Enhancing Performance Prediction Robustness by Combining Analytical Modeling and Machine Learning*

Diego Didona[1], Francesco Quaglia[2], Paolo Romano[1], Ennio Torre[2]
[1]INESC-ID / Instituto Superior Técnico, Universidade de Lisboa
[2] Sapienza, Università di Roma

ABSTRACT

Classical approaches to performance prediction rely on two, typically antithetic, techniques: Machine Learning (ML) and Analytical Modeling (AM). ML takes a black box approach, whose accuracy strongly depends on the representativeness of the dataset used during the initial training phase. Specifically, it can achieve very good accuracy in areas of the features' space that have been sufficiently explored during the training process. Conversely, AM techniques require no or minimal training, hence exhibiting the potential for supporting prompt instantiation of the performance model of the target system. However, in order to ensure their tractability, they typically rely on a set of simplifying assumptions. Consequently, AM's accuracy can be seriously challenged in scenarios (e.g., workload conditions) in which such assumptions are not matched. In this paper we explore several hybrid/gray box techniques that exploit AM and ML in synergy in order to get the best of the two worlds. We evaluate the proposed techniques in case studies targeting two complex and widely adopted middleware systems: a NoSQL distributed key-value store and a Total Order Broadcast (TOB) service.

1. INTRODUCTION

Predicting the performance of applications and systems is a primary concern for various purposes such as capacity planning, elastic scaling and anomaly detection. Existing approaches to performance prediction typically rely on two, antithetic, techniques, namely Analytical Modeling (AM) and Machine Learning (ML).

AM has been, for decades, the reference technique to carry out performance evaluation and prediction of computing platforms, in a wide range of application contexts (see, e.g., [43, 20]). AM takes advantage of available expertise on the internal dynamics of systems and/or applications, and encodes such knowledge into a mathematical model aimed at capturing how (tunable) parameters map onto performance. AM techniques typically require no or minimal training in order to operatively carry out predictions in the target scenario, and have been shown to achieve a good overall accuracy. On the other hand, in order to be instantiated and/or be made tractable, AMs typically rely on simplifying assumptions on how the modeled system and/or its workload behave. Their accuracy can hence be seriously challenged in scenarios (i.e., areas of the features' space or specific workload conditions) in which such assumptions are not matched.

ML-based modeling lies on the opposite side of the spectrum, given that it requires no knowledge about the target system/application's internal behavior. Specifically, ML takes a black box approach that relies on observing the system's actual behavior under different settings in order to infer a statistical behavioral model, e.g., in terms of delivered performance. Over the last years, ML techniques have become more and more popular as tools for performance prediction of complex systems. Two are the main reasons behind this trend. On one side, the ever-increasing complexity of modern computing architectures represents a challenge for the accuracy of existing white box modeling techniques. On the other side, difficulties arise when employing white box models in virtualized, multi-tenant Cloud Computing environments, where details about the infrastructure physically hosting the application are normally (intentionally) hidden away from the users, restricting the possibility of employing detailed white box models for relevant parts of the system (e.g., the interconnection/networking infrastructure).

However, ML-based approaches are not the silver bullet for the problem of performance prediction. Their key drawback is that the accuracy they can reach strongly depends on the representativeness of the dataset used during the initial training phase. In fact, predictions targeting areas of the features' space that have not been sufficiently explored during the training process have typically very poor accuracy [2]. Unfortunately, the space of all possible configurations for a target application grows exponentially with the number of variables (a.k.a. features in the ML terminology) that can affect its performance — the so called *curse of dimensionality* [1]. Hence, in complex systems comprising large ecosystems of hardware and software components, the cost of conducting an exhaustive training process, spanning all possible input configurations, can quickly become prohibitive. Overall, pure ML approaches appear as not fully suited for contexts, like the Cloud, in which it is relevant

*This work has been supported by FCT - Fundação para a Ciência e a Tecnologia through PEst-OE/EEI/LA0021/2013, project specSTM (PTDC/EIA-EIA/122785/2010) and project GreenTM EXPL/EEI-ESS/0361/2013

ICPE'15, Jan. 31–Feb. 4, 2015, Austin, Texas, USA.

http://dx.doi.org/10.1145/2668930.2688047.

to promptly build models capable of determining configurations that guarantee optimal performance (and consequently resource usage).

In this paper we explore the problem of how to combine white and black box performance modeling and prediction methodologies by proposing and evaluating three techniques based on the common idea of building an ensemble of different methodologies. By exploiting AM and ML in synergy, we aim at building a performance model that is more *robust*, i.e., less prone to error than a model based on any of the two techniques implemented alone. The gray box techniques that we propose serve this purpose in a twofold fashion: *i*) by incorporating some ML component, they allow for increasing the prediction accuracy over time as new data from the operational system are collected; *ii*) by relying on a pre-built analytical performance model, they can be instantiated with a lower training time than conventional, pure ML-based predictors.

Particularly, we take inspiration from the literature on ensembles of ML models, which has been targeted at studying how to combine multiple black box ML techniques, and propose three algorithms that allow for the synergistic use of AM and ML models:

- *K Nearest Neighbors (KNN)*: during the learning process, this algorithm evaluates the accuracy that can be achieved by the selected AM model(s) of the target system and by one (or several) black box ML approaches (e.g., Decision Trees, Artificial Neural Networks, Support Vector Machines) in points of the features' space that were not included in the training sets used to build the ML-based learners (namely, a *validation set*). When used to predict the performance achievable in a configuration $\boldsymbol{c}$, the average error achieved by the AM model(s) and by the ML-based learner(s) across the K Nearest Neighbors configurations belonging to the validation set is used to determine which prediction method to choose.

- *Hybrid Boosting (HyBoost)*: in this technique a chain (possibly of length one) of ML algorithms is used to learn the residual errors of some AM. The intuition is that the function that characterizes the error of the AM may be learned more easily than the original target function that describes the relation between input and output variables. With this approach, the actual performance prediction in operative phases is based on the output by AM, adjusted by the error corrector function.

- *Probing (PR)*: The idea at the basis of this algorithm is to use ML to perform predictions exclusively on the regions of the features' space in which the AM does not achieve sufficient accuracy (rather than across the whole space). To this end two learners are exploited. Initially a classifier is used to learn in which regions of the features' space the AM incurs a prediction error larger than some predetermined threshold. In these regions, a second black box regressor is trained to learn the desired performance function.

All of the above algorithms allow for reducing the performance model instantiation time compared to pure ML techniques. In fact, either (i) the employed ML predictors do not need to reach extremely high precision across the whole features' space — given that they are complemented by white-box predictors (as it occurs in KNN) that can normally provide good accuracy in broad areas of the features' space; or (ii) they are targeted at estimating a function, namely the error curve associated with AM, which can be simpler (i.e., require less samples) to learn than the actual performance function (as it occurs in HyBoost); or (iii) they need to be trained only in circumscribed regions of the features' space (as it occurs in PR), which again can reduce the number of samples to be observed during the training phase.

Also, the structure of the framework is open to the possibility of using a family of AM techniques of recent interest (see, e.g., [12]), where parametric meta-models (requiring fewer assumptions on the target system than classical analytical models, hence widening their applicability) are fast trained, in order give rise to the actual AM instance suited for the target system. This has been shown to be doable by relying on a very reduced amount of samples of the real system behavior. Hence, the same (or a reduced portion) of training data that are used for the ML models envisioned in our framework, could be also used to carry out the meta-model training phase.

We assess the validity of our proposal through an extensive experimental evaluation carried out in two different application domains: throughput prediction of a popular open-source NoSQL distributed key-value store, Red Hat's Infinispan [25], and response time prediction of a total order broadcast service, a key building block for fault-tolerant replicated systems.

Our experimental results show that the best performing of our proposed techniques can reduce the Root Mean Square Error on average by about 40% with respect to AM and ML, with maximum gains that extend up to a factor 3× vs AM and 5× vs ML. On the other hand, they also show that none of the proposed ensemble techniques outperforms all the others in all the considered scenarios, and that their accuracy is strongly dependent on the correct determination of their internal meta-parameters. In this work we extensively investigate this issue and we highlight various interesting trade-offs that affect the parameters' tuning of the proposed algorithms.

The remainder of the paper is organized as follows. Section 2 discusses related work. In Section 3 we provide some background on ML techniques, which will form the basis for the comprehension of our proposal. The three innovative ensemble algorithms are presented in Section 4. The experimentation-based evaluation of the effectiveness of our proposals is provided in Section 5. Finally, Section 6 concludes the paper.

2. RELATED WORK

The body of literature on solutions relying either on AM or ML to predict applications' performance is extremely vast [29, 10, 35, 23, 8, 39, 42, 40]. On the other hand, to the best of our knowledge, only a few approaches rely on the synergistic exploitation of AM and ML. We group them in the discussion depending on how the combination of the two techniques is achieved.

Estimate and model. These works rely on ML to perform workload characterization and to estimate the service demand of the requests in the system. Next, this informa-

tion is used to instantiate an AM, e.g. based on queuing theory. Techniques employed to identify the parameters' values for the AM include regression [44, 9], clustering [34], Genetic Programming [18] or a combination of Kalman Filters and autoregressive models [45]. As ML is only employed to characterize the workload, the accuracy of these solutions is ultimately dependent on the accuracy of the adopted AM technique. The ensemble techniques proposed in our work, on the other hand, rely on ML to correct the inaccuracies of an analytical model, and can hence improve accuracy over time, as new sampling data is collected from the system being modeled.

Divide and conquer. This technique consists in building performance models of individual parts of the entire system, which are either based on AM or on ML. The sub-models are then combined according to some formula in order to achieve the prediction curve of the system as a whole. We find applications of this technique in the context of performance modeling of distributed transactional applications [14, 16] and response time prediction of Map-Reduce jobs [22]. In the former case, AM is employed to capture the effects of data and CPU contention on performance, whereas ML is employed to forecast response time of network-bound operations. In the latter one, AM is exploited to compute some performance metrics that are input features for the ML predictor.

The solution we propose in this work is fully complementary with respect to the divide and conquer approach. In fact, performance predictors resulting from the adoption of this technique can still show the limitations typical of the base AM and ML techniques at their core (resp. inaccuracies due to approximations and lengthy training phases). Our solution is specifically aimed at mitigating such limitations, by relying on ensembles of learners to increase accuracy (e.g., by discarding the output of some AM/ML predictor in specific operating points) while jointly reducing the cost of the training process. We demonstrate the effectiveness of our proposal by considering the divide and conquer-based model presented in [16] as the reference performance predictor for the NoSQL transactional platform case study.

Bootstrapping. This technique, which has been applied in various contexts ranging from automatic resource provisioning to anomaly detection, consists in relying on an AM predictor to generate an initial synthetic training set for the ML, with the purpose of avoiding the initial, long profiling phase of the target application under different settings [15]. Then, the ML is retrained over time in order to incorporate the knowledge coming from samples collected from the operational system [38, 32, 37, 33].

With respect to this solution, that only employs the AM to generate the initial training set for the ML, our ensemble-based forecasting techniques maintain the AM as a base predictor, and exploit different ML-based techniques to train complementary black box models aimed at correcting the AM's inaccuracies.

In a previous work [13], we have explored the possibility to infer at runtime, via a single ML, a *corrective function* that, applied to the output of some AM predictor, is able to increase the overall accuracy. The HyBoost ensemble that we propose in this work improves over that solution, particularly by allowing for the combination of multiple learners to compensate for the error of the base AM.

Generally speaking, one (additional) common shortcoming of the above discussed literature solutions is that they rely on a single ML in combination with an AM. This represents a major limitation to the degree of accuracy and predictive power that AM and ML, combined, could achieve: in fact, several independent results in the ML field identify in models' diversity and heterogeneity the key means to build a robust and accurate model with low training time [17, 4]. Our results back and extend this claim: by investigating different techniques of combining white box and black box models, relying in their turn on the exploitation of several MLs, not only we do assess the benefits of combining the two techniques, but we show evidence that there is not a single hybrid ensemble model that always outperforms the others.

Finally, it is worthy to note that nothing prevents our framework to be usable for combining ML with other kinds of white box predictors like simulation models. Although in principle these are generally considered as more expensive (in terms of time for being solved) as compared to AM ones, the vast literature on high performance parallel simulation techniques provides a good support for instantiating simulators allowing to promptly evaluate the behavior of complex systems (thanks to speedups by parallel runs [11, 3]). This would lead to the availability of white box simulation models with features that are still complementary to ML ones, such as reduced instantiation time, leading not to subvert the possibility to reach the actual targets of our proposal when employed as an alternative to AM in the presented ensemble algorithms.

3. BACKGROUND ON ML MODELING

Before presenting the proposed gray box ensemble techniques, we recall some basic concepts on ML-based techniques and introduce terminology that will be used in the remainder of the paper.

From a mathematical perspective, a ML algorithm, noted γ, is a function defined over a set, called *training set* and noted $D_{tr} = \{< \mathbf{x}, y >\}$, where $\mathbf{x} =< x_1, \ldots, x_n >$ is a point in a $n-$dimensional space, called *features' space* and noted F, and y is the value of some unknown function $\phi : F \rightarrow C$. In this paper we consider the case in which the co-domain C of function ϕ is the set $\mathbb{R}$ of real numbers, namely we consider a *regression* problem. The proposed techniques can however be straightforwardly adapted to cope with problems, known under the name of *classification* problems, in which the co-domain of ϕ is discrete.

The output of a ML algorithm γ is a function, called *model* and noted Γ, which represents an approximator of function ϕ over the features' space F. More precisely, a model $\Gamma : F \rightarrow C$ takes as input a point $\mathbf{x} \in F$, possibly not observed in D_{tr}, and returns a value $\hat{y} \in C$. The process of building a model using a ML algorithm γ over a given training set is also called training phase.

The literature on ML has proposed a number of alternative statistical approaches to infer the model Γ given a training set D_{tr}, like Decision Trees (DT), Artificial Neural Network (ANN) and Support Vector Machines (SVM). Independently of the specific approach used to derive Γ, these techniques pursue the same objective: minimizing the error of Γ on the training set, while preserving the ability to gen-

eralize the information observed during the training phase in order to provide accurate estimations of ϕ even in regions of the features' space that were not observed during the training phase.

Various definitions of error can be adopted to evaluate this trade-off, and, more in general, the accuracy of a prediction model (independently of whether it adopts a black or white methodology). In this paper we adopt as error function the Root Mean Square Error (RMSE), whose definition we recall in the following. Given a set of actual values $y_i \in Y$ and of corresponding predictions $\hat{y}_i \in \hat{Y}$, with $\hat{y}_i, y_i \in C$, the RMSE of $\hat{Y}$ with respect to Y is defined as:

$$RMSE(\hat{Y}, Y) = \sqrt{\sum_{\hat{y}_i \in \hat{Y}} \frac{(\hat{Y}_i - Y_i)^2}{|\hat{Y}|}}$$

4. GRAY BOX ENSEMBLE ALGORITHMS

In this Section we present the three different algorithms that exploit ML techniques in ensemble with a white box analytical model, denoted as Γ_{AM}. Before presenting the proposed techniques, we provide a generic mathematical formalization of Γ_{AM}.

Analogously to a ML-based model, an analytical model Γ_{AM} is a function $F_{AM} \rightarrow C$, which can be queried to predict the performance of the modeled system $\hat{y} = AM(\mathbf{x})$ over a given configuration $x \in F_{AM}$. For simplicity, we will assume in the following that $F_{AM} = F_{ML}$ and refer to them by simply using the notation F. In other words, we assume that the domain F_{AM} over which the analytical model Γ_{AM} is defined coincides with the features' space, noted F_{ML}, used by the ML techniques that will be used to learn a correction function for Γ_{AM}. In practice, this assumption is not strictly required, and we simply require that the variables defining the features' space are observable, i.e., they can be measured in the target system. For instance, the white box model AM may actually use a smaller subset of the variables defining the features' space of the black box learners used in ensemble with AM. This could happen, for instance, if the AM were not to account for a set of parameters, say $P \notin F_{AM}$, whose effects on system's performance may be too hard to model explicitly via analytical models. The parameters in P could, however, be incorporated in the features' space F_{ML}, so as to keep their value into account when learning the target function.

The key difference of an analytical model Γ_{AM} with respect to a ML-based model Γ is that the latter is obtained by running a ML algorithm over a training set D_{tr} (i.e., $\Gamma = \gamma(D_{tr})$). Hence, whenever new observations are incorporated in the training set, yielding an updated training set $D'_{tr} \supseteq D_{tr}$, an updated version of the ML-based model $\Gamma' = \gamma(D'_{tr})$ can be computed by training the ML-based learner on D'_{tr}.

Conversely, an analytical model Γ_{AM} incorporates *a priori* domain knowledge on the target system, and it does not require a training phase nor can be dynamically updated. In other words, we consider the analytical model Γ_{AM} to be a static/immutable object, which cannot be updated based on the feedback obtained from the target system.

One may note that analytical models typically rely on a number of internal parameters, which can be used to calibrate the model's output. Such parameters could be updated, via fitting techniques [26], in order to minimize the error achieved by the AM over the set of performance samples gathered over time from the target system. Also, as discussed in Section 2, gray box performance modeling techniques based on the *divide-and-conquer* approach, couple analytical and ML-based models targeting different, but dependent, subcomponent of the system. Whenever the ML-based models are updated, this leads to changes of the input parameters for the white box analytical models. From this perspective, hence, these gray box techniques can be seen as equivalent to white box analytical models whose internal parameters can be dynamically adjusted.

It is worth noting that, by assuming the analytical model Γ_{AM} to be an immutable object, we can ensure that the proposed techniques can also be employed in case Γ_{AM} can be dynamically updated. To this end, it simply suffices to treat the updated white box model $\Gamma_{AM'}$ as a new/different model. On the other hand, having not to impose such an assumption, we would allow the usage of techniques (e.g., ensemble techniques designed for "re-trainable" ML-based learners) that may not be applicable in case the analytical model was actually static.

As already mentioned, we present in the following three ensemble techniques that pursue the same objectives (minimizing training time and achieving an accuracy better or comparable to that of both black and white box techniques) using different algorithmic approaches. In the light of the above considerations, the proposed techniques can be seen as instances of ensemble techniques for ML-based learners, specialized for the case in which one of the learning algorithms in the ensemble outputs always the same model, namely the one coded in the AM formulas, which is essentially independent of the actual ML training set.

Algorithm 1 K Nearest Neighbors

```
1: Set Γ = ∅                                          ▷ Set of models to use
2: Set γ = {γ_1, ..., γ_M}                            ▷ Set of ML regressors
3: Set D_val = ∅                                      ▷ Validation set
4:
5: function INIT(Analytical Model Γ_AM, Training Set D_tr)
6:     Γ = {Γ_AM}                                     ▷ Initialize with the AM model
7:                                  ▷ Build the training set for ML regressors
8:     Set D_regr = StatifiedSample(D_tr)
9:                                  ▷ Use a disjoint data set as Validation set
10:    D_val = D \ D_tb
11:    for m = 1 → M do
12:        Γ_m = γ_m(D_tb)                            ▷ Train m-th regressor
13:        Γ = Γ ∪ {Γ_m}
14:    end for
15: end function

16: function FORECAST(x_s)
17:    Set D_k={<x_i,y_i> ∈KNN(x_s, D_val) s.t. ||x_i,x_s|| < c}
18:    for each Γ_i ∈ Γ do
19:      RMSE[i] = compute RMSE of model Γ_i on set D_k
20:    end for each
21:    μ = argmin_{i=1...M} RMSE[i]                   ▷ Find learner with lowest RMSE
22:    return Γ_μ(x_s)
23: end function
```

4.1 K Nearest Neighbors

The pseudo-code of the first presented technique, which we call K Nearest Neighbors (KNN), is reported in Algorithm 1. This technique relies on an analytical model, noted Γ_{AM}, and on a set γ of M alternative prediction models, noted $\gamma_1, \ldots, \gamma_M$. These predictors in γ should be selected to maximize model diversity, which can be achieved in various ways. A first technique consists in considering different ML algorithms, e.g., DT and ANN. One can also train each

learner γ_i using a different training set, with the purpose of specializing the various models to predict performance in different regions of the features' space. Model diversity can also be promoted by using different analytical models (focused on capturing different systems' dynamics), or even alternative modeling techniques such as simulation.

The KNN algorithm is initialized via the INIT function, by providing Γ_{AM} and a data set of samples, $D_{tr} =< \mathbf{x}_i, y_i >$, which conveys information on the performance $y_i \in C$ of the target system over a set of observed configurations $\mathbf{x}_i \in F$. The data set D_{tr} is not entirely used to train the set Γ of regressors. Conversely, D_{tr} is split into two disjoint data sets, namely D_{regr} and D_{val}.

D_{regr} is used as training set for the learners in Γ, and it should be obtained by extracting a random subset amounting to a percentage p_{regr} of D_{tr}. In order to enhance the representativeness of the samples included in D_{regr}, the process of extraction of D_{regr} from D_{tr} is performed by means of the stratified sampling technique [2], which ensures that the distribution of the values $y_i \in C$ is the same in the two sets.

D_{val} is obtained as the complementary subset of $D_{regr} \in D_{tr}$, which ensures the disjointness of the two sets D_{regr} and D_{val} by construction. The D_{val} is used at query time (Function FORECAST), when one wants to predict the expected performance of the target system, noted y_s, in the configuration $\mathbf{x}_s$. To this end, it is first computed the set D_k that contains the k nearest neighbors $\{\mathbf{x}_1,\ldots,\mathbf{x}_k\} \in D_{val}$ within distance c from point $\mathbf{x}_s$. The samples in D_k, for which we have available also the corresponding actual performance, are then used to compute the average accuracy of each of the models in the set Γ (Line 19). This allows for determining the model, noted γ_μ in the pseudo-code (Line 21), which is expected to maximize prediction accuracy in the region surrounding $\mathbf{x}_s$. Based on this geometrical interpretation, the c parameter can be interpreted as a cut-off threshold, which allows discarding samples of the validation set that are too far away from $\mathbf{x}_s$ and which may not be representative of the target configuration $\mathbf{x}_s$.

The relevance of ensuring the disjointness of D_{val} and D_{tr} can be understood by recalling that samples $\mathbf{x} \in D_{tr}$ are used to train the regressors in Γ. Estimating the accuracy of these models using the same samples that were used to derive the models during the training phase would lead to significantly overestimate the accuracy achievable by, so called, over-fitted models, i.e., models that minimize (or even nullify) the error with respect to the configurations observed during the training phase, but which are unable to generalize and thus incur large errors even in regions in the proximity of points contained in the training set.

4.2 Hybrid Boosting

The second algorithm we present applies a well-known technique from the literature on ensembles of black box learners, which is known as Boosting [2]. In particular, as we are considering a regression problem (whereas the boosting technique was defined for classification problems), we draw inspiration from the *Adaptive Logistic Regression* technique [19]. This is a boosting algorithm that was originally conceived to operate with ML-based regressors, and which we adapted to support the joint usage of one analytical model and of a set of black box learners.

Algorithm 2 Hybrid Boosting

1: Set $\gamma^{red} = \{\gamma_1^{red}, \ldots, \gamma_M^{red}\}$ ▷ ML regressors for residue pred.
2: Set $\Gamma^{red} = \{\Gamma_1^{red}, \ldots, \Gamma_M^{red}\}$ ▷ Models for residue pred.
3: Set $\Gamma^{per} = \{\Gamma_0^{per}, \Gamma_1^{per}, \ldots, \Gamma_M^{per}\}$ ▷ Models for perf. pred.
4:
5: **function** INIT(Analytical Model Γ_{AM}, Training Set D_{tr})
6: $\Gamma_0^{per} = \Gamma_{AM}$ ▷ Set the AM as the 1^{st} predictor
7: **for** $m = 1 \rightarrow M$ **do**
8: $D_m = \emptyset$
9: **for each** $<\mathbf{x}_n, y_n> \in D_{tr}$
10: $y_{m,n} = y_n - \Gamma_{m-1}^{per}(x_n)$ ▷ Compute the residual error
11: $D_m = D_m \cup < x_n, y_{m,n} >$ ▷ of previous learner
12: **end for each**
13: $\Gamma_m^{red} = \gamma_m^{red}(D_m)$ ▷ Train on the residuals
14: $\beta_m = \underset{\beta}{\operatorname{argmin}} \sum_{n=1}^{N} y_n - (\Gamma_{m-1}^{per}(x_n) + \beta\Gamma_m^{red}(x_n))$
15: $\Gamma_m^{per} = \Gamma_{m-1}^{per} + \beta_m\Gamma_m^{red}$ ▷ Set the m-th predictor
16: **end for**
17: **end function**

18: **function** FORECAST(x_s)
19: **return** $\Gamma_0^{per}(\mathbf{x}_s) + \sum_{m=1}^{M} \beta_m\Gamma_m^{red}(x_s)$
20: **end function**

The pseudo-code of this technique, which we name Hybrid Boosting (HyBoost), is reported in Algorithm 2. In addition to the analytical model Γ_{AM}, also in this case we assume the availability of a set of M regressors based on machine learning techniques, which we denote γ^{red}. Unlike in KNN, however, these learners are not used to build alternative models of the performance of the target system. Conversely, the learners are stacked in a chain (i.e., an ordered set) and used to learn the error (residue) introduced by the previous learner in the chain.

More in detail, HyBoost uses two (ordered) sets of predictive models, noted Γ^{red} and Γ^{per}, composed by, respectively, m and $m+1$ models. The first model in Γ^{red}, i.e., Γ_1^{red}, is obtained by training the first regressor γ_1^{red} with a training set D_i that characterizes the error (defined as the difference between the actual and predicted value) of the analytical model Γ^{AM} for each point in the original training set D_{tr}. Any other model Γ_i^{red}, with $i \in [1, M]$, is trained to learn the prediction error of the model Γ_{i-1}^{per}, which incorporates the knowledge of the AM and of the first $i-1$ ML-based learners by means of the following recurrence equation (Line 15):

$$\Gamma_m^{per} = \Gamma_{m-1}^{per} + \beta_m\Gamma_m^{red}$$

where β_m is a coefficient (computed in Line 14) such that the cumulative training error of the resulting m-stage regressor is minimized.

The key intuition at the basis of this algorithm, as already hinted, is that learning the residual errors of an analytical model may be easier than learning the original function for which we are trying to build a robust predictor. Also HyBoost, analogously to KNN, can exploit machine learners using different algorithms. Moreover, it may be further extended and optimized using well-known techniques in the literature on boosting ML-algorithms, such as adaptively weighting the elements in the training set of the i-th learner in order to focus it on minimizing its fitting error on samples over which the $i-1$-th learner incurred the largest errors.

4.3 Probing

We named the last of the three presented techniques *Probing*, and we reported its pseudo-code in Algorithm 3. This approach, which to the best of our knowledge has no direct

Algorithm 3 Probing

```
1: Classifier γ_cls                                  ▷ Detects when Γ_AM is wrong
2: ClassificationModel Γ_cls
3: Regressor γ_reg                                   ▷ Learns φ in areas where Γ_AM is wrong
4: RegressionModel Γ_reg
5: Set D_bad, D_cls, D_good = ∅                      ▷ Initialize data sets
6:
7: function INIT(Analytical Model Γ_AM, Training Set D_tr)
8:     for each <x_n,y_n> ∈ D_tr
9:     if |(y_n − Γ_AM(x_n))/y_n| ≥ c then
10:        D_bad = D_bad ∪ {< x_n, y_n >}
11:        D_cls = D_cls ∪ {< x_n,"bad">}
12:    else
13:        D_cls = D_cls ∪ {< x_n,"good">}
14:    end if
15:    end for each
16:    Γ_cls = γ_cls(D_cls)                          ▷ Train the "good/bad" classifier
17:    Γ_reg = γ_reg(D_bad)                          ▷ Train the regressor for samples in D_bad
18: end function

19: function FORECAST(x_s)
20:    if Γ_cls(x_s) ="bad" then
21:        return Γ_reg(x_s)
22:    else
23:        return Γ_AM(x_s)
24:    end if
25: end function
```

correspondence in the literature on ensembles of ML-based learners, uses 2 ML-based learners:

1. a classification algorithm, noted γ_{cls}, to learn *where* (i.e., in which regions of the features' space) the analytical model is not sufficiently accurate (based on a parametric threshold c over the absolute percentage error);

2. a regression algorithm, noted γ_{reg} which is trained to learn the function ϕ describing the performance of the target systems exclusively in the regions in which the analytical model does not achieve adequate accuracy.

To this end, during the initialization phase, each sample $< \mathbf{x}_i, y_i >$ in the training set is classified as either "good" or "bad" based on the absolute percentage error achieved by the analytical model when queried for $\mathbf{x}_i$ (Lines 8-15). In addition, whenever a sample $< \mathbf{x}_i, y_i > \in D_{tr}$ is classified as bad, it is included in the data set D_{bad} that is used to train the black box regressor γ_{reg}.

When queried to predict the performance of the system on configuration $\mathbf{x}_s$, it is first determined, using the classification model Γ_{cls} whether the analytical model Γ_{AM} is expected to achieve good or bad accuracy, and use, accordingly, either Γ_{AM} or the black box model Γ_{reg}.

The intuition underlying this technique is that, if the errors of the AM are focused in restricted and easily identifiable regions (via Γ_{cls}), one can then specialize the training phase of a black box learner exclusively on those regions. By narrowing the scope in which the ML-based learner Γ_{reg} is used to the regions of high error for Γ_{AM}, the complexity of the function that needs to be learnt via ML may be reduced, which may ultimately benefit the accuracy of Γ_{reg}.

5. EVALUATION

This section is devoted to assess the effectiveness of the proposed gray box modeling techniques by means of an extensive experimental evaluation based on two case studies on middleware systems: the Appia Group Communication Toolkit [28] and a popular open-source distributed key-value store, Infinispan [25] by Red Hat. For each middleware platform, we consider two recently proposed performance prediction models [14, 32], which we use as a first baseline and which we combine with different ML approaches (Decision trees, Neural Networks and SVM) via the proposed gray box techniques.

We start by presenting the two case studies, and the corresponding performance models in Section 5.1. In Section 5.2 we evaluate the impact of the key parameters of the presented gray box modeling techniques on their accuracy. Finally, Section 5.3 focuses on comparing the accuracy and training/querying time of the proposed solutions and of a number of alternative performance modeling approaches.

5.1 Case studies

In order to evaluate experimentally the effectiveness and the sensitivity to parameters of the gray box ensemble methods discussed in the previous section, we consider two case studies: *i*) response time prediction of a Sequencer-based Total Order Broadcast (STOB) service, implemented in Appia [28] and *ii*) throughput prediction of an application deployed over a popular distributed transactional key-value data store, Red Hat's Infinispan (v. 5.2) [25]. We consider these two middleware systems for two main reasons. First, because of their relevance and wide adoption, they allow to demonstrate the viability of the proposed techniques when applied to mainstream software. Second, because of the diversity of the corresponding performance prediction problems: the features' spaces of the two case studies have very different dimensionality (2 for STOB vs 7 for Infinispan), and the corresponding analytical models exhibit different distribution of errors. This allows us to evaluate the proposed solutions in very heterogeneous scenarios, increasing the representativeness of our experimental study.

STOB primitive. Total Order Broadcast (TOB) [21] is a primitive that allows a group of processes to achieve consensus on a common delivery order of messages that can be broadcast (possibly concurrently) by processes in this group. TOB is a fundamental building block at the basis of a number of fault-tolerant replication mechanisms for databases [30], transactional memory [7] and highly-available objects [27]. TOB algorithms based on sequencer processes (STOB) [28] are probably among the most widely deployed TOB protocols, as they achieve the minimum bound on message latency for the TOB problem. In failure-free runs of the

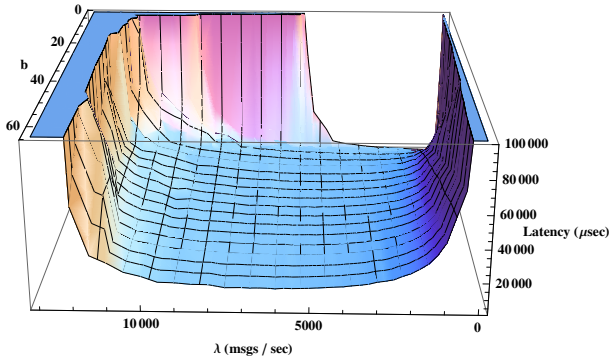
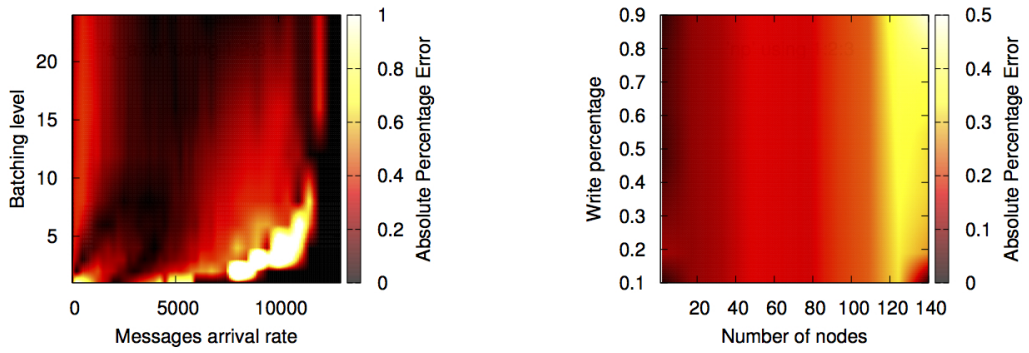


Figure 1: Response time of the STOB service



(a) STOB

(b) Infinispan

Figure 2: Error distribution of Γ_m of the two case studies.

STOB algorithm, if no processes leave or join the group, the processes agree on the identity of a single process, before starting to totally order broadcast (TO-Bcast) messages. Such a process, called *sequencer*, has the role to impose a common total order of delivery on messages to all processes in the group. A total order broadcast of a message is supported via the execution of a plain broadcast of the message by the sender process. When a process receives a message from the network, however, it cannot immediately deliver it to the application. In order to guarantee group-wide agreement on the final delivery order, in fact, it has first to wait to receive from the sequencer the corresponding *sequencing* message, and to ensure that all previously ordered messages have been delivered. The batching level, denoted in the remainder as b, defines how many messages the sequencer waits to receive before generating a sequencing message. Setting b to 1 ensures minimal latency at low load; at high load, however, higher values of batching lead to amortize the cost of sequencing each message, and allow the sequencer to sustain higher throughput.

The AM adopted in this work as white box predictor of the latency of a STOB algorithm has been proposed in [32], in order to automate the tuning of the batching level in function of the message arrival rate λ. This is a relatively simple model, which represents the sequencer node as a $M/M/1$ queue [24], and, based on purely analytical methods, computes the STOB message delivery as the traversal time of a client in the queue.

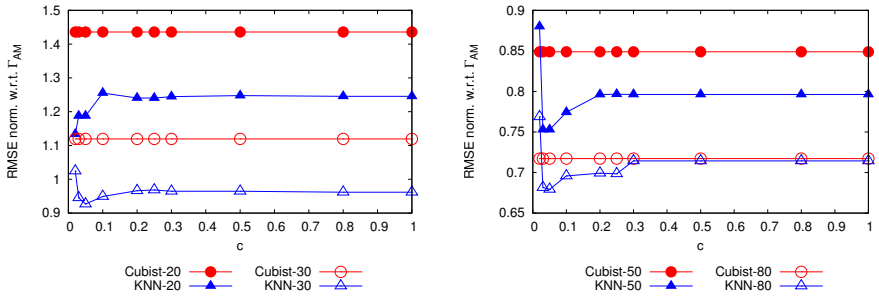
This case study is interesting because, although the parameters characterizing the system's behavior are only two (message arrival rate and batch size), the resulting performance function (shown in Figure 1) exhibits non-linear behavior. This is typical of queuing systems [24], as the response time quickly grows to infinite when the message arrival rate approaches the maximum service rate sustainable by the sequencer (given the current batching level b). It is well known that most ML techniques can be challenged when faced with functions having accentuated non-linear behaviors. Moreover, the error distribution of the corresponding analytical model is particularly interesting as the error looks generally low, except for a very specific zone of the input parameters' space. Such localized error is depicted in Figure 2(a), where the the messages arrival rate is on the x-axis and the batching level on the y-axis.

Infinispan. NoSQL data stores have emerged as reference data platforms for the Cloud: they adopt less expressive data models than the classic relational one, and opt for simpler, yet more scalable, paradigms, as in key-value stores; moreover, in order to enhance performance, these systems typically maintain data fully in-memory and rely on replication as their primary mechanism to ensure fault- tolerance and data durability. Infinispan is a popular NoSQL data store which, analogously to other recent cloud platforms [6, 36], provides support for strong consistency via the abstraction of atomic transactions.

Predicting the performance of such platforms is far from being a trivial task, as it is impacted by several factors: contention on physical (i.e., CPU and network) and logical (i.e., data) resources, characteristics of the transactional workload (e.g., conflict likelihood and transactional mix) and configuration of the platform itself (e.g., scale and replication degree). This case study is, thus, an example of a modeling/learning problem defined over a very vast dimensional space (spanning 7 dimensions in our case) and characterized by a very complex performance function.

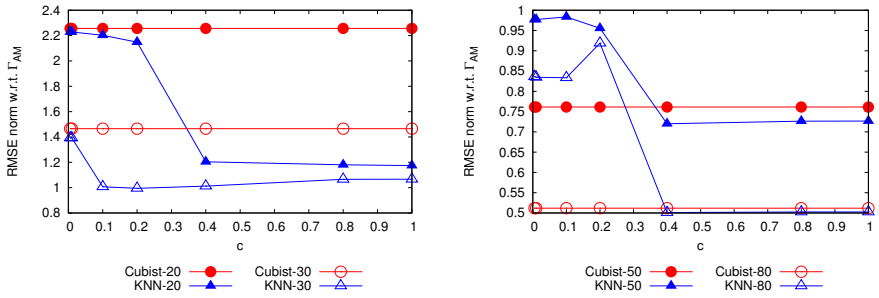
The reference model that we employ as base predictor for this case study is PROMPT [16]. PROMPT relies on the divide-and-conquer approach described in Section 2. On one hand, it exploits the knowledge of the concurrency and replication scheme (e.g., Two-Phase Commit) employed by the data platform to capture the effects of workload and platform configuration on CPU and data contention via a white box analytical model. On the other hand, it relies on ML to predict latencies of network bound operations.

Overall, besides involving a much wider features' space, this case study is particularly interesting as it allows us to evaluate the effectiveness of our approach also when used in combination with another gray box modeling technique. In Figure 2(b) we also visualize, again by means of a heat-map, the error distribution of PROMPT, obtained by projecting the features' space over two dimensions, namely number of nodes in the system (on the x-axis) and percentage of write transactions (on the y-axis). It can be noted that the error distribution of PROMPT is more diffuse than for the case of



(a) Training set 20% and 30% (b) Training set 50% and 80%

Figure 3: Sensitivity analysis of KNN w.r.t. the c parameter (STOB)



(a) Training set 20% and 30% (b) Training set 50% and 80%

Figure 4: Sensitivity analysis of KNN w.r.t. the c parameter (Infinispan)

STOB (see Figure 2(a)), which was characterized by intense spikes in clearly circumscribed regions of the features' space.

Experimental dataset and test bed[1]**.** For the STOB case study, we consider a data set containing a total of five hundred observations drawn from a cluster of 10 machines equipped with two Intel Quad-Core XEON at 2.0 GHz, 8 GB of RAM, running Linux 2.6.32-26 server and interconnected via a private Gigabit Ethernet. In the experiment performed to collect the samples, the batching level was varied between 1 and 24, and 512-byte messages were injected at arrival rates ranging from 1 msgs/sec to 13K msgs/sec.

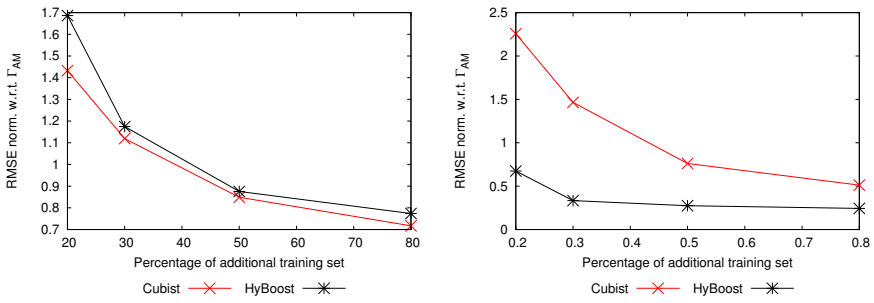
As for the transactional application case study, we consider a dataset composed by approximately five hundred samples, collected by deploying Infinispan on a cloud infrastructure composed by 140 Virtual Machines (VM) equipped with 1 Virtual CPU and 2GBs of RAM; each VM runs a Fedora 17 Linux distribution with kernel 3.3.4- 5.fc17.x86 64. The physical infrastructure hosting the cloud is composed by 18 physical servers equipped with two 2.13 GHz Quad-Core Intel(R) Xeon(R) processors and 32 GB of RAM and interconnected via a private Gigabit Ethernet; the employed virtualization software is Openstack Folsom.

The considered application is a porting of YCSB [5], the *de facto* standard benchmark for key-value stores, which has been modified in order to support transactions. The generated workloads are A, B and F: workload A has a mix of 50/50 reads and writes, and models a session store recording recent actions; workload B is the one of a photo tagging application, which contains a 95/5 reads/update mix; workload F models a user database, in which records are first read and modified within a transaction. In order to generate a wider set of workloads, we also vary the number of reads and writes performed by transactions between 1 and 5. Finally, we consider two different data access patterns: Zipfian, i.e., the popularity of data items follows the zipf distribution (with zipfian constant 0.7), and Hot Spot, according to which the x% of the data accesses are biased towards the y% of the data items (with $x = 99$ and $y = 1$ in our case); the data set is always composed of 500K keys. The samples relevant to the application's throughput are collected while varying workloads and the data platform configuration, deployed on a number of nodes, noted N, ranging from 2 to 140 and set up with a replication factor in the set $\{1, 2, 3, \frac{N}{2}, N\}$.

[1]Dataset in Weka format and source code are available at https://github.com/EnnioTorre/CombiningMultiplePredictors

5.2 Analysis of Parameters' Sensitivity

In this section we evaluate the sensitivity of the proposed ensemble techniques with respect to their key parameters, namely the cut-off threshold c (for KNN and Probing) and the number M of black-box learners exploited in the ensemble. We consider as baselines, in this phase, the performance models described in Section 5.1, as well as a state of the art ML-based regressor, Cubist [31]. Cubist adapts and extends the popular C4.5 decision tree classification algorithm to cope with regression problems, by interpolating arbitrary functions by means of peace-wise linear functions. The choice of Cubist as reference base learner for the results presented in this section is due to the fact that, at least in the considered case studies, Cubist consistently resulted to be the most accurate individual (non-ensembled) ML tech-



(a) STOB (b) Infinispan

Figure 5: Evaluating the accuracy of HyBoost.

nique, when compared to (Weka's implementations of) ANN and SVM[2].

In order to quantify accuracy of the compared alternatives, we select as metric the RMSE normalized with respect to the RMSE of the performance models described in Section 5.1. Whenever we assume the availability of a given training set D_{tr}, we always compute the RMSE over a disjoint test set that comprises all the elements in the entire data set but the ones in D_{tr}.

KNN. Figure 3 and Figure 4 show the normalized accuracy achieved by KNN while varying the cut-off parameter c for the case, respectively, of STOB and Infinispan. Each plot reports results obtained by letting KNN and Cubist observe two percentages of the training set, namely: 20% and 30% in Figures 3(a) and 4(a); 50% and 80% in Figures 3(b) and 4(b). In this experiment we configured KNN to use at most $k = 10$ neighbors, and a single black-box regression model, Γ, which is built by using Cubist.

The plots highlight that the optimal settings of the c parameter is quite different in the two case studies. In particular, for STOB the experimental data clearly show that small cut-off values (on the order of 1% to 5%) are preferable to larger ones. The opposite is true for the Infinispan's case study, where the cut-off parameter that provides optimal accuracy is around 40%. This can be explained by considering that the performance function ϕ that needs to be learnt in the STOB case study is highly non-linear and defined over a small bi-dimensional space. Indeed, given the low dimensionality of the features' space, this data set allows for a quite dense (and hence accurate) sampling of ϕ. Such a dense sampling, combined with the fact that ϕ is subject to quick variations, implies that, by increasing the cut-off parameter, one also increases the probability of including in the set of points D_k (which, we recall, is used to estimate the accuracy of the various prediction models) samples belonging to regions of F in which ϕ exhibits very different dynamics. On the other hand, given the much higher dimensionality of F for the case of Infinispan (and the correspondingly much sparser sampling of F provided by the considered data set), using large cut-off values does pay off, as it increases the likelihood of finding suitable neighbors. In the negative case, being most of the points in the training set quite far apart, for a large fraction of the queries (especially with low percentages of training set), no suitable neighbor is found — in which case, KNN uses, as fall-back, the analytical model.

Overall, both Figures highlight that KNN, when properly tuned, consistently outperforms both the analytical models and the regression models built by Cubist with gains that are more consistent, at low percentage values of the training set, with respect to Cubist, and, vice-versa, more accentuated with respect to the analytical models for higher percentage values of the training set.

HyBoost. The internal parameters of HyBoost are M, namely the depth of the chain of black-box learners that are used to learn a correction function for Γ_{AM}, and the ML algorithms that compose such chain. We experimented with chains of learners of length up to 10 and considered ensembles of black-box learning algorithms including Cubist, Neural Networks and SVM. We present, however, results only with $M = 3$, as the results have shown that, both with STOB and Infinispan, using additional learners did not have any added value on the accuracy. We argue that this depends on the fact that the error function of the ensemble composed by Γ_{AM} and by one ML-based regressor was extremely irregular, hence resulting not easy to learn using additional black-box regressors.

Therefore, we report, in Figure 5, the RMSE (again normalized with respect to the RMSE of Γ_{AM}) achieved by HyBoost while varying the percentage of samples observed during the training phase. The plots highlight remarkable differences between the performances achieved by HyBoost in the two considered case studies. With Infinispan, HyBoost yields substantial improvements with respect to both Γ_{AM} and Cubist, with maximum gains for 30% of training set where it reduces the RMSE by a factor 3x with respect to PROMPT and 5x with respect to Cubist. As for STOB, instead, the error function of the analytical model, as shown in Figure 2(a), is strongly non-linear and irregular, and, consequently, the chain of corrective MLs fails in effectively compensating for Γ_{AM}'s inaccuracies.

Probing. In Figure 6 and Figure 7 we report the results of a study aimed at assessing the sensitivity of Probing with respect to the cut-off parameter c for the case, respectively, of STOB and Infinispan. Let us analyze first the case of Infinispan, where we can see that, with small training sets, this method yields the best results with high cut-off values, i.e.,

[2]It stems from the no-free-lunch-theorem [41] that no ML algorithm can universally outperform all the others. Hence, we do not exclude that for specific parameters' tunings, ANN, SVM, or any other alternative ML algorithm may outperform Cubist.

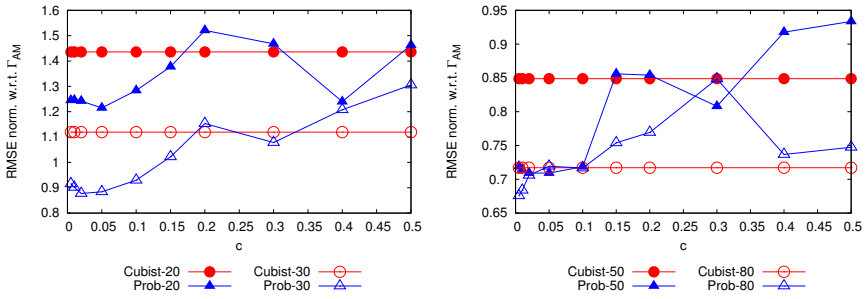


Figure 6: Sensitivity analysis of Probing w.r.t. the c parameter (STOB)

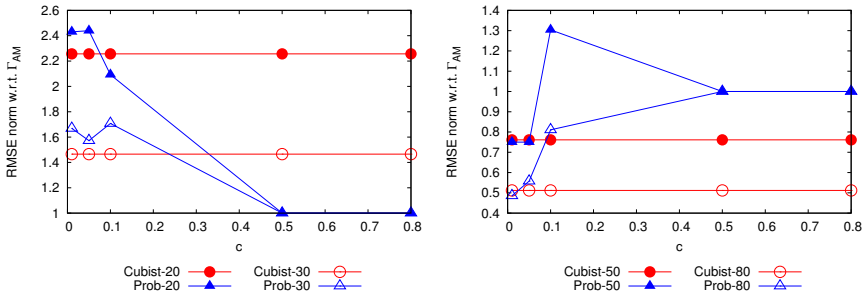


Figure 7: Sensitivity analysis of Probing w.r.t. the c parameter (Infinispan)

greater than 50%. This is indeed expectable, considering that, after having observed only a few samples in the training set it is very hard to build a reliable black-box model in a scenario with such a high dimensional features' space. It follows that in this scenario and for this setting of c, Probing will always query the analytical model. This also explains why Probing performs exactly like Γ_{AM} in these cases. For larger percentages of training sets, the optimal configuration is instead obtained for very small values of c, i.e., close to 1%. With such a configuration, clearly the ML regressor is used most of the times, as the prediction error of Γ_{AM} will be larger than 1% for the vast majority of the points in D_{TR}. Hence we may observe performance very close to those of the ML in this case. Overall, in the Infinispan scenario, Probing never really manages to outperform significantly both the white box and the black box approaches. A possible explanation can be found in Figure 2(b). As already mentioned, the error distribution of Γ_{AM} is extremely disperse in this scenario. In these conditions, generalizing rules capable of accurately determining when to use Γ_{AM} or the black-box learner is quite a challenging task for Γ_{cls}.

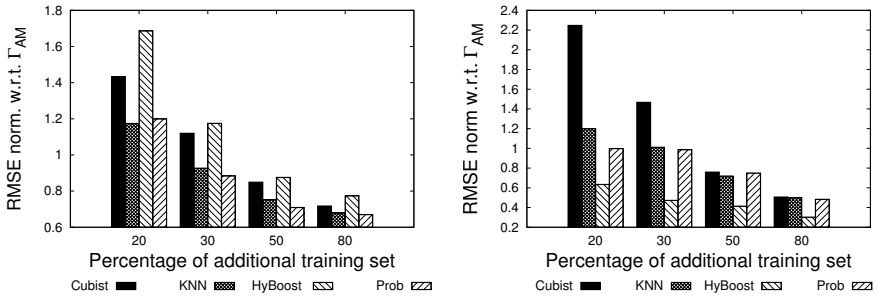
As already mentioned, instead, in the case of STOB, Γ_{AM} exhibits a much more irregular error distribution that, however, exhibits high values only in a clearly localized portion of the features' space. This simplifies considerably the problem of classifying automatically the regions in which the white box model is expected to achieve good/bad accuracy — explaining why Probing performs so much better in this scenario with respect to the case of Infinispan. Concerning the optimal tuning of c, the STOB scenario confirms what our results had already highlighted for Infinispan, when using relatively large training sets (50% and 80%): picking low cut-off values, and trusting consequently more the ML, is the optimal strategy. Somewhat surprisingly, picking very low cut-off values (and hence trusting excessively the ML) is the most rewarding strategy also when considering small training sets in STOB. This may be explained by considering that since in this case ϕ is relatively simple, the ML could already learn a very robust approximator of ϕ, which can hardly be further improved by exploiting Γ_{AM}.

5.3 Mutual Comparison

So far, the proposed ensemble techniques have been only evaluated individually. This section compares their mutual performance, assuming that each ensemble technique is using the optimal parameter values determined in the previous section[3].

Figure 8 compares the three proposed gray box ensemble techniques with respect to each other, to Γ_{AM} and to a pure black-box model built using Cubist. The left plot reports the results obtained for STOB. In this scenario, KNN and Probing are the two techniques that deliver the best results, consistently outperforming both Γ_{AM} and Cubist when the percentage of training set used to initialize the ensemble is larger than or equal to 30%. Conversely, HyBoost exhibits quite disappointing performance in this use case. As already discussed above, the cause of these differences is imputable to the strong non-linearity of the error distribution of Γ_{AM}, which HyBoost tries unsuccessfully to learn via ML techniques. On the other hand, as the regions in which Γ_{AM}

[3]The correct settings of these parameters can be identified recurring to the standard methodology used to tune the internal parameters of ML-based algorithms: performing a parameter's sweep during the training phase, and using cross-validation to evaluate the accuracy achieved when using a candidate parameter configuration over a test set disjoint from the training set used to initialize the ensemble [2].



(a) STOB (b) Infinispan

Figure 8: Mutual comparison between KNN, HyBoost and Probing.

incurs the largest errors are relatively circumscribed, solutions like KNN and Probing, which are based on the idea of determining in which regions to use which learner, result the most effective.

The landscape changes significantly in the Infinispan case study. Here, HyBoost is by far the most effective technique, reducing RMSE on average by a 2x factor vs Γ_{AM} and a 3x factor vs Cubist. KNN and Probing, instead, fail to achieve significant gains with respect to both the baselines, although normally remaining competitive with the best of the two across the entire spectrum of considered training set percentage values. The reason underlying these results is again identifiable by looking at the error distribution of Γ_{AM}, which, as shown in Figure 2(b), varies slowly in this case and exhibits pronounced linear trends.

Overall, these results suggest two main considerations.
No one size fits all: None of the proposed techniques was able to consistently outperform all others in all the considered scenarios. This result is indeed not surprising if one takes into account seminal results, such as the no-free-lunch-theorem [41], which states precisely the impossibility of building a universal statistical learning technique. It is therefore imperative not to blindly rely on any of the proposed ensemble techniques, but to always verify, using cross-validation during the training phase, the actual effectiveness of each technique based on the problem at hand.
The error distribution of Γ_{AM} is crucial: Our experimental study suggests that one of the key factors that affects the performance of the proposed solutions is the "shape" of the error distribution of Γ_{AM}. A natural research question triggered by this finding is whether it is possible to identify rigorous conditions under which each of the proposed ensemble algorithms is favored. Another, probably even more interesting, question is whether one could intentionally introduce biases in the design of analytical models to make them more amenable to be used within a gray box ensemble such as the ones proposed in this work. For instance, one may prefer a simpler, yet globally less accurate analytical model, if it could be guaranteed (even probabilistically) that the error distribution of the adopted model was easier to learn using techniques like HyBoost.

6. CONCLUSIONS

In this paper we explored the problem of how to combine white and black box performance modeling methodologies by proposing and evaluating three techniques that aim at taking the best of the two worlds, namely avoid incurring the drawbacks (e.g., non-minimal errors in specific operative conditions) of any individual technique, while minimizing the time for instantiating a reliable performance model of the target system/application.

Our proposal is aligned with the needs arising in modern computing systems, namely (a) their extremely high complexity and the reliance on virtualization, factors that tend to be adverse to white box, e.g., analytical, performance modeling techniques (e.g., given that system internal operations may be not perfectly known, hence being difficult to be reliably expressed in term of their behavior via analytical formulas), and (b) the need for timely instantiating scenario-specific performance models, which can then be used for prompt optimization of the usage of, e.g., Cloud rented resources. The latter requirement is particularly challenging for black box approaches based on machine learning, given that the building of reliable machine learning predictors generally requires lengthy training phases, leading to delays in the actual optimization based on the performance model.

We evaluated the effectiveness of our proposals by relying on case studies related to two highly relevant open-source middleware platforms, namely a key-value data store and a group communication system. We feel our proposal stands as a relevant achievement in terms of the construction of supports for coping with the problem of performance (and hence resource usage) optimization of highly complex computing systems.

7. REFERENCES

[1] R. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957.
[2] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. 2007.
[3] C. D. Carothers and K. S. Perumalla. On deciding between conservative and optimistic approaches on massively parallel platforms. In *Winter Simulation Conference*, 2010.
[4] R. Caruana et al. Ensemble selection from libraries of models. In *Proc. of ICML*, 2004.
[5] B. F. Cooper et al. Benchmarking cloud serving systems with ycsb. In *Proc. of SOCC*, 2010.
[6] J. C. Corbett et al. Spanner: Google's globally-distributed database. In *Proc. of OSDI*, 2012.

[7] M. Couceiro et al. D2stm: Dependable distributed software transactional memory. In *Proc. of PRDC*, 2009.

[8] M. Couceiro et al. A machine learning approach to performance prediction of total order broadcast protocols. In *Self-Adaptive and Self-Organizing Systems (SASO), 2010 4th IEEE International Conference on*, pages 184–193. IEEE, 2010.

[9] P. Desnoyers et al. Modellus: Automated modeling of complex internet data center applications. *ACM Trans. Web*, 6(2):8:1–8:29, June 2012.

[10] P. Di Sanzo et al. On the analytical modeling of concurrency control algorithms for software transactional memories: The case of commit-time-locking. *Perform. Eval.*, 69(5):187–205, May 2012.

[11] P. Di Sanzo et al. A framework for high performance simulation of transactional data grid platforms. In *Proc. of SIMUTools*, 2013.

[12] P. Di Sanzo et al. Regulating concurrency in software transactional memory: An effective model-based approach. In *Proc. of SASO*, 2013.

[13] D. Didona et al. Identifying the optimal level of parallelism in transactional memory applications. *Computing*, 2013.

[14] D. Didona et al. Transactional auto scaler: Elastic scaling of replicated in-memory transactional data grids. *ACM Trans. Auton. Adapt. Syst.*, 9(2):11:1–11:32, July 2014.

[15] D. Didona and P. Romano. On Bootstrapping Machine Learning Performance Predictors via Analytical Models. *ArXiv e-prints*, Oct. 2014.

[16] D. Didona and P. Romano. Performance modelling of partially replicated in-memory transactional stores. In *Proc. of MASCOTS*, 2014.

[17] T. G. Dietterich. Ensemble methods in machine learning. In *Proc. of MCS Workshop*, 2000.

[18] M. Faber and J. Happe. Systematic adoption of genetic programming for deriving software performance curves. In *Proc. of ICPE*, 2012.

[19] J. H. Friedman. Stochastic gradient boosting. *Comput. Stat. Data Anal.*, 38(4):367–378, Feb. 2002.

[20] V. Grassi et al. On the optimal checkpointing of critical tasks and transaction-oriented systems. *IEEE Trans. Software Eng.*, 18(1):72–77, 1992.

[21] R. Guerraoui and L. Rodrigues. *Introduction to Reliable Distributed Programming*. Springer-Verlag New York, Inc., 2006.

[22] H. Herodotou et al. No one (cluster) size fits all: automatic cluster sizing for data-intensive analytics. In *SOCC*, 2011.

[23] D. Jiang et al. Autonomous resource provisioning for multi-service web applications. In *Proc. of WWW*, 2010.

[24] L. Kleinrock. *Queueing Systems*, volume I: Theory. Wiley Interscience, 1975.

[25] F. Marchioni and M. Surtani. *Infinispan Data Grid Platform*. Packt Publishing, 2012.

[26] D. W. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *SIAM Journal on Applied Mathematics*, 11(2):431–441, 1963.

[27] H. Meling et al. Jgroup-arm: A distributed object group platform with autonomous replication management. *Softw. Pract. Exper.*, 38(9):885–923, July 2008.

[28] H. Miranda et al. Appia: A flexible protocol kernel supporting multiple coordinated channels. In *ICDCS*, 2001.

[29] G. Pacifici et al. Performance management for cluster-based web services. *Selected Areas in Communications, IEEE Journal on*, 23(12):2333–2343, 2005.

[30] F. Pedone et al. The database state machine approach. *Journal of Distributed and Parallel Databases and Technology*, 14:2003, 1999.

[31] J. R. Quinlan. Rulequest Cubist. http://www.rulequest.com/cubist-info.html, 2012.

[32] P. Romano and M. Leonetti. Self-tuning batching in total order broadcast protocols via analytical modelling and reinforcement learning. In *Proc. of ICNC*, 2011.

[33] D. Rughetti et al. Analytical/ml mixed approach for concurrency regulation in software transactional memory. In *Proc. of CCGRID*, 2014.

[34] R. Singh et al. Autonomic mix-aware provisioning for non-stationary data center workloads. In *Proc. of ICAC*, 2010.

[35] R. Singh et al. Analytical modeling for what-if analysis in complex cloud computing applications. *SIGMETRICS Perform. Eval. Rev.*, 40(4):53–62, Apr. 2013.

[36] Y. Sovran et al. Transactional storage for geo-replicated systems. In *SOSP*, 2011.

[37] G. Tesauro et al. On the use of hybrid reinforcement learning for autonomic resource allocation. *Cluster Computing*, 2007.

[38] E. Thereska and G. R. Ganger. Ironmodel: Robust performance models in the wild. *SIGMETRICS Perform. Eval. Rev.*, 36, June 2008.

[39] B. Trushkowsky et al. The scads director: scaling a distributed storage system under stringent performance requirements. In *Proc. of FAST*, 2011.

[40] L. Wang et al. Fuzzy modeling based resource management for virtualized database systems. In *MASCOTS*, 2011.

[41] D. H. Wolpert. The lack of a priori distinctions between learning algorithms. *Neural Comput.*, 8(7):1341–1390, Oct. 1996.

[42] J. Xu et al. On the use of fuzzy modeling in virtualized data center management. In *Proc. of ICAC*, 2007.

[43] P. S. Yu, D. M. Dias, and S. S. Lavenberg. On the analytical modeling of database concurrency control. *J. ACM*, 40(4):831–872, 1993.

[44] Q. Zhang et al. A regression-based analytic model for dynamic resource provisioning of multi-tier applications. In *ICAC*, 2007.

[45] T. Zheng et al. Integrated estimation and tracking of performance model parameters with autoregressive trends. In *Proc. of ICPE*, 2011.