2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing

On the use of machine learning to predict the time and resources consumed by applications

Andréa Matsunaga and José Fortes
Advanced Computing and Information Systems Laboratory
Department of Electrical and Computer Engineering, University of Florida
PO Box 116200, Gainesville, FL, 32611-6200, USA
e-mail: {ammatsun, fortes}@ufl.edu

Abstract— **Most datacenters, clouds and grids consist of multiple generations of computing systems, each with different performance profiles, posing a challenge to job schedulers in achieving the best usage of the infrastructure. A useful piece of information for scheduling jobs, typically not available, is the extent to which applications will use available resources once they are executed. This paper comparatively assesses the suitability of several machine learning techniques for predicting spatiotemporal utilization of resources by applications. Modern machine learning techniques able to handle large number of attributes are used, taking into account application- and system-specific attributes (e.g., CPU microarchitecture, size and speed of memory and storage, input data characteristics and input parameters). The work also extends an existing classification tree algorithm, called Predicting Query Runtime (PQR), to the regression problem by allowing the leaves of the tree to select the best regression method for each collection of data on leaves. The new method (PQR2) yields the best average percentage error, predicting execution time, memory and disk consumption for two bioinformatics applications, BLAST and RAxML, deployed on scenarios that differ in system and usage. In specific scenarios where usage is a non-linear function of system and application attributes, certain configurations of two other machine learning algorithms, Support Vector Machine and k-nearest neighbors, also yield competitive results. In addition, experiments show that the inclusion of system performance and application-specific attributes also improves the performance of machine learning algorithms investigated.**

Keywords-application resource usage; machine learning; regression; classifier tree.

I. INTRODUCTION

Resource consumption by an application in the form of CPU time, amount of memory, network bandwidth, and disk space consumed, is a useful piece of information when available before execution. It can be used by schedulers to accommodate the most number of applications without resource contention, it can help estimate the waiting time on queued systems, it can identify the best resource to run an application and analyze what-if scenarios, or it can provide an estimate of the cost of running an application on a pay-per-use facility (e.g., a cloud). Nonetheless, this information is often not available to users or computational systems. In cases where predicted resource consumption is provided, the prediction rarely takes into consideration both application characteristics and system performance. The use of Machine Learning (ML) algorithms to predict application resource consumption is an appealing approach that has been pursued by several previous studies [1]-[12]. These studies have proposed the use of specific ML algorithms applied to scenarios ranging from generic jobs in batch systems to specific applications that could be part of a workflow distributed across a grid. The abundance of solutions and the diversity of dataset attributes and reported prediction performance metrics combined with limited comparative evaluation make it difficult for users and system developers to choose an appropriate prediction method. To remedy this situation, this paper makes the following contributions:

- It identifies the best ML algorithm and provides reasoning for the results, for predicting execution time, memory and disk requirements for two bioinformatics applications, namely BLAST [13] and RAxML [14], on different types of computer clusters. The regression algorithms considered in this study include k-nearest neighbor (k-nn) [3][6], linear regression [10], decision table [9], Radial Basis Function network (RBFn) [11], Predicting Query Runtime (PQR) [8], and Support Vector Machine (SVM) [19][20]. The datasets comprising more than 2000 cpu-hours of execution are made publicly available for future research.
- It proposes the Predicting Query Runtime Regression (PQR2) algorithm, a generalization of the PQR classification tree approach, to the regression problem. The extension consists of adding regression functions at the leaves of the PQR tree in order to provide fine-grained prediction.
- It investigates the impact of attributes on prediction accuracy and makes the case for increasing the number of attributes (data space), in particular taking into account heterogeneous systems performance and detailed application-specific characteristics.

Experimental results show that PQR2, when compared to other algorithms, offers the best accuracy for the scenarios studied, including cases where the predicted resource usage presents linear and non-linear dependencies on application attributes. The diverse characteristics of the studied datasets suggest that the conclusions of this work may be generally applicable to many other applications and systems.

978-0-7695-4039-9/10 $26.00 © 2010 IEEE
DOI 10.1109/CCGRID.2010.98

This paper is organized as follows. Section II summarizes the algorithms investigated in this study and discusses previous work. Section III explains the extension to PQR for handling regression problems. Section IV describes the collected training and testing datasets used in the experiments. Section V analyzes the experimental results and Section VI draws the conclusions from this work.

II. Machine Learning and Related Work

Predicting application resource consumption, such as the amounts of CPU, memory, disk and network required by an application and the lengths of times during which the resources are occupied, can be viewed as a supervised machine learning problem. The system needs to learn a concept based on a collection of historical data of *n* previous runs of the application. Each previous observation is used as training data, providing a set of *m* attributes and the actual outcome $y = f(a_1, a_2, \dots, a_m)$. Several parametric (e.g., linear regression, polynomial regression) and non-parametric (e.g., k-nn, locally weighted linear regression, decision trees) solutions exist in the field of machine learning, some of which are presented in [15][16][17]. Parametric methods define a hypothesis space and a loss function. The training data is used to extract the model parameters of the problem at hand by minimizing the loss function. After the parameters are extracted there is no need for the system to store the training data and predictions can frequently be computed fast, i.e., in O(1) time. On the other hand, non-parametric methods use the actual data as the model, usually presenting prediction computation time that linearly grows with the size of the training data, i.e., takes O(n) time. The advantage in this case is that adding more training data into the learning process is trivial since the next estimate calculation simply takes the new data into account. ML algorithms considered by previous work and evaluated in this paper, are discussed next.

k-nearest neighbor (k-nn) algorithm: given a query point, the k nearest training data points are considered when computing the prediction. The predicted value can be calculated as the weighted average of selected points. When non-identical weights are used, the weight function usually decreases as the distance from the query point to the training data increases with the goal of minimizing the influence of the distant neighbors on the prediction. While the Euclidean distance is often selected, other distance functions such as Manhattan or Hamming may better represent the gap between data points in certain scenarios. One of the challenges of this algorithm is to find the ideal value for *k*, which is dependent on the distribution of the training data. A high value for k can reduce the influence of noise, but in regions scarce in data, the prediction can be highly influenced by distant points. In [6], the best value for k is searched applying a genetic algorithm. *Locally weighted polynomial regression (LWPR)* is similar to k-nn algorithm, in that it considers data close to the query point for the prediction according to a weight function and its kernel bandwidth. Instead of generating the prediction through averaging, the algorithm attempts to fit the data with a polynomial (e.g., a first order polynomial $y = a_1 x + a_0$). The advantage of this method is that predictions can better track data that is not well represented by a plateau, avoiding the rough edges of the k-nn algorithm. It has been demonstrated that the computation demands of both algorithms grows with the size of the knowledge-base, with an additional computation demand for LWPR, but methods combining caching and filtering [3] or kd-trees and approximate weight functions [2] have shown to reduce the computation time manyfold. Previous work [3] has shown that 1-nn and 3-nn with Gaussian weighted averaging can produce lower error rates when predicting the runtime of a short-running scientific application than a locally weighted linear regression; the opposite result applies for a randomly generated dataset.

Linear Regression (LR) algorithm: simple algorithm that finds a linear hyperplane $y = a_m x_m + \dots + a_2 x_2 + a_1 x_1 + a_0$ with minimum error in relation to existing data points, defined as the sum of Euclidean distance of each data point to the hyperplane. Once the linear model is defined, prediction takes O(1) time. Linear regression was applied in [10] for predicting execution time of an application in a workflow, and it is used by LWPR and SVM algorithms.

Decision table/tree (DT) algorithm: a divide-and-conquer strategy that utilizes a tree structure to separate data according to their characteristics. The nodes of the tree store rules that determine how their children have been split and the leaves store either the actual outcome, a function that determines the outcome, or the actual data from where the prediction can be made. This method provides good computational scalability, can handle well data that are dispersed in input space, and its result is reasonably easy to interpret. However, correlations between input characteristics are not well captured since nodes usually split their children according to a single characteristic. The C4.5 classification tree was applied in [9] for predicting execution times in a distributed system that has a centralized scheduler with backfilling, using ranges of execution time as classes. Although implemented using the so called templates instead of a tree graph, [1][4][5][6][12] all used job characteristics to select a subset of the data points on which an statistical function is applied. The choices for the statistical function included the mean [5][6][12], the mean plus 1.96 standard deviations [1], mean plus 1.5 standard deviations [4], linear regression of a single characteristic x ($y = a_1 x + a_0$) [1][5], inverse regression of a single characteristic x ($y = x/a_1 + a_0$) [5], and logarithmic regression of a single characteristic x ($y = a_1 \log x + a_0$) [5]. Using the well known space-shared system workloads from the Parallel Workload Archive [18] as the dataset, the mean function has been shown to provide smaller error rates when compared to more complex regressions for the set of characteristics chosen for splitting the training data [5][1].

Artificial Neural Network (ANN): an interconnected group of functions that emulates a biological neural system and can collectively perform complex tasks. During the

[1] The Parallel Workload Archive is a collection of traces; they were not used in this study because they lack application and system performance information needed to evaluate the algorithms under consideration.

training phase, an ANN changes its structure based on external or internal information that flows through the network. *Radial Basis Function network* (RBFn) is a feedforward ANN, typically with three layers: input, hidden and output. A variable number of neurons form the key hidden layer, where Euclidean distance from the center of each neuron to the test case is computed and the RBF function applied. RBFn is very similar to k-nn, except for the fact that RBFn is a parametric method. The number of neurons in the hidden layer (number of neighbors in k-nn) affects the performance and the computational demands of RBFn. In the extreme case (overfitting), each data point can be the center of a neuron (single neighbor in k-nn). The use of RBFn was proposed in [11] combined with a Bayesian network for eliminating attributes with low correlation with the outcome.

Support Vector Machine (SVM) algorithm: a kernel method for solving classification [19] and regression [20] problems, especially for scenarios with non-linear learning pattern. The attribute space is transformed by the kernel function into possibly a high-dimension feature space where an optimal linear hyperplane is found by maximizing the margin between the so called support vectors. The advantages of this method include the limited number of parameters to choose, and the ability to handle large numbers of attributes and non-linear scenarios without local minima problems. A comparison of radial basis function neural networks with SVM has been presented in [11], with lower errors for SVM in some scenarios at the cost of higher computational demand.

Time series methods: in scenarios where the data points of the attribute to be predicted exhibit temporal patterns, algorithms that consider only the order of data points have been proposed. The prediction could be as simple as predicting the future value to be the same as the last observed value, to as complex as considering multiple algorithms in parallel, as proposed by the Network Weather Service (NWS) [22] or a probabilistic approach as a Markov-chain. Triple-C [10] presents a scenario where jobs in a workflow present time dependencies, making Markov-chains appropriate for predicting execution time. In this study, the attributes to be predicted are assumed to have low or no ordering dependency and thus this class of solutions is not considered. However, the idea of choosing between several learners as in NWS is at the core of the PQR algorithm discussed in the next section.

III. Predicting Query Runtime Regression

PQR [8] is an algorithm that generates a binary tree that can combine a variety of classifiers. Each node of the tree is represented by a 2-class classifier that is chosen from a pool of classification algorithms according to its accuracy, and the leaves of the tree correspond to ranges of the attribute to be predicted and thus present a coarse granularity of output. PQR has been proposed in the context of predicting execution time of database management systems queries, where cost models are often not good predictors and analytical models are complex and difficult to create [8], yet only coarse classification of queries are necessary.

The algorithm to discretize the numerical attribute into two classes for creating a node consists of ordering all m training attribute values $(a_i, i = 1, \dots, m)$ and finding the k largest gaps δ defined by $\delta_i = \frac{a_{i+1} - a_i}{a_i}, i = 1, \dots, m-1$ that can become potential split locations. The best combination of classifier and split location determines the two attribute ranges.

The growth and quality of the tree can be controlled by tuning several parameters of the PQR algorithm that dictate the minimum number of training data in a node, the minimum accuracy required by a classifier, the minimum attribute interval covered by a node, and the percentage of data points from the extremes of prediction attribute that should not be considered for split. Although many combinations are possible, in practice, the use of default parameters suffices to achieve good accuracy.

In this work we generalize PQR to the regression problem by training regression algorithms from a pool of known methods with the data on PQR leaves and choosing the best algorithm. Any regression algorithm can be placed on the pool, including different configurations of the same algorithm, but preference for parametric methods should be given when fast predictions are required. Figure 1 highlights through an example the differences between a PQR and a PQR2 tree. The nodes of the tree, shown as circles with the name of selected classifier, are produced using the algorithm described in [8]. Each node is responsible for classifying data into two classes, defined by a threshold that is chosen as to maximize the accuracy of the predictor. The values between brackets show the range of attribute values that were observed by each node or leaf during training, followed by the amount of historical information. The only difference is found at the leaves: instead of outputting classes (a broad range) or a static value (e.g., range median), PQR2 selects the best regression model for the available data (LR and SVM in the case of the leaves shown in the figure). The percentage error (displayed in normal face for PQR and in bold for PQR2) in each subspace is lower with PQR2. PQR2 offers fine-grained prediction through models that adapt better to the characteristics of the data in each leaf.

IV. Experimental Setup

To generate data for this study, a heterogeneous environment consisting of four different hardware resources specified in TABLE I were used to run two popular bioinformatics applications, namely Basic Local Alignment Search Tool (BLAST) [1] and Randomized Axelerated Maximum Likelihood (RAxML) [14].

BLAST version 2.2.18 (either 32-bit or 64-bit depending on the resource) was executed against the non-redundant (NR) protein sequence database from NCBI split into 1 fragment (total of 3.5 GB of data). Given an input sequence, BLAST searches a database for similar sequences and calculates the best alignment of the matched sequences. Single nucleotide sequences of varying lengths served as input (see sequence length distribution in Figure 2) in this study.

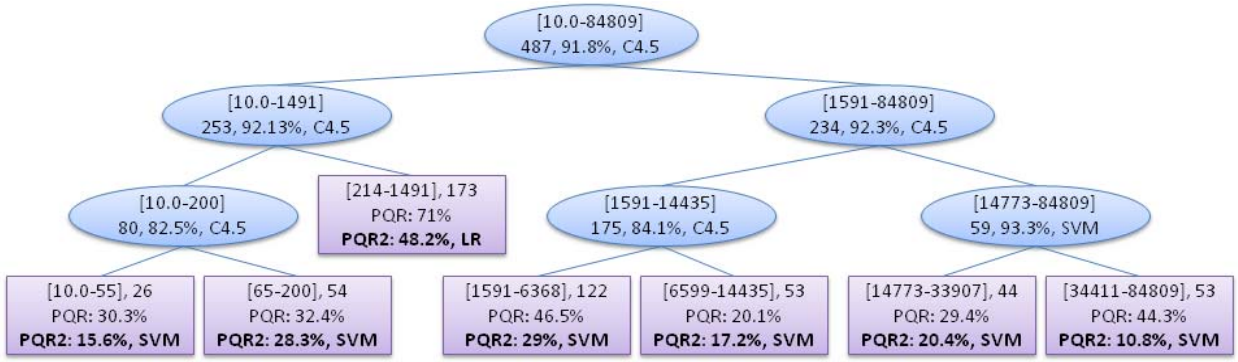

Figure 1. Tree generated by PQR and PQR2 algorithms. The nodes of the tree (circles) are common to both methods, while leaves (squares) of PQR2 (boldface) yield lower errors than PQR (normal face). The improvement comes from the ability of selecting the best regression method from a pool, whereas leaf range median is used in PQR. The number between square brackets represent the range of values of the attribute to be predicted, which is followed by the number of historical data points in each node/leaf. The percentage value indicates the accuracy of each classifier (nodes) or the percentage error of each regressor (leaves). The last value indicates the name of each classification (PQR and PQR2) or regression (PQR2) algorithm selected.

In addition to running BLAST on all different resources on local disk, BLAST was also run placing the database on two Network File System (NFS) servers with different hardware (the * on TABLE I indicates the physical resources used as NFS servers; the disk performance, according to the number of clients, has been used to model the load on the server). The purpose of including runs with the database placed on remote servers is to include scenarios where applications space-share local resources, but time-share other resources such as network or a file system, a typical scenario on clouds that serve virtual machine images or user's data from a shared server. BLAST runs resulted in a collection of 6592 data points. In order to provide insight about the characteristics of this dataset, partial data is plotted on Figure 3 (top) to show the existence of a linear relationship between BLAST execution time and input sequence length that is different for each of the cluster resources used. This linearity is affected when the database is placed on a file server that is concurrently accessed by different number of clients, especially for shorter input sequences (Figure 3 bottom).

RAxML version 7.0.4, a software application for constructing phylogenetic trees using maximum likelihood analysis, was executed for 35 nucleotide datasets combining five different taxa sizes (50, 100, 150, 200 and 250), and seven different sequence lengths (5000, 10000, 15000, 20000, 25000, 40000, and 50000). Runs for different numbers of threads, up to the number of cores in each resource, were also performed, resulting in a dataset with 487 data points.

RAxML is not only computationally intensive, but also memory intensive. Non-linearity of execution time with respect to resources and application characteristics occur in two situations in this dataset: as number of threads increase and as dataset increase. For inputs with250 taxa, Figure 4 shows that single-threaded runs on c3 are faster than on c4 (which has slower memory). This fact slowly changes as the number of threads is increased. Similar trends are observed for other datasets

TABLE I. CHARACTERISTICS OF RESOURCES UTILIZED.

Cluster	c1	c2_512M	c2_3.5G*	c3*	c4
CPU	2-way Pentium III (2 Tualatin cores)	2-way Xeon (2 Prestonia cores)	2-way Xeon (2 Prestonia cores)	2-way Quad Core Xeon (8 Clovertown cores)	16-way Quad Core Xeon (64 Tigerton cores)
CPU clock	1.4 GHz	2.4 GHz	2.4 GHz	2.33 GHz	2.93 GHz
CPU cache◊	512 KiB/core	512 KiB/core	512 KiB/core	4 MiB/2-cores	4 MiB/2-cores
CPU speed	659 MFlops	879 MFlops	879 MFlops	1436 MFlops	1798 MFlops
Kernel◊◊	2.6.23.8-i686	2.6.18-i686	2.6.18-i686	2.6.22.2-x86_64	2.6.18-x86_64
Xen VMM	-	3.1.0	3.1.0	-	-
Memory◊	2 GiB	512 MiB	3.5 GiB	3.9 GiB	503 GiB
Memory speed◊	437 MiB/s	1932 MiB/s	1932 MiB/s	3468 MiB/s	1257 MiB/s
Disk read speed	65.5 MB/s	37.8 MB/s	37.8 MB/s	60.2 MB/s	240.9 MB/s

◊Following the IEC 80000-13:2008 standard, the prefixes KiB (kilobinary bytes), MiB (Megabinary bytes) and GiB (Gigabinary bytes) are used to represent multiples of 2^{10}, 2^{20}, and 2^{30} respectively.
◊◊ i686 suffix indicates a 32-bit kernel while x86_64 suffix indicates a 64-bit kernel

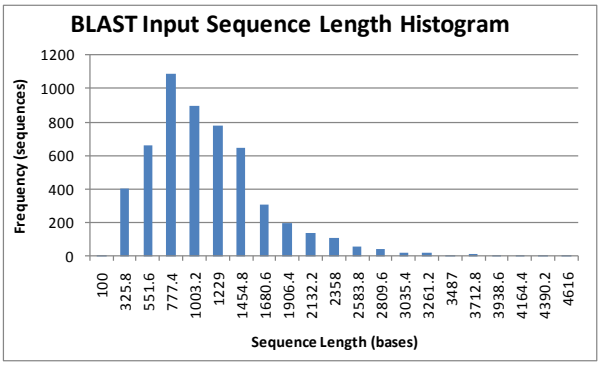

Figure 2. Histogram of input sequence length for BLAST.

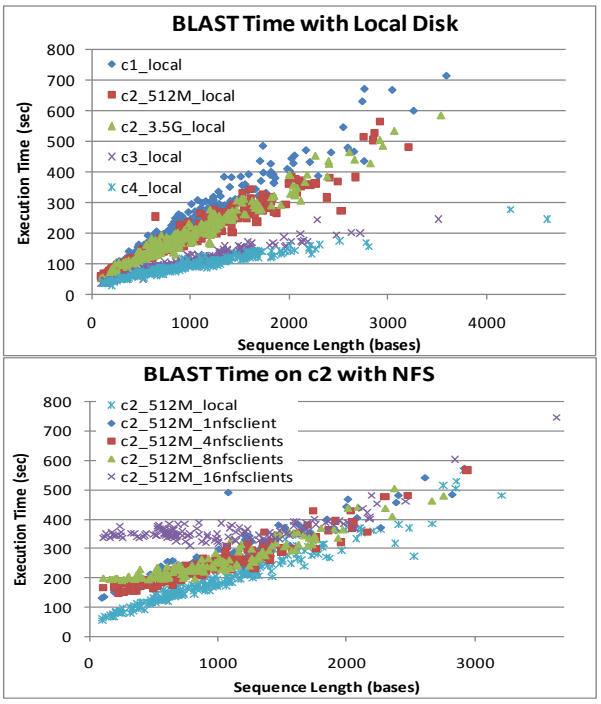

Figure 3. Execution Time of BLAST. A linear relationship between time and input sequence length that is distinct for each different resource can be observed (top). When data is placed on a NFS server, varying the number of concurrent clients on a particular resource (c2) affects the linearity, especially for short sequences (bottom).

and resources (not shown due to space limitations; can be found in [21]). Furthermore, although larger inputs are expected to require more execution time, in some cases, larger inputs have shorter execution time. For the experiments shown in the next section, attributes of applications (sequence length, taxa size, taxa size multiplied by sequence length) and resources (cluster name, CPU clock, amount of cache, amount of memory) were selected. However, as information such as CPU clock does not reflect well the processing capability of each resource, the following benchmarks were used to better characterize the performance of each resource (included in TABLE I):

- A simple matrix multiplication application was used to extract the number of floating point operations performed in a fixed amount of time (CPU speed).
- A cache benchmark application was used to collect the performance of a read-modify-write operation (memory speed).
- The Linux `dd` command was used to read a 1 GiB file from different sources (disk read speed).

The overall average and standard deviation of BLAST and RAxML datasets used in the experiments evaluated in the next section are shown in TABLE II.

TABLE II. OVERALL CHARACTERISTICS OF LEARNING DATASETS.

Dataset	Mean	Standard Deviation
BLAST output	50025 bytes	129743 bytes
BLAST time	208.4 secs	97.7 secs
RAxML RSS	110442 bytes	96077 bytes
RAxML time	5720 secs	10702 secs

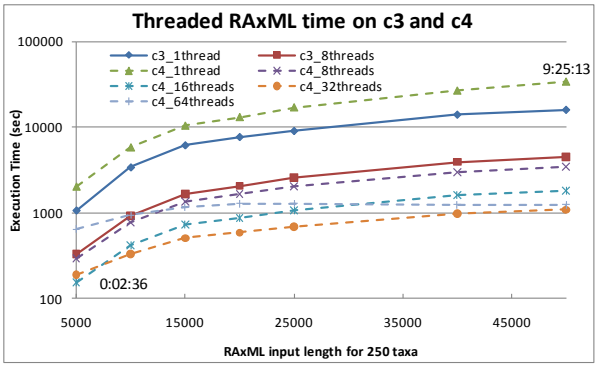

Figure 4. Execution time of RAxML on c3 and c4 resources for input datasets with 250 taxa, lengths varying from 5,000 to 50,000 and different number of threads. The numbers next to extreme points indicate the minimum and maximum execution time (hh:mm:ss).

V. EXPERIMENTAL RESULTS AND EVALUATION

Application resource usage prediction is evaluated in this section for 9 different configurations of machine learning algorithms (see TABLE III), including the new PQR2. All experiments were performed using Weka version 3.0.7, a Java-based open source collection of ML algorithms and statistical tools with graphical user interface for visualizing data, processing information, and carrying data mining experiments. PQR and PQR2 were newly implemented in Weka, extending defined interfaces, making use of data manipulation utilities, and reusing existing ML algorithms such as Naïve Bayes, C4.5, SVM and LR.

Three attributes often considered for prediction are the application execution time, the amount of resident memory (Resident Set Size or RSS) required, and the size of output produced. Evaluating these attributes for BLAST and RAxML datasets described in the previous

section, produces six scenarios. However, due to the constant nature of BLAST memory usage (defined by the NR database size) and RAxML output for the runs performed, evaluations presented in this section did not include these two scenarios. For the remaining four scenarios, 10 iterations of 10-fold cross-validation were performed for each algorithm and scenario to obtain results that are statistically relevant, resulting in 100 evaluations of each experiment.

TABLE III. ML ALGORITHMS INVESTIGATED.

Abbreviation	ML	Configuration
Knn1nInv	k-nn	Single neighbor
Knn30nInv	k-nn	30 neighbors with weight inversely proportional to the distance
Knn30nNow	k-nn	30 neighbors without distance weight
LinearGdy	LR	Greedy selection of attributes
Dectable	DT	Best first attribute selection
RBFn50c01s	RBFn	50 clusters (neurons), 0.1 minimum standard deviation
RBFn200c01s	RBFn	200 clusters, 0.1 minimum standard deviation
Pqr1_75	PQR	0.75 minimum accuracy threshold, considering Naïve Bayes, C4.5 and SVM as node classifiers
Pqr2_75	PQR2	Same as Pqr1_75, with range median, average, LR and SVM as leaf regressors
SmoPoly1	SVM	linear kernel
SmoPoly2	SVM	polynomial kernel of degree 2

The Receiver Operating Characteristic (ROC) curve is a method for visually comparing accuracy of classification algorithms - it was generalized for regression algorithms as Regression Error Characteristic Curves (REC) [23]. The REC curve displays the error tolerance in the horizontal axis and the accuracy of an algorithm in the vertical axis, with accuracy defined as the percentage of predictions below the error tolerance threshold. The area over the curve represents the expected error for a regression model. When the curve of a certain algorithm A always appears above the curve of another algorithm B, A is the preferred algorithm and A is said to dominate B. In this study, the percentage error ($\frac{|p_i - a_i|}{a_i}$, where p_i is the predicted value and a_i is the actual value for a test case i) was chosen as the error metric displayed in the REC curves. Thus, the area over the REC curve is close to the mean percentage error (MPE).

Evaluation of ML algorithms is performed by answering four questions.

Question 1: Which ML algorithm offers the best accuracy?

To evaluate accuracy, REC curve for each ML algorithm was generated for all four scenarios, when all system and application attributes are included in the training dataset (Figure 5). The fact that PQR2 dominates other studied algorithms in all scenarios (top left most curve in all graphs), shows that it can adapt better to different scenarios, where the predicted resource usage presents linear (RAxML RSS in the presence of input size information) and non-linear behavior (other scenarios) as a function of system and application characteristics. This adaptation is due to the fact that different classification and regression models are created for different regions of the data, at the cost of additional computation during training phase.

SVM and k-nn algorithms are in general the next best algorithms. In particular, since the BLAST dataset contains a large number of data points, the REC curves show that increasing the number of neighbors to 30, increases the accuracy of k-nn, but this effect is opposite for the RAxML dataset. In fact, due to the limited number of data points, increasing the number of neighbors in the RAxML scenario is extremely harmful, even more so when no weight is used. This result indicates the difficulty in finding an ideal number of neighbors, especially when the training information presents varying density of points across the data space. With respect to SVM, the graphs show that the polynomial kernel of degree 2 can adapt better to the non-linearity of data than the linear kernel, especially in the RAxML scenarios, without requiring ideal algorithm parameters to be found.

Similar to k-nn, RBFn requires the number of neurons to be specified, a configuration that dramatically impacts the prediction accuracy. Empirical tests showed that the ideal number of neurons for BLAST scenarios was 200, while 50 neurons were the best configuration for RAxML. These configurations were used in the comparative graphs.

Question 2: Which attributes should be included in the training dataset?

Previous works [1][5][11][12][24] have proposed and studied methods for systematically selecting attributes independently from the learning algorithm. In this work, the impact of different sets of attributes on the accuracy of prediction is evaluated. Figure 6 and Figure 7 show the MPE for BLAST and RAxML respectively, for different sets of attributes. The number that appears above each group of vertical bars is the MPE of the case including all attributes (leftmost bar).

In the BLAST scenario, the attributes included are: cluster name, CPU clock, amount of memory, location of data (nominal value indicating the name of the resource hosting the data), CPU speed (Flops), memory speed, disk speed (different from TABLE I in configurations with NFS to account for different number of clients) and number of bases in a sequence. Since the

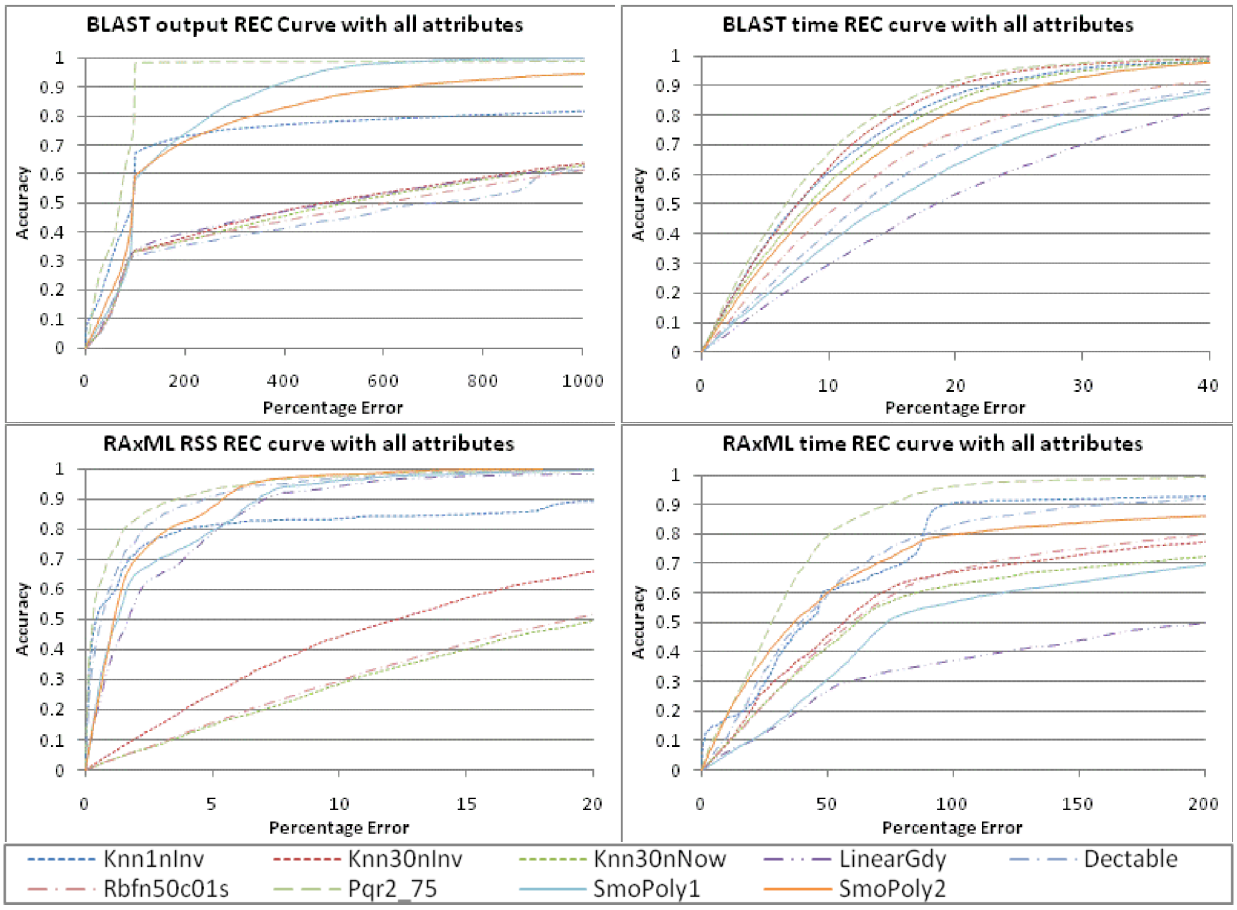

Figure 5. REC curves predicting disk space (top left) and execution time (top right) required by BLAST, and resident memory (bottom left) and execution time (bottom right) required by RAxML when taking all attributes into consideration for various machine learning algorithms. The area over the curve represents the mean percentage error, which is presented in Figure 6 and Figure 7.

output size is not dependent on any of the system attributes, the selection of attributes has almost no impact on the accuracy of BLAST output prediction (Figure 6 left). Thus, maintaining them in the dataset is not detrimental to the performance of ML algorithms. On the other hand, the addition of attributes, by collecting either more detailed information about the application being executed or more system benchmark and monitoring data, can highly impact the prediction accuracy. From the fact that for k-nn, decision table and PQR2 results, the system performance attributes offer better prediction (Fmdb and Cfmd cases) than simply using system specification attributes (Cmdb and Ccmdb), this make the case for clusters, grids and clouds middleware to provide this information to users and learning systems.

In the RAxML case, cluster name, number of threads, CPU clock, amount of cache, amount of memory, CPU speed (Flops), memory speed, disk speed, taxa size, number of bases in a sequence, and input size (taxa size multiplied by number of bases), were considered. The graphs with MPE, displayed in logarithmic scale (Figure 7), show PQR2 as the best algorithm for predicting memory and execution time requirements. Due to the linear relationship between the input size and RSS required by RAxML ($mem(t,b) = (t-2) * b * 32$, where t is the taxa size, and b is the number of bases), the input size is the attribute with the most impact on RSS prediction (cases ending with "t"), especially for the linear models (LR and SVM with linear kernel). As seen in the previous section, the execution time of RAxML is non-linear with respect to input size, number of threads and resources, leading in general to a high MPE. Still, PQR2 can adapt better to this situation, followed by decision table, k-nn, and SVM with polynomial kernel of degree 2. The decision table yields good prediction in this case due to the clusterized distribution of data points. The input size attribute continues to impact accuracy of time prediction more than the other attributes, including system performance attributes, making the case for Software-as-a-Service providers and job management systems to automatically collect more detailed application-specific information.

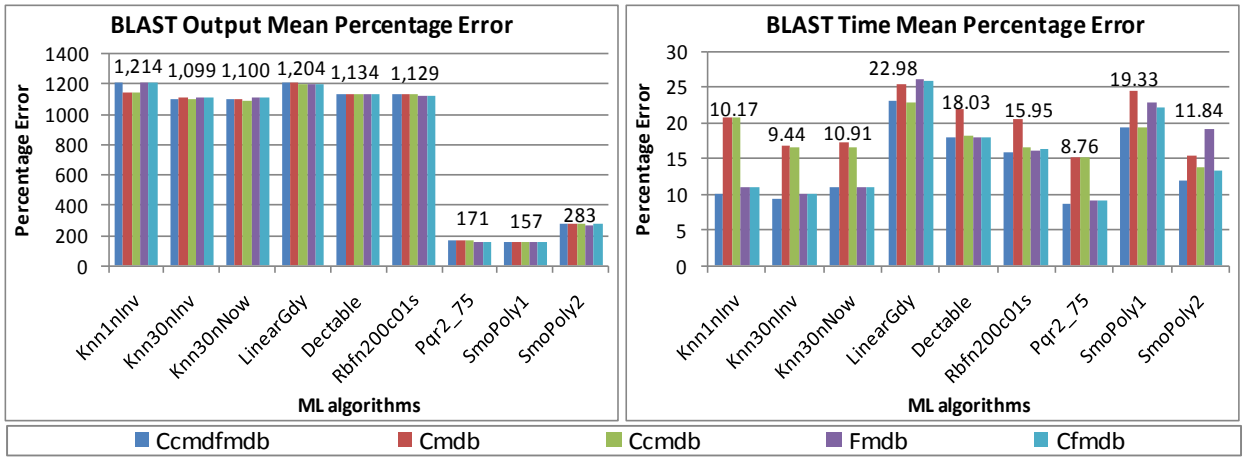

Figure 6. Average percentage errors in predicting disk space and execution time required by BLAST for various machine learning algorithms.

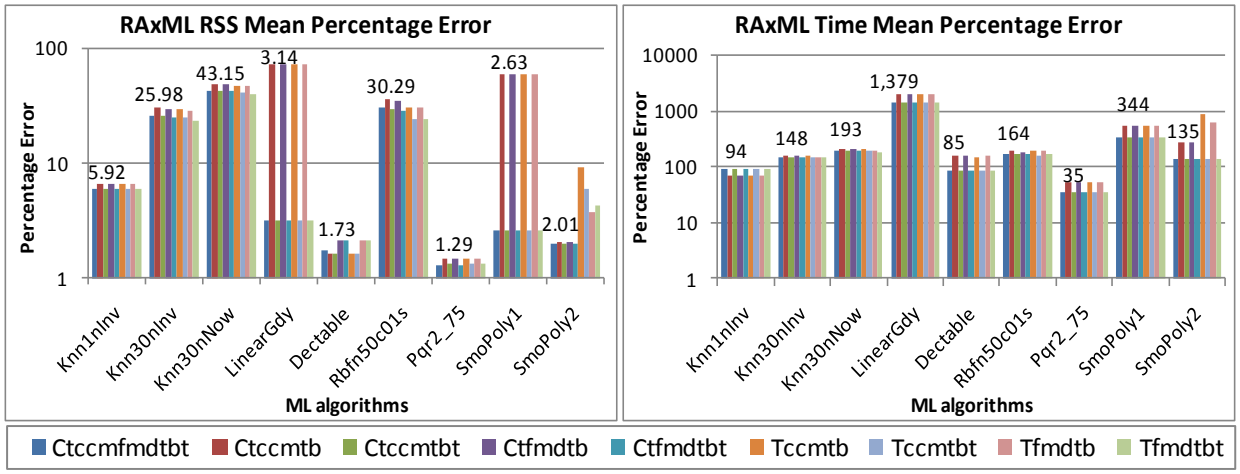

Figure 7. Average percentage errors in predicting resident memory and execution time required by RAxML for various machine learning algorithms (error axis in logarithmic scale).

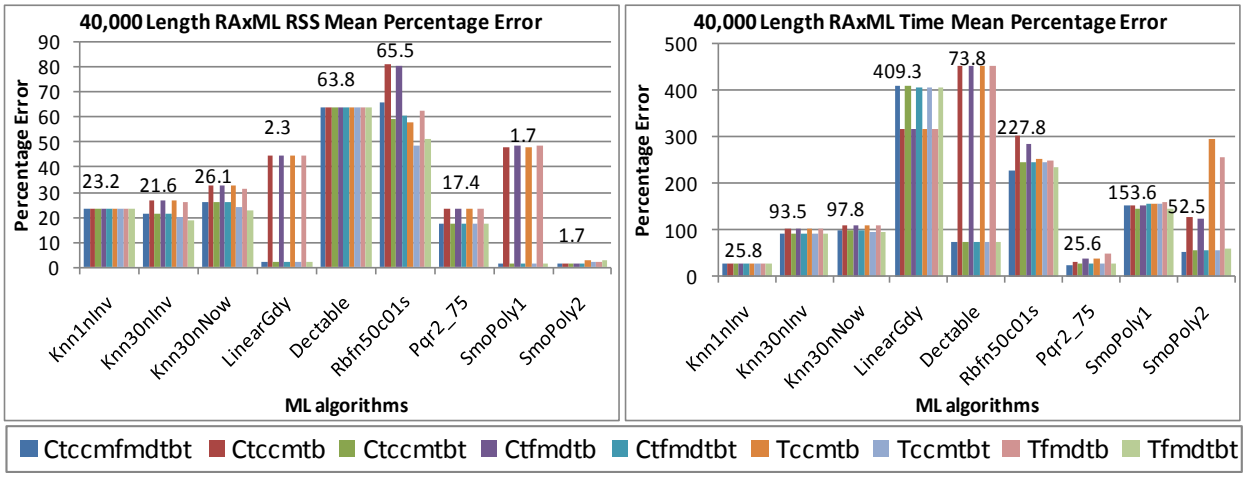

Figure 8. Average percentage errors in predicting resident memory (RSS) and execution time required by RAxML for all cases with 40,000-long sequences for various machine learning algorithms, using other data points as training data.

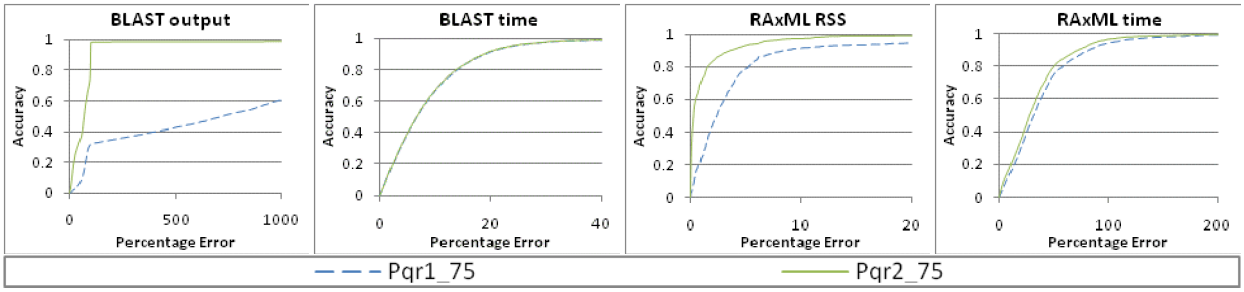

Figure 9. REC curves comparing accuracy of PQR and PQR2 for predicting BLAST output and execution time (2 graphs on the left) as well as RAxML memory consumption and execution time (graphs on the right).

Question 3: Which ML algorithm provides better accuracy when dealing with training datasets with low coverage?

To evaluate this situation, instead of separating training and test data in a random fashion, all data points of RAxML executions with 40,000 bases long input are used as test data and omitted from the training set. The MPE obtained for each combination of ML algorithm and attribute set show that for RSS prediction, the linear models can capture very well the overall model when the input size attribute is present, as expected (Figure 8 left). However, in general this outcome does not outweigh the benefits of other solutions in the more general scenario. The split of data in PQR2 diminishes the performance in this condition when compared to methods that produce a global model, but PQR2 still models better the general trend than the decision table and the k-nn, which always select the 50,000 data points as the closest neighbors. It can also be observed that the clustered nature of this dataset makes RBFn perform poorly since regions without data points are not included in the model.

For time prediction, the algorithms that can deal with non-linear execution times, PQR2 and k-nn, provide the best models even though no data points for 40,000-long inputs are included in the training set.

TABLE IV. MPE COMPARING PQR AND PQR2.

Dataset	Pqr1_75	Pqr2_75	Leaves
BLAST output	962%	171%	15
BLAST time	8.81%	8.76%	37
RAxML RSS	4.54%	1.29%	24
RAxML time	40.82%	35.30%	24

Question 4: Does PQR2 offer better accuracy than PQR?

REC curves (Figure 9) and the MPE (TABLE IV) show that PQR2 offers considerable improvement for predicting BLAST output, small benefits for predicting RAxML memory and time consumption, and essentially no difference predicting BLAST execution time, when compared to PQR. This is explained by the average number of leaves produced by PQR/PQR2 (TABLE IV). When the tree has a large number of leaves (37 for BLAST time model), the amount of data and range in each leaf, on which PQR2 can work to improve its accuracy, are small.

PQR2 models created with the entire dataset selected either the linear regression or SVM regression for modeling the data on the leaves, with a clear tendency to select SVM. This confirms that adding regression algorithms to the leaves is beneficial (PQR2) when compared to simply using the range mean or median (PQR). Considering the selection of classifiers, common to both PQR and PQR2, the resulting models included all classifiers, with strong tendency towards selecting C4.5.

VI. CONCLUSIONS AND DISCUSSIONS

The research reported in this paper considered the problem of accurately predicting application resource usage, reviewed and discussed several noteworthy machine learning algorithms considered by previous work, proposed and implemented PQR2, an extension of an existing classification tree algorithm, and compared all solutions (including the new PQR2 algorithm) under several conditions. Experiments predicting execution time, memory and disk requirements for two popular bioinformatics applications, BLAST and RAxML, were performed on a heterogeneous environment. Overall, PQR2 exhibited better accuracy when compared to other algorithms, due to its ability to better adapt to scenarios with different characteristics (linear and non-linear relationships, high and low density of training data points) by choosing different models for its nodes and leaves.

At a more general level, the two main conclusions from the work reported in this paper are as follows:

1. The scenarios requiring application resource prediction present a diverse behavior, making different algorithms perform better in different situations. The use of methods that can adapt to these situations by considering different configurations and algorithms is key for improving the quality of the prediction without requiring manual tuning. The resulting algorithms may be computationally more demanding during training, but this is usually not a concern as there is no need to generate a new model very often. Nevertheless, the

time and memory consumed by the prediction procedure for all ML algorithms is the subject of future work, with room for optimizations. Roughly, using the largest dataset (BLAST), PQR2 required a few minutes to create the model and a few milliseconds to produce a single prediction, indicating practicality of PQR2 for production deployments. PQR2 proved to be the best solution for BLAST and RAxML and should be considered as candidate solution for other applications.

2. Attributes can have high impact on the performance of the learning algorithms. The use of system performance attributes showed to be relevant for execution time prediction whereas application specific attributes were pertinent for all scenarios. This work makes the case for including as many attributes as available, while letting the algorithms analyze the relevance of the attributes when necessary. For cloud and grid computing scenarios, where resources are outsourced, the provision of this information to its users (or services acting on behalf of the users) through the use of benchmarks and runtime monitoring, especially of shared resources, can bring several benefits. Although this information is not readily available on a per application run basis, especially for shared resources, we expect it to become available in the near future (Amazon CloudWatch is one such example limited to a virtual machine instance). Improved prediction can result in better system utilization [1], can avoid application abortion in system that enforce accurate resource reservation, as well as significant savings when choosing the appropriate pay-as-you-go resource.

ACKNOWLEDGMENTS

This work is supported in part by NSF grants No. OCI-0721867, CNS-0821622, IIP-0758596 and the BellSouth Foundation. BLAST input sequences used in the experiments were provided by Dr. Leonid Moroz and RAxML alignments were provided by Mr. Michael Ott. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF or BellSouth Foundation.

REFERENCES

[1] R. Gibbons, "A Historical Application Profiler for Use by Parallel Schedulers," in Proc. Workshop Job Scheduling Strategies Parallel Process., 1997, pp. 58-77.

[2] A. W. Moore, J. Schneider, and K. Deng, "Efficient Locally Weighted Polynomial Regression Predictions," in Proc. 14th Int. Conf. Machine Learning, 1997, pp. 236-244.

[3] N. H. Kapadia, "On the Design of a Demand-Based Network-Computing System: The Purdue University Network-Computing Hubs," Purdue University, 1999.

[4] A. W. Mu'alem, and D. G. Feitelson, "Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling," IEEE Trans. Parallel Distrib. Syst., vol. 12, no. 6, pp. 529-543, 2001.

[5] W. Smith, I. Foster, and V. Taylor, "Predicting application run times with historical information," J. Parallel Distrib. Comput., vol. 64, no. 9, pp. 1007-1016, 2004.

[6] W. Smith, "Prediction Services for Distributed Computing," in Proc. 21st Int. Parallel Distributed Processing Symp., 2007.

[7] D. Tsafrir, Y. Etsion, and D. G. Feitelson, "Backfilling Using System-Generated Predictions Rather than User Runtime Estimates," *IEEE Trans. Parallel Distrib. Syst.,* vol. 18, no. 6, pp. 789-803, 2007.

[8] C. Gupta, A. Mehta, and U. Dayal, "PQR: Predicting Query Execution Times for Autonomous Workload Management," in Proc. 2008 Int. Conf. Autonomic Computing, 2008, pp. 13-22.

[9] I. Rodero, F. Guim, J. Corbalan et al., "The Grid Backfilling: a Multi-Site Scheduling Architecture with Data Mining Prediction Techniques," Grid Middleware and Services, pp. 137-152, 2008.

[10] R. Albers, E. Suijs, and P. H. N. de With, "Triple-C: Resource-usage prediction for semi-automatic parallelization of groups of dynamic image-processing tasks," in Proc. 23rd Int. Parallel Distributed Processing Symp., 2009.

[11] R. Duan, F. Nadeem, J. Wang *et al.*, "A Hybrid Intelligent Method for Performance Modeling and Prediction of Workflow Activities in Grids," in Proc. 2009 9th IEEE/ACM Intl. Symp. Cluster Computing and the Grid, 2009, pp. 339-347.

[12] F. Nadeem, and T. Fahringer, "Using Templates to Predict Execution Time of Scientific Workflow Applications in the Grid," in Proc. 2009 9th IEEE/ACM Intl. Symp. Cluster Computing and the Grid, 2009, pp. 316-323.

[13] S.F. Altschul, W. Gish, W. Miller, E.W. Myers, and D.J. Lipman., "Basic Local Alignment Search Tool", Journal of Molecular Biology, 1990, v. 215(3), pp.403-410, doi:10.1006/jmbi.1990.9999.

[14] A. Stamakis, "RAxML-VI-HPC: Maximum Likelihood-based Phylogenetic Analyses with Thousands of Taxa and Mixed Models," Bioinformatics 22(21):2688–2690, 2006.

[15] E. Alpaydin, Introduction to machine learning, MIT Press, 2004.

[16] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern classification*, 2 ed., Wiley, 2001.

[17] I. H. Witten, and E. Frank, Data mining: practical machine learning tools and techniques, 2 ed., Morgan Kaufmann, 2005.

[18] D. Feitelson. Parallel Workloads Archive. [Online]. Available: http://www.cs.huji.ac.il/labs/parallel/workload.

[19] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in Proc. fifth annual workshop on Computational learning theory, Pittsburgh, Pennsylvania, United States, 1992, pp. 144-152.

[20] H. Drucker, C. J. C. Burges, L. Kaufman et al., "Support Vector Regression Machines," in Advances in Neural Information Processing Systems 9, 1997, pp. 155—161.

[21] BLAST and RAxML execution traces. http://www.acis.ufl.edu/prediction

[22] R. Wolski, "Dynamically forecasting network performance using the Network Weather Service," Cluster Computing, vol. 1, no. 1, pp. 119-132, 1998.

[23] J. Bi, and K. P. Bennek, "Regression error characteristic curves," in Proc. 20th Int. Conf. Machine Learning, Washington DC, 2003, pp. 43-50.

[24] S. Krishnaswamy, S. W. Loke, and A. Zaslavsky, "Estimating computation times of data-intensive applications," *IEEE Distributed Systems Online,* vol. 5, no. 4, 2004.