Performance Prediction for Data-driven Workflows on Apache Spark

Andrea Gulino, Arif Canakoglu, Stefano Ceri and Danilo Ardagna
Politecnico di Milano, *name.surname@polimi.it.*

Abstract— **Spark is an in-memory framework for implementing distributed applications of various types. Predicting the execution time of Spark applications is an important but challenging problem that has been tackled in the past few years by several studies; most of them achieving good prediction accuracy on simple applications (e.g. known ML algorithms or SQL-based applications). In this work, we consider complex data-driven workflow applications, in which the execution and data flow can be modeled by Directly Acyclic Graphs (DAGs). Workflows can be made of an arbitrary combination of known tasks, each applying a set of Spark operations to their input data. By adopting a hybrid approach, combining analytical and machine learning (ML) models, trained on small DAGs, we can predict, with good accuracy, the execution time of unseen workflows of higher complexity and size.**
We validate our approach through an extensive experimentation on real-world complex applications, comparing different ML models and choices of feature sets.

Index Terms— **performance prediction, workflow applications, Spark, machine learning**

I. INTRODUCTION

In the past decade, we have witnessed an increasing spread of big data applications in several domains, such as business analytics [1], social media analysis [2], healthcare [3], and natural language processing [4]. Big data applications are characterized by data-intensive and computationally intensive tasks which are typically implemented on top of parallel algorithms and distributed computing frameworks. Among such frameworks, Apache Spark has emerged as the de-facto platform for analytical processing, with broad adoption in both industry and academia, due to its simplicity, fault tolerance, and scalability [4].
At the same time, users and enterprises have started moving their big data applications from traditional local server architectures to cloud computing platforms (e.g. Amazon EC2, Google Cloud, Microsoft Azure[1]), which provide configurable environments suiting the needs of big data applications. These systems usually offer services at the Platform level, where big data frameworks are already installed, and allow their users to choose among several configurations, e.g. specifying the number of instances in a cluster and their characteristics (CPUs, memory, . . .). Each choice might drastically affect the application execution time and the monetary cost of using the cloud computing service.
Predicting the performance of an application is therefore useful for a proper allocation of the available resources, aimed at reducing resource wasting and extra costs.

[1]https://aws.amazon.com, https://cloud.google.com and https://azure. microsoft .com

The problem of performance prediction for big data applications on the cloud has been tackled in several studies. Some of them rely on traditional techniques, such as analytical models [5]–[7] and simulation [8]. More recently, machine learning (ML) has been used to predict the performance of large systems [9]–[12]. The idea is to collect training data offline and use ML models, such as regression, to predict the runtime performance of future executions. Those studies mainly differ for the chosen set of features (*black-box* vs. *gray-box* approaches), i.e. capturing more or fewer details of the system, and for the applied ML technique (simple regression vs. more complex ML techniques). In [12] authors show how black-box models (specifically Ernest, an approach proposed by Spark inventors [9]) fail when applications exhibit irregular patterns and/or when extrapolating on bigger data set sizes. Despite achieving good results in terms of accuracy, those studies are performed on simple monolithic applications (e.g. static programs, known ML algorithms), which complexity is far from modern data analytics pipelines. Indeed, nowadays, scientific jobs are rather represented as workflow applications that consist of many complex tasks, with logical or data dependencies, that can be implemented on top of several parallel frameworks [13]–[15].

A workflow application can be represented by a Directed Acyclic Graph (DAG), i.e. a directed graph with no cycles, in which vertices and their connecting edges are used to represent, respectively, application tasks and their dependencies. Each task cannot be executed until all its parent tasks have completed their execution and moved their results to their child tasks. Specifically, we target workflow applications implemented on Spark, i.e. workflows in which each task of the workflow applies a set of Spark operations to the task inputs. Moreover, a workflow can be potentially implemented by multiple Spark applications.

A simple way of predicting the execution time of a workflow could consider the workflow as a monolithic application with known inputs. In this case, a ML model should be trained for each feasible workflow, i.e. for each possible combination of tasks that could appear in the workflow. Besides being a solution that does not scale, the complexity of large workflows could be hardly captured by machine learning.

The solution proposed in this paper, instead, builds a separate ML model for each task allowed in a workflow application. The workflow execution time is eventually estimated as a combination of the individual tasks execution time predictions. This type of solution allows training ML models on minimal workflows and uses them to predict the performance of *unseen* workflows of arbitrary complexity. Moreover, since the input data of intermediate tasks is not known offline, we estimate their characteristics (profiles) by mixing ML and analytical models.

The overall solution is a hybrid approach to estimate the execution time of arbitrary complex workflow applications based on Spark. To the best of our knowledge, no previous work focused on performance prediction for Spark applications presenting this level of complexity and no similar approaches, explicitly tackling the problem of intermediate result estimation, were used for workflow performance prediction.

We validate our approach on a real-world complex system, comparing different ML techniques and choices of feature sets (black-box vs. gray-box). Eventually, an extrapolation analysis compares the robustness of different ML models against the variation of the input data size and of the cluster computational power.

This paper is organized as follows: in Section II we describe data-driven workflow applications and introduce the real-world system used in our experimental evaluation; in Section III we describe our three-phase approach for performance prediction, validating the approach in section IV. A discussion of related literature proposals is reported in Section V. Conclusions are finally drawn in Section VI.

II. Data-driven Workflow Applications

A data-driven workflow application can be represented as a Directly Acyclyc Graph (DAG), i.e. a directed graph with no cycles. Each vertex in this DAG represents a task, while edges represent the data and control dependencies between tasks. A task is executed only when all its input data have been computed, i.e. when all parent tasks have completed their execution. Formally, a DAG representing a workflow application can be described as a tuple $G = (V, E)$, where V is the set of vertices (tasks), E the set of directed edges (dependencies) s.t. $E \subseteq V^2$. We call *entry task* (*exit task*) a task with no incoming (outgoing) edges. For simplicity, we assume that workflows have a unique exit task. While entry tasks represent the reading of the workflow input data, the exit task stores back its final result. We further assume that each task produces a single output, which can be the input to multiple child tasks (through multiple edges).
Moreover, we characterized each task defining:

- *Output Profile*: a set of features quantitatively describing its result.
- *Task Arguments*: any input parameter given to the task, in addition to the input data produced by its parent tasks. Examples can be strings, flags or other options that might change the behaviour of the task.
- *Environment Parameters*: a set of features describing the execution environment in which the task will run. Examples are the number of cores and amount of memory of a cluster.
- *Execution Time*: time required for processing the input data and produce the task result.

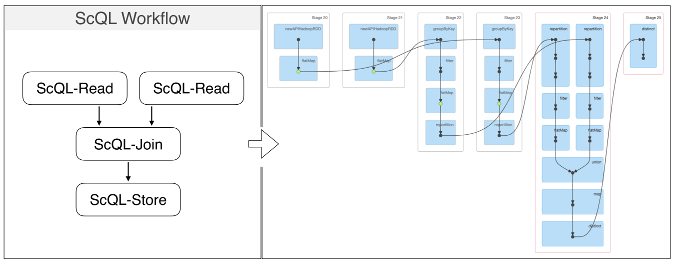

Fig. 1: A ScQL Workflow is mapped to Spark DAG Application.

.

A. Target Application

For our experimental evaluation, we chose a real-world cloud system that maps complex workflows to Spark applications. This system implements a SQL-like language for interval data called ScQL [16], [17].

Like for other query languages relying on relational algebra, a ScQL query can be easily mapped to a workflow, in which each task represents a query operator. A simplified version of a ScQL query is the following:

```
DS1 = READ() dataset_1;
DS1 = READ() dataset_2;
RES = JOIN(dist<100) DS1 DS2;
STORE RES;
```

Once a query is submitted, the ScQL processing system translates the query into a ScQL-Workflow, similar to the one depicted on the left side of Fig. 1. Each task of this workflow applies a set of Spark operations on input data to implement the semantics of the corresponding ScQL operator. In this specific example, the entire workflow corresponds to a single Spark application, which DAG structure is depicted on the right side of Fig. 1.

In the ScQL Data Model, an interval-dataset D is made of several files (which in memory are called *samples*). Each entry in a file represents an interval, by means of its start and stop values.
Although ScQL includes several operators, in our experimental evaluation we will only mention two of its most complex and peculiar operators, namely ScQL-Map and ScQL-Join, that are here briefly described:

- **ScQL Map**: given two input datasets, namely a *reference* and an *experiment* dataset, this operator computes, for each interval in the reference dataset, aggregates on the overlapping experiment intervals.
- **ScQL Join**: given two input datasets, namely a *reference* and an *experiment* dataset, this operator computes, for each couple of reference-experiment files, the couples of overlapping reference-experiment intervals. A `distance` parameter (dist) can be provided to match also experiment intervals which are at a maximum distance from a reference interval.

Lastly, we mention a partitioning technique called *interval-binning* [18]. The *bin-size* parameter determines the amount of intervals that end up in the same partition. Depending on this parameter, some intervals of the initial dataset might be replicated in several partitions.

III. Workflow Performance Prediction

Our *three-phase* solution for predicting the execution time of a workflow application performs the following three steps:

1) **Intermediate profile estimation**: for each task in the workflow, its output profile is estimated. This implies that, at the end of this phase, the input data profile of every task in the workflow will be available. This phase is detailed in Section III-C.
2) **Task execution time prediction**: for each task of the workflow, its execution time is predicted. This is discussed in section III-A.
3) **Workflow execution time prediction**: the overall execution time is computed by combining the predictions of the individual tasks execution times. Generally, the workflow execution time depends on the underlying engine and its scheduling algorithm. In this work we target workflows that are mapped to Spark applications. By assuming that: i) every task of a workflow is executed within the same Spark context; ii) every workflow task corresponds to a number of Spark tasks that is greater or equal to the number of available cores (in other words, assuming that each workflow task will keep busy all the available cores) then we can reasonably approximate the workflow execution time as the sum of the individual tasks execution times. The validity of these assumptions is proven by our experimental results (see Section IV).

A. Task execution time prediction

The goal of task execution time prediction is to build, for each different type of task that can be present in a workflow, a machine learning model that, with *adequate* accuracy, is able to predict the task execution time. A ML model is trained on a set of features characterizing the task, such as a quantitative description of its input data, its arguments, and features describing the execution environment, with the objective of predicting its execution time.
Identifying and extracting a complete set of features is hard to achieve in this context. First of all, the knowledge on the application and on the execution environment might be limited; secondly, extracting some features might be non-trivial and time-consuming, hence not convenient. Third, a super-set of features usually leads to overfitting.

State-of-the-art ML solutions for performance prediction are mostly based on simple feature sets that provide an high-level description of the input data and of the execution environment. These are usually called *black-box* features, e.g. the input size and the number of cores can be considered a complete set of black-box features for the performance prediction of a generic parallel application. Although it has been proven that black-box features can be powerful enough to get good prediction accuracy (e.g. in Ernest [9]), choosing a simple feature set might become limiting when trying to describe the performance of complex workflows.

TABLE I: Feature-set categorization

	Basic	Full
Black-box	Input Data (BBI) Task Parameters (BBT) Execution Environment (BBE)	BBI ∪ BBT ∪ BBT Composite Features (BBC)
Gray-box	Input Data (GBI) Task Arguments (GBT) Execution Environment (GBE)	GBI ∪ GBT ∪ GBT Composite Features (GBC)

In this work, we considered different feature sets, showing how different choices can impact the prediction accuracy of the produced models. Without constraining the choice of features, we propose a categorization of the possible feature sets. The first distinction we make is between:

- **Black-box** features, that do not require detailed knowledge on the application, data model and execution environment.
- **Gray-box** features, that extend black-box features with application, data-model and environment-specific features. We assume that gray-box feature sets include also black-box features.

Orthogonality to the previous distinction, we discriminate among:

- **Basic features**, organized into:
  - **Input Data Profiles**: features quantitatively describing the input data of a task (e.g. data size, number of files, number or entries ...).
  - **Task Arguments**: any input parameter given to the task.
  - **Execution Environment**: a set of features describing the environment in which the task is executed, e.g. number of cores and amount of memory, or more advanced features characterizing the Spark environment.
- **Composite features**. Even though several machine learning methods are able to capture non-linear dependencies on the features, it is sometimes "helpful" to include non-linear combinations of some basic features in the final feature set. For example, Ernest [9] introduced the logarithm of the number of cores, which encodes the cost of reducing operations in parallel frameworks, and the ratio between data size and number of cores, which approximates the time spent in parallel processing. We define composite features as linear or non-linear combinations of basic features.

Given the aforementioned distinctions, we organize the possible feature sets in four categories, summarized in Table I: basic black-box features (**BB-Basic**); basic and composite black-box features (**BB-Full**); basic gray-box features (**GB-Basic**); basic and composite gray-box features (**GB-Full**).

According to the proposed categorization, Ernest, i.e. the state-of-the-art for Spark application performance prediction, uses a black-box full (BB-Full) feature set. In the experimental evaluation, our choice of features for the BB-Full feature set included all the Ernest features and will be considered a baseline to evaluate the benefits of our approach.

For each category, we defined a set of candidate features on which we applied Sequential Forward Selection (SFS) [19] to remove non-relevant ones.

B. Model Techniques

In the previous works, Venkararaman et al. [9] (Ernest's authors) and Maros et al. [12] have applied ML to estimate the execution of simple Spark applications. While Ernest is based on linear regression, with coefficients estimation based on non-negative least squares (NNLS), in Maros et al. authors took into account more complex ML techniques such as Decision Tree and Random Forest.
In Section IV we compare the performance of three suitable ML techniques: (i) Linear regression (LR), which produces models that can be easily interpreted, but hardly captures complex interactions between features, (ii) Decision Trees (DT) and (iii) Random Forest (RF), having the ability to capture non-linear relationships, still allowing a good interpretability of the model.

Given a target task type, for which we want to predict the execution time, the training set provided to the chosen machine learning technique is made of several executions of that task. As proven in the experimental section, depending on the feature set and on the model technique, it might not be necessary to run on big inputs and extremely powerful environments. Even running small executions on medium-size clusters can still produce prediction models with good prediction error on larger inputs and clusters.

C. Predicting intermediate input features

In order to perform the task execution time prediction step, all the input features, including the input data profiles, must be available offline. However, this does not hold for intermediate tasks, which input data have not been computed yet. To address this problem, we perform, as the first step, an estimation of all such data profiles.
We assume that entry tasks import data which have already been profiled. This assumption is in general acceptable, since application users typically download a limited number of datasets which are then re-used for several workflow runs and can therefore be profiled once for all.

The goal is to estimate, for every task $v_i \in V$, which is not the exit task, its output profile P_i. Specifically, an estimation model should be built for each feature in P_i. (e.g. data size, number of entries, etc.).
Similarly to the execution time prediction, the feature set used for this type of estimation includes: i) the input profiles of task v_i, ii) the task arguments. Moreover, the same feature set categorization proposed in section III-A has been applied, reasonably excluding any environment-related feature.
Each feature in P_i can be estimated:

- **Analytically**: there is a known combination of features of the chosen feature set which allows to exactly compute the output feature.
- **Heuristically**: the exact formula for computing the output feature is unknown or there is not enough information to compute it; in this case, heuristics on the available feature set can be defined.
- by **Machine Learning**: applying ML to estimate the output feature, similarly to what was has been said for task execution time prediction.

The intermediate profile estimation phase estimates each task output profile, starting from entry nodes, which input data profile was computed offline (previous assumption), up to the exit node.
In the experimental section (IV) we compare different ways of predicting the output profiles, showing the advantage of using heuristics/analytical models w.r.t. ML. Even though heuristics and analytical models require more knowledge on the application, they are able to guarantee lower prediction error. Analytical and heuristic-based based estimations for ScQL (GMQL) are described in [18].

IV. Experimental Evaluation

We applied the proposed performance prediction model to ScQL DAGs, introduced in II-A, focusing on two of its most complex and peculiar operators. In this section we:

- describe the experimental setup (IV-A), the evaluation metric and how hyper-parameter tuning was performed (IV-B)
- assess the accuracy of task execution time prediction for different choices of feature sets, different ML techniques and execution environment configurations (IV-C).
- assess the accuracy of output profile estimation, either using analytical models/heuristics, and ML (IV-D).
- measure the error in predicting the workflow execution time, comparing gray-box models to the state of the art black-box based (IV-E) and validating on workflows that were much more complex than the workflows used for collecting training data.
- present an extrapolation analysis, in which we compare the *robustness* of different ML models against the increase of the input data size and of the cluster computational power (IV-F). Specifically, we prove that models trained on small input data (number of cores) are good at predicting execution with larger input data (higher number of cores). This is particularly beneficial, given the nature of big data analyses, in which the size of processed data gets bigger and bigger, as well as the computational resources required for processing.

A. Experimental setup

All the experiments in this section were run on Amazon AWS (EMR service) choosing a variable number of `r5d.2xlarge` worker nodes (each having 8 vCores, 64 GiB RAM). Specifically, we used clusters with 6,8,10 and 12 worker nodes, running the `emr-5.19.1` release with `Spark 2.3.2` and the `Amazon 2.8.5` Hadoop distribution. Training sets were built running more than 4500 executions, collecting all the information required for execution time prediction and output profile estimation, using an in-house developed profiling solution.

B. Evaluation metric and hyper-parameter tuning

a-MLLibrary, an open source library[2] built on top of scikit-learn 0.19.1[3], was used to test different values for the hyper-

[2]https://github.com/eubr-atmosphere/a-MLLibrary
[3]https://scikit-learn.org

TABLE II: Most used hyper-parameters values (RF and DT)

Hyper-parameter	**DT**	**RF**
Max Depth	5	8
Max Features	auto	auto
Min samples to split	4	2
Min samples per leaf	1	2
Criterion	MSE	MAE
# Estimators	NA	300

TABLE III: Most used hyper-parameters values (LR)

Hyper-parameter	**LR**
Penalty α	0.01
Fit-intercept	True

parameters characterizing each learning methods along the lines of the work in [12]. Several combinations of values for the hyper-parameters were tested, and the best combination, corresponding to the lowest MAPE obtained with 5-fold cross-validation, was selected.

The most frequently used hyper-parameters are reported in tables II and III. Some parameters, e.g. the alpha penalty and the maximum three depth for Random Forest and Decision Tree help to prevent overfitting. Minimum Samples to Split/per Leaf represent, respectively, the minimum number of samples required to split a node and the minimum number of samples required to be a leaf. More details on the hyper-parameters are provided in the scikit-learn documentation. For each feature set category, we first defined a set of candidate features and then applied SFS to exclude non-relevant features.

C. Task execution time prediction

Tables IV and V report the measured MAPE for different ScQL operators, i.e. ScQL-Map (Table IV) and ScQL-Join (Table V). Within each table, we report the prediction error for:

- different choices of the feature set, i.e. black-box vs gray-box and basic vs full, as described in Section III-A;
- different machine learning methods, including DTs, RFs, and LR. We excluded methods for which the best MAPE was higher than 20%, like XGBoost and Neural Networks.

Tables VI and VII, show, respectively, the relevant features for ScQL-Map and ScQL-Join provided by the ML technique giving the lowest MAPE (RF), for a given feature set choice (black-box vs. gray-box, basic vs. composite). The feature called *binned-result-size* represents the (known-by-semantics) result size, accounting for the replication of some intervals due to binning. The definition of this composite feature includes gray-box features such as the *bin-size* and the *average-interval-length*. Similarly applies to the feature called *binned-total-size*, which definition includes also a task parameter *distance*.

Results show that:

- RF is able to give good predictions even with black-box basic features. Therefore, even without having detailed knowledge on the application and on the environment, and without "helping" the model by defining composite (i.e. non-linear) features, it is still possible to get an acceptable error.
- LR performs well only when all the complex features are in place (gray-box), including the user-provided non-linear features (composite).
- Basic gray-box features do not significantly reduce the prediction error unless they are properly combined into composite features.
- The higher complexity of the ScQL-Join operation w.r.t. the ScQL-Map is reflected in the higher prediction error.

D. Intermediate profiles estimation

In Table VIII we report the prediction error (MAPE) for the main features describing the result of a ScQL-Join operation. While the output number of files can be analytically determined, the output total size and the output average interval length cannot be known a priori, since they depend on the number of intersections between input intervals. For those features, we defined two heuristics which give a low prediction error. For instance, the number of output samples (files) of a ScQL-Join operation is simply estimated as the product between the number of samples in the reference dataset and the number of samples in the

TABLE IV: Prediction MAPE for SciQL-MAP - AWS

	Basic			**Full**		
	DT	LR	RF	DT	LR	RF
BB	16%	34%	**12%**	16%	33%	**12%**
GB	16%	34%	**12%**	12%	14%	**9%**

TABLE V: Prediction MAPE for SciQL-JOIN - AWS

	Basic			**Full**		
	DT	LR	RF	DT	LR	RF
BB	20%	32%	**17%**	19%	26%	**15%**
GB	19%	31%	**14%**	16%	22%	**13%**

TABLE VI: SciQL-MAP Relevant Features (RF)

	Basic	**Full)**
BB	ref-total-size (0.5) exp-num-files (0.3) cores (0.2) ...	ref-total-size (0.5) exp-num-files (0.3) **input-total-size/cores** (0.3) ...
GB	ref-total-size (0.5) exp-num-files (0.3) cores (0.2) ... **bin-size** (<0.1)	**binned-result-size / cores** (0.8) **binned-result-size**(0.1) ...

TABLE VII: SciQL-JOIN Relevant Features (RF)

	Basic	**Full**
BB	input-total-size (0.5) exp-num-files (0.3) ref-num-entries (0.1) cores (0.1) ... distance (<0.1)	input-total-size (0.3) **input-total-size/cores** (0.3) exp-num-files (0.3) ref-num-entries (0.1) ...
GB	input-total-size (0.4) exp-num-files (0.3) ref-num-entries (0.1) cores (0.1) ... **bin-size** (<0.1)	**(binned-total-size / cores)**^{2} (0.5) exp-num-files (0.1) ...

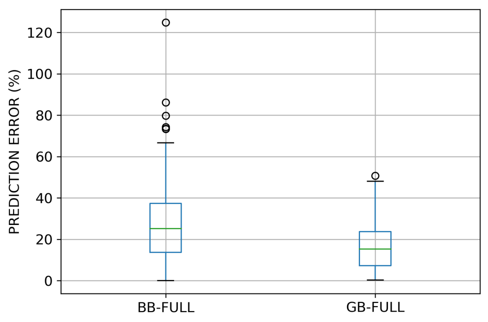

Fig. 2: Workflow execution time prediction error distribution for BB-FULL and GB-FULL

experiment dataset. For all the three features we reported the prediction error obtained using ML models (specifically, Decision Tree). Although the output total size is a black box feature, the prediction error reported in the table was obtained using a model which took into account gray box features; using only black-box features we were not able to obtain a MAPE lower than 22%. We do not report ScQL-Map prediction errors since all the output features can be easily estimated, both analytically and by applying ML, with a MAPE lower than 1%.

E. Workflow execution time prediction

In order to measure the workflow execution time prediction error we automatically generated 168 workflows with different topologies including ScQL-Map and ScQL-Join, together with other mandatory entry/exit tasks which, for simplicity, are not discussed in this paper. Each ScQL workflow was mapped to a Spark Application. On average, each generated Spark application contained 23 jobs, 117 stages and more than 16K tasks per critical stage. Note that those workflows have higher complexity w.r.t. to the workflows used to build the training sets, mostly containing only the task for which the ML model was going to be build. In other words, here we demonstrate the ability of our modular approach to predict the execution time of workflows of unseen complexity, although minimal workflows were used for training the ML models.

As described in Section III, we first estimated the input profiles for all the tasks in the DAG and then predicted each task execution time using ML models that we previously built. We compared the workflow execution time prediction error (MAPE) for BB-FULL, i.e. the state of the art Ernest approach, and GB-FULL feature sets. The error distribution is depicted in Fig. 2. While using BB-FULL we observed an average MAPE of **28%**, using GB-FULL features the average MAPE dropped to **17%**. Moreover, while with BB-FULL (Ernest) the error variance is higher, reaching more than 120% prediction error in one case, the highest error measured for GB-FULL is slightly higher than 50%.

F. Extrapolation analysis

A desirable way to build a prediction model for performance estimation would consist in learning with small input data and few computational resources and expect a low prediction error even when the input data is much bigger and the execution environment is more powerful. In this way, the training dataset could be built in a short-time and renting expensive resources would not be necessary. If a ML model is able to guarantee a stable prediction error for unseen value ranges of a given feature, we can say that the model is *robust* against the scaling of that feature. In this section, we compare the robustness of models, built with different ML techniques and trained on different feature sets, w.r.t. the scaling of the number of cores and of the input data size. Since we observed that DT models are as robust as the models produced by RFs, we only show the comparison between RF and LR.

In the first two rows of Fig. 3, we tested the robustness against the scaling of the input data size for ScQL-MAP (first row) and ScQL-Join (second row). The first plot in each row shows on the x-axis the input data size and on the y-axis the number of executions in our dataset performed with an input of that size. We trained the models on small executions (with input size lower than x_0 = 20% of the maximum size) and measured the validation MAPE on executions belonging to unseen input size ranges: 20-40%, 20-60%, 20-80% and 20-100% of the maximum input size. The scaling of the MAPE is depicted in the plots belonging to the second and third columns of Fig. 3. The first point in each plot (positioned at x_0) represents the validation MAPE computed on some executions randomly selected from the 0-20% range that were not used for training.

Results show that RF guarantees a good scaling, while LR fails (overfits in 0-20% range), unless gray-box full features are provided. Similarly, we tested the robustness against the scaling of the number of cores, which, for our experiments, corresponds to the scaling of the number of worker nodes in the EMR cluster. We trained our models on executions with 16 and 32 cores (i.e. 2 and 4 worker nodes), and measured the validation MAPE on executions using 48 cores, (48 & 64) cores and (48 & 64 & 96) cores.

Again, results show that RF guarantees a good scaling, while LR fails (overfits), unless composite features are used. In this case, LR is robust even using black-box features, given that non-linear features involving the number of cores are defined (in our feature sets *input-total-size/cores* has been selected by SFS).

TABLE VIII: Output-Estimation MAPE for ScQL-Join

Feature	Analytical | Heuristics	ML
out-num-files	0%	0.004%
out-total-size	0.2%	0.11%
out-interval-length	0.005%	0.03%

V. Related Work

To the best of our knowledge, this is the first comprehensive work on performance estimation for scientific data-driven workflows, implemented on Spark, based on machine learning and analytical models. Moreover, we did not find any modular workflow performance prediction solutions which explicitly address the problem of estimating

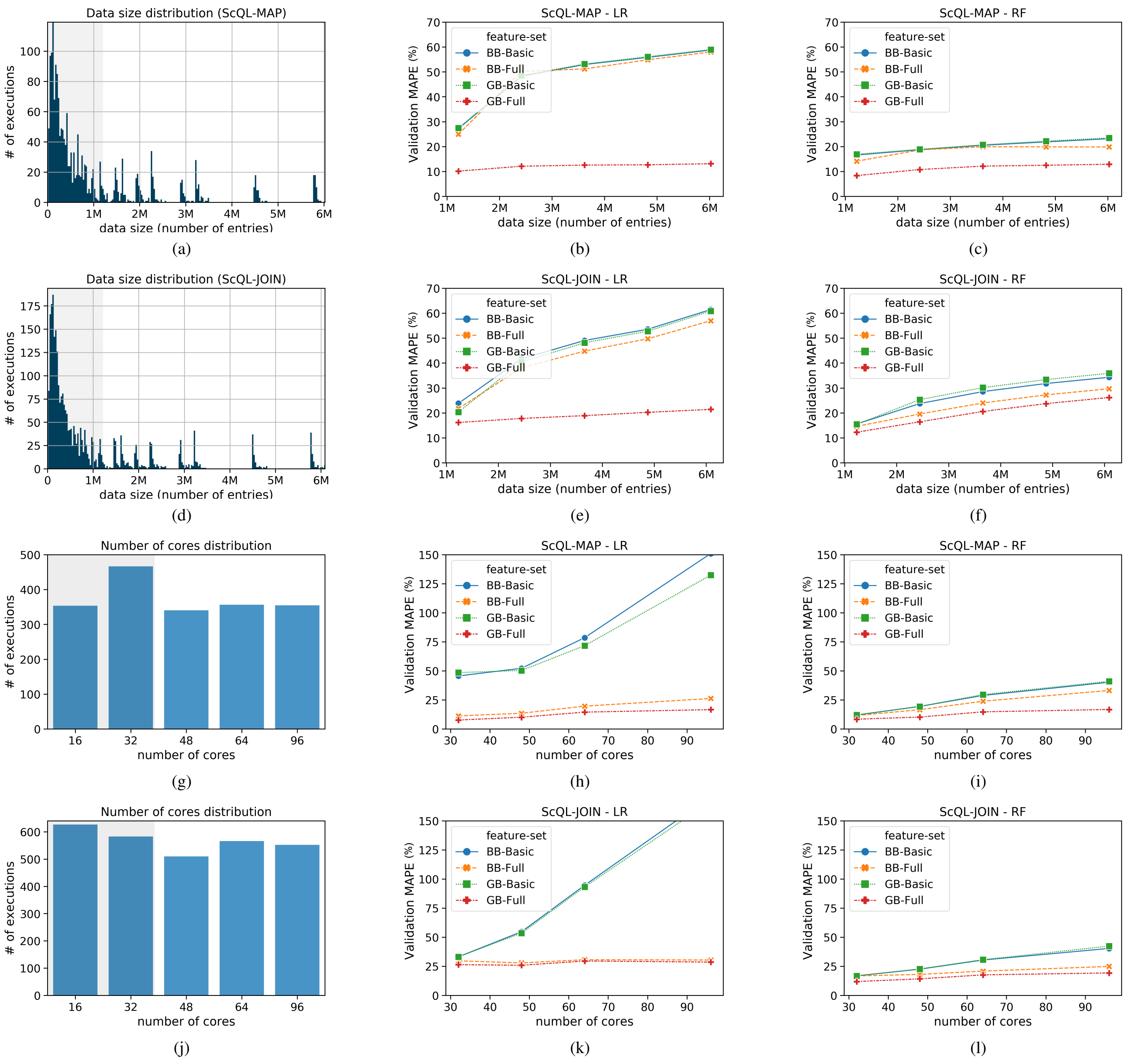

Fig. 3: Validation error (MAPE) variation w.r.t. the scaling of the input data size and the number of cores.

intermediate tasks input data profiles, which are not known offline.

Performance prediction for cloud-based applications has been tackled in several ways. Some studies apply traditional techniques, such as analytical models [6], [20]–[22] and simulation [23]. These techniques usually require detailed knowledge of the system, which is available only at runtime, and make simplifying assumptions that produce models unable to capture the complexity of cloud-based big data applications, losing in accuracy. Some studies, e.g. [24], propose a change in the Spark architecture that makes the job completion time estimation easier. More recently, there have been studies exploiting machine learning (ML) models for performance prediction of large systems [9]–[12], [25]. Most of these works use a *black-box* approach, in which historical data is used to predict future execution time, without knowledge of the system internals. Some works, instead, try to apply *gray-box* approaches, taking into account some system detail [12], [26]. *Ernest* [9], using simple features (functions of the input size and of the number of cores) and non-negative least squares (NNLS) regression, is considered the state-of-the-art in using ML. In [12], several ML models and several approaches (*black-box* vs. *gray-box*) are compared, showing better accuracy w.r.t. *Ernest*.

In the area concerning workflow performance prediction, most of the works focus on the individual task execution time prediction [27]–[29], highlighting the important role that this prediction plays in the context of optimal workflow scheduling [30]. In those papers, machine learning techniques are applied to predict the execution time of tasks contained in different real-word static workflows, i.e. workflows with a fixed structure, executed on the cloud. feature sets mostly accounts for environment parameters and only the workflow input data size, not the individual input to each task, is used to describe the input data.

Compared to previous work, we extend performance prediction to complex Spark applications, showing the limitation of using black-box features, and we consider complex workflows with arbitrary combinations of tasks, whose performance can be accurately predicted thanks to intermediate output estimation.

VI. CONCLUSIONS

In this paper, we presented a hybrid three-phase modular solution for predicting the performance of complex data-driven workflows which can be mapped to Apache Spark applications. The workflow performance is predicted by combining individual task execution time predictions. Compared to previous works in this area, we targeted dynamic workflows, with arbitrary tasks composition, introduced intermediate-profile estimation, essential for the proposed approach, and considered a real-word complex system as a benchmark. In the experimental evaluation we compared the accuracy of different ML methods using different feature sets, which depend on how much knowledge on the application is available. Results show that, even for complex systems, performance can be predicted with good accuracy, which improves when the semantics of tasks is known, i.e. when using gray-box features. Moreover, the produced models keep a low prediction error even for *unseen* input data size, amount of cluster resources and workflow dimensions.

ACKNOWLEDGMENT

This work was supported by the AWS Machine Learning Research Award (MLRA) and by the Data-Driven Genomic Computing (GeCo) project, funded by the European Research Center (ERC) (Advanced ERC Grant 693174).

REFERENCES

[1] Z. Sun, L. Sun, and K. Strang, "Big data analytics services for enhancing business intelligence," *Journal of Computer Information Systems*, vol. 58, no. 2, pp. 162–169.

[2] N. A. Ghani, S. Hamid, I. A. T. Hashem, and E. Ahmed, "Social media big data analytics: A survey," *Computers in Human Behavior*, vol. 101, pp. 417–428.

[3] A. J. Kulkarni, P. Siarry, P. K. Singh, A. Abraham, M. Zhang, A. Zomaya, and F. Baki, *Big Data Analytics in Healthcare*. Springer.

[4] J. Hirschberg and C. D. Manning, "Advances in natural language processing," *Science*, vol. 349, no. 6245, pp. 261–266.

[5] V. Mak and S. Lundstrom, "Predicting performance of parallel computations," *IEEE Trans. on Parallel & Distributed Systems*, vol. 1, no. undefined, pp. 257–270, 1990.

[6] D. Ardagna, E. Barbierato, A. Evangelinou, E. Gianniti, M. Gribaudo, T. B. Pinto, A. Guimarães, A. P. Couto da Silva, and J. M. Almeida, "Performance prediction of cloud-based big data applications," in *ICPE 2018*.

[7] E. Vianna, G. Comarela, T. Pontes, J. Almeida, V. Almeida, K. Wilkinson, H. Kuno, and U. Dayal, "Analytical performance models for mapreduce workloads," *International Journal of Parallel Programming*, vol. 41, no. 4, pp. 495–525, 2013.

[8] A. Verma, L. Cherkasova, and R. H. Campbell, "ARIA: Automatic Resource Inference and Allocation for MapReduce Environments," in *ICAC 2011*.

[9] S. Venkataraman, Z. Yang, M. Franklin, B. Recht, and I. Stoica, "Ernest: Efficient performance prediction for large-scale advanced analytics," in *13th NSDI*, 2016.

[10] X. Pan, S. Venkataraman, Z. Tai, and J. Gonzalez, "Hemingway: modeling distributed optimization algorithms," *arXiv preprint arXiv:1702.05865*.

[11] O. Alipourfard, H. H. Liu, J. Chen, S. Venkataraman, M. Yu, and M. Zhang, "Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics," in *14th NSDI*, 2017.

[12] A. Maros, F. Murai, A. P. Couto da Silva, J. M. Almeida, M. Lattuada, E. Gianniti, M. Hosseini, and D. Ardagna, "Machine learning for performance prediction of spark cloud applications," in *2019 IEEE CLOUD*.

[13] M. Atkinson, S. Gesing, J. Montagnat, and I. Taylor, "Scientific workflows: Past, present and future," 2017.

[14] W. A. Warr, "Scientific workflow systems: Pipeline pilot and knime," *Journal of computer-aided molecular design*, vol. 26, no. 7, pp. 801–804.

[15] F. Lordan, E. Tejedor, J. Ejarque, R. Rafanell, J. Álvarez, F. Marozzo, D. Lezzi, R. Sirvent, D. Talia, and R. M. Badia, "Servicess: An interoperable programming framework for the cloud," *Journal of grid computing*, vol. 12, no. 1, pp. 67–91, 2014.

[16] P. Pinoli, S. Ceri, D. Martinenghi, and L. Nanni, "Metadata management for scientific databases," *Information Systems*, vol. 81, pp. 1–20.

[17] M. Masseroli, P. Pinoli, F. Venco, A. Kaitoua, V. Jalili, F. Palluzzi, H. Muller, and S. Ceri, "Genometric query language: a novel approach to large-scale genomic data management," *Bioinformatics*, vol. 31, no. 12, pp. 1881–1888, 2015.

[18] A. Gulino, A. Kaitoua, and S. Ceri, "Optimal binning for genomics," *IEEE Transactions on Computers*, vol. 68, no. 1, pp. 125–138, 2018.

[19] M. Kudo and J. Sklansky, "Comparison of algorithms that select features for pattern classifiers," *Pattern recognition*, vol. 33, no. 1, pp. 25–41, 2000.

[20] R. Nelson and A. N. Tantawi, "Approximate analysis of fork/join synchronization in parallel queues," *IEEE transactions on computers*, vol. 37, no. 6, pp. 739–743.

[21] V. W. Mak and S. F. Lundstrom, "Predicting performance of parallel computations," *IEEE Transactions on Parallel and Distributed Systems*, vol. 1, no. 3, pp. 257–270.

[22] D.-R. Liang and S. K. Tripathi, "On performance prediction of parallel computations with precedent constraints," *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, no. 5, pp. 491–508.

[23] M. Bertoli, G. Casale, and G. Serazzi, "Jmt: performance engineering tools for system modeling," *ACM SIGMETRICS Performance Evaluation Review*, vol. 36, no. 4, pp. 10–15.

[24] K. Ousterhout, C. Canel, S. Ratnasamy, and S. Shenker, "Monotasks: Architecting for performance clarity in data analytics frameworks," in *SOSP 2017*.

[25] S. Mustafa, I. Elghandour, and M. A. Ismail, "A machine learning approach for predicting execution time of spark jobs," *Alexandria Engineering Journal*, vol. 57, no. 4, pp. 3767 – 3778.

[26] J. Shon, H. Ohkawa, and J. Hammer, "Scientific workflows as productivity tools for drug discovery." *Current opinion in drug discovery & development*, vol. 11, no. 3, pp. 381–388.

[27] T. P. Pham, J. J. Durillo, and T. Fahringer, "Predicting workflow task execution time in the cloud using a two-stage machine learning approach," *IEEE Transactions on Cloud Computing*, 2017.

[28] R. F. Da Silva, G. Juve, M. Rynge, E. Deelman, and M. Livny, "Online task resource consumption prediction for scientific workflows," *Parallel Processing Letters*, vol. 25, no. 03, p. 1541003, 2015.

[29] M. H. Hilman, M. A. Rodriguez, and R. Buyya, "Task runtime prediction in scientific workflows using an online incremental learning approach," in *IEEE/ACM UCC*, 2018.

[30] G. Kousalya, P. Balakrishnan, and C. P. Raj, "Workflow scheduling algorithms and approaches," in *Automated Workflow Scheduling in Self-Adaptive Clouds*, 2017, pp. 65–83.