A Combined Analytical Modeling Machine Learning Approach for Performance Prediction of MapReduce Jobs in Hadoop Clusters

Ehsan Ataie[1], Eugenio Gianniti[2], Danilo Ardagna[2], and Ali Movaghar[1]

[1]Department of Computer Engineering, Sharif University of Technology, Tehran, Iran
[2]Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Milan, Italy

Abstract

Nowadays MapReduce and its open source implementation, Apache Hadoop, are the most widespread solutions for handling massive dataset on clusters of commodity hardware. At the expense of a somewhat reduced performance in comparison to HPC technologies, the MapReduce framework provides fault tolerance and automatic parallelization without any efforts by developers. Since in many cases Hadoop is adopted to support business critical activities, it is often important to predict with fair confidence the execution time of submitted jobs, for instance when SLAs are established with end-users. In this work, we propose and validate a hybrid approach exploiting both queuing networks and support vector regression, in order to achieve a good accuracy without too many costly experiments on a real setup. The experimental results show how the proposed approach attains a 21% improvement in accuracy over applying machine learning techniques without any support from analytical models.

Keywords: Analytical performance modeling, machine learning, cloud computing, MapReduce.

1 Introduction

The MapReduce framework is a programming model and a scalable and fault-tolerant run-time environment [1] that became the most popular platform for data analytics because of its simplicity, generality, and maturity [2]. A data processing request under the MapReduce framework, called a job, consists of two types of tasks: map and reduce. A map task reads one data chunk and processes it to produce intermediate results. Then, reduce tasks fetch these intermediate results and carry out further computations to generate the final result [3]. Cloud platforms make MapReduce an attractive framework for organizations that need to process large datasets, but lack the computing and human resources to install and manage a cluster. Hadoop is an open source implementation of MapReduce used in production deployments and is used for applications like log file analysis, database querying, web indexing, report generation, machine learning research, scientific simulation, bioinformatics, and financial analysis [4], [5]. IDC estimates that Hadoop touched half of the world data last year [6], supporting both traditional batch and interactive data analysis applications [7].

One of the main challenges in MapReduce environments is to predict the execution time of Hadoop jobs. One approach to do such prediction is to develop analytical models based on queueing networks (QNs), Petri nets (PNs), stochastic activity networks, and so on for predicting performance metrics. However, analytically modeling big data applications is very challenging due to the great number of parameters that have to be investigated, especially in large Hadoop 2.x environments, where resources are

dynamically allocated between the map and reduce stages. Furthermore, stakeholders of performance prediction of a Hadoop process are its users in our scenarios. Actually, a non-expert MapReduce user is allowed to provision a cluster of any size on the target cloud within minutes to meet her data-processing needs [4]. To ensure analytical model (AM) tractability in such complex systems, AM-based performance models typically rely on simplifying assumptions that reduce their accuracy.

On the other hand, 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. In recent years, a growing number of successful researches were done to explore the possibility of using ML techniques to predict performance of complex computer systems [8]–[10]. Therefore, ML can also be exploited for predicting the execution time of Hadoop jobs. To be able to predict accurately, the ML model should be built during a training phase with a sufficient number of experimental data from different workloads, using different parameters and different configurations. However, running several experiments in cloud environment would be costly and time consuming. On the other hand, though ML often provides good accuracy in regions for which it is well trained, it shows poor decision in regions for which none or very few samples are known.

Gray box modeling [11]–[13] is a new approach for performance modeling and prediction that tries to achieve the best of the AM and ML worlds by mixing the two. Such models can be exploited to support design-time decision-making during the development and deployment phases of big data applications. These models can then be also kept alive at run-time to conduct the dynamic adjustment of the system configuration [14].

Our focus in this paper is to provide a design-time combined AM/ML model to estimate MapReduce jobs execution time in Hadoop clusters. At first, an AM based on QNs is proposed to initially model the execution of MapReduce jobs. The results obtained from simulation are then exploited as analytical data. 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 form a more accurate performance predictor. In addition, some intuitions are exploited in our approach to output more accurate predictions while consuming less data from noisy environment operational systems. Then, our hybrid approach and the pure ML approach are compared in terms of the prediction accuracy and the size of consumed operational data. The results show that our hybrid approach outperformed the pure ML one in terms of accuracy by about 21% when up to half of the real data points of the configuration set are not used on the higher end of the configuration space, and by about 25% when up to one fifth of the points are left out on the lower end.

The remainder of this paper is organized as follows. In Section 2, the background and problem statement is presented. Section 3 represents our approach for combined AM/ML performance prediction of big data applications in a cloud environment. Section 4 reports some experimental results to validate and study the properties of our models. In Section 5 we compare our work with other researches available in the literature. Finally, the conclusions and future work are drawn in Section 6.

2 Problem Statement

There are cases where MapReduce users need to know how long their job execution will take using different number of available resources to decide on the configuration of the infrastructure they want to rent. In other words, they want to determine how a job execution time changes when the number of available resources 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. The experiments used in this study have been run on PICO[1], the big data infrastructure of CINECA, the Italian supercomputing center.

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

Bootstrapped hybrid performance modeling [15] is a combined AM/ML modeling approach that brings the strengths of AM methods to compensate the weaknesses of ML techniques, and vice versa. Such an approach exploits an early analytical model to generate an initial set of synthetic data points, which are then fed into a machine learning model to enable it to predict the application performance. The knowledge-base (KB) of synthetic data is then updated over time to incorporate new samples collected from the real operational system. The machine learning model is also updated and trained according to the new KB. However, since the experimental environment used for executing MapReduce jobs in this work is shared among different users' applications, performance metrics are usually varying a lot. Therefore, real data samples are likely to be noisy and should be consumed conservatively.

On the one hand, hybrid approaches use analytical modeling, which relies on a-priori knowledge of the internals of the target system and is known as *white box* technique. On the other hand, they use machine learning, which relies on inferring the input/output relationships that map application and system characteristics onto the target performance indicators, and on encoding such relationships via statistical models, requires no knowledge of the internal details of the system, and is known as *black box* technique. While white box techniques are good for their *extrapolation* capabilities, i.e., to predict values in regions of the parameters space not sufficiently explored, black box ones are good for *interpolation* capabilities, i.e., to predict values in areas of the features space that have been sufficiently observed during the training phase. So, utilizing bootstrapped hybrid approaches enables us to achieve the best of both worlds. In particular, it provides: (i) a more robust performance predictor that requires a small training phase in order to instantiate a performance model (borrowing from AM), (ii) good extrapolation capabilities (borrowing from AM), (iii) the ability to progressively enhance the accuracy of the performance predictor as new data samples from the operational system are gathered (borrowing from ML), and (iv) good interpolation capabilities (borrowing from ML).

The initial idea of incorporating the combination of analytical modeling and machine learning in this research starts from the work by Didona and Romano [16], in which the authors proposed different strategies based on *merge* and *replacement* for updating the initial synthetic data KB.

In our work, both synthetic and real data come from a limited-size configuration set. Thus, if the replace 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 set of added data points. Such a model is not acceptable in action, because our experiments show that the data from the operational system is very noisy and so relying on one data sample for each configuration will often generate very inaccurate predictions.

Furthermore, when the merge strategy is chosen and implemented, all the synthetic and real data are kept in the KB and are used for model selection and training. Therefore, the results are more accurate and dependable than the ones based on the replace strategy, when 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, 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.

Thus, though the bootstrapped hybrid approach is a good candidate for performance prediction of cloud applications, the work done by Didona and Romano requires some extension to guarantee the generation of sufficiently accurate results while being less dependent to the data samples from the operational system with respect to our configurations and usage scenarios.

3 Description of the Proposed Approach

In this section, the QN model used for producing analytical data samples for our hybrid approach is 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 reviewed. Using the QN and an appropriate ML technique as the main building blocks, the detailed steps of our proposed hybrid algorithm is introduced afterwards.

3.1 Analytical Model

The QN used for analytically modeling MapReduce job execution in a cluster of computing servers is depicted in Figure 1. It is a closed QN model where the number of concurrent users is assumed to be one and she start off in the delay center, characterized by the average think time Z. When the user submits her job, it is forked into as many map task requests as stated in the job profile, 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. Hence, the 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 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 in order to model that when map tasks complete, they release containers, which can be assigned to tasks ready in the FCR FIFO queue. Moreover, the reduce fork is also external to the FCR to model correctly applications characterized by a number of reducers larger than the total cluster capacity.

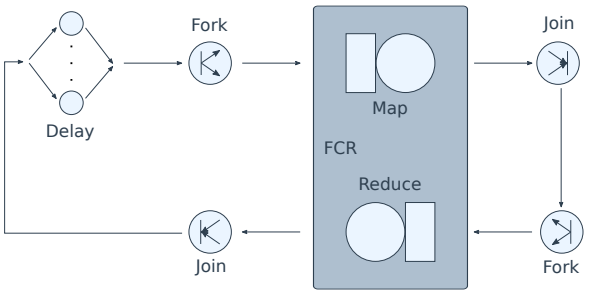

Figure 1: QN model for MapReduce job execution

3.2 Machine Learning Model

In this work, machine learning is used to regress execution time of MapReduce jobs in a cloud cluster. Different techniques are investigated 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 map and reduce tasks, average and maximum values of map execution time, average and maximum values of reduce execution time, average and maximum values of shuffling time, dataset size, and the number of available cores. The set of relevant features are obtained by considering the analytical bounds for MapReduce clusters published in [17], [18].

Our initial experiments showed that Gaussian SVR and polynomial SVR do not predict accurately and the errors they produce are usually large. On the other hand, we found that the linear regression and the linear SVR are the ones that most often achieved the best results. However, the linear regression model was unstable in frequent cases when we had linearly dependent features, leading to meaningless results. Thus, we finally chose linear SVR as the main technique and $1/nCore$ was the single feature for our study, where $nCore$ is the total number of cores of the configuration.

Whenever training ML models and selecting the best among them is needed, available data samples in

the KB are partitioned into three disjoint sets, called training set, cross validation (CV) set, and test set. While the training set is used for training different alternative models, the CV set is exploited for selecting the best amongst the alternatives. Furthermore, the test set is used for evaluating the generalization capability and accuracy of the selected model.

3.3 Main Algorithm

The pseudo-code of our proposed hybrid algorithm is shown in Algorithm 3.1. A synthetic data set which is used to form an initial KB is generated by simulating the reference AM at line 2. The KB is then used to select and train an initial ML model at line 3. Since the real data samples are very noisy, we need to avoid the dependency on this data as much as we can. So, an iterative procedure is adopted for merging real data from the operational system into the KB, which is explained at lines 4–15 of Algorithm 3.1. 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 10 and 11 as stated before. Using these sets, line 12 is dedicated to the selection of an ML model between alternatives and retraining it. Then some error metrics are measured at line 13.

At lines 14 and 15, two conditions are checked. Both conditions consider the mean absolute percentage errors (MAPEs) on the training set and on the test set (shown as $trError$ and $tstError$ in Algorithm 3.1) 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 test 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 is not small enough, the algorithm jumps to line 9 to reshuffle the KB and choose a different model. This condition is critical at the final iteration, when training the model output by the whole process, but since we suppose that the number of iterations is not known beforehand, we check this condition at the end of each iteration to prohibit the emission of a weak model. In other words, if a good value is chosen for the threshold, this condition will prevent the oscillation problem we talked about in Section 2.

Algorithm 3.1 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
9:          do
10:             shuffle KB
11:             partition KB into train, CV, and test sets
12:             select and train new ML model
13:             measure trError and tstError
14:         while ¬(trError < itrThr ∧ tstError < itrThr)
15:     while ¬(trError < stopThr ∧ tstError < stopThr) ∧ (more new data is available)
16: end procedure
```

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 can be stopped to avoid consuming more operational data for the matter of time and cost of real experiments and the noisiness of real samples. On the other hand, if no new data is available, the algorithm should be stopped and output the last ML model.

4 Experimental Analysis

The goal of this section is to compare our hybrid approach with the pure ML approach. The pure ML approach is supposed to obey the structure of Algorithm 3.1, except for line 2, which relies on the AM. Actually, we also examined a base version of pure ML approach that does not include checking of the two conditions, for which the mean relative error for one missing point (i.e., 120) was around 24% while 20 data samples from the operational system were consumed for each available point. This result is far away from that of our hybrid approach. So, for the sake of brevity, we ignore the details of the results of this base version of pure ML in the remainder of this paper.

All these experiments have been performed 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@2.5GHz, with 128 GB of RAM. Out of the 74 nodes, 66 are available for computation. In our experiments on PICO, we have used several configurations ranging from 20 to 120 cores and set up the scheduler to provide one container per core. The storage is constituted by 4 PB of high throughput disks based on the GSS technology. The cluster is shared among different users, hence resources are managed by the Portable Batch System (PBS), which allows for 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 nodes and to define how many CPUs and how much memory are needed on each of them.

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 overall usage of the cluster. In particular, storage is not handled directly by PBS, thus leading to an even greater impact on performance.

The dataset used for running the experiments has been generated using the TPC-DS benchmark data generator, creating at a scale factor of 250 GB several files directly used as external tables by Hive. We chose the TPC-DS benchmark as it is the industry standard for benchmarking data warehouses. We performed experiments on a Hive query, called R1, as shown in Figure 2. The profiling phase has been conducted extracting average task durations from at least twenty runs of each query. GNU Octave[2] is used for numerical computation, while LibSVM [19] is chosen as machine learning library.

```
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 2: R1 query

In the following subsection, the accuracy of our AM is considered by comparing the results of simulating the QN model shown in Figure 1 with the real experiments in terms of response time. To fairly compare our hybrid approach with the pure ML one, the results of finding the combination of the two thresholds that minimizes the error of each approach is introduced afterwards. Finally, the extrapolation capabilities of the two approaches are investigated when some points lack from either the right-most or left-most side of the configuration set.

4.1 Simulation Results of the Analytical Model

At first, the analytical model shown in Figure 1 is used to generate the set of synthetic data samples. To do this, Java Modeling Tool [20] is exploited for trace-based simulation of the queueing model, with a 10% accuracy and 95% confidence interval. The think time, Z, is set to 10 seconds. The configura-

[2] www.gnu.org/s/octave

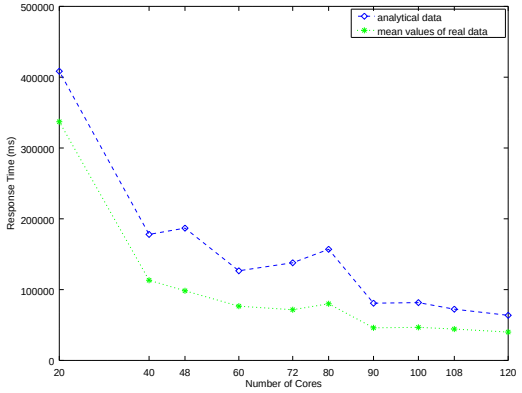

Figure 3: Comparison of simulation results and mean values of real data

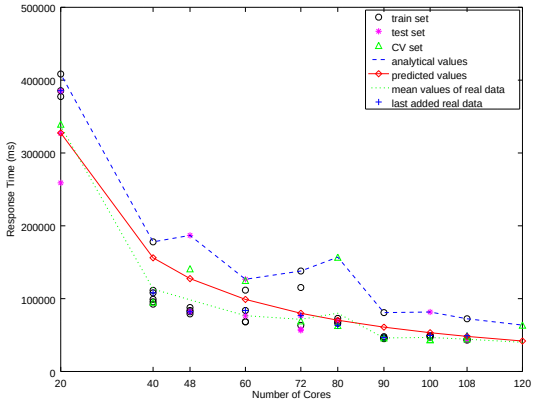

Figure 4: A snapshot of the final state of the hybrid approach

tion set for analytical data contains 10 points, which are representatives of the total number of available cores when executing MapReduce jobs. These points include 20, 40, 48, 60, 72, 80, 90, 100, 108, and 120. This set is the same as the one for real data samples coming from the operational system.

In Figure 3, the response times acquired from simulation runs are compared with those obtained from real experiments. The point is that the average relative error of the values observed from simulations is around 65% with respect to the mean values of the real samples, and in the worst case, the relative error reaches up to 96%. These results show that the analysis provides only an inaccurate and conservative estimate.

4.2 Finding the Optimal Thresholds

To have a fair comparison between the two approaches, we try to find the optimal combination of the $(itrThr, stopThr)$ couple once for our hybrid approach and once for the pure ML approach. By optimal combination of the two thresholds, we mean the values that minimize the MAPE on the missing configurations, provided that the latter are known from the operational system, but are not fed into the algorithm.

To achieve this, the data for the point 120 is assumed to lack, while the data for the other nine points is supposed to be available from the real system. We change the value of $itrThr$ from 25 to 41 and the value of $stopThr$ from 10 to 30. For every combination of the two thresholds, the algorithms are run against 100 different values of the randomization seed. Then the generated results are averaged to compute the mean value of MAPE for every thresholds combination. The results of running these experiments showed that the optimum combination of the two thresholds are $(38, 24)$ and $(30, 19)$ for the hybrid and pure ML approaches, respectively.

Figure 4 shows the state of our hybrid approach at the end of the sixth iteration for some value of the seed, where both conditions of the Algorithm 3.1 are satisfied and, thus, the algorithm exits while predicting the response time of the missing point with a 4.4% error. The x-axis shows the number of cores for different configurations and the y-axis represents the response time of MapReduce jobs, in milliseconds. All the points in the configuration set except 120 are assumed to be available. In this snapshot, the KB contains one synthetic and six operational samples for each configuration—except for 120, for which only one synthetic sample is available—and is entirely partitioned into training, CV, and test sets, which are discriminated using different symbols in

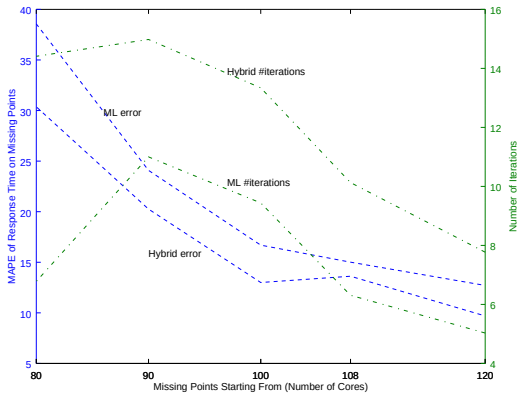

Figure 5: Extrapolation on many cores

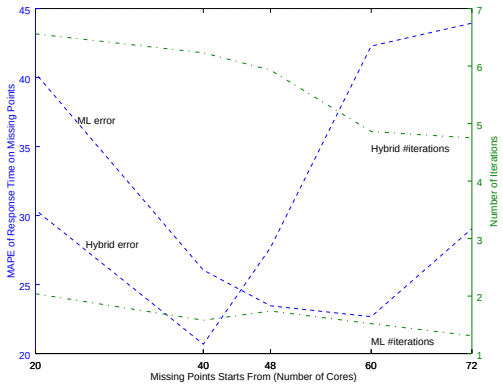

Figure 6: Extrapolation on few cores

Figure 4. The prediction curve obtained from the final ML model is also plotted. The total number of real data samples used in this scenario is equal to the number of available configuration points (i.e., 9) multiplied by the number of iterations (i.e., 6).

4.3 Extrapolation Capability on Many Cores

To examine the extrapolation capability of the two approaches in the upper region of the configuration set, we started running them using their optimal thresholds when only point 120 is set aside. To avoid bias, we used 100 values for the seed, different from those used in the optimum finding process. Then we compared the value of MAPE and the number of iterations for the two approaches. Moreover, moving from the right side of the configuration axis to the left, we gradually added other points to the set of left out real data. The results of the comparison are depicted in Figure 5.

As can be seen from Figure 5, our hybrid approach outperforms pure ML in terms of MAPE in all the scenarios, ranging from when the right-most point of the configuration set (i.e., 120) is the only missing point, to when the five right-most points of the configuration set (i.e., 80, 90, 100, 108, and 120) are set aside. When the only missing point is 120, the error of our hybrid approach is around 11% in contrast to around 14% of the pure ML. On the other hand, when the right-most half of real data points in the configuration set are not used, the hybrid approach finishes with 30% of error in contrast to about 38% of pure ML. These values show an improvement of more than 21% on average on the value of MAPE.

The total number of real samples consumed in the algorithm are proportional to the number of iterations that each approach executes. Specifically, the used real samples are 9 times the number of iterations. The number of iterations of the hybrid approach, despite being bigger than in the pure ML one, changes between 8 and 15, which means it needs a rather small amount of real data to output relatively accurate predictions. The point is that all the improvements were acquired in situations where the average error of the AM with respect to the mean values of real data samples is more than 73% for the missing points, which means the AM was not so accurate.

4.4 Extrapolation Capability on Few Cores

Next, we examined the extrapolation capability of the two approaches in the lower region of the configuration set by running them using their optimum

thresholds when only point 20 lacks. Similarly to the previous extrapolation analysis, 100 different values for the seed were used. Then we compared MAPE and the number of iterations for the two approaches. Moreover, moving from the left side of the configuration axis towards the right, we gradually added other points to the set of missing points. The results of the comparison are shown in Figure 6.

As can be seen from Figure 6, our hybrid approach outperforms the pure ML one with respect to MAPE by around 25% when one (i.e., 20) or two (i.e., 20 and 40) points are missing. However, when three, four, and five points are set aside, the MAPE of the hybrid approach becomes greater than the one from pure ML. The number of iterations of the hybrid approach, even if bigger than in pure ML, changes between 4.5 and 7, which means it needs absolutely small amounts of operational data to output its predictions.

Furthermore, the errors in both approaches are generally bigger than the ones in the previous extrapolation scenarios. Moreover, starting from 20, the slope of the MAPE curves are negative at first and then become positive. We can enumerate a few reasons for the strange behavior of the MAPE curves in left extrapolation scenarios in contrast with the right extrapolation ones: first, the left side of the prediction curve is more informative than the right side as depicted in Figure 4. So, it is reasonable to be harder to predict when some data points lack on the left side of the configurations. Second, starting from 20 cores, we have operational data for almost every tens of CPUs. However, due to the existing limitations on running experiments in real cloud, we could not exceptionally run any experiment for around 30 cores, which resides on the left side of the configuration set. Third, the optimization process for finding the optimal threshold combinations were run setting aside the 120-core configuration, so it is logical that the results for right extrapolation be better than the ones for left extrapolation.

5 Related Work

There are several works for performance modeling and evaluation that adopt machine learning techniques: Ipek *et al.* [10] adopt artificial neural networks to predict the impact of architectural changes on performance metrics while studying memory system, microprocessor, and multithreaded chip multiprocessor. To predict performance of Hadoop clusters, Yigitbasi *et al.* [8] compare several ML methods, ranging from ordinary linear regression to advanced techniques like artificial neural networks, model trees, and SVR with diverse MapReduce applications and cluster configurations. The authors in [9] propose a system, AROMA, 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 Service Level Agreements incurring minimum cost, with an average accuracy on completion time prediction around 12%.

Although combining AM and ML in synergy can lead to get the best of both worlds, hybrid AM/ML techniques for performance modeling and evaluation have received little attention by the literature.

Tesauro *et al.* [21] propose an autonomic resource allocation in a multi-application prototype data center with the goal of maximizing the total expected business value summed over the applications. 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 [22] present 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 and give hints to the system and model designer regarding the root-cause of the problem. Queuing-based mathematical models are coupled with decision tree (DT) regressors that estimate by keeping track of historical information about their predictions. Herodotou *et al.* [4] propose *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 MapReduce jobs is broken down into four smaller steps and, for each step, a suitable white box or black box modeling approach is chosen. Chen *et al.* [23] present *Ensemble*, a framework for performance inquiring in modern database environments such as data centers and clouds. Ensemble uses *model templates*, provided by a user/analyst or retrieved from a repository of models previously derived for other applications. Model templates, which can be either white box or black box, express the beliefs of analysts about analytical performance models for Ensemble to fit or validate with experimental data. Combining white-box models for deriving the count and sizes of communication and storage operations and black-box models for the performance of the individual operations, Isaila *et al.* [12] present a hybrid model for I/O autotuning to evaluate execution time as the performance metric. For the learning algorithm, the authors test a wide range of solutions such as neural networks, support vector machines, linear regression, random forest, and Cubist [24], thus showing that no single algorithm performs the best.

There are also a considerable number of works that target in-memory transactional data stores: Rughetti *et al.* [25] use 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 their research are parametric analytical modeling and neural networks, respectively. Didona *et al.* [26] introduce Transactional Auto Scaler (TAS), a system for automating the scaling of fully-replicated in-memory transactional data grids in cloud data platforms. In TAS, analytical and ML models are 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 to predict the effects of contention due to CPU utilization. On the other hand, black-box techniques, based on DT regression, are used to predict performance at the network level. In a similar work, Didona and Romano [27] present *PROMPT*, a performance model for partially-replicated in-memory transactional cloud stores, in which the effects of replication degree and data locality on data contention require more complicated models in contrast to [26]. The authors validate PROMPT's accuracy through experiments based on Infinispan [28] and the YCSB benchmark [29], using both private and public cloud infrastructures. In [27], queuing theory and DT regression are used as white-box and black-box techniques, respectively. Sanzo *et al.* [13] propose a framework for modeling cloud transactional in-memory data stores. Since the actual data-transport/networking infrastructure on top of which the data grid is deployed might be unknown, it is not feasible to model it via white-box techniques such as simulation. So, the proposed framework integrates simulation and DT regression techniques, the latter being used to capture the dynamics of the data-exchange layer across the cache servers. Didona *et al.* [30] consider 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 [26] as the analytical-based performance model, while DT regression is utilized as the machine learning technique.

There are some works in the literature that consider the application of Hybrid approaches on Total-Order Broadcast (TOB) protocols: Romano and Leonetti [31] use hybrid techniques to automate the tuning of the batching level of a Sequencer-based Total Order Broadcast (STOB) protocol. Their model relies on a combination of queuing theory and RL techniques. Didona *et al.* [11] propose three algorithms that allow for the synergistic use of AM and ML models. The algorithms are based on the common idea of building an ensemble of different methodologies. In order to evaluate the proposed algorithms, the response time prediction of a STOB service and the throughput prediction of an application deployed over Infinispan are considered as case studies and the results compared with the results obtained from the Cubist tool.

The closest research to our work is done by Didona and Romano [16], who investigate a technique

whose main idea consists in 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 propose different algorithms based on merge and replacement. As case-studies, the authors consider Infinispan and TOB. While for both case-studies DT regression is used as ML technique, PROMPT and queueing theory are used as AM techniques for Infinispan and TOB, respectively. The effect of the proposed parametrized algorithms on the mean average percentage error of the gray box model is evaluated by means of ten-fold cross validation.

6 Conclusions

In this paper, we used the power of AM and ML in synergy to model and predict the execution time of MapReduce jobs in Hadoop environments in a more accurate, less expensive, and less time consuming manner. Our preliminary results have shown how our hybrid approach is effective in predicting the execution time by reducing the MAPE of pure ML by more than 21% when extrapolation over many cores is examined, thus making it suitable for capacity planning decisions at design-time. For extrapolation on few cores, our approach outperforms the pure ML one by 25%, reducing the MAPE when one or two points are left out. We think that adding more configurations to the lower end of the configuration set will increase the accuracy of the hybrid approach, hence this will be our next plan.

In future, we also plan to investigate the interpolation capability of our hybrid approach in contrast to pure ML. For interpolation, we will assume that some points in the middle of the configuration set lack, while the other points are available. Future work will also include adding weight parameters to both synthetic and real data. Weighting can be used as a means to suggest the ML to give more relevance and trust to real than to synthetic samples. Another plan will be to use more accurate analytical models for generating the synthetic data, like fluid PNs: increasing the initial data samples accuracy, we expect a better prediction power of the final hybrid models.

Extending the current models and algorithm to investigate multi-user scenarios and also to consider Tez and Spark applications, where a Tez directed acyclic graph node or Spark stage is associated to a corresponding multi-server queue, is another research line that we are going to study in the future.

References

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[2] Y. Zhang, S. Chen, Q. Wang, and G. Yu, "i2MapReduce: Incremental MapReduce for Mining Evolving Big Data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 7, pp. 1906–1919, 2015.

[3] W. Wang, K. Zhu, L. Ying, J. Tan, and L. Zhang, "MapTask Scheduling in MapReduce With Data Locality: Throughput and Heavy-Traffic Optimality," *IEEE/ACM Transactions on Networking*, vol. 24, no. 1, pp. 190–203, 2016.

[4] H. Herodotou, F. Dong, and S. Babu, "No One (Cluster) Size Fits All: Automatic Cluster Sizing for Data-intensive Analytics," in *2nd ACM Symposium on Cloud Computing*, 2011.

[5] X. Xu, M. Tang, and S. Member, "A New Approach to the Cloud-based Heterogeneous MapReduce Placement Problem," *IEEE Transactions on Services Computing*, vol. PP, no. 99, pp. 1–12, 2015.

[6] K. Kambatla, G. Kollias, V. Kumar, and A. Grama, "Trends in Big Data analytics," *Journal of Parallel and Distributed Computing*, vol. 74, no. 7, pp. 2561–2573, 2014.

[7] C. Shanklin, *Benchmarking Apache Hive 13 for enterprise Hadoop*, `http://hortonworks.com/blog/benchmarking-apache-hive-13-enterprise-hadoop`, [Online; accessed 10-July-2016].

[8] N. Yigitbasi, T. L. Willke, G. Liao, and D. Epema, "Towards Machine Learning-Based Auto-tuning of MapReduce," in *IEEE 21st International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems*, 2013, pp. 11–20.

[9] P. Lama and X. Zhou, "AROMA: automated resource allocation and configuration of mapreduce environment in the cloud," in *9th international conference on Autonomic computing*, 2012, pp. 63–72.

[10] E. Ipek, S. A. McKee, B. R. de Supinski, and R. Caruana, "Efficiently exploring architectural design spaces via predictive modeling," in *12th international conference on Architectural support for programming languages and operating systems*, 2006, pp. 195–206.

[11] D. Didona, F. Quaglia, P. Romano, E. Torre, and U. Roma, "Enhancing Performance Prediction Robustness by Combining Analytical Modeling and Machine Learning," in *6th ACM/SPEC International Conference on Performance Engineering*, 2015, pp. 145–156.

[12] F. Isaila, P. Balaprakash, S. M. Wild, D. Kimpe, R. Latham, R. Ross, and P. Hovland, "Collective I/O tuning using analytical and machine learning models," in *2015 IEEE International Conference on Cluster Computing*, 2015, pp. 128–137.

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

[14] J. Polo, Y. Becerra, D. Carrera, M. Steinder, I. Whalley, J. Torres, and E. Ayguad, "Deadline-Based MapReduce Workload Management," *IEEE Transactions on Network and Service Management*, vol. 10, no. 2, pp. 231–244, 2013.

[15] D. Didona, P. Romano, I.-i. I. S. Técnico, and U. D. Lisboa, "Hybrid Machine Learning/Analytical Models for Performance Prediction: a Tutorial," in *The 6th ACM/SPEC International Conference on Performance Engineering*, 2015, pp. 341–344.

[16] D. Didona and P. Romano, "On Bootstrapping Machine Learning Performance Predictors via Analytical Models," 2014. arXiv: `:1410.5102v1`.

[17] M. Malekimajd, D. Ardagna, M. Ciavotta, P. Milano, and A. M. Rizzi, "Optimal Map Reduce Job Capacity Allocation in Cloud Systems," *ACM SIGMETRICS Performance Evaluation Review*, vol. 42, no. 4, pp. 51–61, 2015.

[18] A. Verma, L. Cherkasova, and R. H. Campbell, "ARIA: Automatic Resource Inference and Allocation for MapReduce Environments," in *8th ACM international conference on Autonomic computing*, 2011, pp. 235–244.

[19] C.-C. Chang and C.-J. Lin, "LIBSVM: a library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, no. 27, pp. 1–27, 2011.

[20] M.Bertoli, G.Casale, and G.Serazzi, "JMT: performance engineering tools for system modeling," *ACM SIGMETRICS Performance Evaluation Review*, vol. 36, no. 4, pp. 10–15, 2009.

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

[22] E. Thereska and G. R. Ganger, "IRONModel: Robust Performance Models in the Wild," in *ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, 2008, pp. 253–264.

[23] J. Chen, G. Soundararajan, S. Ghanbari, and C. Amza, "Model Ensemble Tools for Self-Management in Data Centers," in *Data Engineering Workshops (ICDEW), 2013 IEEE 29th International Conference on*, 2013, pp. 36–43.

[24] *Data Mining with Cubist*, `http://www.rulequest.com/cubist-info.html`, [Online; accessed 25-June-2016].

[25] D. Rughetti, P. D. Sanzo, B. Ciciani, F. Quaglia, and S. Universit, "Analytical/ML Mixed Approach for Concurrency Regulation in Software Transactional Memory," in *Cluster, Cloud and Grid Computing (CCGrid), 14th IEEE/ACM International Symposium on*, 2014, pp. 81–91.

[26] D. Didona, P. Romano, S. Peluso, and F. Quaglia, "Transactional Auto Scaler : Elastic Scaling of Replicated In-Memory Transactional Data Grids," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 9, no. 2, pp. 1–32, 2014.

[27] D. Didona and P. Romano, "Performance Modeling of Partially Replicated In-Memory Transactional Stores," in *IEEE 22nd International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems*, 2014, pp. 265–274.

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

[29] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," in *Proc. of SOCC*, 2010, pp. 143–154.

[30] D. Didona, P. Felber, D. Harmanci, P. Romano, and J. Schenker, "Identifying the optimal level of parallelism in transactional memory applications," *Computing*, vol. 97, no. 9, pp. 939–959, 2015.

[31] P. Romano and M. Leonetti, "Self-tuning Batching in Total Order Broadcast Protocols via Analytical Modelling and Reinforcement Learning," in *International Conference on Computing, Networking and Communications, Network Algorithm & Performance Evaluation Symposium*, 2012, pp. 786–792.