundancy is a workaround for situations in which one node is performing poorly and hinders the overall availability and performance of the cluster. Moving in this direction, the model can also serve availability purposes.

TABLE 6
The MILP model 1 output for query 20 and some approximation parameters

Approximation of executors with server type 1	Approximation of executors with server type 2	Predicted time (ms)
1	1	27340.419
2	1	27340.962
3	9	27341.203
4	2	27341.278
5	12	27341.350
6	3	27341.364
7	9	27341.410
8	7	27341.429
9	5	27341.426
without approximation (i.e., 9, 12)		27341.446

ACKNOWLEDGMENTS

This work was supported by the National Research Foundation of Korea under grants 2022R1F1A1074967 and 21A20151113068 (BK21 Plus for Pioneers in Innovative Computing – Dept. of Computer Science and Engineering, SNU). ICT at Seoul National University provided research facilities for this study.

REFERENCES

[1] GIA, "Big data - global market trajectory and analytics," Global Industry Analysts (GIA), Tech. Rep. MCP-7749, 2022. [Online]. Available: https://www.strategyr.com/market-report-big-data-forecasts-global-industry-analysts-inc.asp

[2] J. Goepfert, "Worldwide big data and analytics spending guide," International Data Corporation (IDC), Tech. Rep. US48965120, 2021. [Online]. Available: https://www.idc.com/tracker/showproductinfo.jsp?containerId=IDC_P33195

[3] Apache tez – welcome to apache tez. Accessed on September 2021. [Online]. Available: https://tez.apache.org/

[4] Apache spark - unified analytics engine for big data. Accessed on September 2021. [Online]. Available: https://spark.apache.org/

[5] C. A. Ardagna, V. Bellandi, M. Bezzi, P. Ceravolo, E. Damiani, and C. Hebert, "Model-Based Big Data Analytics-as-a-Service: Take Big Data to the Next Level," *IEEE Transactions on Services Computing*, vol. 14, no. 2, pp. 516–529, mar 2021.

[6] J. Xu and B. Palanisamy, "Optimized Contract-Based Model for Resource Allocation in Federated Geo-Distributed Clouds," *IEEE Transactions on Services Computing*, vol. 14, no. 2, pp. 530–543, mar 2021.

[7] S. Venkataraman, Z. Yang, M. Franklin, B. Recht, and I. Stoica, "Ernest: Efficient Performance Prediction for Large-scale Advanced Analytics," in *Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation*. Santa Clara, CA, USA: USENIX Association, 2016, pp. 363–378.

[8] A. Maros, F. Murai, A. P. C. da Silva, J. M. Almeida, M. Lattuada, E. Gianniti, M. Hosseini, and D. Ardagna, "Machine learning for performance prediction of spark cloud applications," in *12th IEEE International Conference on Cloud Computing, Milan, Italy, July 8-13, 2019*, E. Bertino, C. K. Chang, P. Chen, E. Damiani, M. Goul, and K. Oyama, Eds. IEEE, 2019, pp. 99–106.

[9] G. P. Gibilisco, M. Li, L. Zhang, and D. Ardagna, "Stage aware performance modeling of DAG based in memory analytic platforms," in *9th IEEE International Conference on Cloud Computing, San Francisco, CA, USA, June 27 - July 2, 2016*. IEEE Computer Society, 2016, pp. 188–195.

[10] S. Kim, N. Pham, W. Baek, and Y. R. Choi, "Holistic VM Placement for Distributed Parallel Applications in Heterogeneous Clusters," *IEEE Transactions on Services Computing*, vol. 14, no. 5, pp. 1411–1425, 2021.

[11] M. Alalawi and H. Daly, "Designing a Hadoop MapReduce Performance Model using Micro Benchmarking Approach," in *Proceedings of the International Conference on Innovation in Computer Science and Artificial Intelligence*, London, UK, jul 2019, pp. 1–11.

[12] Z. Fu and Z. Tang, "Optimizing Speculative Execution in Spark Heterogeneous Environments," *IEEE Transactions on Cloud Computing*, vol. 10, no. 1, pp. 568–582, 2022.

[13] Z. Chao, S. Shi, H. Gao, J. Luo, and H. Wang, "A Gray-box Performance Model for Apache Spark," *Future Generation Computer Systems*, vol. 89, pp. 58–67, 2018.

[14] J. L. Berral, N. Poggi, D. Carrera, A. Call, R. Reinauer, and D. Green, "ALOJA: A Framework for Benchmarking and Predictive Analytics in Hadoop Deployments," *IEEE Transactions on Emerging Topics in Computing*, vol. 5, no. 4, pp. 480–493, 2017.

[15] Z. Jalalian and M. Sharifi, "Autonomous Task Scheduling for Fast Big Data Processing," in *Proceedings of the High Performance Computing and Big Data Analytics, TopHPC*, Tehran, Iran, 2017, pp. 137–154.

[16] O. Alipourfard, H. Harry Liu, J. Chen, S. Venkataraman, M. Yu, and M. Zhang, "CherryPick: Adaptively Unearthing the Best Cloud Configurations for Big Data Analytics," in *Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI '17)*, Boston, MA, USA, 2017, pp. 469–482.

[17] Á. B. Hernández, M. S. Perez, S. Gupta, and V. Muntés-Mulero, "Using Machine Learning to Optimize Parallelism in Big Data Applications," *Future Generation Computer Systems*, vol. 86, pp. 1076–1092, 2018.

[18] D. Cheng, X. Zhou, Y. Xu, L. Liu, and C. Jiang, "Deadline-Aware MapReduce Job Scheduling with Dynamic Resource Availability," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 4, pp. 814–826, 2019.

[19] L. Cai, Y. Qi, W. Wei, J. Wu, and J. Li, "mrMoulder: A Recommendation-based Adaptive Parameter Tuning Approach for Big Data Processing Platform," *Future Generation Computer Systems*, vol. 93, pp. 570–582, 2019.

[20] Y. Li, F. Liu, Q. Chen, Y. Sheng, M. Zhao, and J. Wang, "MarVeLScaler: A Multi-View Learning based Auto-Scaling System for MapReduce," *IEEE Transactions on Cloud Computing*, vol. 10, no. 1, pp. 506–520, 2022.

[21] L. Thamsen, T. Renner, I. Verbitskiy, and O. Kao, "Adaptive Resource Management for Distributed Data Analytics," in *Proceedings of the High Performance Computing and Big Data Analytics, TopHPC*, Tehran, Iran, 2017, pp. 155–170.

[22] D. J. Reid, "Optimal distributed execution of join queries," *Computers and Mathematics with Applications*, vol. 27, 1994.

[23] S. Papadomanolakis and A. Ailamaki, "An integer linear programming approach to database design," 2007.

[24] J. Sun and G. Li, "An end-to-end learning-based cost estimator," vol. 13, 2020.

[25] A. Dutt, C. Wang, A. Nazi, S. Kandula, V. Narasayya, and S. Chaudhuri, "Selectivity estimation for range predicates using lightweight models," vol. 12, 2018.

[26] Y. Park, S. Zhong, and B. Mozafari, "Quicksel: Quick selectivity learning with mixture models," 2020.

[27] J. Ding, Y. Xie, and M. Zhou, "Performance Modeling of Spark Computing Platform," *Studies in Computational Intelligence*, vol. 810, pp. 121–133, 2020.

[28] Y. Chen, P. Goetsch, M. A. Hoque, J. Lu, and S. Tarkoma, "d-Simplexed: Adaptive Delaunay Triangulation for Performance Modeling and Prediction on Big Data Analytics," *IEEE Transactions on Big Data*, vol. 8, no. 2, pp. 458–469, 2022.

[29] M. Nguyen, S. Alesawi, N. Li, H. Che, and H. Jiang, "A Black-box Fork-join Latency Prediction Model for Data-intensive Applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 9, pp. 1983–2000, 2020.

[30] X. Li, F. Chen, R. Ruiz, and J. Zhu, "MapReduce task scheduling in heterogeneous geo-distributed data centers," *IEEE Transactions on Services Computing, early access*, 2021.

[31] S. Karimian-Aliabadi, D. Ardagna, R. Entezari-Maleki, E. Gianniti, and A. Movaghar, "Analytical Composite Performance Models for Big Data Applications," *Journal of Network and Computer Applications*, vol. 142, pp. 63–75, 2019.

[32] S. Karimian-Aliabadi, D. Ardagna, R. Entezari-Maleki, and A. Movaghar, "Scalable Performance Modeling and Evaluation of MapReduce Applications," in *Proceedings of the Communications in Computer and Information Science*, vol. 891. Tehran, Iran: Springer, apr 2019, pp. 441–458.
[33] S. Karimian-Aliabadi, M. M. Aseman-Manzar, R. Entezari-Maleki, D. Ardagna, B. Egger, and A. Movaghar, "Fixed-point Iteration Approach to Spark Scalable Performance Modeling and Evaluation," *IEEE Transactions on Cloud Computing, early access*, 2021.
[34] E. Gianniti, A. M. Rizzi, E. Barbierato, M. Gribaudo, and D. Ardagna, "Fluid Petri Nets for the Performance Evaluation of MapReduce and Spark Applications," *ACM SIGMETRICS Performance Evaluation Review*, vol. 44, no. 4, pp. 23–36, 2017.
[35] A. Castiglione, M. Gribaudo, M. Iacono, and F. Palmieri, "Exploiting Mean Field Analysis to Model Performances of Big Data Architectures," *Future Generation Computer Systems*, vol. 37, pp. 203–211, 2014.
[36] Z. Shmeis and M. Jaber, "A Rewrite-based Optimizer for Spark," *Future Generation Computer Systems*, vol. 98, pp. 586–599, 2019.
[37] R. Krishna, C. Tang, K. Sullivan, and B. Ray, "ConEx: Efficient Exploration of Big-Data System Configurations for Better Performance," *IEEE Transactions on Software Engineering*, vol. 48, no. 3, pp. 893–909, 2022.
[38] S. Sidhanta, W. Golab, and S. Mukhopadhyay, "Deadline-Aware Cost Optimization for Spark," *IEEE Transactions on Big Data*, vol. 7, no. 1, pp. 115–127, 2019.
[39] F. Xu, H. Zheng, H. Jiang, W. Shao, H. Liu, and Z. Zhou, "Cost-effective Cloud Server Provisioning for Predictable Performance of Big Data Analytics," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 5, pp. 1036–1051, 2019.
[40] R. Han, X. Lu, and J. Xu, "On big data benchmarking," in *Big Data Benchmarks, Performance Optimization, and Emerging Hardware*, J. Zhan, R. Han, and C. Weng, Eds. Cham: Springer International Publishing, 2014.
[41] R. Han, L. K. John, and J. Zhan, "Benchmarking Big Data Systems: A Review," *IEEE Transactions on Services Computing*, vol. 11, no. 3, pp. 580–597, may 2018.
[42] "Apache hadoop yarn," accessed on September 2021. [Online]. Available: https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html
[43] "Tpc-ds homepage," accessed on May 2022. [Online]. Available: https://www.tpc.org/tpcds/
[44] "Wikimedia downloads," accessed on June 2022. [Online]. Available: https://dumps.wikimedia.org/
[45] F. Ahmad, S. Lee, M. Thottethodi, and T. Vijaykumar, "PUMA: Purdue mapreduce benchmarks suite," no. 437, 2012. [Online]. Available: https://docs.lib.purdue.edu/ecetr/437/
[46] "Email address regular expression that 99.99% works," accessed on June 2022. [Online]. Available: https://emailregex.com

Mohammad-Mohsen Aseman-Manzar received the M.S. degree and B.S. degrees in Computer Engineering from Sharif University of Technology and Iran University of Science and Technology (IUST), respectively. He is currently a Ph.D. student of Computer Engineering at the Sharif University of Technology. His research interests include performance modeling and evaluation of distributed systems, Big Data frameworks, and the Internet of things. In his industrial life, he is a senior software developer and works with various languages and frameworks. More information is available at https://www.asemanmanzar.ir/resume.

Soroush Karimian-Aliabadi is a Ph.D. graduate in Computer Engineering from the Sharif University of Technology in 2021. He received the BS degree from the Tehran University and MS degree from the Sharif University of Technology in 2012 and 2014, respectively. His primary focus is on analytical performance modeling and evaluation of distributed processing systems using high-level derivations of stochastic petri nets. He gained experiences in modeling and evaluation of Cloud and Big Data environments during MS and Ph.D. in performance and dependability laboratory under supervision of Prof. Ali Movaghar. In 2016, he joined Prof. Danilo Ardagna's group at Politecnico di Milano as a visiting researcher, where he worked on experimental validation of MapReduce performance model.

Reza Entezari-Maleki is an assistant professor in the School of Computer Engineering, Iran University of Science and Technology (IUST), Tehran, Iran. He received the B.S. and M.S. degrees from IUST, in 2007 and 2009, respectively, and the Ph.D. degree from the Sharif University of Technology (SUT), in 2014. He worked at the School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran, as a post-doctoral researcher, from 2015 to 2018. His main research interests are performance and dependability modeling and evaluation of distributed computing systems. More information is available at http://webpages.iust.ac.ir/entezari.

Bernhard Egger received the diploma in computer science from the Swiss Federal Institute of Technology, Zürich (ETHZ) in 2001 and the PhD degree in computer science and engineering from Seoul National University in 2008. After spending three years at the Samsung Advanced Institute of Technology, Samsung Electronics' central research insitute, as a senior research engineer, he joined Seoul National University where he is a professor in the Department of Computer Science and Engineering. His research interests include cyber security, programming language design, compilers, runtimes, and operating systems for parallel and heterogeneous architectures with the goal to maximize resource utilization through the reconciliation of static (compile-time) and dynamic (run-time) information. He is a member of the IEEE and ACM. More information is available at https://csap.snu.ac.kr/bernhard.

Ali Movaghar received the BS degree in electrical engineering from the University of Tehran in 1977, and the MS and PhD degrees in computer, information, and control engineering from the University of Michigan, Ann Arbor, in 1979 and 1985, respectively. He is a professor in the Department of Computer Engineering at the Sharif University of Technology in Tehran, Iran and has been on the Sharif faculty since 1993. He visited the Institut National de Recherche en Informatique et en Automatique in Paris, France and the Department of Electrical Engineering and Computer Science at the University of California, Irvine, in 1984 and 2011, respectively.He worked at AT&T Information Systems in Naperville, Illinois in 1985-1986, and taught at the University of Michigan, Ann Arbor, in 1987-1989. His research interests include performance/dependability modeling and formal verification of wireless networks, and distributed real-time systems. He is a senior member of the IEEE and the ACM.