Hybrid Machine Learning/Analytical Models for Performance Prediction: a Tutorial*

Diego Didona and Paolo Romano
INESC-ID / Instituto Superior Técnico, Universidade de Lisboa

ABSTRACT

Classical approaches to performance prediction of computer systems rely on two, typically antithetic, techniques: Machine Learning (ML) and Analytical Modeling (AM).

ML undertakes a black-box approach, which typically achieves very good accuracy in regions of the features' space that have been sufficiently explored during the training process, but that has very weak extrapolation power (i.e., poor accuracy in regions for which none, or too few samples are known).

Conversely, AM relies on a white-box approach, whose key advantage is that it requires no or minimal training, hence supporting prompt instantiation of the target system's performance model. However, to ensure their tractability, AM-based performance predictors typically rely on simplifying assumptions. Consequently, AM's accuracy is challenged in scenarios not matching such assumptions.

This tutorial describes techniques that exploit AM and ML in synergy in order to get the best of the two worlds. It surveys several such hybrid techniques and presents use cases spanning a wide range of application domains.

1. INTRODUCTION

Performance modeling of applications and systems is a critical requirement in tasks like anomaly detection, optimization, capacity planning, and, with the advent of Cloud computing, automatic resource provisioning.

Analytical Modeling (AM) has been for decades the reference technique in this context [11, 12, 20, 21]. AM relies on exploiting *a-priori* knowledge of the internal dynamics of target applications/systems in order to express the input/output relation by means of a set of analytical equations. For this reason, AM modeling is often referred to as *white box modeling*. The most appealing feature of AM is that, once instantiated, a white box model exhibits a good accuracy when working in *extrapolation*. That is, AM is capable of predicting several Key Performance Indicators (KPI) for a wide set of application's workload, input parameters and characteristics of the underlying hosting platform. Yet, AM comes with the downside of relying on assumptions and approximations (necessary to ensure the analytical tractability of the model) that can hamper prediction's accuracy.

The successful application of AM technique has been progressively challenged over the last years, primarily due to two main causes: *i*) the ever increasingly complexity of applications makes it increasingly difficult to derive detailed AMs capable of capturing all the dynamics and relations that map characteristics of an application/system onto KPIs; *ii*) the advent of Cloud Computing has led applications to be deployed over virtualized infrastructures, whose details are typically intentionally hidden. This limited knowledge about the underlying platform poses serious challenges to the derivation of accurate white box models.

Fortunately, the last years have also witnessed the maturing of research in the area of Machine Learning (ML) [1], which resulted into the development of a wide range of freely-available high quality ML toolkits [15]. This has led a growing number of researchers to explore the possibility of using ML techniques to build *black box* predictors of complex computer systems [16, 18, 19]. Black box modeling is antithetic with respect to its white box counterpart: it relies on inferring the input/output relationships that map application's and system's characteristic onto the target KPI, and on encoding such relationships via statistical models. Such models are built on the basis of a so called *training phase*, during which the application is tested with different workloads and parametrized with different configurations, with the purpose of observing the corresponding achieved performance. The most appealing property of such modeling approach is that it is sufficient to identify *which* are the inputs –a.k.a. features– of the performance functions, and the ML algorithm will take care of inferring *how* they map to the target KPI, without exploiting any additional knowledge about the application.

Unfortunately, ML does not represent the "silver bullet" for the performance modeling problem, as the lack of *a priori* information about the target application/system does come with a price. The accuracy of ML-based performance predictors, in fact, ultimately depends on the representativeness of the input/output samples collected during the training phase. In order to exhaustively cover the whole space of possible inputs, the training phase should sweep all combinations of possible workloads and system configurations.

*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/10.1145/2668930.2688823.

Unfortunately, the cardinality of the resulting set grows exponentially with the number of input features, making it cumbersome, or even impossible, to carry out an exhaustive training phase for complex systems. As a result, black box models typically delivers a very good accuracy when working in *interpolation*, i.e., in regions of the features' space that they have been sufficiently explored; conversely, their accuracy is typically poor when working in extrapolation.

Gray box modeling [5, 6, 4, 8, 7, 10, 14, 9, 3] has emerged in the last years as an attempt to achieve the best of the AM and ML world. It relies on exploiting both methodologies, in order to compensate the weaknesses of the one with the strengths of the other. In particular, gray box modeling aims at making performance predictors more robust to the limitations of the two base methodologies by *i*) inheriting from ML-based models the ability to progressively enhance the accuracy of the performance predictor as new data from the operational system are collected; *ii*) requiring, like AM models, little or no training time in order to instantiate a performance predictor. Different gray box methodologies have been proposed, targeting very diverse techniques to combine AM and ML. In this paper we overview the most prominent approaches that have been proposed in literature. This document is intended to be a companion of the ICPE15 tutorial, which will cover this topic in deeper detail.

The next section is devoted at introducing some basic concepts and terminology on ML. Section 3 describes the three most prominent gray box modeling methodologies, whereas Section 4 overviews example applications for each of these methodologies. Finally, Section 5 concludes the paper and discusses open research questions in the area of gray box modeling.

2. BACKGROUND ON MACHINE LEARNING

Machine Learning deals with the construction and study of systems that can learn from data [1]. In this section we are going to provide basic background on ML focused on Supervised Learning and Reinforcement Learning, which are the ML techniques that are most frequently employed in the domain of performance modeling. We shall also introduce two of the most commonly used ML algorithms, namely Decision Trees and Support Vector Machines.

Supervised Learning (SL) is the task aimed at learning the relation between a set of input parameters, called *input features*, and a set of outputs, called *target features*. More formally, a supervised machine learner infer a function (also called model) $\phi : X \rightarrow Y$ based on the observed output corresponding to a set $\tilde{X} \subset X$, called *training set*. Such a function can be exploited to predict the output value y corresponding to values of the input parameters that are not present in the training set. If the codomain of the ϕ function is continuous, then the learner is defined *regressor*. If it is discrete, then the learner is defined *classifier*; in this case, the values that the output can assume are called *classes*.

Two of the most prominent Supervised Learning algorithms are Decision Trees (DT) and Support Vector Machines (SVM) [1].

DTs [1] structure their model as a tree: each interior node of the tree corresponds to one of the input variables; edges from a parent to a child are labeled with a predicate about the value of the input feature relevant to the parent node. In the case of DT classifiers, each leaf represents the class of the target feature; in the case of regressors, it is a function of the input values.

SVMs [1] map the points of the training set over a multidimensional space W such that elements in the same class occupy a specific portion of that space and are as far away as possible from elements of other classes. Although introduced to solve classification problems, SVMs have been extended to be used also as regressors [1].

Reinforcement Learning (RL) is the branch of ML that investigates the issue of how an *agent* should perform actions in an environment in order to maximize a cumulative *reward*. One of the most important issues in reinforcement learning techniques is the trade-off between exploration and exploitation: in absence of an explicit model capable of determining *a priori* the optimality of an action over another one, the agent must explore the environment in order to gather feedback on the rewards of the action. A RL algorithm aims to identify which and how many exploration steps should be performed in order to maximize the long-term reward (typically modeled as an unknown random variable). When applied to the domain of self-tuning, RL techniques are used to implement a controller that is in charge of determining the optimal configuration for a system. In this context, the environment is represented by the set of tunable parameters and external factors (e.g., the workload) and the reward is defined as a function of some target KPI.

3. GRAY BOX MODELING TECHNIQUES

Existing works in the area of gray box performance modeling and optimization can be grouped in three different classes, which we call *Boostrapping*, *Divide-and-conquer*, and *Ensembling*. We overview each of them in the following.

Bootstrapping. This technique relies on an AM in order to produce a *synthetic* training set over which a black box learner is initially trained. This training set is composed of $< input, output >$ tuples where the input normally spans different workload metrics and platform configurations, and the corresponding output is obtained by querying the analytical model. As this set is obtained without actually gathering samples from the operational system, this technique allows for significantly reducing the time necessary an initial black-box leaner (e.g., based on DT or SVM). Clearly, the performance function inferred by the ML will be very similar to the one encoded in the AM and, as such, it will inherit also its possible inaccuracies. However, as $< input, output >$ tuples are collected from the actual system (e.g., while it is running in production), such synthetic training set can be complemented with *real* samples. The ML can be, thus, periodically re-trained over time, in order to benefit from this additional knowledge and compensate for possible inaccuracies of the original AM.

This technique has been implemented in solutions integrating analytical models both with RL [13, 9] and SL algorithms [14, 10], and has been applied for data centers management [13, 14] and applications optimization [10, 9]. A recent detailed study of the bootstrapping technique investigates the design space of such technique and highlights some pitfalls that may arise in its implementation [7].

Divide and conquer. This approach consists in building specialized models, for different components of the target system, that rely either on AM or on ML; such models

are then coupled to carry out the performance prediction about the behavior of the overall system. Normally, AM is exploited to capture performance of components whose internal dynamics are known and easy to monitor; ML, conversely, is typically employed to predict the behavior of components whose internals are hidden (and, thus, not observable), or whose performance dynamics are too complex to be accurately captured via analytical methods.

This approach has been implemented for modeling performance of distributed transactional Cloud data platforms, where details of the underlying physical architecture is hidden by the virtualization layer; it has been integrated in hybrid models encompassing both queueing-theory based AMs [5, 8] as well as simulation-based ones [3].

Ensembling. This technique entails combining the outputs of multiple AMs and MLs according to different schemes. Various implementations have been proposed, which differ in the way the base models are combined to produce the final performance prediction of the target KPI.

One approach we find in literature [4, 6] consists in building an AM to predict the target KPI; then, a chain of M black box learners is progressively trained not on the KPI itself, but on the residual prediction errors of the previous model. In this way, the i-th model learns a corrective function that compensates for the inaccuracies of the chain composed by previous $i-1$ models.

Other solutions rely on building a battery of independent predictors for the target KPI; at query time, only the predictor that is estimated to be the most reliable for a given input i is employed to carry out the final prediction. All the ensemble-based predictors that we are aware of carry out this decision based on the expected accuracy of the individual learners in the "proximity" of the target input i, but adopt different notions of proximity. Existing approaches include solutions that partition statically the input space (based on some *a-priori* knowledge) and assign different partitions to each model [2], techniques that employ a classifier to determine which predictor to use or that pick the most accurate learner over the k nearest neighbors of the target input in the training set [6] .

4. APPLICATIONS

This section is devoted to presenting in greater detail one use case for each of the gray box modeling methodologies described in the previous section.

4.1 Divide and conquer

For this category we present Transactional Auto Scaler (TAS) [5]. We choose this use case as, to the best of our knowledge, it represents the first implementation of such gray box modeling methodology.

TAS is a performance model for replicated transactional Cloud data stores; a transactional data store is a platform that allows applications deployed over it to perform atomic operations on shared data in spite of concurrent accesses via the *transaction* abstraction [17]. In a replicated transactional platform, there are multiple machines (a.k.a. nodes) that host the same data set, and each node maintains a local copy of it. Whenever a transaction completes its execution, it starts a distributed *commit* phase during which it contacts all the nodes in the systems to determine the final outcome for the transaction, i.e., commit or abort. In the former case the transaction completes successfully, otherwise it needs to be aborted and restarted.

TAS captures the effect of data and CPU contention by means of white box models: it models the CPU of different nodes using queuing theory in order to compute the response time of CPU-bound operations, like reading or writing data; moreover, it also models each datum as a queue, in order to compute the probability for two transactions to abort due to conflicting accesses on the same data. On the other hand, TAS exploits ML, specifically DTs, in order to predict the execution time of network-bound operations, i.e., the distributed commit phase. As already discussed, the rationale behind this choice is that the virtualization layer of Cloud environments hides the physical details of the hosting infrastructures; it is, thus, extremely cumbersome to derive a network performance model without knowing, for example, the network topology according to which physical machines are organized.

The two models are built independently, but they are solved according to an iterative scheme that couples them in order to predict the overall execution time of a transaction. First, the AM is queried to produce an estimate of some intermediate performance variables that are among the input features of the ML, e.g., the rate at which commit procedures are initiated; then, the ML is queried to produce a prediction of the network-bound operations execution times, which is in turn needed by the AM to compute the overall response time of a transaction. This two steps are repeated until the difference in the transactions response time computed in two consecutive iterations becomes marginal.

Bootstrapping. We choose as representative for this methodology IRONModel [14], which is a performance modeling framework for anomaly detection in data centers.

IRONModel relies on a set of AMs that are built to predict the expected behavior of different components in a data center, from routers to storage systems. The synthetic training sets generated starting from these models are given in input to a DT, which ultimately serves all the performance prediction queries. Note that, unlike other use cases in which any ML could be employed, IRONModel specifically relies on DTs because of their characteristic to produce a model that is interpretable by humans; this is an important requirement for anomaly detection and data centers management.

This black box model is exploited not only to perform what-if analysis, but it is also periodically queried so at to monitor whether the behavior of a component deviates from its normalcy. Upon detecting such a deviation, an alarm is triggered, which is handled by the system administrator. If the anomalous behavior is not recognized to be caused by a bug or a failure of a sub-component, and it is, then, a regular but unforeseen behavior of the target component, then the system administrator performs the following operations: *i*) she examines the logs about the utilization patterns of the target component, in order to detect and isolate workloads of the target component that caused the anomalous performance; *ii*) she sets-up experiments aimed at reproducing such workloads and conditions, and schedules them to be run either immediately, if the current workloads can be redirected to other components in the data center, or as soon as they can be executed in isolation, e.g., at night; *iii*) once the experiments are completed, the logs encoding the measured input/output relation is given as input to the DT, which is re-trained in order to update its rule-set and prop-

erly predict the component's behavior that was regarded as anomalous.

Ensemble. The use case chosen as example for this methodology is Chorus, a model ensemble tool for self-management in data centers. Chorus' performance predictor relies on ensembles of both black and box models. Chorus' white box models are typically simple, as they are designed to capture the target system's behavior in well specified operational conditions, for example in cases in which the workload is CPU bound, disk bound or memory bound. Black box models include both SVMs, as well as simpler regressive models (linear, polynomial or exponential functions), whose parameters are determined by fitting the output of the various models to the data in the training set.

Chorus is trained by performing the following steps. First of all, the input space is partitioned into R disjoint regions. Then, an immutable validation set D_v is generated, starting from some input/output observations; such set is used to evaluate the accuracy of the overall performance model and to stop the training phase when a target average accuracy has been achieved. The training set D_t is expanded incrementally in rounds, until the stopping criterion is met. At each round, the accuracy of the M models is evaluated by means of k-fold cross validation on D_t. This entails partitioning D_t into k bins $D_{t,1} \dots D_{t,k}$ and then, iteratively for $i = 1 \dots k$, training the models over $D_t \setminus D_{t,i}$ and evaluating its accuracy against $D_{t,i}$. The models are then sorted on a per region basis, according to the achieved accuracy in predicting performance for samples falling in a given region.

Once trained, Chorus serves a query for an input sample belonging to region $r \in R$ by returning the output of the most accurate performance predictor for r.

5. CONCLUSIONS AND DISCUSSION

Analytical Modeling and Machine Learning are typically regarded as two alternative methodologies to model performance of computer systems and applications. In this paper we have overviewed three different hybrid modeling methodologies, which leverage on both AM and ML to get the best of the two worlds, namely reducing the model's training time and increasing its accuracy as new training data become available.

While research on AM and ML has already reached maturity, investigation on hybrid methodologies is still at its infancy. A recent work [6] has shown that none of such techniques outperforms the others in terms of accuracy for every application and for every training data set. An interesting research line to pursue, in the light of this result, is to identify which characteristics of the applications being modeled, or of the AM and ML techniques employed for modeling may lead a given hybrid methodology to outperform the others.

Another interesting issue to investigate is whether it is possible to fruitfully combine further the described methodologies, with the goal of building a unique, more accurate, meta-hybrid model.

6. REFERENCES

[1] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. 2007.

[2] J. Chen et al. Model ensemble tools for self-management in data centers. In *Proc. of ICDE Workshops*, 2013.

[3] P. Di Sanzo et al. A flexible framework for accurate simulation of cloud in-memory data stores. *ArXiv e-prints*, Dec. 2014.

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

[5] 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.

[6] D. Didona et al. Enhancing Performance Prediction Robustness by Combining Analytical Modeling and Machine Learning. In *Proc. of ICPE*, 2015.

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

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

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

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

[11] Y. C. Tay. *Analytical Performance Modeling for Computer Systems*. Morgan & Claypool Publishers, 2013.

[12] L. Kleinrock *Queueing Systems, Theory, Volume 1.* Wiley Interscience, 1975.

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

[14] E. Thereska and G. Ganger. Ironmodel: Robust performance models in the wild. In *Proc. of SIGMETRICS*, 2008.

[15] M. Hall et al. The WEKA Data Mining Software: An Update. *SIGKDD Explor. Newsl.*, 11(1) 10–18, June 2009.

[16] M. Couceiro et al. A machine learning approach to performance prediction of total order broadcast protocols. In *Proc. of SASO*, 2010.

[17] P. Bernstein and E. Newcomer Principles of Transaction Processing: For the Systems Professional *Morgan Kaufmann Publishers Inc.*, 1997

[18] M. Couceiro et al. Chasing the optimum in replicated in-memory transactional platforms via protocol adaptation. In *Proc. of DSN*, 2013

[19] A. Ganapathi et al. Predicting Multiple Metrics for Queries: Better Decisions Enabled by Machine Learning. In *Proc. of ICDE*, 2009

[20] J. Padhye et al. Modeling TCP throughput: A simple model and its empirical validation. *SIGCOMM Comput. Commun. Rev.*, 28(4) 303-314, Oct. 1998.

[21] P. Di Sanzo et al. On the analytical modeling of concurrency control algorithms for software transactional memories: The case of commit-time-locking. Performance Evaluation 69(5), May, 2012