2012 SC Companion: High Performance Computing, Networking Storage and Analysis

Predicting the Execution Time of Workflow Activities Based on Their Input Features

Tudor Miu and Paolo Missier
School of Computing Science
Newcastle University, Newcastle-upon-Tyne, UK
{t.a.miu@newcastle.ac.uk, paolo.missier@newcastle.ac.uk}

***Abstract*—The ability to accurately estimate the execution time of computationally expensive e-science algorithms enables better scheduling of workflows that incorporate those algorithms as their building blocks, and may give users an insight into the expected cost of workflow execution on cloud resources. When a large history of past runs can be observed, crude estimates such as the average execution time can easily be provided. We make the hypothesis that, for some algorithms, better estimates can be obtained by using the histories to learn regression models that predict execution time based on selected features of their inputs. We refer to this property as *input predictability* of algorithms. We are motivated by e-science workflows that involve repetitive training of multiple learning models. Thus, we verify our hypothesis on the specific case of the C4.5 decision tree builder, a well-known learning method whose training execution time is indeed sensitive to the specific input dataset, but in non-obvious ways. We use the case study to demonstrate a method for assessing input predictability. While this yields promising results, we also find that its more general applicability involves a trade off between the black-box nature of the algorithms under analysis, and the need for expert insight into relevant features of their inputs.**

I. Introduction

Computational science is frequently characterised by data-intensive, long-running applications that are executed a large number of times on Grid-based cyberinfrastructures, and increasingly on public cloud platforms. This paper describes our investigation into the prediction of execution times for such class of applications. Accurate runtime prediction is not only a pre-requisite for efficient scheduling [1], [2], but it may also result in monetary savings, as host time is one of the dominant factors in cost models for public cloud resources.

The problem of predicting execution time of tasks for the purpose of job scheduling has been addressed in various forms in the past [3], [4] and more recently in a cloud setting, to establish trade-offs between execution times and monetary cost [5]. While a more in-depth analysis of existing approaches can be found in Sec. IV, here we begin by clarifying the specific scope and goals of our work in the broader context of execution time prediction for e-science workflow-based applications.

A. Motivating example: a chemical engineering workflow

Our prime motivation for this work comes from workflow-based e-science [6], specifically from the case where the workflow includes computationally complex tasks which are executed a large number of times and whose execution time may vary significantly from one run to the next. We use one exemplar of such class of problems as a practical motivation for our work. The *Discovery Bus* (DB) [7] is a Chemical Engineering workflow used in the context of Quantitative Structure-Activity Relationships research (QSAR)[1]. DB uses a variety of machine learning algorithms, also known as *model builders*, to generate a large number of predictive models, which can be later used to predict the activity of chemical compounds based on their structure. An implementation of the DB workflow and its deployment on the Azure cloud platform, done by members of our group, is described in [8]. In this implementation, DB takes datasets consisting of molecular structures labelled with known activity as input, and runs a number of model builders on each such input dataset concurrently and in a competitive fashion, i.e., by choosing the one that performs best, using a pattern known as *Panel of Experts*. Operating on a large set of input datasets, the DB workflow generated over 750,000 predictive models using the eScience Central cloud-based workflow system[2], requiring a total of over 8 CPU-months [8]. For such workflows, the ability to predict the execution time of each of its experts, i.e., the model builders, readily translates in efficient workflow activity scheduling and it is useful for cost estimation.

B. Input predictability

This work is focused on services S, and more generally software components, which could be encapsulated as workflow activities for which a history H_S of executions is available, which minimally contains pairs $\langle d_k, t_k \rangle$ of input datasets d_k and corresponding execution times t_k. For such services, we would like to use H_S to learn a model that computes an estimate $\hat{t}_S(d)$ of the execution time for any new input d[3]. We use the term *input-predictability* to denote the accuracy of estimates $\hat{t}_S(d)$ that are obtained using only features extracted from d.

A notable example of an easily data-predictable algorithm is matrix multiplication. Here the dimensions of the input matrix determines the exact number of multiplications and summations performed. Multiplying an $m \times n$ matrix with

[1]www.openqsar.com.

[2]www.esciencentral.co.uk

[3]Note that, unlike similar approaches to performance estimation, this definition does not consider system-specific features. To achieve this isolation, we run our experiments on a single idle machine, as described later in Sec. III.

978-0-7695-4956-9/13 $26.00 © 2013 IEEE
DOI 10.1109/SC.Companion.2012.21

an $n \times p$ matrix requires mnp multiplications and $m(n-1)p$ additions. In this case, one can use regression to correlate these simple features with execution time.

Looking at this example, it is tempting to be drawn to traditional algorithmic complexity analysis as the main way of predicting performance. In contrast, our approach is purely based on learning from historical data. This has the main advantage that we can deal with *grey box* software components, such as web services or other types of scripted workflow tasks, whose implementation details may be unavailable.

On the other hand, clearly not all algorithms are input-predictable. Obvious counter-examples are algorithms whose behaviour is affected by random variables that are part of the input or initial states, such as iterative algorithms where the number of iterations depends on a randomly chosen initial solution. A well-known example is another model builder, namely the training phase of a feed-forward neural network using backpropagation [9]. Training is an iterative process that normally stops when a certain criterion measure, such as an error function on the training or validation set, reaches a sufficiently low value. In this case, making a cost-effective and accurate prediction of the number of iterations is not feasible.

C. Goals and contributions

In this paper we provide an operational definition of input predictability, we describe an experimental method for its quantitative assessment, and illustrate its use on a significant case study. The method involves exploring a space of estimation functions (regression models), each operating on a possibly different combination of features extracted from the input, to find the ones that maximize prediction accuracy. The ability to reliably predict execution time from the input has obvious benefits for scheduling, provided that computing the estimate is substantially less expensive than running an instance of the algorithm itself.

The determination of input predictability is a knowledge-intensive process and not all workflow services enjoy this property. To further define the scope of our work, we have chosen to focus on "commodity" services that appear frequently in e-science applications, such as the machine learning (model builder) algorithms mentioned earlier as part of the DB workflow. To illustrate the method, we have used the C4.5 decision tree builder (a well-known machine learning algorithm [10], [11]) as a case study.

Our main experimental findings are as follows.

- For C4.5, a variety of regression models can be trained to reach acceptable asymptotic prediction errors of about 20%, substantially better than the baseline predictor, consisting of the average of past observed execution times. This error measure is relative to the accuracy of the baseline predictor and is defined in Sec. III;
- The model only requires a small set F of features, which can be computed in time that is linear in the number of C4.5 input instances, thus with little overhead and much less expensively than running C4.5 itself;
- While input predictability with good asymptotic error can be obtained for C4.5, the optimal combination of regression methods and choice of input features depends on the specific type of data that makes up the history H_S, and on how homogeneous the data is over the history of observations. To make this observation precise, in our experiments we simulate histories of different length and with different data types for the same C4.5 implementation, and compare the asymptotic error rates obtained in each case.

Our work is aimed at delivering execution time predictions for individual data-driven algorithms, when their input is known. As these algorithms could be encapsulated as workflow activities, the predictions are for individual activities enactments, as opposed to predictions for entire workflow compositions.

D. Paper organization

The rest of this paper is organized as follows. In the next section we provide a more formal problem formulation, and describe our techical approach. Sec. III presents experimental results, followed by a discussion of related work (Sec. IV) and conclusions.

II. Technical approach

A. Problem formulation

One enactment s of a service S takes an input dataset d, and performs some operation in time $t = time(s(d))$. We assume that multiple enactments can be observed, resulting in a history $H_S = \{\langle d_k, t_k \rangle\}$ where $t_k = time(s_k(d_k))\}$. We also assume that the inputs d_k are all of the same type D, which is defined by S's input interface specification. For the sake of the example, and to set the context for our experiments, let us say that S is a *model builder*. In this case, all d_k consist of a set of training and test instances, and each $s_k(d_k)$ computes a predictive model (in the case of C4.5, a decision tree) in time t_k. Here the input type D defines a table with a set number of attributes (which are either numerical or categorical) and a variable number of records, each providing one training or testing example. An input d is such a table. A number of different features can be used to characterise D. For instance, in our example these may include the number of training instances, the number of numerical attributes, the average and standard deviation of the values of a numerical attribute over all instances, and so forth.

Our problem is to identify a combination F of features of D, along with a regression model M, such that $M(f) = \hat{t}$ is an estimate of $t = time(s(d))$, where $f = F(d)$ is the feature vector consisting of the values of F in d. Model M is trained on history H_S and is characterized by a prediction error ϵ, which in general is a function of the size $|H_S|$ of H_S. Regardless of the specific choice of metric used for ϵ, there is an expectation that, under our assumption of homogeneity of the inputs d_i, the error will converge asymptotically as $|H_S|$ increases, i.e., as more observations are accumulated during service operation.

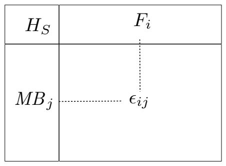


Fig. 1: Problem space: Model builders, features, and prediction error

Just as one can select a number of feature combinations F_i to characterise D, multiple model builders MB_j are available to induce an appropriate regression model M (in our experiments we have used the M5P model tree builder, the k-Nearest Neighbors algorithm for regression, and the Multi-Layer Perceptron, all described in [11]). The combination of these choices forms a two-dimensional space, where $\langle F_i, MB_j \rangle \rightarrow \epsilon_{ij}$ denotes that builder MB_j is trained using features F_i of D, resulting in asymptotic prediction error ϵ_{ij} (Fig. 1).

Thus, the problem of characterizing input predictability of S translates into seeking the combination with minimal error within this space. In the rest of the paper we report on our experience in addressing this problem for the specific case where S is C4.5. We find that this investigation is both knowledge-intensive, in the sense that human expertise is required to narrow down the space of possible combinations that are worth exploring, and data-intensive, as computing ϵ_{ij} requires a large number of executions s_i[4].

In summary, for a sequence of datasets d_k of type D, our approach involves (*i*) generating an execution history based on the d_k, (*ii*) selecting a small set of regression models, M_j $j : 1 \dots m$, (*iii*) selecting a set of feature sets F_i, $i : 1 \dots n$ for type D, and implementing corresponding feature extraction functions defined on D, and (*iv*) training each of the models M_j using each of the feature sets F_i.

B. Experimental datasets for C4.5

Our case study is aimed at showing that C4.5 has good input predictability. This involves populating the table in Fig. 1. Since predictability may depend on the specific data type D from which features are extracted, we performed three independent sets of experiments, using three separate sources of input datasets, corresponding to three different types D_i. These are drawn from the UCI Machine Learning Repository [12], namely *Poker Hand*[5], *Adult*[6], *KDD Cup 1999 10% sample*[7]. We chose these datasets because they have a large but manageable number of instances and attributes and they can be successfully used to train C4.5 models.

[4]Incidentally, note that this is a case of applying machine learning algorithms to predicting the execution time of a machine learning algorithm, using features of target model builder's training inputs.

[5]http://archive.ics.uci.edu/ml/datasets/Poker+Hand

[6]http://archive.ics.uci.edu/ml/datasets/Adult

[7]http://archive.ics.uci.edu/ml/datasets/KDD+Cup+1999+Data

C. Feature Selection

We experiment with two different types of features F that, for each d_k, can be computed in a time that is linear to the number of attributes in d_k. The first consists of a vector of *indicator* variables, one for each attribute defined by D, as well as the number of instances in d. Consistent with vertical sampling (i.e., selection of attributes from D), the indicator I_{kl} is a binary attribute that is set to 1 if d_k contains the l-th column of D, and 0 otherwise. This type of feature vector has the advantage that it mostly describes the schema of d_k. However its main drawback is the high dimensionality imposed on the feature space, which may result in low prediction accuracy on a small training set.

The second type of features includes the vector of *counts* of the numeric and categorical attributes in the input dataset, in addition to the number of instances. This results in a low dimensional feature space for predictors which may reduce time for training the predictor and/or for delivering predictions. The downside is that this feature space may not be expressive enough and this may lead to a pronounced bias component in the prediction error. For example, this method does not account for the presence of individual attributes or combinations of attributes which have greater impact on the training time of C4.5 (i.e. if they cause substantially more splits).

In practical terms, all scenario datasets have been converted into Weka's native ARFF format [11]. This is a tabular format encoded as a text file similar to CSV, which includes schema metadata such as attribute names and types.

D. Experimental setup

We restricted our analysis to three types of regression models: the M5P model tree builder, k-Nearest Neighbors (kNN) and Multi- Layer Perceptron (MLP). In addition, while we are interested in a general method that is agnostic of the nature of an algorithm, and thus ignores its theoretical complexity, in our experiments we have also compared our predictions with a linear regression model that is based on the complexity of C4.5 (see Sec. II-F).

The overall experimental setup is depicted in Fig.2, where the `FEx_*` blocks represent extractors for each of the selected feature sets, which are then selectively fed to the various regression models. As the figure shows, the `LR` (linear regression) model, discussed in Sec. II-F, is only trained on its own specific features.

All experiments were performed using J48, Weka's implementation of the C4.5 classifier [11], [13]. We trained J48 with every generated input and its execution time was measured in milliseconds on a single, idle desktop machine with an Intel Q9400 Core 2 Quad processor and 4GB of memory. We also used Weka's machine learning algorithms to train and evaluate all execution time predictors, except for the linear regression model which was implemented and evaluated in R [14].

E. Building histories

In the following, we consider each of the three D separately from the others. To generate a history of size n, we sample from d to obtain n different subsets $d_1 \dots d_n$. Each data point $\langle d_k, t_k \rangle$ in the history, called a *scenario*, is obtained by training C4.5 on d_k, and recording the corresponding execution time t_k. To give an idea of the distribution of the execution times, Fig. 3 reports these times for the Adult scenario. The shape of the distribution is very similar to the other scenarios. For practical reasons, the inputs were generated by limiting the maximum number of instances in an input such that the operational range of the execution time predictors be bounded.

Each d_k is obtained by sampling both *horizontally* (instances/rows) and then *vertically* (attributes/columns) from d. Horizontal sampling was performed using Weka's *Resample* filter. The filter requires as input the percentage of instances in the intended sample relative to the initial dataset. This percentage was generated randomly from a uniform distribution bounded by specified lower and upper bounds. The bounding was necessary to limit the execution time of the target algorithm. Vertical sampling was done by removing each non-class column from the initial dataset with a probability of 50%.

This sampling method generates inputs which, although synthetic, come close to real-world usage of machine learning algorithms. For example, vertical sampling emulates repeated enactments of the same algorithm for identifying an optimal subset of predictive attributes. This is known as *scheme-specific attribute selection* [11] where each subset of attributes is evaluated with the target machine learning algorithm. The subset of attributes with optimal performance, low dimensionality and perhaps domain knowledge bias is chosen for prediction purposes. Iterating through attribute subsets may reveal associations between attribute combinations and C4.5 training times.

Horizontal sampling is widely used for assessing the predictive power of machine learning schemes, i.e., when using k-fold cross-validation for improving performance estimation.

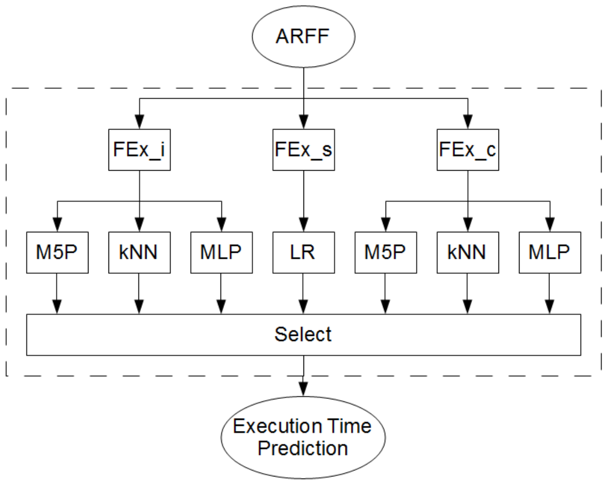


Fig. 2: Architecture for Evaluating C4.5 Execution Time Predictors.

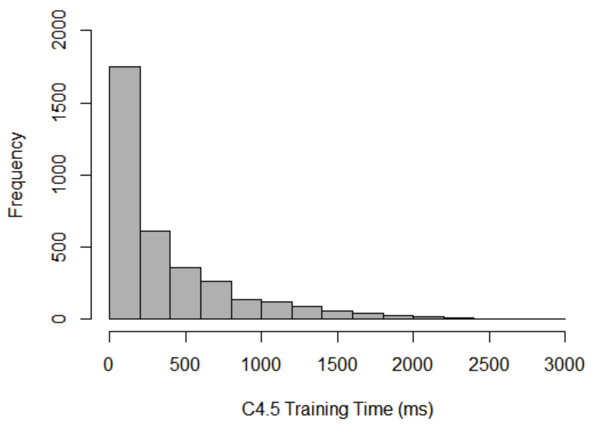


Fig. 3: Distribution of the execution times in the Adult scenario

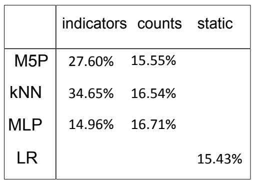
It is also a component of ensemble methods, such as bagging [15], which require building several models from subsets of a relation. These methods seek to derive a set of models whose predictions are combined by taking votes or averages. For example, bagging is a method of deriving a predefined number of models trained on datasets of equal number of instances that have been randomly sampled horizontally from a given training set. During prediction, each model gives an estimate and an unweighted vote or average is output as the final prediction. It is argued in [16] that bagging works especially well with trees, such as C4.5. Our prediction method would apply naturally to bagging because the execution time estimate is the same for all bagged trees, given that they are trained on datasets with equal number of instances and identical schemas.

F. *Parametric linear regression from analytical complexity*

The expected execution time of C4.5 for an input with N instances and p attributes is usually $O(pNlogN)$, but could be as high as $O(pN^2)$ [16]. The causes for such variations are encoded in the input data and this is what makes C4.5's training performance hard to predict[8]. In particular, training C4.5 involves the successive partitioning of the training set into subsets that contain a population of class labels which maximizes a certain performance criterion, typically cross-entropy [11]. For a fixed parametrization of C4.5, the set of splits, and thus the overall performance, is a function of the input, but it is hard to provide an analytical model of such function.

Although we are interested in regression methods that do not rely on knowledge about the computational complexity of algorithms, it is interesting to use the latter to construct a model for comparison with other regression models. From the complexity mentioned above, we derive the following linear regression parametric expression (referred to as `LR` in the

[8]Note that we focus exclusively on the training phase of C4.5, rather than in the operational/classification phase, which in comparison takes negligible time.



	indicators	counts	static
M5P	27.60%	15.55%	
kNN	34.65%	16.54%	
MLP	14.96%	16.71%	
LR			15.43%

Fig. 4: Populating the problem space with results from KDD Cup.

results), for a polynomial constructed from the terms that appear in the theoretical complexity formulation:

$$\hat{t} = \beta_1 p_{num} N log N + \beta_2 p_{cat} N log N + \beta_3 p_{num} N^2 + \beta_4 p_{cat} N^2$$

where p_{num} and p_{cat} are the counts of the numerical and, respectively, categorical attributes in the input dataset. The actual expression we used also accounted for the lower-order interactions terms between N, p_{num}, p_{cat} and $logN$ (not shown for simplicity). $\hat{t}$ is the expected response and β_j are the linear regression coefficients and are estimated by minimizing the expression with the method of ordinary least squares.

III. Experimental Results

In this section we present our experimental results. All machine learning and data mining tools we used are included in Weka [13] and R [14].

The error type we report is Relative Absolute Error (RAE) [11]. It is defined as

$$RAE = \frac{\sum_{i=1}^{N} |a_i - p_i|}{\sum_{i=1}^{N} |a_i - \bar{a}|}$$

where p_i and a_i are the predicted and, respectively, actual values of the class attribute in the test set. $\bar{a}$ is the mean of the class attribute in the training set. By definition, the RAE for a simple baseline predictor, one that consistently outputs the mean of the class attribute, is 100%. A value of less than 100% signifies improvement over the simple predictor.

A. Relative Performance of Runtime Predictors

We trained a single C4.5 model for every input dataset generated by vertical and horizontal sampling, recording the corresponding histories. We then sampled 3400 such runs from the histories, for each 3 scenarios, and used them to train and evaluate execution time predictors. Table I shows the 10-fold cross-validation RAE for each of the three selected regression models. Note that each set of results can be represented as one instance of the table in Fig. 1 above. For instance, the KDD Cup results can be represented as in Fig. 4.

For every predictor and feature extraction method we attempted several parametrizations, but we reported only the lowest RAE. Also included are the errors of linear regression models that use the terms discovered by static analysis. Figure 5 contrasts the performance of these models graphically.

We can observe that, since all errors are below 100%, all predictors exhibit improvement over a naive predictor. Overall, we were able to obtain an RAE below 20% on all datasets.

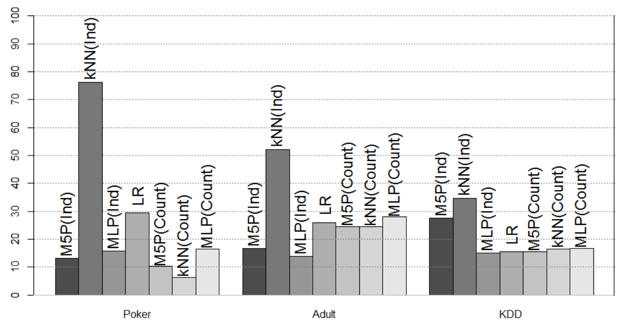


Fig. 5: Relative Absolute Error Measures for all predictors

Of all predictors, the Multi-Layer Perceptron, when trained with indicator variables, performs well across all datasets. However, this model builder is computationally expensive to train. Other model builders that are faster to train are the M5P and Linear Regression models, but they do not always perform well. Once trained, these models deliver predictions at a very small computational cost. kNN, on the other hand, compared to other model builders, has a relatively low training cost, but it requires significantly more computational effort for every prediction it makes, because it is a so-called *instance-based* learning method.

No interesting results can be reported from mixing histories across datasets, i.e., when execution times obtained from subsets of KDD Cup are considered together with those from Adult, for example. This confirms the need for our earlier assumption of homogeneity of the inputs observed within a

Scenario	No. Attributes	Model Builder/features	RAE
Poker Hand	11	M5P_indicators	13.09%
		kNN_indicators	76.23%
		MLP_indicators	15.80%
		LR_static	29.35%
		M5P_counts	10.29%
		kNN_counts	6.21%
		MLP_counts	16.50%
Adult	15	M5P_indicators	16.73%
		kNN_indicators	52.10%
		MLP_indicators	13.92%
		LR_static	25.97%
		M5P_counts	24.55%
		kNN_counts	24.40%
		MLP_counts	27.98%
KDD Cup	40	M5P_indicators	27.60%
		kNN_indicators	34.65%
		MLP_indicators	14.96%
		LR_static	15.43%
		M5P_counts	15.55%
		kNN_counts	16.54%
		MLP_counts	16.71%

TABLE I: Performance of execution time predictors across experimental datasets

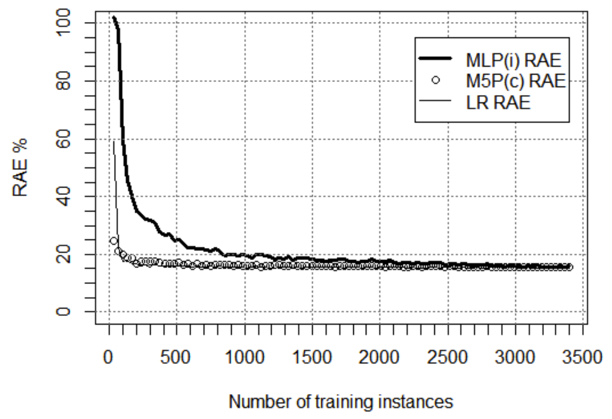


Fig. 6: RAE estimate as a function of the number of training instances for the KDD scenario.

history. One possible reason for this is that the execution time is dominated by hidden variables in the input dataset which we were unable to identify at a low computational cost. However, when not crossing scenario boundaries, intra-scenario variations can still be explained using the identified sets of features.

B. Predictive Power of Input Features

The Adult and KDD Cup scenarios show that using indicators to account for the presence of combinations of attributes is competitive, if not best. However, for Poker Hand, using the counts of numeric and categorical attributes as predictors is superior, except when compared to MLP. Closer examination of the dataset reveals that the numeric attributes correspond to the numbers on the five poker cards and the categorical attributes to the suit of each card. Clearly, there must be only two hidden causes which govern the splits and these must be uniformly distributed, the first one, across all numeric attributes and, the second one, across all categorical attributes. In this case, a feature space of attribute indicators does not account for substantially more information than a feature space of attribute counts. Results show that the latter compensates, by having a lower dimension and being more densely populated, which helps in training a more accurate predictive model.

The performance of linear regression is not always good. In Adult and KDD Cup it is comparable to other generic methods, but in Poker Hand it performed poorly. This shows that generic predictors may perform substantially better than methods that make use of intimate knowledge of the target algorithm, such as static analysis.

C. Asymptotic Error

Figure 6 shows the how the RAE evolves as the number of instances in the training set increases. We chose the KDD scenario and the best model builder for each feature set: MLP for indicators, M5P for counts. Also plotted is the RAE evolution of linear regression.

The RAE values were obtained by sampling the history incrementally from 1% to 100% of its size. For each sample size, we randomly drew 10 samples and performed 10-fold cross-validation on each sample for each of the three model builders. We then averaged the RAEs for each sample size and model builder and reported them in the plot.

Each RAE function graph exhibits an "elbow" - an abscissa value after which the model builder does not improve or oscillate substantially. This means that, after training the model with a sufficiently large number of instances, it makes little sense to rebuild the model with additional newly observed data because this will add very little in terms of accuracy. However, monitoring its performance is possible as actual execution times will always be available to be contrasted with the predicted ones. Some predictors may converge more slowly - their elbows occur later. An interesting example for this occurrence is the one plotted in Figure 6. All three model builders have very similar asymptotic RAEs, but the MLP model's elbow occurs later than the other two.

D. Effects of Horizontal and Vertical Sampling on Execution Time

Horizontal and vertical sampling have different effects on the execution of the target algorithm. A random horizontal sample of a dataset, if large enough, is likely to maintain the proportions of the populations of class labels of the original set. An initial split in the sample is decided solely on these proportions and, consequently, is likely to be the same as the split in the original dataset. By transitivity, it is likely that two different samples will initially split in the same way. Recursively, the same is true for the splits in sub-populations until the sizes of the populations become very small and proportions deviate significantly. Because this happens toward the leaves of the tree and the data is relatively less time-consuming to process, then the execution time for splitting this data is unpredictable, but small, and accumulates into a moderate prediction error.

Vertical sampling, which implies selecting a subset of columns, effectively removes potential splits that exist in the original subset. This will likely cause the sample to be split in a different way than the original dataset and may be associated to a different execution time. However, the predictive models will associate this combination of attributes with the recorded deviation in the execution time. How this is done depends on the internals of a predictor type. For instance, some predictors will not account for rare associations or for associations that correspond to very small deviations in the response variable.

It should be noted, however, that the results are affected by our data generation method. Firstly, the reported value at which the elbow occurs is arguably a pessimistic estimate of what would happen in reality. Suppose we start making predictions for bagging on a subset of attributes of a larger relation. Because all training datasets share the same schema and are of identical size, they are relatively homogeneous. In this case we would be able to obtain a stabilization of the RAE much sooner than with our data generation method because the latter yields more heterogeneous datasets.

Secondly, the lack of oscillations is an optimistic estimate. Continuing with our previous example, if we subsequently do bagging on a (partially) different set of attributes, then we can reasonably expect that the prediction errors on the first few iterations to be higher than the last ones from the previous round of bagging. This is because there is a sudden manifestation of heterogeneity. However, we do expect a stabilization of the error after more iterations in this case as well.

Finally, because the RAE is an error measure for an entire test set, not just a single test instance, the oscillations depend largely how often the model is rebuilt and re-evaluated. There are two ways in which to use the recent sequence of RAEs in a cost-effective manner. First, one may decide to stop refining the predictor if it does not show definite improvement after a few times it was re-trained with a reasonable amount of additional runs of the target algorithm. Second, by computing the RAE of an unchanged predictor on the most recent history of predictions, it is possible to detect oscillations which could justify the retraining of the predictor.

E. Choosing the Best Predictions and Best Predictors

Not all model builders are suitable for every dataset, at least not across the entire range of parametrizations, and there are many different feature sets on which to train these model builders. Clearly, there is a great deal of variation in the quality of the predictions, so, in the absence of any other prior knowledge, choosing the best model is a difficult problem. Fortunately, we have shown empirically that the quality of the predictor exhibits some characteristics which can be used to simplify the problem of choosing the best predictor or subset of predictors for a certain execution time prediction task.

If we are confident that we have explored sufficient heterogeneity in the input datasets, then, if the error sequence is stabilized, it means that the elbow has occurred and that we now have an accurate estimate of the lowest error of the predictor. If, at this point, there are other predictors exhibiting lower errors, either stabilized or not, then we can discard the predictor that stabilized at a high error.

However, while we can disconnect some predictors, we can still evaluate, without retraining, active predictors when new data is being used as input for prediction. Because the real execution time can be contrasted with its prediction, we can still evaluate active predictors by computing their error on the unseen data. If there is a subsequent oscillation this will be noticed by evaluation and this is an indicator for the need to re-train the predictors on the more recent case base.

IV. Related Work

Execution time prediction is an important topic in scheduling tasks on large computational infrastructures [17]–[20] and a number of studies have focused on workflow deployments on such infrastructures. For example, the authors of [18], [19] discuss predictions for entire workflows while in [20], [21] the modular character of workflows is exploited and the focus is on predictions for individual workflow activities. In those studies, the input is not anatomized to the extent it is in this paper. At most, generic attributes such as file names or problem sizes [18]–[20] are considered.

Many previous efforts have the merit of delivering predictions for different computational platforms modelled with system-specific parameters such as architectural parameters of processing stations [22] or even for platforms affected by variability [17], [21], [23], [24]. Unlike them, our approach focuses strictly on examining the input of a problem in great detail. Moreover, to the best of our knowledge, this paper is also the first to successfully attempt the prediction of the execution time of the complex C4.5 algorithm.

This research stems from our earlier, initial attempt at learning from execution histories, specifically to predict the accuracy of experts in our *Panel of Experts* pattern (mentioned in the Introduction) on particular input problems [25].

Focusing more on execution time prediction, Iverson *et al.* [26] provide execution time predictions for two algorithms, a Cholesky matrix decomposition algorithm and an algorithm determining whether a number is prime through trial divisions. Their input features are the dimension of the square matrix for the former algorithm and the actual input number of the latter. In addition, they share cases between different architectures because they consider code profiles as part of the input space. For runtime prediction, the authors use a version of k-Nearest Neighbours.

Matsunaga *et al.* [22] present a study of several machine learning schemes for predicting the execution time of two bioinformatics applications: BLAST and RAxML. Similar to Iverson *et al.* [26], they consider not only input features, such as the number of bases in a sequence for BLAST or taxa size and number of bases in a sequence for RAxML, but also system-specific attributes in order to characterize the effect of different computing architectures on the execution time of the application. We purposely factored out variability due to the computational platform by executing all experiments on a single idle machine in order to isolate the variability due to the input. While our experiments can be replicated on any platform, we do not consider how to transfer knowledge gained from a platform to another, even if performing experiments from a single scenario.

Kuperberg *et al.* [27] describe a method for deriving execution time predictions for Java components. They benchmark the application by counting the individual bytecode instructions that are being executed. These instructions are, on the one hand, translated alongside their counts to actual execution times on a given platform and, on the other hand, associated with the parameters input to the component. However, the only parameters extracted correspond to primitive Java types or basic collections and they do not provide an generic method for extracting features from custom objects. Instead, they suggest that a domain expert should specify the important features in this case. The downside of their method is that, by making use of bytecode benchmarking, it requires platform instrumentation.

Perhaps the most similar work to ours is by Krishnaswamy *et al.* [28], where they too consider estimating the execution time of Weka algorithms. However, they do not show evidence of what variability they introduced in their input. Although they consider additional input-agnostic parameters, the input to the algorithm is characterized only by the disk size of the relation. We believe that disk size is not a good predictor because this it is only marginally consistent with the information contained within a relation. Rather, the disk size depends solely on the encoding of the relation which is only weakly correlated with the contained information. Consider for example two categorical attributes that are perfectly correlated. Suppose that the domain of one is spanned by string values which are much longer than the strings in the other domain. All other things being equal, if we were to consider only one of the two attributes at a time, the execution times would be equal, but the sizes will differ. The argument is similar for numeric attributes. A string of trailing decimal zeros does not affect the numerical values, but it increases the size of the relation. The all-encompassing reason is that the information contained within each attribute can be encoded in infinitely many ways with no upper bounds for disk size.

Unlike previously mentioned work, our research stands out through the emphasis placed on characterizing the input to the target algorithm over several dimensions. We believe that this data-centric approach is suitable given that the target algorithm under study is essentially data-driven. A paper similar in focus to ours is by Kapadia *et al.* [29]. They made predictions for the execution time of a data-driven application using input features only, but in a different domain. Their reported shape of the error of the predictor as a function of the number of training runs is very similar to ours. However they, unlike us, do not discuss the potential advantages posed by such a regular error behaviour and they do not explore more than one feature space using varying degrees of prior knowledge about the algorithm under study. In addition, our work might be extended to other machine learning algorithms.

Selecting a subset of predictive models for execution time is not new, but previous efforts do not make use of input features and prior knowledge of the evolution of prediction errors. For example, Tao *et al.* [24] use time-series analysis for three cluster traces in turn. Glasner *et al.* [23] build clusters of applications according to historical attributes. Each cluster has a dynamically changing set of active time-series predictors. When a prediction for a task needs to be made, the clusters to which the task belongs are selected. For each such cluster, the predictions from each model are combined. Finally, the output predictions from the previously selected clusters are combined. However, neither approach is suitable for data-driven applications because the effects of input features on execution time may outweigh the effects of platform variability implicitly captured by the submission time or ordering of successive runs. In contrast to previous approaches, our proposed architecture includes the critical input feature extraction step where different sets of predictive attributes are defined, each with varying impact on the accuracy of the predictions. In addition, prior knowledge of the elbow or potential lack of oscillations can simplify the process of discarding poor predictors.

V. Conclusions

In the context of repetitive, computationally expensive e-science workflows, the ability to accurately estimate the execution time of the entire workflow or of their composing blocks is a pre-requisite for enabling better scheduling of the workflow on cloud resources, as well as to inform users of the expected cost of execution. We have investigated the hypothesis that, for certain algorithms, execution time can be accurately predicted based on features of the input that can be extracted inexpensively, given a sufficiently long history of observed past executions.

We have shown experimentally how effective predictive models for this property, which we call *input predictability*, can be built and have demonstrated the method on a specific case study involving the Weka implementation of the C4.5 machine learning algorithm. The method is designed to be applicable to black-box algorithms, however it involves exploring a space of candidate input features and predictive model builders. Exploring such a space efficiently remains a knowledge-intensive task. We plan to address this main limitation in future work.

Because of the encouraging results we obtained for C4.5, we contemplate to extend our method to other machine learning algorithms and evaluate which of these are amendable to input-based execution time prediction. Another potential research path is testing our prediction method against provenance traces from realistic workflow enactments, such as Discovery Bus, mentioned in Sec. I-A. In such a complex example where there are potentially many numerous input features that could be used, we would in addition consider automated feature selection techniques that could improve the accuracy of execution time prediction models.

References

[1] I. Rao and E.-N. Huh, “A probabilistic and adaptive scheduling algorithm using system-generated predictions for inter-grid resource sharing,” *J. Supercomput.*, vol. 45, no. 2, Aug. 2008.

[2] C. Lin and S. Lu, “Scheduling scientific workflows elastically for cloud computing,” in *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, july 2011.

[3] D. Tsafrir, Y. Etsion, and D. G. Feitelson, “Modeling user runtime estimates,” in *In 11th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP 2005*, 2005.

[4] D. Tsafrir, Y. Etsion, and D. Feitelson, “Backfilling using system-generated predictions rather than user runtime estimates,” *Parallel and Distributed Systems, IEEE Transactions on*, vol. 18, no. 6, june 2007.

[5] V. Viana, D. de Oliveira, and M. Mattoso, “Towards a cost model for scheduling scientific workflows activities in cloud environments,” in *Services (SERVICES), 2011 IEEE World Congress on*, july 2011.

[6] E. Deelman, D. Gannon, M. Shields, and I. Taylor, “Workflows and e-Science: An overview of workflow system features and capabilities,” *Future Generation Computer Systems*, vol. 25, no. 5, pp. 528–540, 2009. [Online]. Available: http://www.sciencedirect.com/science/article/B6V06-4SYCPKX-2/2/bb631979e3dd7071ddede90bbff65a91

[7] J. Cartmell, S. Enoch, D. Krstajic, and D. Leahy, "Automated QSPR through Competitive Workflow," *Journal of Computer-Aided Molecular Design*, vol. 19, no. 11, pp. 821–833, 2005. [Online]. Available: http://dx.doi.org/10.1007/s10822-005-9029-8

[8] J. Cala, P. Watson, and S. Woodman, "Cloud Computing for Fast Prediction of Chemical Activity," in *Procs. 2nd International Workshop on Cloud Computing and Scientific Applications (CCSA)*, Ottawa, Canada, 2012.

[9] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Neurocomputing: foundations of research," J. A. Anderson and E. Rosenfeld, Eds., 1988, ch. Learning representations by back-propagating errors.

[10] J. R. Quinlan, *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[11] I. H. Witten, E. Frank, and M. A. Hall, *Data Mining: Practical Machine Learning Tools and Techniques*, 3rd ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011.

[12] A. Frank and A. Asuncion, "UCI machine learning repository," 2010. [Online]. Available: http://archive.ics.uci.edu/ml

[13] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: an update," *SIGKDD Explor. Newsl.*, vol. 11, no. 1, Nov. 2009.

[14] R Development Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2008, ISBN 3-900051-07-0. [Online]. Available: http://www.R-project.org

[15] L. Breiman, "Bagging predictors," *Mach. Learn.*, vol. 24, no. 2, Aug. 1996.

[16] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition*, ser. Springer Series in Statistics. Springer, Sep. 2009.

[17] M. Dobber, R. van der Mei, and G. Koole, "Effective prediction of job processing times in a large-scale grid environment," in *High Performance Distributed Computing, 2006 15th IEEE International Symposium on*, 2006.

[18] F. Nadeem and T. Fahringer, "Predicting the execution time of grid workflow applications through local learning," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, ser. SC '09, 2009.

[19] ——, "Using templates to predict execution time of scientific workflow applications in the grid," in *Cluster Computing and the Grid, 2009. CCGRID '09. 9th IEEE/ACM International Symposium on*, may 2009.

[20] R. Duan, F. Nadeem, J. Wang, Y. Zhang, R. Prodan, and T. Fahringer, "A Hybrid Intelligent Method for Performance Modeling and Prediction of Workflow Activities in Grids," *2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pp. 339–347.

[21] X. Liu, Z. Ni, D. Yuan, Y. Jiang, Z. Wu, J. Chen, and Y. Yang, "A novel statistical time-series pattern based interval forecasting strategy for activity durations in workflow systems," *J. Syst. Softw.*, vol. 84, no. 3, Mar. 2011.

[22] A. Matsunaga and J. A. B. Fortes, "On the use of machine learning to predict the time and resources consumed by applications," in *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, ser. CCGRID '10, 2010.

[23] "Adaps a three-phase adaptive prediction system for the run-time of jobs based on user behaviour," *Journal of Computer and System Sciences*, vol. 77, no. 2.

[24] M. Tao, S. Dong, and L. Zhang, "A multi-strategy collaborative prediction model for the runtime of online tasks in computing cluster/grid," *Cluster Computing*, vol. 14, no. 2, Jun. 2011.

[25] P. Missier, "Incremental workflow improvement through analysis of its data provenance," in *Procs. TAPP'11 (Theory and Practice of Provenance)*, Heraklyion, Crete, Greece, June 2011.

[26] M. A. Iverson, F. zgner, and L. C. Potter, "Statistical prediction of task execution times through analytic benchmarking for scheduling in a heterogeneous environment," *IEEE Transactions on Computers*, 1999.

[27] M. Kuperberg, K. Krogmann, and R. Reussner, "Performance prediction for black-box components using reengineered parametric behaviour models," in *Proceedings of the 11th International Symposium on Component-Based Software Engineering*, ser. CBSE '08, 2008.

[28] S. Krishnaswamy, A. Zaslavsky, and S. W. Loke, in *9th International Conf. on Information Processing and Management of Uncertainty in Knowledge-Based Systems*.

[29] N. Kapadia, J. Fortes, and C. Brodley, "Predictive application-performance modeling in a computational grid environment," in *High Performance Distributed Computing, 1999. Proceedings. The Eighth International Symposium on*, 1999.