A Hybrid Machine Learning Approach for Performance Modeling of Cloud-based Big Data Applications

Ehsan Ataie[1], Athanasia Evangelinou[2], Eugenio Gianniti[2] and Danilo Ardagna *[2]

[1]Department of Engineering and Technology, University of Mazandaran, Babolsar, Iran
[2]Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Milan, Italy
Email: ataie@umz.ac.ir, athanasia.evangelinou@polimi.it, eugenio.gianniti@polimi.it, danilo.ardagna@polimi.it

Nowadays, Apache Hadoop and Apache Spark are two of the most prominent distributed solutions for processing big data applications on the market. Since in many cases these frameworks are adopted to support business critical activities, it is often important to predict with fair confidence the execution time of submitted applications, for instance when Service Level Agreements (SLAs) are established with end-users. In this work, we propose and validate a hybrid approach for the performance prediction of big data applications running on clouds, which exploits both Analytical Modeling (AM) and Machine Learning (ML) techniques and it is able to achieve a good accuracy without too many time consuming and costly experiments on a real setup. The experimental results show how the proposed approach attains improvement in accuracy, number of experiments to be run on the operational system, and cost over applying machine learning techniques without any support from analytical models.

Keywords: Analytical Performance Modeling; Machine Learning; Cloud Computing; MapReduce; Hadoop; Tez; Spark

1. INTRODUCTION

The implementation of big data applications is steadily growing today. According to a recent analysis, the big data market is expected to reach $189.1 billion in 2019 with a Compound Annual Growth Rate (CAGR) of 12.0%, about three times the one of the overall ICT market [1]. In 2020, the global data volume is predicted to be around 40,000 Exabytes which represents a 300 times growth factor compared to the global data volume in 2005 [2]. Most of big data applications implemented today are based on the MapReduce (MR) programming model, which is the most common adopted solution. MapReduce-based systems have risen as a scalable and cost effective solution for massively parallel data processing [3] The majority of large-scale data intensive applications designed by MR model are deployed and executed on a large-scale distributed Hadoop system [4].

Clouds are cost-effective platforms to support such systems, as resources (e.g., nodes) can be allocated and deallocated on demand, in response to the applications requirements and Quality of Service (QoS) needs [5]. As a matter of fact, Forrester research predicts that by 2020 nearly 40% of big data analytics will be supported by public clouds [6].

From the technological perspective, the rigid division between map and reduce requires to subdivide a complex application into a Directed Acyclic Graph (DAG) of MR jobs, comprising tasks that perform a specific computation on partitions/splits of the input data. In this case, the MR paradigm forces the storage of each intermediate phase results on disk, thus being unsuitable for applications requiring a low latency between different phases, along with general application QoS guarantees. Other frameworks, such as Tez and Spark, have been introduced [7] to address this problem. Although Tez can handle general DAGs of map, reduce and reduce/reduce phases, it still requires to write each stage results on disk. On the other hand, Spark can exploit a set of primitives to request the caching of partial results in memory, thus allowing lower latency and better performance [8]. In practice, Spark can easily obtain a 10x to 100x speedup over Hadoop on specific scenarios [9, 10] and it will stay prominent in the field for years to come [11].

In this context, one of the main challenges [12, 13, 14] is that the execution time of big data applications is generally unknown in advance. Because

of this, performance analysis is usually done empirically through experimentation, requiring a costly setup [15].

In addition, big data systems are becoming a central force in society, thus requiring the development of intelligent systems, which provide QoS guarantees to their users. Performance prediction models are extremely useful to aid development and deployment of big data applications either for design time decisions or run time system reconfiguration. Design time models can help, e.g., to determine the appropriate size of a cluster or to predict the budget required to run big data analyses in public clouds. Such models can be used also at run time, allowing a dynamic adjustment of the system configuration [16, 17], e.g., to cope with workload fluctuations or to reduce energy costs.

One approach for performance prediction is to develop white box Analytical Models (AMs) based on Queueing Networks (QNs), Petri Nets (PNs), Stochastic Activity Networks (SANs), and so on for predicting performance metrics. However, analytically modeling big data applications is very challenging since it requires a deep knowledge of the system behavior. Moreover, performance prediction in a cloud context is even more complex, since applications execution time may also vary as a consequence of resource contention and performance degradation introduced by the underlying virtualization layer [18]. To ensure AM tractability in such complex systems, AM-based performance models typically rely on simplifying assumptions that reduce their accuracy.

On the other hand, black box Machine Learning (ML) deals with the study and construction of algorithms that can learn from data and make predictions on it without a priori knowledge about the internals of the target system. To be able to predict accurately, ML models should be built during a training phase with a sufficient amount of experimental data from different workloads, using different parameters and configurations. However, running several experiments in cloud environments would be costly and time consuming. Though ML often provides good accuracy in regions for which it is well trained, it shows poor precision in regions for which none or very few samples are known.

Hybrid Machine Learning (HML), which can be considered as a gray box modeling technique [19, 20, 21], [22] is a relatively new approach for performance prediction that tries to achieve the best of the AM and ML worlds by mixing the two. Such models can be used to support design-time decision-making during the development and deployment phases of big data applications.

In this paper, a combined AM/ML model is presented to estimate big data applications execution time built on the most popular frameworks, i.e., MR, Tez, and Spark, running in cloud clusters. At first, AMs are proposed to initially model the response time of a big data application and initiate some synthetic samples. Then, this analytical data is used to train an initial ML model. During an iterative and incremental process, new data from the operational system is fed into the ML model to build a more accurate performance predictor. In addition, some intuitions are exploited to provide more accurate predictions while consuming less data from the operational systems, which, due to resource contention, might be affected by noise. After proposing the hybrid model, the accuracy of the proposed model is evaluated on real systems by performing experiments based on the TPC-DS industry benchmark for business intelligence data warehouse applications. PICO[3], the big data infrastructure of CINECA, the Italian supercomputing center, is considered as the target deployment environment. Results and experiments performed on real systems have shown that the achieved percentage error is around $15 - 18\%$ for MR and Tez, and about 10% for Spark with respect to the actual measurements on average. Moreover, our hybrid approach is compared against two state of the art ML-based techniques in terms of different metrics defined to indicate the prediction accuracy, as well as the time and cost of constructing final models. The results show that our hybrid approach outperforms both baseline models. With respect to previous literature, to the best of our knowledge, this paper is one of the first contributions able to study the performance of big data frameworks in cloud environments using HML approaches.

The paper extends our previous work [19] in several directions, in terms of implementation and testing of various analytical models. In particular, a formula-based approximation is proposed as AM technique to generate synthetic data samples; interpolation and extrapolation capabilities of the proposed algorithm are examined, and finally, the proposed algorithm is applied not only to a simple MR application, but also to Tez and Spark queries.

The rest of this paper is organized as follows. In Section 2, the background and modeling problem settings are presented. Section 3 introduces our approach for hybrid AM/ML performance prediction. Section 4 reports an extensive experimental campaign, which has been performed to explore and validate the behavior of our modeling approach. In Section 5 we compare our work with other researches proposed in the literature. Finally, the paper is concluded in Section 6 with some notes for future work.

2. BACKGROUND AND MOTIVATIONS

Often big data applications users need to know a priori how long their job execution will take using different configurations for the cloud infrastructure they want to rent. In other words, they want to determine how jobs execution time changes when the available resources (in terms of, e.g., Virtual Machines (VMs) type and

[3] http://www.cineca.it/en/news/pico-cineca-new-platform-data-analytics-applications

their number) changes. However, running experiments in real cloud environments is generally expensive and time consuming. So, exploiting a reasonably accurate model for performance evaluation and prediction of cloud applications is of great importance.

Our work starts from the bootstrapped hybrid performance modeling proposed by Didona and Romano in [23], which is a combined AM/ML modeling approach that brings the strengths of AM methods to compensate the weaknesses of ML techniques, and vice versa. On the one hand, hybrid approaches use analytical modeling, which relies on a priori knowledge of the internals of the target system, known as white box approach. On the other hand, machine learning infers the input/output relationships that map application and system characteristics onto the target performance indicators through statistical models, without requiring the knowledge of internal system details, known as black box approach. While white box techniques pose good *extrapolation* capabilities, i.e., they are able to predict values in regions of the parameters space not sufficiently explored, black box ones provide good *interpolation* capabilities, i.e., they are able to predict values in areas of the features space that have been sufficiently observed during the training phase. So, utilizing bootstrapped hybrid techniques allows for achieving the best of both worlds. In particular, hybrid methods provide: (i) more robust performance predictors that require a small training phase in order to instantiate a performance model, (ii) good predictive performance because of extrapolation capabilities (both borrowed from AM), (iii) the ability to progressively enhance the accuracy of the performance predictor as new data samples from the operational system are gathered, and (iv) good interpolation capabilities (both borrowed from ML).

Didona and Romano's approach exploits an early analytical model to generate an initial set of synthetic data points, which are then fed into a ML model to predict the application performance. Then, the Knowledge Base (KB) of synthetic data is updated by real samples over time following either the merge or replacement strategies to achieve a good accuracy. According to the merge strategy, real samples are collected from the real operational system and added to the synthetic set, while in the replacement case, the nearest neighbor is removed and a new real sample is incorporated to the KB. Consequently, the ML model is also updated and trained according to the new KB. A critical consideration during the aggregation of real data for the training process, is that a limited number of configurations can be analyzed at design time for cloud big data applications. Moreover, in some cases real data samples might be noisy, specially when big data applications run on shared cloud infrastructure [24], and should be consumed conservatively.

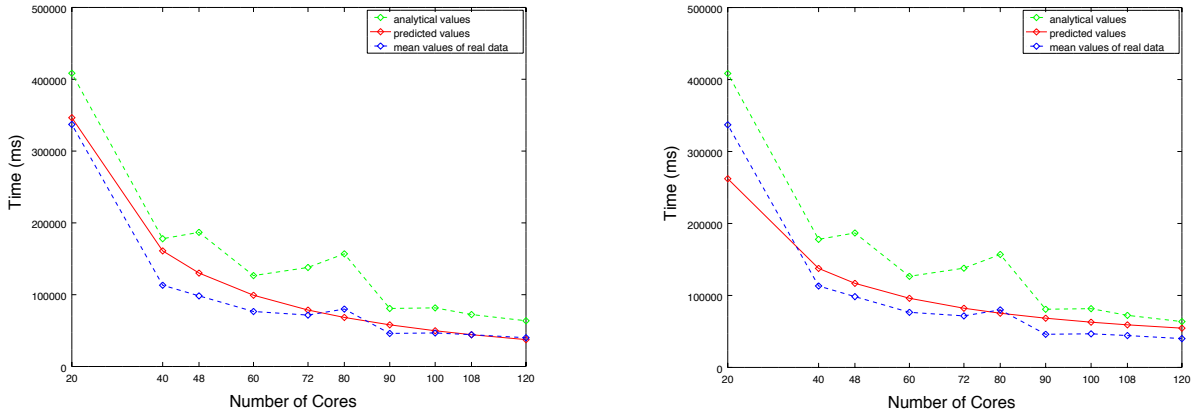
Regarding our work, an iterative procedure was adopted for merging real data from the operational system into the KB. Since in our case, both synthetic and real data come from a limited-size configuration set, if the replacement strategy is selected, in the first iteration of our incremental and iterative process of model selection and training, new real data points evict the synthetic ones of the same configuration. Then, in each subsequent iteration, new real data points evict the old real ones. Therefore, the output model of each iteration will be trained on the latest available set of added (possibly noisy) data points, and the accuracy will not necessarily improve over time. Hence, a model obtained through the replacement strategy cannot be used in action, since relying on few possibly noisy data samples often generates very inaccurate performance predictions.

On the other hand, if the merge strategy is implemented, the results are more accurate and dependable than the ones obtained from the replacement strategy, as soon as the number of iterations becomes large enough. However, since both the size of the configuration set and the number of iterations are rather small in our scenarios, naively applying Didona and Romano's approach, the prediction curve oscillates occasionally during successive iterations and the error may become large even in the last iteration, which can produce unacceptable prediction outputs.

Figures 1a and 1b show some representative results from our initial experiments, which demonstrate the limitations of Didona and Romano's work. The x-axis reports the number of cores supporting the execution of MR jobs on CINECA PICO while the y-axis represents the jobs response time. The response times obtained from the ML model, represented by red curve, are compared with the mean expected values from real experiments, which are represented by the blue curve. Moreover, the green curve refers to the simulation values deriving from the AM and used to form an initial synthetic KB. While Figure 1a is obtained by adding two real operational data points to the initial synthetic data set, Figure 1b is obtained by adding five operational data points.[4] Importing a larger number of operational data points, one would expect that the *prediction curve* gets closer to the *expected values* curve. However, comparing the two graphs, it is evident that adding new points, instead of enhancing the ML model prediction, actually makes the ML output worse.

Thus, though the bootstrapped hybrid approach is a good candidate for performance prediction of cloud big data applications, the work performed by Didona and Romano required a significant extension to provide the generation of sufficiently accurate results while being less dependent to the data samples from the operational

[4]With *adding a data point*, we mean adding an application execution time measure on the operational system for all the considered configurations. Since 10 different configurations have been analyzed, 20 and 50 runs on the target systems were considered in Figure 1 ((a)) and ((b)) when two and five data points are added.



(a) (b)

FIGURE 1. Didona's algorithm output when two operational data points (a) and five operational data points (b) are included in the KB

system with respect to the configurations and usage scenarios.

3. DESCRIPTION OF THE PROPOSED APPROACH

The goal of this paper is to identify hybrid ML models to predict the performance of Hadoop MR, Tez, and Spark applications running on clusters managed by the YARN capacity scheduler. In this section, two analytical models for producing artificial data samples for our hybrid approach are introduced at first. Then, the ML techniques that were examined in this study and the features that were investigated for training and selecting models are discussed. The detailed steps of our proposed hybrid algorithm are introduced afterwards.

3.1. Analytical Models

The first AM we considered is a QN, which is shown in Figure 2. For the sake of simplicity, the model is used for modeling a MapReduce job execution in a cluster of computing servers. Since in general, big data applications have a DAG structure (see e.g., Figure 3), QN fork-joins can describe in the most effective way the parallel execution of DAG nodes tasks [25]. This model can be easily generalized to model also Spark applications and other DAGs in memory frameworks [15, 25].

This analytical model is a closed QN model where the number of concurrent users is assumed to be one, and the user starts submitting jobs at the delay center characterized by the average think time Z. When the user submits her/his job, it is forked into many map task requests, which then enter the Finite Capacity Region (FCR). FCRs model situations where several service centers access resources belonging to a single limited pool, competing to use them [26]. Hence, the

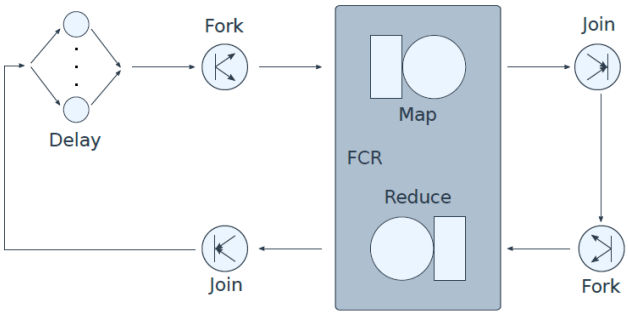


FIGURE 2. Queueing model for MapReduce job execution

FCR enforces an upper bound on the total number of requests served at the same time within itself, allowing tasks to enter according to a First In First Out (FIFO) policy. The FCR includes two multi-service queues that model the map and reduce execution stages. The FCR and multi-service queues capacities are equal to the total number of cores available in the cluster. In this way, we can model the dynamic assignment of YARN containers to map and reduce tasks whenever they are ready. Map tasks are executed by the first multi-service queue and synchronized after completion by joining back to a single job request; the reduce phase is modeled analogously. Note that, the map join is external to the FCR to model that when map tasks complete, they release container cores, which can be assigned to tasks ready in the FCR FIFO queue. Moreover, the reduce fork is also external to the FCR to correctly model applications characterized by a number of reducers larger than the total cluster capacity.

Regarding the second model, we consider a first principle formula which approximates DAGs execution as a sequence of stages respecting precedence constraints (see Figure 3). Such approximation has been proven to

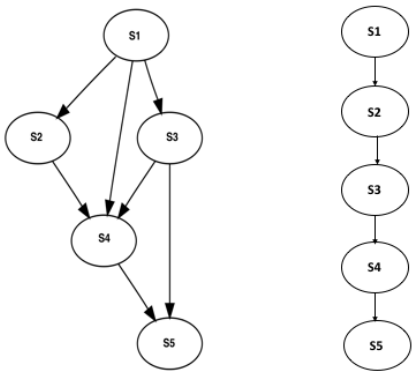


FIGURE 3. DAG and order preserving sequential execution

be accurate in scheduling theory for systems in which the number of tasks is greater than the available resources. This happens very frequently in practice, especially for Spark jobs characterized by thousands of tasks [27] which take few milliseconds. A DAG execution time is estimated as:

$$RTime = \sum_{i=1}^{k} \lceil \frac{n_i}{n_c} \rceil \, t_i \qquad (1)$$

where n_i is the number of tasks, and t_i is the average execution time of each task associated with stage i. Moreover, n_c is the number of available cores during the jobs[5] execution, and k is the total number of stages.

The term $\lceil \frac{n_i}{n_c} \rceil$ equals to the number of waves requested to execute tasks sequentially at stage i. During the first $\lceil \frac{n_i}{n_c} \rceil - 1$ waves, tasks statistically keep the n_c cores busy, while during the last wave, the final tasks which in number are lower than or equal to n_c, complete stage i execution.

3.2. Machine Learning Model

In this work, machine learning is used to regress execution time of Hadoop and Spark applications in a cloud cluster. Different techniques have been investigated (for additional details see [28]) including linear regression, Gaussian Support Vector Regression (SVR), polynomial SVR with degree ranging between 2 and 6, and linear SVR. As feature set, we started from a diverse collection of features including the number of tasks in a map/reduce phase (Spark stage), average and maximum values of tasks execution time, average and maximum values of shuffling time, dataset size, and the number of available cores. The set of relevant features have been obtained by considering the analytical bounds for MR clusters proposed in [29, 13].

Our initial experiments showed that Gaussian SVR and polynomial SVR do not predict accurately and the errors they produce are usually large (up to 175% for the R1 query introduced in Section 4). On the other hand, we found that the best results were most often achieved by linear regression and the linear SVR. However, the linear regression model was unstable in frequent cases when linearly dependent features existed. Filtering linear dependent features required ad hoc analysis and resulted to be query specific. For this reason, we preferred to adopt SVR, which is also more robust to noisy data. Instead of considering the number of cores feature n_c, we considered $1/n_c$ according to the approximate formula (1).

In our work, we perform a model selection process. A four-way data splitting method has been applied for identifying the *best* ML candidate model. The available operational data samples in the KB are partitioned into four disjoint sets, called training set, cross validation set CV_1, cross validation set CV_2, and test set. Moreover, since our goal is to identify models able to achieve generalization, the operational data obtained with the largest configurations are included only in CV_2 (this has been demonstrated a relevant choice for the thresholds optimization step, see the next section and Section 4). While the training set is used for training different alternative models, the CV_1 set is exploited for SVR model selection, whereas CV_2 is used as stopping criterion and to determine the best values for the thresholds of our iterative algorithm. The test set is used for evaluating the accuracy of the selected model [30], and it is restricted to include some specific configurations in order to test interpolation or generalization capabilities.

3.3. Main Algorithm

The pseudo-code of our proposed HML algorithm is shown in 1. A synthetic data set, used to form an initial KB, is generated at line 2 based on the AM discussed in Section 3.1 (i.e., approximated formula or simulation). The KB is then used to select and train an initial ML model at line 3. An iterative procedure is adopted for merging real data from the operational system into the KB, which is implemented at lines 4–17. The operational data for all available configurations is gathered and then merged into KB at lines 5–8. Then the updated KB is shuffled and partitioned at lines 11 and 12 as stated before. Using these sets, line 13 is dedicated to the selection of an ML model between alternatives and retraining it. Then, some error metrics are evaluated at line 14.

At lines 16 and 17, two conditions are checked. Both conditions consider the Mean Average Percentage Error (MAPE) on the training set and on the cross validation set CV_2 (denoted with tr_Error and CV_2_Error in 1)

[5]Spark applications are considered as a single DAG obtained by adding precedence constraints from the very last stage of each job and the initial stages of the following one.

Algorithm 1 Hybrid algorithm

```
1: procedure Hybrid–Algorithm
2:     create a KB using synthetic data generated from AM
3:     select and train an initial ML model
4:     do
5:         for all conf in AvailableConfigs do
6:             gather new data from operational system
7:         end for
8:         merge new data into KB using specified weights
9:         InnerIterations ← 1
10:        do
11:            shuffle KB
12:            partition KB into train, CV1, CV2, and test sets
13:            select and train new ML model
14:            evaluate tr_Error and CV2_Error
15:            InnerIterations ← InnerIterations + 1
16:        while ¬(tr_Error < itrThr ∧ CV2_Error < itrThr) ∧ InnerIterations < MaxInnerIt
17:    while ¬(tr_Error < stopThr ∧ CV2_Error < stopThr) ∧ (additional data is available)
18: end procedure
```

to check whether they are less than specific thresholds ($itrThr$ and $stopThr$, respectively) or not. The error on the training set determines if the model fits well on its training set itself. So, if this error is small enough, the model will avoid underfitting or high bias. On the other hand, the error on the CV_2 set determines if the model has generalization capability. So, if this error is sufficiently small, the model will avoid overfitting or high variance.

If the values of errors for the first condition are not small enough and the maximum number of inner iterations is not reached, the algorithm jumps to line 10 to reshuffle and split the KB into train, CV_1, CV_2, and test sets and choose a different model. On the other side, the second condition prohibits the emission of a weak model. As it will be demonstrated in the following, if a good value is chosen for the two thresholds, the first condition prevents the oscillation problem discussed in Section 2, and the second one stops the procedure as soon as a good model is obtained.

As an example, Figure 4a shows a prediction model which is generated after adding some new noisy data into the KB. The x-axis represents the number of cores for different configurations and the y-axis shows the response time of MR jobs, in milliseconds. In this snapshot, the KB contains one synthetic and five operational samples for each configuration except for 120, for which only one synthetic sample is available and is included in CV_2. The MAPE of this model on point 120 is 72%. Then, this model is rejected by the condition at line 16 since the values of both error metrics tr_Error and CV_2_Error are too high. Therefore, the model of Figure 4a is replaced with the model of Figure 4b by our algorithm. The MAPE of the model of Figure 4b on point 120 is now 19.5%.

On the other hand, if the value of errors for the second condition is not small enough, the algorithm jumps to line 4 to start another iteration. Otherwise, i.e., if both error values are smaller than the $stopThr$ or no new data from operational system is available, the algorithm stops. If the errors are sufficiently small, the current model seems to be good enough for performance prediction, and the iterative process will be stopped to avoid consuming more operational data for the matter of time and cost of real experiments. On the other hand, if no new data is available, the algorithm stops and outputs the last ML model.

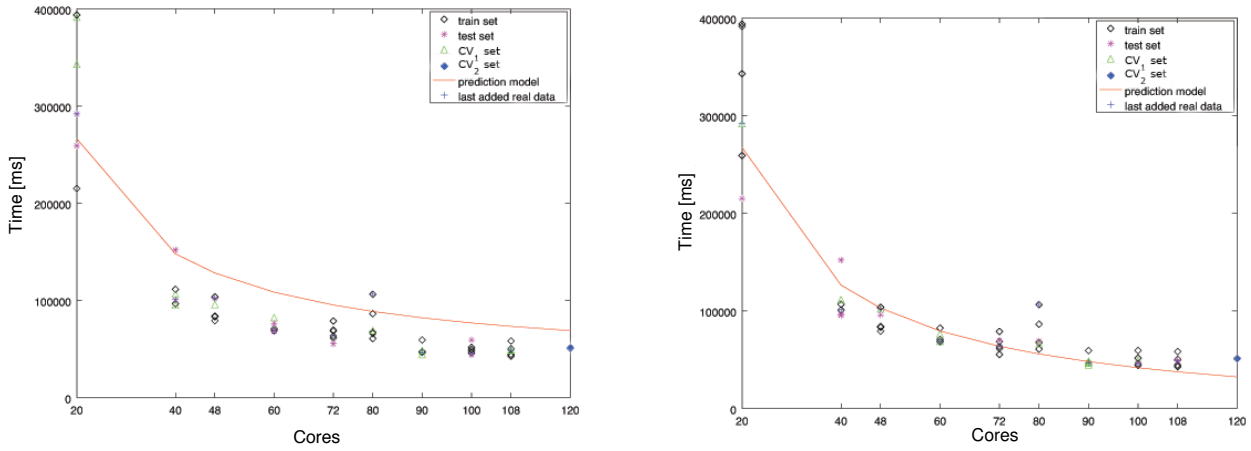
4. EXPERIMENTAL ANALYSIS

In this section, we explore in depth the behavior of our modeling technique when it is applied to MR, Tez, and Spark applications and draw conclusions for the extrapolation and interpolation capability in each case. Initially, a complete description of the experimental setting is provided. Afterwards, we introduce two baseline techniques used to compare the effectiveness of our approach in terms of the adopted performance measures. Finally, four sets of experiments are described extensively.

4.1. Experiments Setting

The datasets used for running the experiments has been generated using the TPC-DS benchmark data generator. We chose TPC-DS [6] as it is the industry standard benchmark for data warehouse systems. Three sets of experiments were conducted and different queries were tested for each technology. In particular, for MapReduce validation the experiments were performed on a Hive ad hoc query called R1 (Figure 5) while for Tez jobs a different query, called Q1 (Figure 6), was considered. Finally, for Spark, query Q40 from the official TPC-DS benchmark was analyzed. The data set size has been set to 250 GB. The experiments were executed on CINECA, the Italian supercomputing center. PICO, the big data cluster available at CINECA, is composed of 74 nodes, each of them boasting two Intel Xeon 10-core 2670 v2@2.5 GHz, with 128 GB of RAM. Out of the 74 nodes, 66 nodes are available for computation. The storage is constituted of 4 PB of high throughput disks based on the GSS technology. For MR experiments we deployed Hadoop 2.5.1 and Apache Tez 0.6.2, while for Spark applications version 1.6.0 was considered. The cluster is shared among different users; resources are managed by the Portable Batch System (PBS), which allows submitting jobs and checking their progress, configuring at a fine-grained level the computational requirements. For all submissions, it is possible to request a number of

[6] http://www.tpc.org/tpcds



(a) (b)

FIGURE 4. An oscillated prediction model after adding noisy data to the KB (a) and the new model obtained by hybrid algorithm (b)

nodes and to define how many CPUs and what amount of memory are needed. The YARN Capacity Scheduler is set up to provide one container per core.

Since the cluster is shared among different users, the performance of single jobs depends on the overall system load, even though PBS tries to split the resources. Due to this, it is possible to have large variations in performance according to the total usage of the cluster and network contention. In particular, storage is not handled directly by PBS, thus leading to an even greater impact on performance. Overall, queries execution required about 20,000 CPU hours on the PICO cluster.

For each configuration, execution runs lying farther from the mean more than three times the standard deviation have been considered outliers and discarded (we verified these measurements were due to exceptionally high resource contention experienced on PICO). In SVR training, weighting is used as a means to suggest the ML to give more relevance and trust to real than to synthetic samples. Therefore, the weights of real data are assumed to be *five* times the weight of analytical data in all experiments regarding the hybrid approach: even if some data points might be noisy, higher weights assigned to real data allows for achieving higher accuracy than pure AMs, noisy data are managed within the inner loop. The number of inner iterations of 1 was set to 10. For numerical computation, GNU Octave is used, while LibSVM [31] is exploited as ML library.

To validate the effectiveness of the proposed HML approach in a comparative manner, two techniques that lack AM are considered:

- *Basic Machine Learning (BML):* this approach relies on SVR for the computation of the regression function. In this case, the algorithm is fed with the same operational data used by our hybrid ML at the last iteration.

```
SELECT avg(ws_quantity),
       avg(ws_ext_sales_price),
       avg(ws_ext_wholesale_cost),
       sum(ws_ext_wholesale_cost)
FROM   web_sales
WHERE  (web_sales.ws_sales_price
        BETWEEN 100.00 AND 150.00) OR
        (web_sales.ws_net_profit
        BETWEEN 100 AND 200)
GROUP BY ws_web_page_sk
LIMIT 100;
```

FIGURE 5. R1 query

```
SELECT avg(ws_quantity),
       avg(ss_net_profit)
FROM   store_sales,catalog_sales
WHERE cs_bil_customer_sk=ss_customer_sk
      AND
      ss_quantity > 10
      AND
      ss_net_profit > 0
LIMIT 100;
```

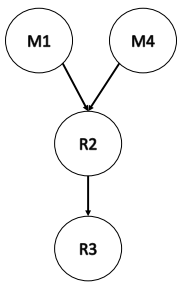


FIGURE 6. Q1 query

- *Iterative Machine Learning (IML):* in this approach, operational data are added iteratively but the initial KB is empty lacking the AM information. In other words, we consider the general structure of 1 except lines 2 and 3 that corresponds to AM involvement.

To compare quantitatively our hybrid approach with BML and IML baseline methods, three performance measures are defined:

- *The MAPE of response time*: this measure focuses on the prediction accuracy. It is defined as the

percentage of relative error of the response time predicted from the learned model with respect to the expected value of response times measured on the operational system. MAPE is used to evaluate both extrapolation and interpolation capabilities.

- *The number of iterations*: this measure indicates the number of iterations of the external loop of 1, which equals the number of real data samples fed into the ML model for each configuration.
- *The cost*: this measure focuses on the cloud expenses for model construction and is defined as:

$$cost = \sum_{i \in AC} N_{c_i} \cdot \widetilde{R}Time_i \cdot N_{e_i} \cdot P \qquad (2)$$

where AC is the set of configurations used for model selection and training, N_{c_i} and $\widetilde{R}Time_i$ are the number of cores and the execution time associated with each configuration i, and N_{e_i} is the number of operational data points used for training the model. Moreover, P is the price of using one core per time unit.

In the following sections, the aforementioned four sets of experiments are described. For the first and second sets of experiments where the MR query is used (Section 4.2 and Section 4.3), the accuracy of the AM is evaluated by comparing the results obtained from the proposed approximation formula and the proposed QN simulation, respectively, with the real experiments in terms of response time. To fairly compare our hybrid approach with the IML, the results of finding the optimal combination of the thresholds that minimizes the error of response time is introduced afterwards. Then, the extrapolation and interpolation capabilities of the approaches are investigated when some points lack from the configuration set (i.e., they are introduced in the validation set only).

It is important to point out that, for the third and fourth sets of experiments considering DAG-based Tez jobs and Spark applications ((Section 4.4 and Section 4.5) our hybrid approach adopts the same thresholds obtained from the two-phase MR job in Section 3.1. Vice versa, the optimal threshold combination for the IML is calculated. In other words, the thresholds for the IML method are optimized for each set of experiments while for our proposed approach, the optimum combination is calculated only in MR case and then, it is used for Tez and Spark queries performance prediction.

4.2. MapReduce Job Analysis with Approximated Formula

In this section, we report the results we have achieved on R1 Hive query considering the approximated formula (1) as AM technique. Several configurations ranging from 20 to 120 cores have been used for this set of experiments. For each configuration, the profiling phase has been conducted extracting the number of map and reduce tasks and their average durations across 20 runs.

4.2.1. Data from Analytical Model

The configuration set for analytical data includes 11 points that represent the different number of cores used for executing MR jobs. Figure 7a plots the average response time of MR job executions versus the number of cores. The average relative error of the values obtained from Equation 1 is around 16% with respect to the mean values of real samples, the maximum relative error is about 31%.

4.2.2. Finding the Optimal Thresholds

To have a fair comparison between the approaches, we found the optimal combination of the $(itrThr, stopThr)$ pair in both hybrid and IML cases. By optimal combination of the two thresholds, we mean the values that minimize the MAPE on the CV_2 set.

We varied the value of $itrThr$ in the range $[25, 40]$ and of $stopThr$ in the range $[10, 30]$ with step 1. For every combination of the two thresholds, the algorithms are run for 50 different seeds to generate different results. Then, the generated results are averaged to compute the mean value of MAPE for every combination of the thresholds. The results of running these experiments showed that the optimum combination of the two thresholds are $(34, 23)$ and $(30, 19)$ for HML and IML approaches, respectively.

4.2.3. Extrapolation Capability on Many Cores

To examine the extrapolation capability of the approaches in the upper region of the configuration set, analyses considered the optimal thresholds from Section 4.2.2, and progressively removed from the training set and cross validations sets CV_1 and CV_2 the configurations with the largest capacity, which are moved to the test set. In other words, in the first scenario the training and CV_1 set included AM data and the real system data only for configurations from 20 to 100 cores while the CV_2 and test sets included experimental data for 108 and 120 cores, respectively. In the second case, we considered real data for the training and CV_1 set from 20 to 90 cores, the CV_2 set included operational data for 100 cores, while the test set included experimental results in the range $[108, 120]$ cores and so on. Then, the error on response time prediction, the number of iterations, and the associated cost of the alternative techniques were compared.

If not differently stated in the remainder of the paper, both the extrapolation and interpolation capabilities analyses will be performed by training 50 different models obtained by setting 50 different *seeds*. These seeds are also different from the ones used to identify the optimal threshold combination.

As can be seen from Figure 7b, the error on the test

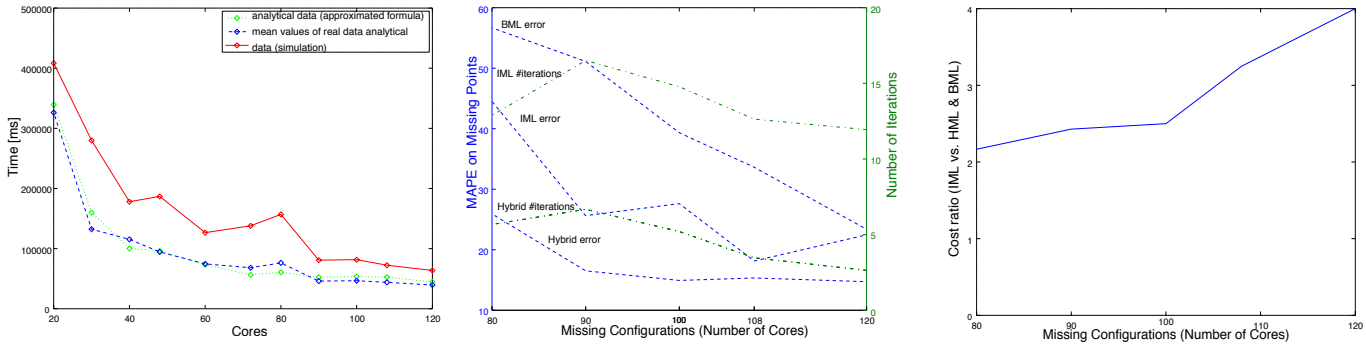


FIGURE 7. (a) Comparison of formula-based approximation, simulation and the mean values of real data, (b) right extrapolation and (c) cost for R1 query

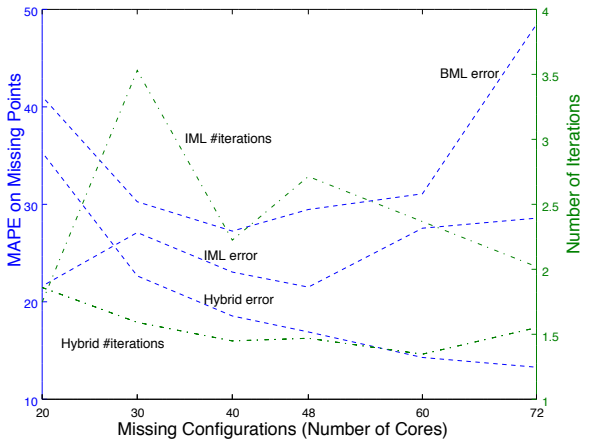


FIGURE 8. Left extrapolation for R1 query

set of the hybrid and the IML across the 50 models are close up to three missing points. By missing points, we mean the number of configurations that are included in the test set but not considered for training. However, when the number of missing points grows, our hybrid approach performs better than the IML. On the other hand, the number of learning iterations of the hybrid approach is between 2 and 6 and is thus rather smaller than the one of IML, which varies between 12 and 16.5. Translating the number of iterations to the cost, the results are shown in Figure 7c, which indicates the cost ratio of IML with respect to our hybrid solution. As the figure shows, IML cost is from 2 up to 4 times larger than our hybrid technique.

4.2.4. Extrapolation Capability on a Few Cores

Next, we examined the extrapolation capability of the approaches in the lower region of the configuration set when only point 20 is included in the test set, and we compared the error on response time prediction, the number of iterations, and the associated costs. In the next steps, moving from the left side of the configuration axis towards the right, we gradually added other points to the test set. As it can be seen from Figure 8, our hybrid approach outperforms both the IML and BML approaches in terms of the MAPE in almost all scenarios. Specifically, as long as the number of missing points is small, the error on the test set of the hybrid and IML are relatively close, but the number of iterations of the hybrid approach is much smaller than the one of the IML. When the number of missing points gradually grows, although the number of iterations of the hybrid and IML models become close, the accuracy of the hybrid approach improves in comparison with the IML.

As shown in Figure 8 and Figure 7b, the values of errors of all approaches in left extrapolation scenarios are generally larger than the ones in right extrapolation scenarios. We can enumerate a few reasons for this behavior: first, the left side of the response time curve is more informative than the right side as depicted in Figure 7a. As a result, the prediction when some data points on the left side of the configurations are missing from the training set is more difficult. Second, the optimization process for finding optimum combinations of the thresholds was run including 120 in the CV_2 set. Hence, it can be expected to get better results for the right than the left extrapolation. Since the accuracy for the left extrapolation on R1 Hive query is low (percentage error greater than 30% is considered too high in performance evaluation practice [32]), as well in the remaining technologies, in the remainder of the paper we omit left extrapolation results for space limitation. Note that from a practical perspective, right extrapolation is more significant than left extrapolation, since end-users are more concerned to predict the performance of a query on larger (and more expensive) system configuration to evaluate the performance gain against the additional cost to be incurred.

4.2.5. Interpolation Capability

To assess the interpolation capability of our proposed HML approach, we considered three different scenarios:

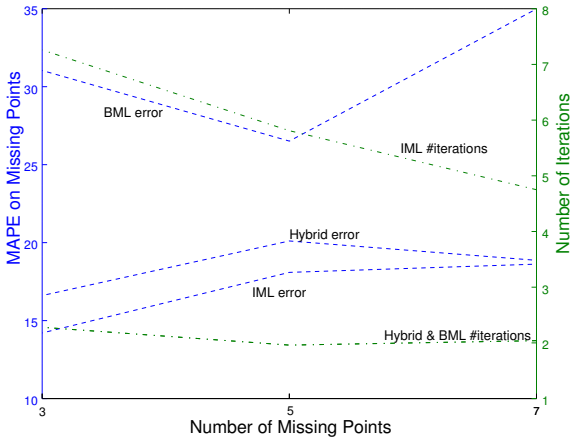


FIGURE 9. Interpolation for R1 query

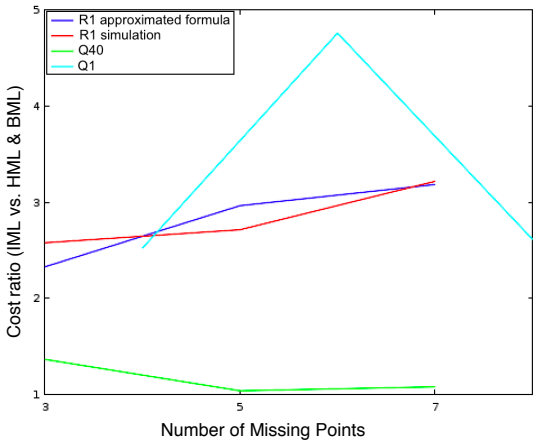


FIGURE 10. Cost of interpolation analyses results

a configuration where the training and CV_1 set included AM data and operational data for: (i) 20, 72, and 120 cores, (ii) 20, 48, 72, 100, and 120 cores, and (iii) 20, 40, 48, 72, 80, 100, and 120 cores. These scenarios are respectively reported in the Figure 9 and Figure 10 on the x-axis with values 7, 5, and 3 as the number of missing points from the full set of configurations available.

As shown in Figure 9, though the error of response time prediction of hybrid approach is slightly worse than that of the IML, the hybrid approach still performs much better than BML. Furthermore, the number of iterations of hybrid ML and BML approaches are between 2 and 3 in all three scenarios, which is smaller than that of IML. Moreover, as shown in Figure 10 (the blue curve), the cost of constructing HML and BML models is much less than that of IML model; e.g., it is almost one third of the cost of the IML approach when three or five points are missing. Thus, our hybrid approach performs better than the BML in terms of accuracy and outperforms the IML in terms of the number of experiments to run and the corresponding cost.

4.3. MapReduce Job Analysis with QN Simulation

The goal of this section is to evaluate our hybrid approach when relying on a less accurate AM, i.e., QN simulation, with respect to the previous section. The lower accuracy was obtained by considering exponential time distribution for map and reduce stages while the best fitting for the reduce stage can be obtained through Erlang (see [25] for further details). In this way, we can verify if our hybrid approach is too sensitive to the accuracy of the AM or if the interpolation and right extrapolation capabilities can be obtained also in such conditions.

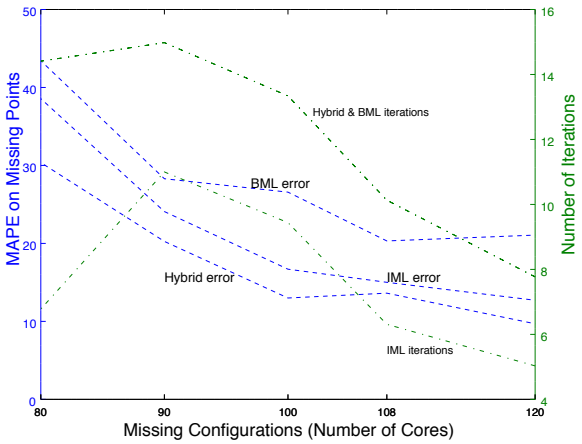


FIGURE 11. Right extrapolation for R1 query (simulation)

4.3.1. Simulation Results

At first, the QN model shown in Figure 2 is used to generate the set of synthetic data samples that feed the initial KB. The JMT [26] with 10% accuracy and 95% confidence interval has been used as QN simulator. The think time, Z, was set to 10 seconds and a single user was considered.

In Figure 7a, the average response times obtained from simulation (red line) are compared with those obtained from real experiments. The average relative error of the values observed from simulations is around 65% with respect to the mean values of real samples, and in the worst case, the relative error reaches 96% which shows that the QN analysis is very conservative.

4.3.2. Finding the Optimum of the Thresholds

To have a fair comparison between the two techniques, the optimum combination of the $(itrThr, stopThr)$ thresholds was determined as in the previous section both for our hybrid and the IML approach. The optimum combination of the two thresholds are $(38, 24)$

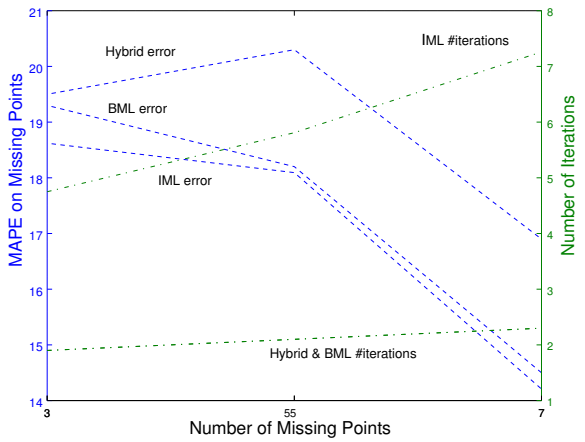


FIGURE 12. Interpolation for R1 query (simulation)

and $(30, 19)$ for HML and IML approaches, respectively.

4.3.3. Extrapolation Capability for Many Cores
The results of the right extrapolation are reported in Figure 11. Our hybrid approach outperforms the IML approach in terms of the MAPE in all scenarios. When the only missing point is 120, the error of our hybrid approach is around 11% in contrast to around 14% of IML. On the other hand, when half of the rightmost points of the configuration set are missing, hybrid approach achieves 30% error on the test set while IML obtains about 38%.

4.3.4. Interpolation Capability
For assessing the interpolation capability of our proposed approach, we followed an identical procedure and considered the same configurations analyzed in Section 4.2.5. As shown in Figure 12, the error of response time prediction of the hybrid approach is slightly worse than the one of IML and BML, while its number of iterations are significantly lower than IML.

The maximum relative error in the case of HML reaches 20.5% while for IML and BML it is equal to 19.5% and 18.5%, respectively. Moreover, as shown in Figure 10, the cost of hybrid and BML models is much less (up to 3x) than the IML model one.

4.4. Tez DAG-based Job Analysis with Approximate Formula

To validate the effectiveness of the hybrid approach and to investigate if the optimal thresholds determined for R1 query in Section 4.2 can also be generally applied to other big data technologies, we perform another set of experiments on a Hive query, called Q1 shown in Figure 6. The profiling phase has been conducted by extracting the number of tasks and the average task durations from around fifteen runs of each configuration. The configuration set for analytical data contains 12 points that are representatives of the number of available cores when executing Apache Tez jobs. These points include 2, 3, 4, 5, 6, 8, 9, 10, 12, 15, 16, and 20 cores. This set is equal to the set of system configurations we used for gathering real data samples from the operational system.

4.4.1. Data from Analytical Model
Equation 1 is used to approximate response time of MR jobs executing Q1 query and to produce the initial set of synthetic data samples.

In Figure 13a, the response times determined through Equation 1 are compared with the expected values of those obtained from real experiments. While the average relative error of the values obtained from approximation is around 15% with respect to the mean values of real samples, the maximum relative error is 45%.

4.4.2. Finding the Optimal Thresholds
In order to examine the generalization capabilities of our proposed method, the optimum combination of the $(itrThr, stopThr)$ determined in Section 4.2.2, $(34, 23)$, is used to run the hybrid approach. Vice versa, the optimum finding process is run again for the IML approach based on Q1 data to provide the best situation for IML. The optimum values of $(itrThr, stopThr)$ thresholds obtained for IML are $(26, 15)$.

4.4.3. Extrapolation Capability on Many Cores
To evaluate the extrapolation capability of the proposed method from the right side, we started running the three approaches when only 20 cores are missing (i.e., 20 cores configuration is included in the test set) and we gradually added other configurations to the set of missing points. Next, the values of performance measures are compared for the three approaches. The results of the comparison are depicted in Figure 13b and Figure 13c.

As can be seen from Figure 13b, the hybrid approach outperforms the IML and BML ones in terms of the MAPE in all scenarios, ranging from when the rightmost point of the configuration set (i.e., 20) is the only missing point, to when five rightmost points of the configuration set (i.e., 10, 12, 15, 16, and 20) are missing. When the only missing point is 20, the error of our hybrid approach is around 19% in contrast to around 34% and 40% of IML and BML approaches, respectively. On the other hand, when five of the rightmost points of the configuration set are missing, hybrid approach achieves around 19% of error on average on the test set in contrast to about 38% and 56% of IML and BML approaches, respectively.

The average number of IML iterations ranges between 2.5 and 3.5 while it is always below 2 in all extrapolation scenarios of HML and BML approaches. The cost of model construction for hybrid, BML, and

IML approaches are compared in Figure 13c. The cost of the IML approach is much larger (up to 2.5x) than the one of the hybrid and BML techniques.

4.4.4. Interpolation Capability

To evaluate the interpolation capability on Q1 query, we considered three different scenarios, which correspond to the following configurations (in terms of number of cores) included in the train and CV_1 sets: (i) 2, 5, 10, and 20, (ii) 2, 3, 8, 9, 16, and 20, and (iii) 2, 3, 5, 6, 10, 12, 16, and 20. These scenarios are respectively determined by 8, 6, and 4 as the number of missing points (with respect to the full set of operational data) on the horizontal axis of Figure 14.

As shown in Figure 14, the relative error increases when the number of missing points rises. The hybrid and the BML approaches have the best and the worst prediction accuracy, respectively. The error of our hybrid approach when the real data for just one third of the configuration set is available remains under 14%. On the other hand, the number of iterations of hybrid approach is smaller than that in IML approach and remains under 3 in all scenarios. Moreover, as shown in Figure 13c, the cost of hybrid model construction is much lower than the IML one. Thus, our hybrid technique performs better than the BML in terms of accuracy and performs better than the IML on all metrics.

4.5. Spark Job Analysis with Approximate Formula

In order to examine the predictions techniques on Apache Spark, we performed the last set of experiments on the official Q40 query of the benchmark TPC-DS, whose DAG is shown in Figure 15, using the approximated formula in Equation 1 as AM. In particular, we followed a similar process as reported in Section 4.2, using the same set of configurations for both analytical and real data as in R1 case. The profiling phase has been conducted extracting the number of tasks and the average task durations from around ten runs with the same configuration.

4.5.1. Data from Analytical Model

In Figure 16a we compare the average execution times for every configuration obtained across the experiments to the ones obtained from Equation 1. The average relative error of the values obtained from the approximation formula, is around 34%.

4.5.2. Finding the Optimal Thresholds

The optimal combination of the $(itrThr, stopThr)$ was determined only for the IML approach and was equal to $(25, 15)$. For every threshold combination the algorithm run for 50 different seed values to generate different results. Vice versa, in the case of hybrid approach, we used the same thresholds $(34, 23)$ obtained for R1 query in Section 4.2.

4.5.3. Extrapolation Capabilities on Many Cores

Right extrapolation capability analysis results are reported in Figure 16 (b) and (c). Figure 16b shows that the use of the proposed approach defeats the IML, providing always a lower MAPE on the test set. What is remarkable is that while we are moving to the left side, where more points are missing the MAPE error of IML increases dramatically, demonstrating the dominance of our proposed approach. In particular, when only 120 cores configuration is missing, the error of the hybrid model is 7% approximately, contrary to 15% of the IML. However, when 5 points are missing from the configuration set, the error of IML shoots up to 30% while in case of hybrid algorithm a small increase is observed (11%).

4.5.4. Interpolation Capability

Concerning interpolation, we considered three different scenarios when applying hybrid algorithm to Q40 query: (i) three points 20, 72 and 120 cores are missing (ii) five points 20, 48, 72, 100, 120 cores are missing, and finally (iii) seven points 20, 40, 48, 72, 80, 100 and 120 cores are missing. Concerning the relative error, we observe from Figure 17 that the IML approach gives better accuracy compared to our technique. Although the hybrid error for three missing points is high (17%), in case of seven missing points the hybrid algorithm gets closer to IML performance. However, the number of iterations of the hybrid approach is smaller than the one of IML and lower costs can be obtained as reported in Figure 10.

5. RELATED WORK

In recent years, ML became popular to predict the performance of complex computer systems. Yigitbasi *et al.* [33] have compared several ML methods to predict Hadoop clusters performance, ranging from ordinary linear regression to advanced techniques like artificial neural networks, regression trees, and SVR with diverse MR applications and cluster configurations. The authors in [34] have proposed AROMA, a system based on SVR for automatic resource allocation and configuration in cloud-based clusters. AROMA mines historical execution data in order to profile past submissions and to match incoming jobs to the available performance signatures for prediction. In this way, the proposed system can avoid deadline violations stated in SLAs incurring minimum cost.

Venkataraman *et al.* [14] have built Ernest, a black box performance prediction framework for large-scale analytics based on ML. They used optimal experiment design to collect the minimum number of training points. The accuracy of the proposed approach was evaluated on Amazon EC2 using different business analytics workloads based on Spark MLlib. The

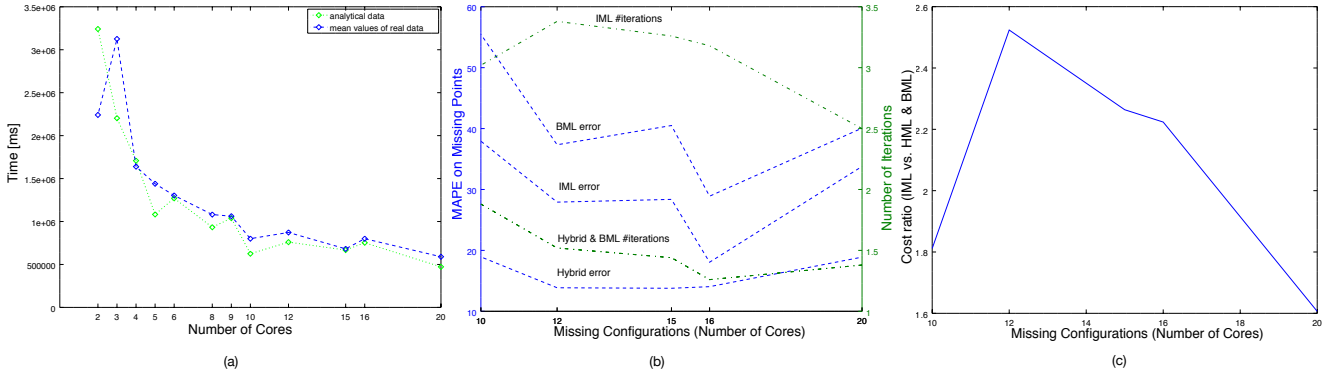


FIGURE 13. (a) Comparison of formula-based approximation and the mean values of real data, (b) right extrapolation and (c) cost for Q1 query

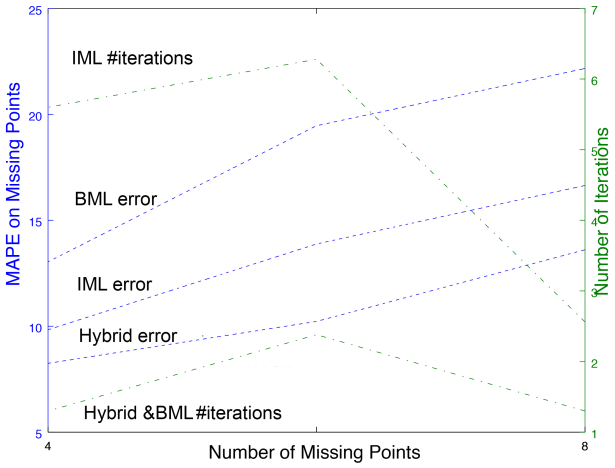


FIGURE 14. Interpolation for Q1 query

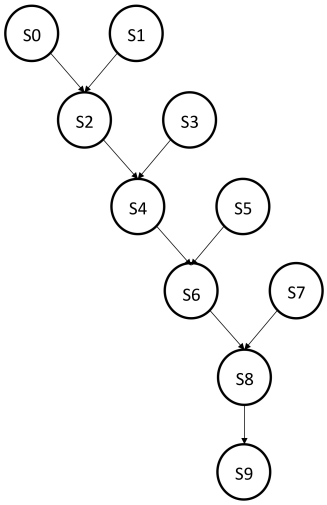


FIGURE 15. Q40 query DAG

evaluation showed that the average prediction error is under 20%.

Alipourfard *et al.* [35] have presented CherryPick, a black box system that leverages Bayesian optimization to unearth the optimal or near-optimal cloud configurations that minimize cloud usage cost guaranteeing application performance. Similar ideas were also exploited in the design of Hemingway [36], which embeds the *Ernest* model and it is specialized in the identification of the optimal cluster configuration for Spark MLlib based applications. Hemingway takes into account some machine learning algorithm peculiarities and adopts experiment design to collect as few training points as possible.

Delimitrou and Kozyrakis [37] have proposed Quasar, a black box cluster management system that maximizes resource utilization while meeting performance and QoS constraints. The authors exploit classification techniques to determine the impact of the type, amount of resources, and workload interference on the system performance.

Some of the related works, exploited the possibility to use AM and ML in synergy to get the best of both worlds. Tesauro *et al.* [39] have proposed an autonomic resource allocation in a multi-application prototype data center with the goal of maximizing the total applications expected business value. They show how to combine the strengths of both Reinforcement Learning (RL) and queuing models in a hybrid approach, in which RL trains offline on data collected while a queuing model policy controls the system. Thereska and Ganger [40] have presented a hybrid performance modeling framework which uses the redundancy of high-level system specifications described through models and low-level system implementation to localize system-model inconsistencies. The final goal is to give hints to the system and model designer regarding the root-cause of the problem. Queuing-based mathematical models were coupled with Decision Tree (DT) regressors in [40]. Herodotou *et al.* [41] have proposed Elastisizer, a system to which users can express cluster sizing problems as queries in a declarative fashion. In this system, the overall process of estimating execution time and cost of MR jobs is broken down into four smaller steps and, for each step, a suitable white-box or black-box modeling approach is chosen.

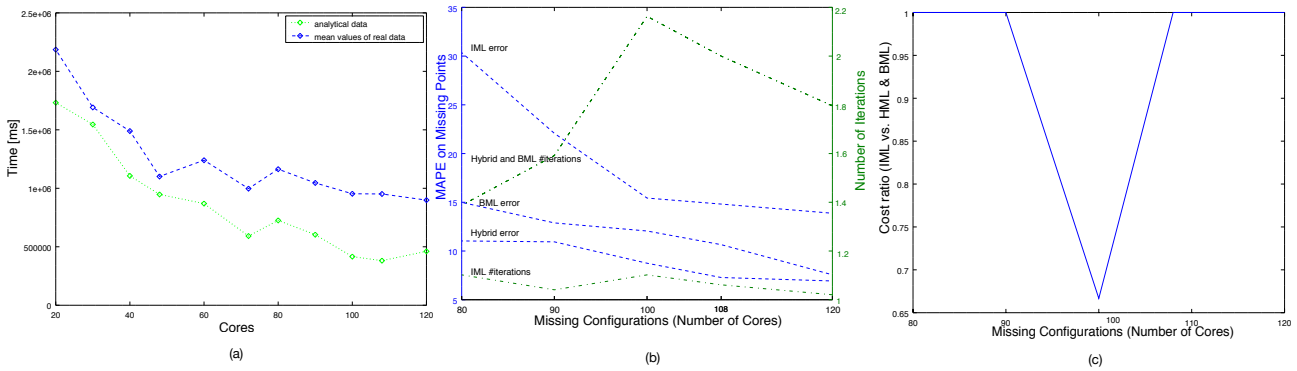


FIGURE 16. (a) Comparison of formula-based approximation and the mean values of real data, (b) right extrapolation and (c) cost for Q40 query

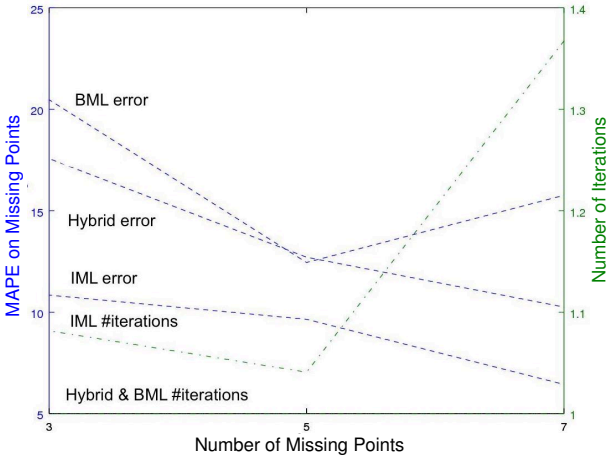


FIGURE 17. Interpolation for Q40 query

There are also hybrid works that target in-memory transactional data stores: Rughetti *et al.* [42] have used a mixed AM/ML approach to dynamically tune the level of concurrency of applications based on software transactional memory to optimize system throughput. The AM and ML techniques used in this research are parametric analytical modeling and neural networks, respectively. Didona *et al.* [43] have introduced Transactional Auto Scaler (TAS), a system for automating the scaling of fully-replicated in-memory transactional data grids in cloud platforms. In TAS, analytical and ML models were incorporated to predict throughput, commit probability, and average response time. White box models, based on queuing theory, are used to capture the dynamics of concurrency control/replication algorithms to forecast the effects of data contention, as well as the effects of contention due to CPU utilization. Didona *et al.* [44] have considered the issue of automatically identifying the optimal degree of parallelism of an application using distributed software transactional memory by introducing a hybrid approach. They exploit TAS [43] as the analytical-based performance model, while DT regression is utilized as the machine learning technique.

Dalibard *et al.* [45] have developed BOAT, a gray box framework, which allows developers to build efficient auto-tuners for complex computer system. BOAT is based on structured Bayesian optimization. The authors demonstrated how to optimally tune the scheduling of a neural network training on a heterogeneous cluster. Didona *et al.* [46], have investigated a technique whose main idea consists of relying on an AM to generate a KB of synthetic data over which a complementary ML is initially trained. The initial KB is then updated over time to incorporate real samples from the operational system. For updating the KB, the authors proposed different algorithms based on merge and replacement. As case studies, they considered Infinispan and Total Order Broadcast (TOB) relying on DT regression and queuing models. The effect of the proposed parametrized algorithms on the mean average percentage error of the gray box model was evaluated by means of ten-fold cross validation.

6. CONCLUSIONS

In this paper, we proposed a novel hybrid machine learning technique which is able to use AM and ML in synergy to model and predict the execution time of jobs running on the most widely used big data frameworks. With respect to the state of the art work, our approach is particularly effective to predict the performance of big data applications when shared physical environments characterized by high resource contention are considered. Moreover, our method outperforms baseline machine learning techniques, providing always better extrapolation capabilities, better interpolation capabilities in many cases and reducing significantly the costs (up to 4.5x) and the number of training samples to be gathered from the operational system, even without optimizing the

threshold parameters. Overall, the MAPE that can be achieved on the test set ranges between 7% and 30%, even when the AM is not accurate.

In our research agenda, we plan to investigate the effectiveness of the approach when using more accurate AMs (e.g., Stochastic Petri Nets) and to adopt the models for run time management of big data applications.

ACKNOWLEDGEMENTS

The results of this work have been partially funded by EUBra-BIGSEA (grant agreement no. 690116), a Research and Innovation Action funded by the European Commission under the Cooperation Programme, Horizon 2020 and the Ministério de Ciência, Tecnologia e Inovação, RNP/Brazil (grant GA0000000650/04). Eugenio Gianniti is also partially supported by the DICE Horizon 2020 research project (grant agreement no. 644869). Machine learning model training have been performed on Microsoft Azure under the *Top Compsci University Azure Adoption* program. CINECA experiments have been performed thanks to the LISA program within the UMQ project.

REFERENCES

[1] Shirer, M. and Goepfert, J. The digital universe in 2020. `https://www.idc.com/getdoc.jsp?containerId=prUS44998419`. Accessed: September 2019.

[2] Mohamed, S. H., El-Gorashi, T. E., and Elmirghani, J. M. (2019) A survey of big data machine learning applications optimization in cloud data centers and networks. *arXiv preprint arXiv:1910.00731* , **?**

[3] Almendros-Jiménez, J. M., Becerra-Terón, A., and Torres, M. (2017) Integrating and querying OpenStreetMap and linked geo open data. *The Computer Journal*, **62**, 321–345.

[4] Wang, H. and Cao, Y. (2019) An energy efficiency optimization and control model for hadoop clusters. *IEEE Access*, **7**, 40534–40549.

[5] Qadeer, A., Waqar Malik, A., Ur Rahman, A., Mian Muhammad, H., and Ahmad, A. (2019) Virtual infrastructure orchestration for cloud service deployment. *The Computer Journal* , **?**

[6] Hopkins, B. Move big data to the public cloud with an insight PaaS. `https://go.forrester.com/blogs/insight-paas-accelerate-big-data-cloud`. Accessed: November 2019.

[7] Saha, B., Shah, H., Seth, S., Vijayaraghavan, G., Murthy, A., and Curino, C. (2015) Apache tez: A unifying framework for modeling and building data processing applications. *Proc. of the 2015 ACM SIGMOD international conference on Management of Data*, pp. 1357–1369. ACM.

[8] Ferreira, R. S. and Pereira, D. A. (2019) Bigfeel—a distributed processing environment for the integration of sentiment analysis methods. *The Computer Journal* , **?**

[9] Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., and Stoica, I. (2010) Spark: Cluster computing with working sets. *Proc. of HotCloud 2010*.

[10] Zaharia, M., Xin, R. S., Wendell, P., Das, T., Armbrust, M., Dave, A., Meng, X., Rosen, J., Venkataraman, S., Franklin, M. J., Ghodsi, A., Gonzalez, J., Shenker, S., and Stoica, I. (2016) Apache spark: A unified engine for big data processing. *Communications of the ACM*, **59**, 56–65.

[11] Woodie, A. A decade later, apache spark still going strong. `https://www.datanami.com/2019/03/08/a-decade-later-apache-spark-still-going-strong/`. Accessed: October 2019.

[12] Lin, M., Zhang, L., Wierman, A., and Tan, J. (2013) Joint optimization of overlapping phases in MapReduce. *SIGMETRICS Performance Evaluation Review*, **41**, 16–18.

[13] Verma, A., Cherkasova, L., and Campbell, R. H. (2011) ARIA: automatic resource inference and allocation for MapReduce environments. *Proc. of the 8th ACM international conference on Autonomic computing*, pp. 235–244. ACM.

[14] Venkataraman, S., Yang, Z., Franklin, M. J., Recht, B., and Stoica, I. (2016) Ernest: Efficient Performance Prediction for Large-Scale Advanced Analytics. *Proc. of 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI16)*, pp. 363–378.

[15] Gibilisco, G. P., Li, M., Zhang, L., and Ardagna, D. (2016) Stage aware performance modeling of dag based in memory analytic platforms. *Proc. of 9th International Conference on Cloud Computing (CLOUD)*, pp. 188–195. IEEE.

[16] Ardagna, D., Ghezzi, C., and Mirandola, R. (2008) Rethinking the use of models in software architecture. *Proc. of international Conference on the Quality of Software Architectures*, pp. 1–27. Springer.

[17] Polo, J., Becerra, Y., Carrera, D., Steinder, M., Whalley, I., Torres, J., and Ayguad, E. (2013) Deadline-Based MapReduce Workload Management. *IEEE Trans. on Network and Service Management*, **10**, 231–244.

[18] Casale, G., Tribastone, M., and Harrison, P. G. (2014) Blending randomness in closed queueing network models. *Perform. Eval.*, **82**, 15–38.

[19] Ataie, E., Gianniti, E., Ardagna, D., and Movaghar, A. (2016) A Combined Analytical Modeling Machine Learning Approach for Performance Prediction of MapReduce Jobs in Cloud Environment. *Proc. of SYNASC*.

[20] Didona, D., Quaglia, F., Romano, P., Torre, E., and Roma, U. (2015) Enhancing Performance Prediction Robustness by Combining Analytical Modeling and Machine Learning. *Proc. of ACM/SPEC International Conference on Performance Engineering*.

[21] Isaila, F., Balaprakash, P., Wild, S. M., Kimpe, D., Latham, R., Ross, R., and Hovland, P. (2015) Collective I/O tuning using analytical and machine learning models. *Proc. of IEEE Cluster Computing*, pp. 128–137.

[22] Sanzo, P. D., Quaglia, F., Ciciani, B., Pellegrini, A., Didona, D., Romano, P., Palmieri, R., and Peluso, S. (2015) A flexible framework for accurate simulation of cloud in-memory data stores. *Simulation Modelling Practice and Theory*, **58**, 219–238.

[23] Didona, D. and Romano, P. (2015) Hybrid Machine Learning/Analytical Models for Performance Prediction: a Tutorial. *Proc. of ACM/SPEC International Conference on Performance Engineering*.

[24] Ardagna, D., Barbierato, E., Evangelinou, A., Gianniti, E., Gribaudo, M., Pinto, T., Guimarães, A., Couto da Silva, A. P., and Almeida, J. M. (2018) Performance prediction of cloud-based big data applications. *Proc. of the ACM/SPEC International Conference on Performance Engineering*, pp. 192–199. ACM.

[25] Ardagna, D., Bernardi, S., Gianniti, E., Aliabadi, S. K., Perez-Palacin, D., and Requeno, J. I. (2016) Modeling performance of hadoop applications: A journey from queueing networks to stochastic well formed nets. *Proc. of international Conference on Algorithms and Architectures for Parallel Processing*, pp. 599–613. Springer.

[26] M.Bertoli, G.Casale, and G.Serazzi (2009) JMT: performance engineering tools for system modeling. *ACM SIGMETRICS Performance Eval. Review*, **36**, 10–15.

[27] Ousterhout, K., Rasti, R., Ratnasamy, S., Shenker, S., and Chun, B.-G. (2015) Making sense of performance in data analytics frameworks. *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI15)*, pp. 293–307.

[28] Rizzi, A. M. (2016) Support vector regression model for BigData systems. *ArXiv e-prints* , **?**

[29] Malekimajd, M., Ardagna, D., Ciavotta, M., and Rizzi, A. M. (2015) Optimal Map Reduce Job Capacity Allocation in Cloud Systems. *ACM SIGMETRICS Perf. Evaluation Review*, **42**, 51–61.

[30] Arlot, S., Celisse, A., et al. (2010) A survey of cross-validation procedures for model selection. *Statistics surveys*, **4**, 40–79.

[31] Chang, C.-C. and Lin, C.-J. (2011) LIBSVM: a library for support vector machines. *ACM Trans. on Intelligent Systems and Technology*, **2**, 1–27.

[32] Lazowska, E. D., Zahorjan, J., Graham, G. S., and Sevcik, K. C. (1984) *Quantitative System Performance*. Prentice-Hall.

[33] Yigitbasi, N., Willke, T. L., Liao, G., and Epema, D. (2013) Towards machine learning-based auto-tuning of MapReduce. *Proc. of 21st International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems*, pp. 11–20. IEEE.

[34] Lama, P. and Zhou, X. (2012) Aroma: Automated resource allocation and configuration of mapreduce environment in the cloud. *Proc. of the 9th international conference on Autonomic computing*, pp. 63–72. ACM.

[35] Alipourfard, O., Liu, H. H., Chen, J., Venkataraman, S., Yu, M., and Zhang, M. (2017) CherryPick: Adaptively Unearthing the Best Cloud Configurations for Big Data Analytics. *Proc. of 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI17)*.

[36] Pan, X., Venkataraman, S., Tai, Z., and Gonzalez, J. (2017) Hemingway: modeling distributed optimization algorithms. *arXiv preprint arXiv:1702.05865* , **?**

[37] Delimitrou, C. and Kozyrakis, C. (2014) Quasar: resource-efficient and QoS-aware cluster management. *ACM SIGPLAN Notices*, pp. 127–144. ACM.

[38] Ibeid, H., Meng, S., Dobon, O., Olson, L., and Gropp, W. (2019) Learning with analytical models. *Proc. of the IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 778–786. IEEE.

[39] Tesauro, G., Jong, N. K., Das, R., and Bennani, M. N. (2007) On the use of hybrid reinforcement learning for autonomic resource allocation. *Cluster Computing*, **10**, 287–299.

[40] Thereska, E. and Ganger, G. R. (2008) IRONModel: Robust Performance Models in the Wild. *Proc. of ACM SIGMETRICS*.

[41] Herodotou, H., Dong, F., and Babu, S. (2011) No One (Cluster) Size Fits All: Automatic Cluster Sizing for Data-intensive Analytics Categories and Subject Descriptors. *Proc. of ACM Symposium on Cloud Computing*.

[42] Rughetti, D., Sanzo, P. D., Ciciani, B., Quaglia, F., and Universit, S. (2014) Analytical/ML Mixed Approach for Concurrency Regulation in Software Transactional Memory. *Proc. of IEEE/ACM International Symposium in Cluster, Cloud, and Grid Computing (CCGrid)*.

[43] Didona, D., Romano, P., Peluso, S., and Quaglia, F. (2014) Transactional Auto Scaler : Elastic Scaling of Replicated In-Memory Transactional Data Grids. *ACM Trans. on Autonomous and Adaptive Systems*, **9**, 1–32.

[44] Didona, D., Felber, P., Harmanci, D., Romano, P., and Schenker, J. (2015) Identifying the optimal level of parallelism in transactional memory applications. *Computing*, **97**, 939–959.

[45] Dalibard, V., Schaarschmidt, M., and Yoneki, E. (2017) Boat: Building auto-tuners with structured bayesian optimization. *Proc. of the 26th International Conference on World Wide Web*, pp. 479–488. International World Wide Web Conferences Steering Committee.

[46] Didona, D. and Romano, P. (2014) On bootstrapping machine learning performance predictors via analytical models. *arXiv preprint arXiv:1410.5102* , **?**