2013 IEEE 21st International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems

Towards Machine Learning-Based Auto-tuning of MapReduce

Nezih Yigitbasi, Theodore L. Willke, and Guangdeng Liao
Intel Labs
Hillsboro, OR
{nezih.yigitbasi,theodore.l.willke,guangdeng.liao}@intel.com

Dick Epema
Delft University of Technology
the Netherlands
D.H.J.Epema@tudelft.nl

Abstract—**MapReduce, which is the de facto programming model for large-scale distributed data processing, and its most popular implementation Hadoop have enjoyed widespread adoption in industry during the past few years. Unfortunately, from a performance point of view getting the most out of Hadoop is still a big challenge due to the large number of configuration parameters. Currently these parameters are tuned manually by trial and error, which is ineffective due to the large parameter space and the complex interactions among the parameters. Even worse, the parameters have to be re-tuned for different MapReduce applications and clusters. To make the parameter tuning process more effective, in this paper we explore machine learning-based performance models that we use to auto-tune the configuration parameters. To this end, we first evaluate several machine learning models with diverse MapReduce applications and cluster configurations, and we show that support vector regression model (SVR) has good accuracy and is also computationally efficient. We further assess our auto-tuning approach, which uses the SVR performance model, against the Starfish auto-tuner, which uses a cost-based performance model. Our findings reveal that our auto-tuning approach can provide comparable or in some cases better performance improvements than Starfish with a smaller number of parameters. Finally, we propose and discuss a complete and practical end-to-end auto-tuning flow that combines our machine learning-based performance models with smart search algorithms for the effective training of the models and the effective exploration of the parameter space.**

I. Introduction

The low cost of data acquisition and storage has enabled industry to store massive amounts of data with the hope of driving their innovation. MapReduce, the programming model of the data center, and Hadoop, the most popular open source MapReduce implementation, have taken a huge step towards solving the problem of processing these *big data* in a scalable and fault-tolerant way, and made it possible to process tens of petabytes of data daily [1]–[3]. However, performance-wise getting the most out of Hadoop is non-trivial; as of Hadoop 0.20.2 there are around 190+ configuration parameters and 10+ of these parameters have impact on the application performance [1].

On the one hand the large parameter space enables a wide range of opportunities for significant performance improvement by careful parameter tuning (Section III), but on the other hand the large parameter space and the complex interactions among the parameters make manual tuning time consuming and difficult. Therefore, the research community has recently started exploring *auto-tuning* Hadoop configuration parameters [4]. Auto-tuning involves two phases: model building and parameter optimization (Section VI). In the first phase, the auto-tuner establishes a performance model to predict the performance of an application given a parameter configuration. Then, using this performance model in the parameter optimization phase, the auto-tuner searches for the optimal parameter values. Therefore, the performance model is of crucial importance to the auto-tuner, and the quality of the auto-tuner depends strictly on the quality of the performance model. In this paper, we explore the feasibility of machine learning-based performance models as we move towards a new framework for the auto-tuning of Hadoop MapReduce.

Cost-based (analytical) performance modeling is a well-known approach that has proven itself useful in diverse computer systems, but it suffers from several drawbacks when modeling MapReduce applications. First, cost-based modeling is a white box approach where a deep knowledge about a system's internals is required, and since the system comprising the software stack (operating system, the Java® virtual machine, Hadoop, and the workloads) and the hardware stack are very complex, it becomes very difficult to capture this complexity with cost-based models. Second, Hadoop has several extension points where users can plug in their own policies, such as scheduling and block placement policies, making it necessary for the performance model to support these different policies. Finally, although cost-based models may be effective, due to the high coupling of the model to the system internals a change in the system, such as the hardware technologies or the Hadoop framework itself, triggers a change in the model. We are motivated by these drawbacks to explore machine learning-based performance models, which are black box models, since building cost-based models for a complex system such as Hadoop with good accuracy while being flexible and robust is challenging.

Compared with the white box approach, black box models have two main advantages. First, black box models and the recommendations of an auto-tuner using these models are based on observations of the actual system performance for a particular workload and cluster. Second, they are usually simpler to build than white box models as there is no need for detailed information about a system's internals. One of the interesting research questions that motivated our work is: Can we build an effective auto-tuner even if we treat the underlying system as a black box? As we demonstrate in Section V, the answer is yes; our machine learning-based approach results in comparable and in some scenarios even better performance than a cost-based approach.

Parameter	Description	Default	Rules of Thumb	
			SNB Cluster	ZT Cluster
`io.sort.mb` (map output sort buffer)	The amount of memory used while sorting files during the map phase	100	200	400
`mapred.tasktracker.map.tasks.maximum` (map slots)	The maximum number of map tasks to run simultaneously by a task tracker	2	7	31
`mapred.tasktracker.reduce.tasks.maximum` (reduce slots)	The maximum number of reduce tasks to run simultaneously by a task tracker	2	2	2
`mapred.reduce.tasks` (reduce tasks)	The number of reduce tasks to run for a job	1	15	15
Job input size	The size of the input dataset	N/A	N/A	N/A

TABLE I. FIVE PARAMETERS THAT WE HAVE USED IN OUR PERFORMANCE MODELS. THE FIRST FOUR PARAMETERS ARE HADOOP CONFIGURATION PARAMETERS SO THEY HAVE DEFAULT VALUES AND RULE OF THUMB SETTINGS FOR THE TWO CLUSTERS WE HAVE USED IN OUR EXPERIMENTS: THE SNB AND ZT CLUSTERS (SECTION IV-B), AND THE LAST PARAMETER IS THE SIZE OF THE JOB INPUT DATASET.

Unlike the cost-based approach, our approach relies on training data to *learn* the performance model, which makes our approach more robust and flexible. The training data have to be collected separately for every application and cluster, and as expected, the data collection process may take more time than profiling the workload once, which is a common method used by cost-based approaches [4]. However, in a real deployment we expect the training to be done once, because in practice applications rarely change, instead their input datasets change [5], and our performance models can easily capture the changes to the input dataset as the input size is one of our model parameters. Moreover, training data collection time can be further reduced by using the logs of the completed jobs as Hadoop daemons already generate a large amount of logging information. Our contributions are threefold:

- We develop several machine learning-based performance models of two Hadoop benchmarks (wordcount and sort) using data collected from two different cluster configurations, and we assess these models in terms of accuracy, computational performance, and their sensitivity to training data size (Section IV).
- We perform an extensive evaluation of the support vector regression model (SVR), which has both good accuracy and computational performance, by comparing it against the cost-based model of the Starfish auto-tuner, the rules of thumb settings, which are industry recommended parameter settings, and the default Hadoop parameter settings. Our results reveal that the SVR model is able to achieve comparable and in some cases even better performance improvements than the cost-based model (Section V).
- We propose a machine learning-based auto-tuning approach that uses smart search algorithms for both training the performance models and exploring the parameter space (Section VI).

II. MAPREDUCE AND HADOOP

MapReduce is a programming model used for processing large amounts of data on commodity clusters. The user specifies a *map* function that processes a key-value pair to produce a list of intermediate key-value pairs, and a *reduce* function to aggregate the output of the map function. Hadoop is a framework that implements the MapReduce programming model, and simplifies cluster programming by taking care of automatic parallelization, load balancing, and fault-tolerance. A typical Hadoop cluster runs over the Hadoop Distributed File System (HDFS) and has a single job tracker (master) that is responsible for managing the task trackers (slaves) that run on each of the nodes in the cluster.

When a user submits a *job* consisting of map and reduce functions, Hadoop first splits the input data that resides in the HDFS into *splits*. Then, Hadoop divides the job into several *tasks* depending on the size of the input data. For every split, Hadoop runs a separate map task, which produces a list of key-value pairs. Hadoop then partitions the map output based on the keys, and runs a reduce task for each key writing the final output to the HDFS.

Hadoop gives users enough flexibility to change its behavior by exposing a large number of configuration parameters (tuning knobs). While some of these parameters, such as the number of map and reduce slots, have impact at the machine level some parameters, such as the number of reduce tasks, have impact at the cluster level. Similarly, parameters may impact different resources in the system; while the size of the map output sort buffer impacts mainly the memory and disk I/O performance, the number of reduce tasks parameter impacts mainly the network and disk I/O performance.

In this work to model the performance of MapReduce applications we have used the five parameters shown in Table I. The first four parameters are Hadoop parameters and the last one is the size of the job input dataset. For the Hadoop parameters, the default column shows the default settings of Hadoop, and the rules of thumb column shows the values that the industry recommends [6], [7]; note the different rule of thumb settings for our SNB and ZT clusters (Section IV-B). Our motivation for using these parameters is threefold. First, these parameters have impact on different resources and the impact of this parameter set covers all the available resources in a cluster. Second, these parameters have impact at different levels (machine-level or cluster-level). Finally, based on our domain expertise we expect these parameters to have significant impact on the performance. However, selecting the parameters to use in a machine learning-based performance model is a non-trivial research problem in its own, which is known as the feature selection problem. Therefore, we have used our domain expertise to select these parameters, and we have left the feature selection problem as an important future work.

III. THE NEED FOR AUTO-TUNING

In this section we make the case for why auto-tuning is the right approach to tuning Hadoop parameters. First, through experiments in our 8-node SandyBridge (SNB) cluster (see Section IV-B) we show that there is significant performance improvement opportunity even by tuning a small number of parameters. Second, we explain why manually tuning a complex system, such as Hadoop, is not feasible.

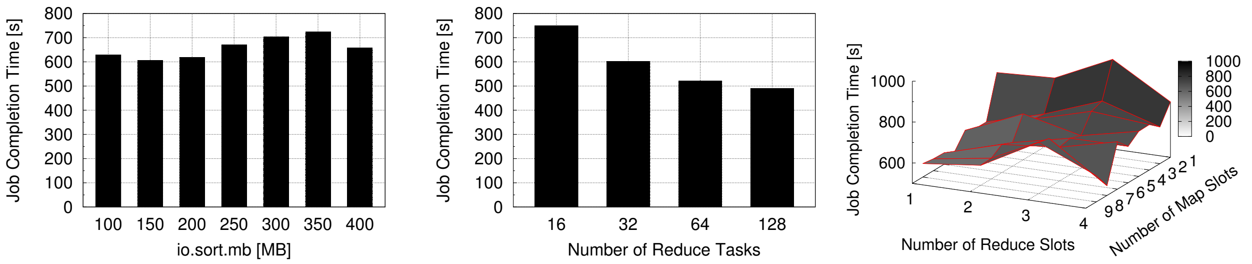


Fig. 1. Performance of the sort benchmark with 80GB input on our 8-node SNB cluster (see Section IV-B) with different parameter settings for the size of the map output sort buffer (left), the number of reduce tasks (middle), and the number of map and reduce slots (right).

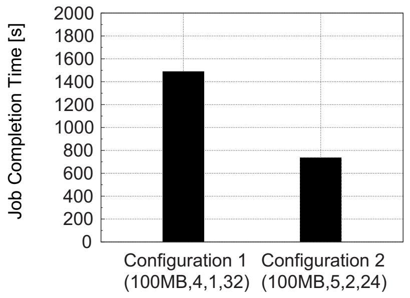


Fig. 2. Performance obtained by tuning several Hadoop parameters at once for the sort benchmark with 120GB input on our 8-node SNB cluster (see Section IV-B) for two configurations (map output sort buffer, map slots, reduce slots, reduce tasks).

Figure 1 shows the performance of different parameter configurations comprising different values for the map output sort buffer parameter (left), the reduce tasks parameter (middle), and the map and reduce slots parameters (right) for the sort benchmark with 80GB input on our 8-node SNB cluster (see Section IV-B). The performance difference between the best and the worst configuration is 16% for the map output sort buffer parameter, 35% for the reduce tasks parameter, and 42% for the map and reduce slots parameters. Our results show that even by only tuning a single parameter in isolation it is possible to obtain up to around 40% improvement in performance.

Figure 2 shows the result when we tune several Hadoop parameters at once for the sort benchmark with 120GB input on our 8-node SNB cluster. For Configuration 1, we set the map output sort buffer size to 100MB, the number of map slots to 4, the number of reduce slots to 1, and the number of reduce tasks to 32. For Configuration 2, we set the map output sort buffer size to 100MB, the number of map slots to 5, the number of reduce slots to 2, and the number of reduce tasks to 24. Our results show that by tuning only a small number of parameters together we can achieve roughly a factor of two performance improvement.

Although there is a vast performance improvement opportunity with careful parameter tuning as we have shown above, manual tuning is not feasible for two reasons. First, it is very difficult to understand the complex interactions among the parameters and reason about their performance impact as the impact of a parameter depends on several factors such as the workload characteristics, the input dataset, the cluster configuration, and even on the values of other parameters [4]. Second, the parameter search space is huge; even if we assume that the parameters take values from different sets of the same size, say 20, for 10 parameters the size of the search space will be 20^{10}, and it is simply impossible to manually explore this search space. Even if we explore a small subset of the search space, which may yield sub-optimal performance in the end, manual tuning will still be inefficient, since the user has to run a large number of trials to carefully search for the performance sweet spots. For example, when tuning the number of reduce slots the user has to address the trade-off between resource utilization and resource contention; while large values can cause resource contention low values may leave resources underutilized.

IV. Modeling the Performance of MapReduce Applications

In this section we study several machine learning algorithms for modeling the performance of MapReduce applications as performance modeling is the key component in an auto-tuning system. We evaluate our models in terms of their accuracy, computational performance, and their sensitivity to various variables such as the training data size, the application characteristics, and the cluster configuration.

A. Machine Learning-based Performance Models

We have explored various machine learning models to use in a practical auto-tuner, such as the one we propose in Section VI. Our models have five inputs (Table I) and one output, which is the job completion time. Although multiple linear regression is also considered as a machine learning model [8], when presenting the results we refer to it as a "traditional statistical model" to discriminate it from the other more complex machine learning models we have explored. We briefly describe our models in turn.

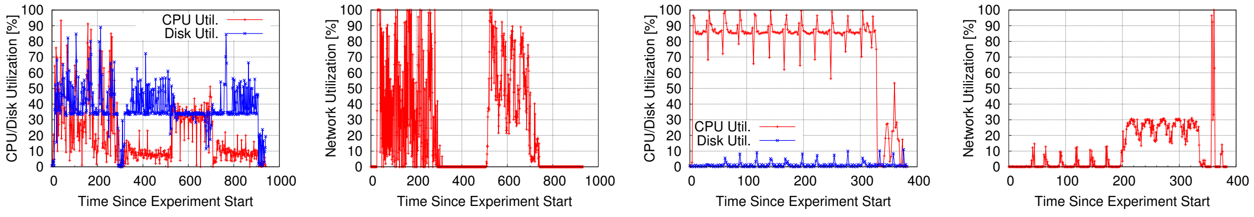
Simple Multiple Linear Regression (MLR): We have used the simplest form of multiple linear regression for modeling the job completion time. This model does not have any parameter interactions and higher order terms.

Multiple Linear Regression with Parameter Interactions (MLR-I): With this model the completion time is modeled as

$$y_i = \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_k x_{ik} + \sum_{p,r=1 \wedge p \neq r}^{k} \beta_{pr} x_{ip} x_{ir} + e_i$$

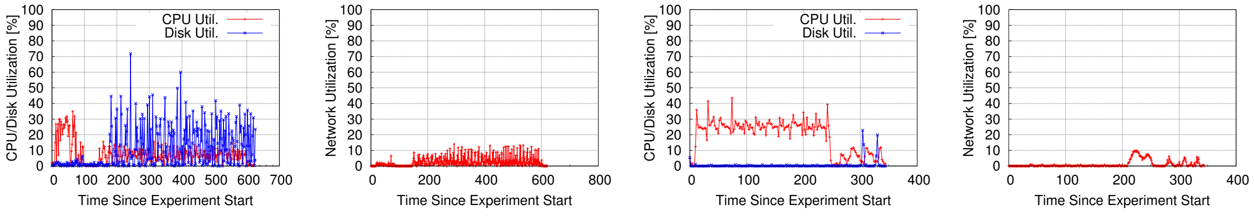
where $\beta_{pr} x_{ip} x_{ir}$ terms represent the parameter interactions. As we don't know which parameters have interactions among them, we have performed an exhaustive search over all possible pairwise interactions to identify the interactions that yield the best performing model.

Multiple Linear Regression with Quadratic Effects (MLR-Q): With this model the completion time is modeled as



(a) Sort CPU/Disk Util. (b) Sort Network Util. (c) Wordcount CPU/Disk Util. (d) Wordcount Network Util.

Fig. 3. CPU, disk, and network utilization when running the sort (a and b) and the wordcount (c and d) benchmarks on the SNB cluster.



(a) Sort CPU/Disk Util. (b) Sort Network Util. (c) Wordcount CPU/Disk Util. (d) Wordcount Network Util.

Fig. 4. CPU, disk, and network utilization when running the sort (a and b) and the wordcount (c and d) benchmarks on the ZT cluster.

$y_i = \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_k x_{ik} + \beta_{k+1} x_{i1}^2 + ... + \beta_{2k} x_{ik}^2 + e_i$ where the x_{ik}^2 terms represent the quadratic effects of each input parameter.

Multiple Linear Regression with Parameter Interactions and Quadratic Effects (MLR-IQ): Finally, we have also explored a regression model that captures both parameter interactions and quadratic effects.

Artificial Neural Networks (ANN): ANNs are one of the powerful learning models for general nonlinear regression between multiple input and output variables. ANNs have several desirable properties for performance modeling that motivated us to assess ANNs for auto-tuning purposes. First, ANNs are learning models that adjust their internal state to the data without any assumptions about the data. Second, ANNs can represent any function to arbitrary precision making them universal function approximators [9]. Third, ANNs can capture complex relationships between the input and output variables, such as nonlinearity and parameter interactions, making them suitable for practical performance modeling. Finally, ANNs have been shown to be useful for performance modeling in several previous studies (Section VII).

Model Trees (M5Tree): Model trees are an important class of machine learning algorithms for constructing tree models from data. In particular, we have used the M5' model tree algorithm provided by the Weka toolkit [8].

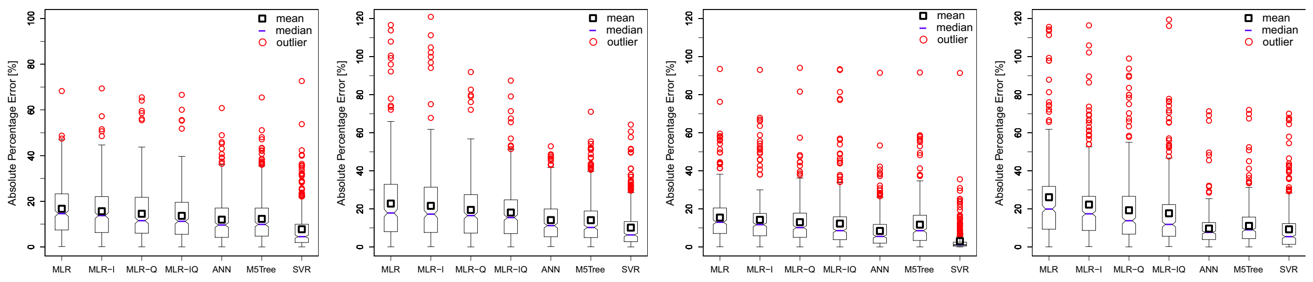
Support Vector Regression (SVR): Support Vector Machines (SVM) are a set of machine learning algorithms used for classification and regression. In the context of regression SVMs are called Support Vector Regression (SVR). Two unique features of SVR have motivated us to explore it for performance modeling. First, SVR is considered as a powerful learning algorithm that has good generalization capabilities and has less risk of overfitting than the other approaches. Second, unlike ANNs, SVR is faster to train (see Section IV-E), and there is no local minima problem during the training phase as SVR transforms the regression problem into a convex optimization problem where all local minima are guaranteed to be global minimums.

B. Experimental Setup

Workloads: We have used the wordcount and the sort benchmarks from the HiBench benchmark suite [10]. Our motivation for using these benchmarks is threefold. First, these workloads are simple, and hence, easy to reason about. Second, these benchmarks are representative of real MapReduce applications as the computation performed by these benchmarks are common use cases of MapReduce, namely extracting a small amount of data from a large dataset and transforming data from one representation to another, respectively. Second, these two benchmarks have different characteristics in terms of resource requirements; wordcount is a CPU bound benchmark while the sort benchmark is mostly I/O bound as shown in Figure 3.

Clusters: We have assessed our models using data collected from two different clusters: SandyBridge (SNB) cluster and ZT cluster. The SNB cluster comprises eight machines connected through a 1 Gbps Ethernet switch, and each machine has a four-core Intel® Core™ i7-2600 processor running at 3.4 GHz, has 16GB main memory and three rotational drives that serve the HDFS data. The ZT cluster has more resources than the SNB cluster; it comprises eight machines connected through a 10 Gbps Ethernet switch, and each machine has a sixteen-core Intel® Xeon® E5-2670 processor running at 2.6 GHz, has 64GB main memory and three rotational drives that serve the HDFS data. In our experiments we have used Hadoop 0.20.2 and the master node running the job tracker and the name node were also responsible for running application tasks.

Finally, to give an idea about the workload characteristics and our cluster configurations, we present the resource utilizations when running the sort and wordcount benchmarks with 80GB input in Figures 3 and 4. While the sort benchmark is mostly network I/O bound on the SNB cluster, it achieves a speedup of around 35% on the ZT cluster where the network is no longer a bottleneck. Similarly, on the SNB cluster the wordcount benchmark is CPU bound, but on the ZT cluster



(a) Sort benchmark on SNB cluster (b) Sort benchmark on ZT cluster (c) Wordcount benchmark on SNB cluster (d) Wordcount benchmark on ZT cluster

Fig. 5. The distributions of the absolute percentage error for the sort and wordcount benchmarks on the SNB (a,c) and ZT clusters (b,d).

the CPU is not a bottleneck anymore. We also observe that the resources on the ZT cluster are utilized less than the SNB cluster as the ZT cluster has more resources.

C. Methodology

Data Collection: To train our models we have collected data from real sort and wordcount executions on the SNB and ZT clusters. During training an important question to address is which parameter values should we explore while collecting the training data. Since our parameter space is huge we have used a sparse sampling approach. In particular, we have collected data both with exhaustive search over different small subsets of the parameter space (exploring a specific range of parameter values) and with random explorations, where we have picked parameter values uniformly random from different range of values. The former approach is guided by our domain knowledge when choosing the specific parameter ranges while the latter approach provides us unbiased observations from the real performance surface of the application. We expect that these two approaches together provide more insights into the real performance surface than the individual approaches. During data collection the clusters were dedicated to our workloads. For each benchmark and each cluster we have explored the same parameter values to collect the training data, and to capture the performance variability we have executed the benchmarks three times for every parameter configuration resulting in around 400 training points for each benchmark.

Training and Testing: After we have collected the raw data we have split the dataset randomly into an 80% training set and a 20% test set. We have trained the models with the training set, and we have evaluated their accuracy with the test set. We have performed this train-test cycle five times with different random splits, and then we report the average performance metrics. To build our models we have used both the Weka toolkit [8] and the R statistical software environment [11].

Model Tuning: Our models have various parameters, such as the number of hidden layers in the ANN and the maximum tree depth of the M5Tree. As these parameters depend on the nature of the data there is no simple way to determine the best parameter values. Therefore, we have used five-fold cross-validation and exhaustive search to find the best parameter values for our models. In particular, we have used a subset of one of the training sets as the validation set, and we have performed exhaustive search using cross-validation to identify the model parameters that yield the minimum error.

Metrics: We have assessed both the computational performance and the accuracy of our models as both are important when using the models for auto-tuning. To characterize the computational performance we have evaluated the time it takes to build the model and the time it takes to make a single prediction using the model. To characterize the accuracy we have used the absolute percentage error, the R^2 statistic, and the root mean squared error (RMSE) statistic.

The absolute percentage error for the ith prediction is defined as $|\frac{P_i - O_i}{O_i}| * 100$ where P_i is the ith predicted value and O_i is the ith observation in the test set. We also present the mean absolute percentage error (MAPE), which is the mean of the absolute percentage error values computed over all the data points in the test set. The smaller the absolute percentage error the better; in particular a zero absolute percentage error denotes a perfect prediction. The R^2 is defined as $R^2 = 1 - \frac{SS_{err}}{SS_{tot}}$ where SS_{err} is the residual sum of squares and SS_{tot} is the total sum of squares. SS_{err} is defined as $SS_{err} = \sum_i (P_i - O_i)^2$, and SS_{tot} is defined as $SS_{tot} = \sum_i (O_i - \bar{O})^2$, where $\bar{O}$ is the mean of the observed values in the test set. R^2 simply shows the predictive power of a model. Models with R^2 values close to 1 are considered as better models in terms of predictive power. Finally, the RMSE statistic is defined as the square root of the mean squared error, which is defined as $\frac{\sum_i |P_i - O_i|^2}{N}$, where N is the number of data points in the test set. A smaller RMSE value denotes a more accurate model.

D. Model Accuracy

In this section we explore the accuracy of our models using the data collected from two benchmarks on two different clusters.

1) Sort Benchmark: Table II shows the basic statistics for the absolute percentage error for all the models for the sort benchmark, and Figure 5 (a) and (b) present the corresponding box plots that show the distributions of the absolute percentage error. In the box plots presented in this section, the outliers are defined as points with values either less than Q1-1.5IQR or greater than Q3+1.5IQR, where Q1 and Q3 are the first and third quartiles, and IQR is the interquartile range (Q3-Q1).

As the multiple linear regression (MLR) model gets more complex, from MLR to MLR-IQ, the model gets more accurate; the mean model error decreases from 17% to 14% for the SNB cluster, and from 23% to 18% for the ZT cluster. Similarly, the median and the max errors also decrease as the

Model	SNB Cluster							ZT Cluster						
	min	Q1	median	mean	Q3	max	IQR	min	Q1	median	mean	Q3	max	IQR
MLR	0	7	15	17	23	68	16	0	8	18	23	33	117	25
MLR-I	0	6	14	16	22	69	16	0	8	17	22	31	129	23
MLR-Q	0	6	11	15	22	66	16	0	7	16	19	27	92	20
MLR-IQ	0	6	11	14	20	67	14	0	7	15	18	25	87	18
ANN	0	4	10	12	17	61	13	0	4	10	12	17	61	13
M5Tree	0	5	10	12	17	65	12	0	5	10	14	19	71	14
SVR	0	2	4	8	10	73	8	0	3	6	10	13	64	10

TABLE II. BASIC STATISTICS FOR THE ABSOLUTE PERCENTAGE ERROR [%] FOR THE SORT BENCHMARK ON THE SNB AND THE ZT CLUSTERS. THE BEST PERFORMING MODEL IS DEPICTED WITH LIGHT GRAY. Q1, Q3, AND IQR DENOTE THE FIRST QUARTILE, THE THIRD QUARTILE, AND THE INTERQUARTILE RANGE, RESPECTIVELY.

Model	SNB Cluster				ZT Cluster			
	R^2		RMSE		R^2		RMSE	
	mean	median	mean	median	mean	median	mean	median
MLR	0.79	0.8	179.23	160.69	0.76	0.73	199.75	213.65
MLR-I	0.81	0.81	170.8	155.9	0.79	0.8	184.38	190.91
MLR-Q	0.82	0.84	165.38	154.64	0.82	0.82	173.66	176.45
MLR-IQ	0.84	0.85	157.08	145.67	0.85	0.87	156.03	160.66
ANN	0.87	0.89	135.71	134.71	0.93	0.94	108.78	106.84
M5Tree	0.83	0.81	159.81	164.2	0.92	0.93	114.62	104.74
SVR	0.89	0.92	125.47	122.2	0.95	0.96	87.51	88.87

TABLE III. THE R^2 AND RMSE STATISTICS FOR THE SORT BENCHMARK ON THE SNB AND THE ZT CLUSTERS. THE BEST PERFORMING MODEL IS DEPICTED WITH LIGHT GRAY.

complexity of the MLR model increases. This result confirms the presence of both nonlinearity and second order effects in the actual performance surface, and further confirms that these effects are captured by the complex MLR models. In general, all machine learning models (ANN, M5Tree and SVR) have better accuracy and lower variability (IQR) than the traditional statistical models (MLR models). ANN and M5Tree models have similar accuracy for both SNB and ZT clusters with a MAPE of around 12% while SVR has a MAPE of 8% for the SNB cluster and 10% for the ZT cluster. Moreover, SVR has a significantly better median error than the other models with a median error of 4% and 6% for the SNB and the ZT cluster, respectively.

For the SNB cluster the minimum and the maximum errors of different models are similar while this is not the case for the ZT cluster; the same modeling techniques can behave differently when trained with data collected from different clusters. Therefore, it is important to assess machine learning models with data from significantly different clusters. In particular, the maximum error of the traditional statistical models increase significantly with the data collected from the ZT cluster whereas the maximum errors obtained with the machine learning models increase only slightly, which shows their robustness. Similarly, for the ZT cluster the MLR models have higher variability (with an IQR of up to 25%) than the machine learning models (with an IQR of 10% to 13%), but in contrast this difference is less pronounced for the SNB cluster.

An important error statistic is the third quartile (Q3) of the error distribution, which constitutes the bulk (75%) of the test data. Our results reveal that at the third quartile machine learning models have smaller errors than the traditional statistical models on both clusters (Table II). In particular, for the SNB cluster 75% of the test data have an error of 20%-23% for the MLR models and an error of 10%-17% for the machine learning models. Similarly, for the ZT cluster 75% of the test data have an error of 25%-33% for the MLR models while for the machine learning models the error is 13%-17%.

Finally, Table III presents the R^2 and RMSE statistics for the sort benchmark on the two clusters. Similarly to the error statistics presented in Table II, machine learning models have higher R^2 values and lower RMSE values confirming their better predictive capabilities than the MLR models. **Considering the results in Table II and Table III we conclude that, overall, machine learning models perform better than the traditional statistical models, and in particular, SVR has the best performance among the models we have explored for the sort benchmark.**

2) Wordcount Benchmark: Table IV presents the basic statistics for the absolute percentage error for all the models for the wordcount benchmark, and Figure 5 (c) and (d) present the corresponding absolute percentage error distributions. Similarly to the results for the sort benchmark, as the complexity of the MLR model increases the mean/median error decreases from 15%/13% to 12%/9% for the SNB cluster, and from 26%/20% to 12%/9% for the ZT cluster. The decreasing error trends confirms that complex MLR models can capture the nonlinearity and second order effects in the performance surface also for the wordcount benchmark. In general, SVR and ANN have better performance than the other models with a median error of 1% and 6% for the SNB cluster, and 5% and 8% for the ZT cluster, respectively. A particularly interesting result is that the M5Tree model performs worse than the other machine learning models with a similar performance as the MLR-IQ model. This result suggests that the workload characteristics definitely have an impact on the performance of different modeling techniques, since for the sort benchmark, M5Tree model has a similar performance as the other machine learning models. Therefore, it is important to assess the models with data collected from workloads with different characteristics.

When we look at the error IQRs, traditional statistical models tend to have a higher variability (IQR) than the machine

Model	SNB Cluster							ZT Cluster						
	min	Q1	median	mean	Q3	max	IQR	min	Q1	median	mean	Q3	max	IQR
MLR	0	7	13	15	21	94	14	0	9	20	26	32	181	23
MLR-I	0	6	12	14	18	187	12	0	9	17	22	27	208	18
MLR-Q	0	5	10	13	18	94	13	0	7	14	19	27	169	20
MLR-IQ	0	4	9	12	16	133	12	0	4	9	12	16	133	12
ANN	0	2	6	8	12	92	10	0	4	8	10	13	71	9
M5Tree	0	3	9	12	17	124	14	0	4	9	11	16	72	15
SVR	0	0	1	4	2	91	2	0	1	5	9	12	70	11

TABLE IV. BASIC STATISTICS FOR THE ABSOLUTE PERCENTAGE ERROR [%] FOR THE WORDCOUNT BENCHMARK ON THE SNB AND THE ZT CLUSTERS. THE BEST PERFORMING MODEL IS DEPICTED WITH LIGHT GRAY. Q1, Q3, AND IQR DENOTE THE FIRST QUARTILE, THE THIRD QUARTILE, AND THE INTERQUARTILE RANGE, RESPECTIVELY.

Model	SNB Cluster				ZT Cluster			
	R^2		RMSE		R^2		RMSE	
	mean	median	mean	median	mean	median	mean	median
MLR	0.45	0.51	241.98	193.85	0.45	0.47	234.24	246.8
MLR-I	0.45	0.48	242.13	194.29	0.53	0.54	215.02	230.04
MLR-Q	0.6	0.72	206.78	142.16	0.67	0.75	174.57	193.76
MLR-IQ	0.59	0.74	209.38	134.38	0.69	0.76	169.93	193.85
ANN	0.73	0.88	162.83	80.36	0.87	0.95	96.77	76.71
M5Tree	0.48	0.5	233.9	192.1	0.83	0.91	115.23	110.67
SVR	0.76	0.95	147.8	56.35	0.81	0.83	131.66	137.92

TABLE V. THE R^2 AND RMSE STATISTICS FOR THE WORDCOUNT BENCHMARK ON THE SNB AND THE ZT CLUSTERS. THE BEST PERFORMING MODELS ARE DEPICTED WITH LIGHT GRAY.

learning models. For the SNB cluster the best performing machine learning model, SVR, has an IQR of 2%, but in contrast the best performing MLR model, MLR-IQ, has an IQR of 12%. Likewise, for the ZT cluster, the best performing machine learning model, ANN, has an IQR of 9% while the best performing MLR model, MLR-IQ, has an IQR of 12%. Since 50% of the predictions have an error in the range Q1 to Q3 (the IQR), a smaller IQR for the machine learning models confirms their better robustness than the traditional models.

Similarly to the results of the sort benchmark, machine learning models also tend to have smaller errors at the third quartiles for the wordcount benchmark (Table IV). For the traditional statistical models, 75% of the predictions have an error in the range 16%-21% for the SNB cluster and 16%-32% for the ZT cluster. On the other hand, for the machine learning models 75% of the predictions have an error in the range 2%-12% for the SNB cluster and 12%-16% for the ZT cluster. We conclude that machine learning models have better accuracy than the traditional statistical models for the 75% of the predictions, which constitute the bulk of the test data.

Finally, Table V shows the R^2 and RMSE statistics for wordcount. Similarly to the error statistics presented in Table IV and similarly to the results for sort (Table III), machine learning models have higher R^2 values and lower RMSE values confirming their better predictive capabilities than the traditional models. In particular, SVR performs the best for the SNB cluster while ANN is the best modeling approach for the ZT cluster. This result is particularly interesting as it demonstrates the impact of the workload and cluster characteristics on the predictive power of the models. **When we consider the results for sort and wordcount together, overall, machine learning models have better predictive capabilities than the traditional models. In particular, SVR is the best performing model except for the dataset collected from the ZT cluster using the wordcount benchmark, for which ANN performs slightly better.**

E. Computational Performance of the Models

In an auto-tuner, the computational performance of the models also matters. To this end, in this section we assess the time to train our models and the time it takes to make a single prediction with each model. We perform the measurements on a machine that has a dual-core Intel® Core™ i5-2540M processor running at 2.60 GHz and 4GB main memory, and we report the average of ten measurements. We present the computational performance of the models only for the SNB cluster using the sort benchmark as the results are similar for the other datasets.

Figure 6 (top) shows the time to train the models, which is measured using the whole dataset; the final models that will be used in the auto-tuner will also be trained using the whole dataset. As the multiple linear regression models get more complex, from MLR to MLR-IQ, the training time increases by 2.08x, from 2.5ms to 5.2ms.

Machine learning models have significantly longer training times than multiple linear regression models with ANN having the longest training time (~29s), SVR having a training time of ~1s, and M5Tree having the shortest training time (~100ms). Overall, we find that all traditional statistical models are relatively lightweight to train. Among the machine learning models, ANN takes the longest time to train as we train the ANN for 100,000 epochs, and for each epoch the network weights are updated using gradient descent, which is a computationally intensive operation. Finally, as shown with the vertical lines in Figure 6 all models have little variability in their training times.

Figure 6 (bottom) shows the time to make a single prediction using the models. Similarly to the results for the training time, as the multiple linear regression models get more complex, the time to make a single prediction increases by 2x, from 2.5ms to 5ms. On the other hand, although machine learning models take significantly longer time to train they

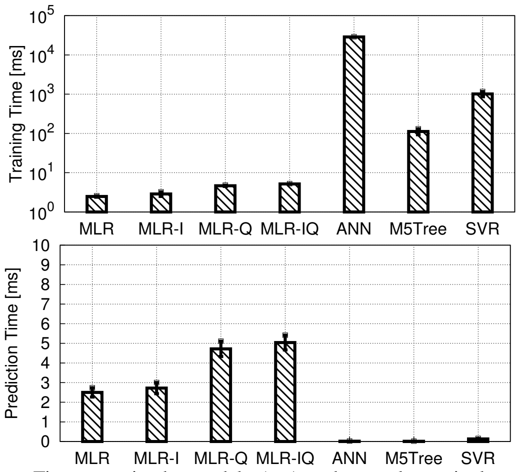


Fig. 6. Time to train the models (top) and to make a single prediction (bottom) for all models. Please note that the vertical axis has a logarithmic scale for the top graph.

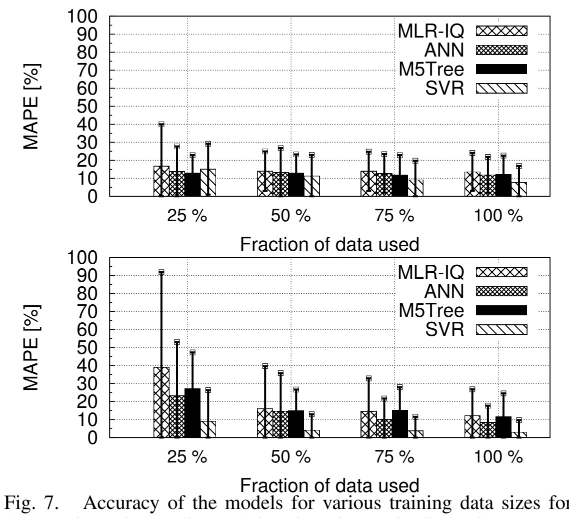


Fig. 7. Accuracy of the models for various training data sizes for the sort (top) and wordcount (bottom) benchmarks on the SNB cluster.

are relatively lightweight when making predictions with a prediction time of less than a millisecond for all the machine learning models.

F. *Model Sensitivity to Training Data Size*

In this section we explore the sensitivity of the models to the size of the training data. To this end, we have trained the models using only a fraction of the data, that is 25%, 50%, 75%, and 100% of the data. For each fraction we have applied the same modeling method described in Section IV-C. Figure 7 presents the results for the sort and wordcount benchmarks on the SNB cluster. The results are similar for the datasets collected from the ZT cluster. Among the multiple linear regression models, we only present the sensitivity of the MLR-IQ model as it performs the best.

For sort, as the training set size increases MAPE decreases for all the models, by 3% for MLR-IQ, 2% for ANN, 1% for M5Tree, and 7% for SVR. Similarly, for wordcount as the size of the training set increases MAPE decreases by 26% for MLR-IQ, 14% for ANN, 16% for M5Tree, and 6% for SVR. Based on these results we conclude that our models can benefit from additional data; if we further increase our training set size we expect the accuracy of our models to increase.

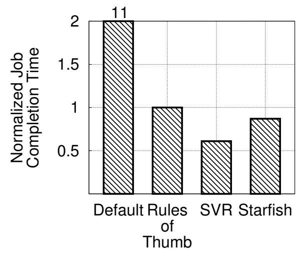
V. Machine Learning-Based vs. Cost-Based Models

In this section, we compare our SVR model against the cost-based model of the Starfish auto-tuner [4] to assess how much performance improvement we can get with each approach. We have performed the experiments in our SNB cluster using two different datasets (80GB and 240GB). Therefore, for each dataset we have a different SVR performance model as the job input size is one of the model parameters. To find the completion time with our approach, first we have identified the best configuration by performing exhaustive search over the performance surface generated by our model, then we have measured the completion time with this configuration. We have compared the completion time of our approach against the completion time achieved with the configuration recommended by Starfish.

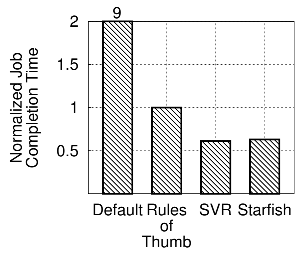
An important difference between Starfish's cost-model and our model is that the cost-based model uses fourteen Hadoop parameters while our model uses five parameters, which gives the cost-based model more flexibility when exploring the available headroom for performance improvement as it has more knobs to tune. Consequently, the absolute completion time numbers are not directly comparable, and for a fair comparison we have normalized the completion time with respect to the completion time obtained with the rules of thumb settings. In addition, we have also compared our approach against the default parameter settings; we refer to Table I for our model parameters, and their default and rules of thumb values.

Figure 8 shows the results of our comparison. For the sort benchmark, our SVR model has improved the performance more than Starfish for the 80GB input. While Starfish has improved the performance by 13% over rules of thumb, our SVR model has provided a 39% improvement. For the 240GB input, both models perform similarly and improve the performance around 40%. Similarly, for the wordcount benchmark, both models have relatively similar performance improvements with Starfish being slightly better (˜5%). Since our model has a smaller number of inputs than the cost-based model, we expect that increasing the number of model inputs will provide more flexibility for parameter tuning and help further improve the performance. As expected, for both benchmarks the default settings perform significantly worse than both approaches as default settings do not involve any tuning at all. Our results reveal that compared with the cost-based model our SVR model achieves comparable or in some cases even better performance improvements.

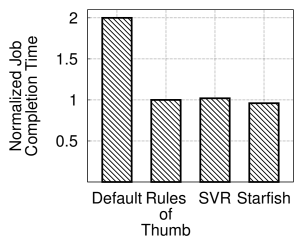
One limitation of our approach may be the time it takes to collect the training data. However, in a real deployment we expect the training to be done once per application on a particular cluster—in practice applications rarely change, but rather their inputs change, and this is easily captured by our models as job input size is one of our model parameters. Moreover, the training time can be further reduced by using the logs of the completed jobs in a Hadoop cluster.



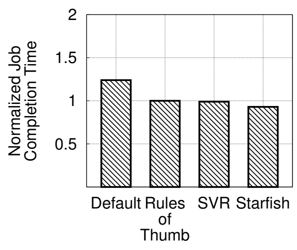
(a) Sort with 80GB input



(b) Sort with 240GB input



(c) Wordcount with 80GB input



(d) Wordcount with 240GB input

Fig. 8. The normalized job completion times with respect to the rules of thumb settings for the sort (a and b) and wordcount benchmarks (c and d). For figures a and b, the numbers on the Default bar show the normalized completion time with the default settings.

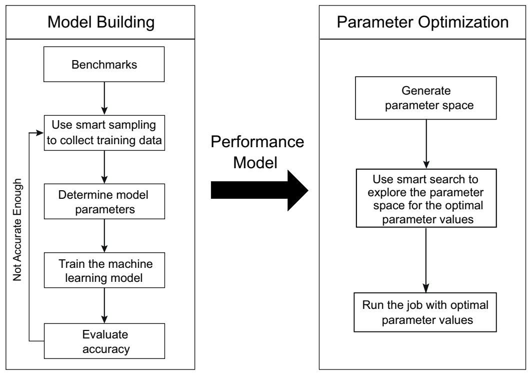

Model Building
Benchmarks
Use smart sampling to collect training data
Determine model parameters
Train the machine learning model
Evaluate accuracy
Not Accurate Enough
Performance Model
Parameter Optimization
Generate parameter space
Use smart search to explore the parameter space for the optimal parameter values
Run the job with optimal parameter values

Fig. 9. A practical end-to-end machine learning-based auto-tuning flow.

VI. Towards A Practical Machine Learning-Based Auto-tuner

A practical auto-tuner has to implement the model building and parameter optimization phases shown in Figure 9. So in addition to the performance model several additional components are required to implement this end-to-end auto-tuning flow. In this section, we propose a practical auto-tuning approach that realizes this flow using machine learning-based performance models and smart search techniques, which are used both in the model building phase for training the models and in the parameter optimization phase for exploring the parameter space.

Our machine learning-based auto-tuning approach learns the underlying performance model from the training data collected from benchmark executions, and thus, is flexible and robust when dealing with different applications and clusters. One of the major challenges with our approach is to train a relatively accurate performance model quickly. Random sampling, which is a simple approach to collect unbiased observations from the performance surface of an application, has been used to collect training datasets in previous studies [12], [13]. However, based on our experience random sampling may require a large number of samples to build an accurate model, and since the overall tuning time is an important consideration in an auto-tuner a long training time can reduce its effectiveness.

To address the above challenge, we propose using smart sampling techniques during the model building phase to reduce the model training time. With smart sampling we employ a direct search algorithm rather than random sampling. The direct search method focuses on regions of interest (e.g., a region with good performance) and searches towards such regions for finding the parameter configuration that yields the best performance. Unlike random sampling, this method can reduce the number of samples required to accurately represent the features of the performance surface in those regions. As our results have already shown that the performance surfaces of MapReduce applications are usually nonlinear and multimodal, we apply a global search algorithm, such as a genetic algorithm or recursive random sampling, to avoid being trapped in a local minimum. The smart search algorithm collects training samples and trains the performance model using these samples as it searches through the actual performance surface of the application (Figure 9). Then, the auto-tuner evaluates the accuracy of the performance model and stops the sampling process if the model is accurate enough (e.g., 90%). Otherwise, sampling continues until either the search algorithm converges, reaches the search budget or the model is accurate enough.

After building a reasonably accurate performance model, the auto-tuner proceeds to the parameter optimization phase. In this phase, the auto-tuner generates a parameter search space and explores it using smart search techniques, and it uses the performance model to find the parameter configuration that has the best predicted performance. When the search algorithm finds the best parameter configuration it terminates, and finally the auto-tuner starts the job with the best configuration.

VII. Related Work

Closest to our work are the previous research efforts on machine learning-based performance modeling and performance auto-tuning, which we describe in turn.

Machine Learning-Based Performance Modeling Machine learning has been successfully used to model the performance of diverse applications. ANNs and multiple linear regression have been shown to be effective for modeling the performance of parallel applications [14]. Similarly, ANNs and M5' model trees have been used to model the performance of virtualized systems [15] and utility computing systems [16], respectively. ANNs, support vector machines, and decision trees have also been used for optimizing the performance of data center applications [17]. Our work contributes to this body of work by applying machine learning to model the performance of an interesting class of applications, namely MapReduce applications, with the goal of performance auto-tuning.

Performance Auto-tuning Previous studies have used different approaches to performance auto-tuning. Cost-based approaches have been successfully used in query optimization in traditional database systems [18]. With this approach, the cost model is the most important component of the optimizer as the accuracy of a cost model directly impacts the resulting performance. Empirical auto-tuning, which is based on an empirical search of the best parameter configuration, has been used in systems such as Atlas [19] and PHiPAC [20]. However, one major drawback of this approach is the long search time for large search spaces, which is usually the case in practice. To auto-tune the performance of MapReduce applications, Babu [21] proposes a competition-based approach where two copies of the same task are started with different parameter configurations and the best configuration is identified empirically. Similarly, Kambatla et al. [22] propose a history-based approach where the system keeps a history of job executions, and similar jobs are executed with the same optimal configuration. Finally, machine learning-based approaches have also been used successfully for auto-tuning the performance of parallel applications [14] and data center applications [17].

Closest to our work is the Starfish auto-tuner [4], which relies on a cost-based performance model. However, building a cost model that is accurate, robust, and flexible at the same time is challenging for a complex system such as Hadoop. Although cost-based models provide deep insights into a system's internals, one of their main limitations is their flexibility; a change in the scheduling policies, the Hadoop framework internals, or the cluster requires the cost models to be rebuilt. However, our approach does not have this limitation, because addressing such a change with our approach only requires the models to be retrained, and model retraining is easier than building a new cost-model. Moreover, Starfish only models the performance impact of job-level parameters (mostly memory-related), and it does not model various important cluster-level parameters, such as the number of map and reduce slots. Finally, our findings show that a machine learning-based approach yields comparable and in some cases better performance improvements than Starfish's cost-based approach.

VIII. Conclusion and Future Work

In this paper we have explored various machine learning-based performance models for auto-tuning Hadoop MapReduce. We have shown that support vector regression model (SVR) has good accuracy and computational performance across diverse workloads and clusters. We have then compared our auto-tuning approach, which uses the SVR performance model, against the Starfish auto-tuner, which uses a cost-based model. Our findings reveal that the SVR model is able to achieve comparable and in some cases even better performance than Starfish by considering less parameters. Unlike the cost-based approach, our approach is more robust and flexible as it is easier to adapt to changes; our models learn the performance surface of the applications rather then being hardwired as is the case for the cost-based models. Our results demonstrate that it is possible to build an effective auto-tuner with a black box approach, that is, by only using observations from the system and without getting exposed to its internals. Finally, we have also proposed a practical end-to-end auto-tuning flow by combining our models with smart search algorithms.

References

[1] "Apache Hadoop Project", http://hadoop.apache.org/.

[2] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[3] "Facebook, hadoop, and hive," 2009, http://www.dbms2.com/2009/05/11/facebook-hadoop-and-hive/.

[4] H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. B. Cetin, and S. Babu, "Starfish: A self-tuning system for big data analytics," in *CIDR*, 2011, pp. 261–272.

[5] P. Bhatotia, A. Wieder, R. Rodrigues, U. A. Acar, and R. Pasquin, "Incoop: Mapreduce for incremental computations," in *Proc. of the 2nd ACM Symposium on Cloud Computing (SOCC)*, 2011, pp. 1–14.

[6] "7 tips for improving mapreduce performance," 2012, http://www.cloudera.com/blog/2009/12/7-tips-for-improving-mapreduce-performance/.

[7] "Optimizing hadoop deployments," 2010, http://software.intel.com/file/31124.

[8] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd ed. Morgan Kaufmann, 2005.

[9] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Netw.*, vol. 4, no. 2, pp. 251–257, Mar. 1991.

[10] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang, "The hibench benchmark suite: Characterization of the mapreduce-based data analysis," in *Intl. Conference on Data Engineering Workshops*, march 2010, pp. 41 –51.

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

[12] G. Fursin, C. Miranda, O. Temam, M. Namolaru, E. Yom-Tov, A. Zaks, B. Mendelson, E. Bonilla, J. Thomson, H. Leather, C. Williams, M. O'Boyle, P. Barnard, E. Ashton, E. Courtois, and F. Bodin, "MILEPOST GCC: machine learning based research compiler," in *Proc. of the GCC Developers' Summit*, 2008. [Online]. Available: http://hal.inria.fr/inria-00294704/fr/

[13] J. Cavazos and M. F. P. O'Boyle, "Method-specific dynamic compilation using logistic regression," *SIGPLAN Not.*, vol. 41, no. 10, pp. 229–240, Oct. 2006.

[14] E. Ipek, B. R. de Supinski, M. Schulz, and S. A. McKee, "An approach to performance prediction for parallel applications," in *Proc. of the 11th Intl. Euro-Par Conference on Parallel Processing*, 2005, pp. 196–205.

[15] S. Kundu, R. Rangaswami, K. Dutta, and M. Zhao, "Application performance modeling in a virtualized environment," in *Proc. of the High Performance Computer Architecture (HPCA)*, 2010, pp. 1–10.

[16] J. Wildstrom, P. Stone, and E. Witchel, "Carve: A cognitive agent for resource value estimation," in *Proc. of the Intl. Conference on Autonomic Computing*, ser. ICAC, 2008, pp. 182–191.

[17] S.-w. Liao, T.-H. Hung, D. Nguyen, C. Chou, C. Tu, and H. Zhou, "Machine learning-based prefetch optimization for data center applications," in *Proc. of the Conference on High Performance Computing Networking, Storage and Analysis (SC)*, 2009, pp. 56:1–56:10.

[18] S. Chaudhuri, "An overview of query optimization in relational systems," in *Proc. of the 17th ACM symposium on Principles of database systems (PODS)*, 1998, pp. 34–43.

[19] R. C. Whaley and J. J. Dongarra, "Automatically tuned linear algebra software," in *Proc. of the ACM/IEEE Conference on Supercomputing (SC)*, 1998, pp. 1–27.

[20] J. Bilmes, K. Asanovic, C. W. Chin, and J. Demmel, "Optimizing matrix multiply using PHiPAC: a portable, high-performance, ANSI C coding methodology," in *Proc. of the 11th Intl. Conference on Supercomputing (ICS)*, 1997, pp. 340–347.

[21] S. Babu, "Towards automatic optimization of mapreduce programs," in *Proc. of the 1st ACM symposium on Cloud computing (Socc)*, 2010, pp. 137–142.

[22] K. Kambatla, A. Pathak, and H. Pucha, "Towards optimizing hadoop provisioning in the cloud," in *Proc. of the Conference on Hot topics in cloud computing (HotCloud)*, 2009.