Efficient Response Time Predictions by Exploiting Application and Resource State Similarities

Hui Li* David Groep† Lex Wolters*

*Leiden Institute of Advanced Computer Science (LIACS), Leiden University
PO Box 9512, 2333 CA
Leiden, The Netherlands

†National Institute for Nuclear and High Energy Physics (NIKHEF)
PO Box 41882, 1009 DB
Amsterdam, The Netherlands

E-mail: hli@liacs.nl davidg@nikhef.nl llexx@liacs.nl

Abstract

In large-scale Grids with many possible resources (clusters of computing elements) to run applications, it is useful that the resources can provide predictions of job response times so users or resource brokers can make better scheduling decisions. Two metrics need to be estimated for response time predictions: one is how long a job executes on the resource (application run time), the other is how long the job waits in the queue before starting (queue wait time). In this paper we propose an Instance Based Learning technique to predict these two metrics by mining historical workloads. The novelty of our approach is to introduce policy attributes in representing and comparing resource states, which is defined as the pool of running and queued jobs on the resource at the time to make a prediction. The policy attributes reflect the local resource scheduling policies and they can be automatically discovered using a genetic search algorithm. The main advantages of this approach compared with scheduler simulation are twofolds: Firstly, it has a better performance to meet the real time requirement of Grid resource brokering; secondly, it is more general because the scheduling policies are learned from past observations. Our experimental results on the NIKHEF LCG production cluster show that acceptable prediction accuracy can be obtained, where the relative prediction errors for response times are between 0.35 and 0.70.

1 Introduction

Large-scale Grids typically consist of many geographically distributed resources. As an example, the LHC Computing Grid (LCG) [5] testbed currently has approximately 140 sites in 34 countries with a total number of 12,516 CPUs and 5 Petabytes storage. Resource brokering or superscheduling in such an environment is a challenging task, where dynamic information about the sites are crucial in the scheduling cycle. Response time of a job, defined as the time elapsed from its submission till completion, is such a dynamic metric that needs to be estimated and provided by the resources. In this paper we address the problems of predicting application response times on resources such as space-shared parallel supercomputers and clusters.

Response time is composed of two main components: application run time and queue wait time. The basic idea of our approach is to derive predictions from similar observations in the past. The key problem here is how to define similarity to compare jobs. For the application run time, job attributes are naturally used for similarity definition. For instance, group name, application name, etc. For the queue wait time, however, new attributes need to be introduced as waiting time of a job is typically the result of interactions of the job, other running and queued jobs, and the scheduling system. We define a *resource state* as the pool of running and queued jobs on the resource at the time to make a prediction. Several new attributes are introduced to represent a resource state, such as the sum of run time multiplying with the number of CPUs ("processor*time") of queued jobs. We further introduce policy attributes to categorize resource state attributes in a more fine-grained level. For example, if group name is used as policy attribute, the "processor*time" of queued jobs will be categorized by groups. An Instance Based Learning (IBL) technique with an extended distance function is adopted to measure job as well as resource state similarity and make predictions. Attributes are selected and other IBL parameters are optimized using a genetic search algorithm.

The rest of the paper is organized as follows. Section 2 summarizes the related work in response time predictions. Section 3 introduces our definitions for job similarity and resource state similarity. New attributes defined for resource states are discussed in detail. Section 4 describes the prediction algorithm. The distance function, induction models, and the genetic algorithm are elaborated. Section 5 presents the experimental results and analysis using workload on the NIKHEF LCG production cluster. Conclusions and future work are discussed in the final section.

2 Related Work

Techniques have been proposed in the literature for predicting application run times using historical information. In [10] "templates of attributes" are defined for categorizing historical jobs and statistical techniques like mean or linear regression are applied to generate predictions. Templates can be automatically discovered using a genetic search. In [4] Instance Based Learning (IBL) techniques are investigated for run time predictions. IBL uses historical data "near" the query point to build a local model for approximation. A proper distance metric has to be defined to measure the distances between data instances. In fact the IBL algorithm is a generalization of the template approach, in which distances are simplified to binary values (belong or not belong to a specific category).

For queue wait times, the basic idea of most proposed techniques is based on scheduler simulation. In [11] and [12] scheduling algorithms like FCFS, LWF, and backfill are simulated for predicting queue wait times, where application run times are estimated using the template approach [11], or by IBL algorithms [12]. In [6] simulation is also adopted to predict queue wait times for a policy-based scheduler "Maui". Although relatively better prediction accuracy can be achieved using simulation, several major disadvantages remain for this approach. Firstly, it has a serious performance problem to meet the real-time requirement in the Grid brokering process. Secondly, it is not a general solution as there are different types of local scheduling systems deployed on different Grid sites. Some sites have schedulers with combinations of basic scheduling algorithms, most sites enforce different kinds of policies in their own fashion.

As an alternative solution, we are trying to derive queue wait times also from similar historical observations. From this perspective, the assumption is that "similar" jobs under "similar" resource states would most likely have similar waiting times, given that the scheduling algorithm and policies remain unchanged. Similar approach has been proposed in [9], in which summary statistics about the resource state (e.g. free CPUs, number of running jobs) is used as attributes in template definition. The template approach to estimate application run times as described above can be applied similarly to predict waiting times. Our work distinguishes from it in two aspects: Firstly we introduce attributes that reflect scheduling policies to represent resource

Abbr.	Job Attribute	Type
g	group name	nominal
u	user name	nominal
e	executable name	nominal
n	number of CPUs	nominal
r	requested run time	numeric-s
tod	arrival time of day	numeric-s

Table 1. Job attributes used (numeric-s: numeric scalar)

Abbr.	Policy Attribute	Type
g	group name	nominal
u	user name	nominal
q	queue name	nominal

Table 2. Policy attributes defined

states in a more fine-grained level for similarity comparison, where these policy attributes can be automatically discovered by a genetic search. Secondly we use Instance Based Learning as the common framework for both application run times and queue wait times. We elaborate our approach with detail in the following sections.

3 Similarity Definition

As mentioned above, the key problem in our approach is how to properly define similarity. We introduce definitions of job similarity and resource state similarity as follows.

3.1 Job Similarity

Jobs can be compared by their attributes recorded in the workload traces. Table 1 shows several main attributes that identify a job in openPBS accounting logs on the LCG production cluster at NIKHEF [8]. These attributes are mostly self-explanatory by their names and they have two main types, namely, nominal scalar (strings) or numeric scalar (numbers). In the Instance Based Learning algorithm described later, similarity between jobs are formulated by a distance function made of attributes. Jobs with smaller distances are considered more similar than others. For instance, jobs from the same group, say "bioinfo", and with same executable name called "proteinmatch" will have smaller distances than those who have nothing in common, therefore have a higher chance being used for predictions of the same kind. To make a good distance function, we employ a genetic search to select only relevant attributes according to the target (application run time or queue wait time). Automatic discovery of relevant attributes enables

Abbr.	State Attribute	Value Calculation	Type
freeCPUs	number of free CPUs		numeric-s
VRunJobs	categorized number of running jobs	$V_i = NR_i$	numeric-V
VQueueJobs	categorized number of queue jobs	$V_i = NQ_i$	numeric-V
VAlreadyRun	categorized sum of elapsed processor*time of running jobs	$V_i = \sum_{j=1}^{NR_i} alrunt(j) \times cpu(j)$	numeric-V
VRunRemain	categorized sum of remaining processor*time of running jobs	$V_i = \sum_{j=1}^{NR_i} remaint(j) \times cpu(j)$	numeric-V
VAlreadyQueue	categorized sum of already queued processor*time of queue jobs	$V_i = \sum_{j=1}^{NQ_i} alrquet(j) \times cpu(j)$	numeric-V
VQueueDemand	categorized sum of demand processor*time of queue jobs	$V_i = \sum_{j=1}^{NQ_i} demandt(j) \times cpu(j)$	numeric-V

Table 3. Resource state attributes (numeric-s: numeric scalar, numeric-V: numeric vector, V_i: the value of the ith category, NR: number of running jobs, NQ: number of queueing jobs, $alrunt(j)$: already run time for the jth job in the category, $remaint(j)$: remaining run time for the jth job in the category, $alrquet(j)$: already queued time for the jth job in the category, $demandt(j)$: run time for the jth job in the category, $cpu(j)$: number of CPUs for the jth job in the category)

the technique to apply when more information is available, such as executable arguments.

3.2 Resource State Similarity

More issues arise when measuring similarities of resource states. Firstly, since a resource state consists of running and queued jobs at a particular time, attributes have to be defined to represent the resource states in a way that they can be compared properly. Secondly, policy attributes that reflect the scheduling policies have to be embedded into the state attributes so that local scheduling information can be included. Table 2 and Table 3 lists the policy attributes and the resource state attributes, respectively. They are explained extensively as follows.

Three attributes are defined as the candidate policy attributes for resource states, namely, group name, user name, and queue name. These job credential information are frequently used in defining local scheduling policies. For instance, group name, which is also referred to "Virtual Organization" in the Grid, is a popular choice to make scheduling policies based on various QoS requirements of different stakeholders. Site administrators may define multiple queues and make different rules for them. Other attributes that may influence scheduling decisions can also be potentially added to this list. Useful policy attributes will be automatically selected in the genetic search.

The policy attributes identify a set of categories to which jobs in a resource state will be assigned. How to represent and compare categorized resource state attributes becomes a central problem to address. We define a new attribute type called "numeric-V", which contains a vector of <key, value> pairs. The key is the values of selected policy attributes and represents a specific category. The value is a numeric scalar variable associated with that category. As is shown in Table 3, most of the resource state attributes (except "freeCPUs") are of the type "numeric-V". For example, assuming that the selected policy attributes are group and queue name (written as "group-queue"), they generate categories such as "bioinfo-qlong", "hep-dque", and "astronomy-qshort". Attribute "VRunJobs" contains a vector of pairs like <"bioinfo-qlong", 32>, <"hep-dque", 8>, etc, where the value in each pair represents the number of running jobs in the category identified by the key. The same representation holds for other resource state attributes. "VQueueJobs" contains policy-categorized number of queueing jobs. "VAlreadyRun" ("VRunRemain") is the elapsed (the remaining) processor*time of the running jobs, which is calculated as the sum of number of CPUs multiply with the already run time (the remaining run time). "VAlreadyQueue" and "VQueueDemand" represent the already queued and the demand processor*time, respectively. Mathematical forms for value calculation are listed in Table 3. Distances between "num-V" type of attributes are calculated using the distance function described in the next section.

4 The Prediction Algorithm

Statistically application run times in the workload traces are distributed with heavy tails and show a high level of self-similarity [2]. This means that run times vary significantly across all time scales and global models such as non-parametric regression or multi-layer sigmoidal neural networks are not likely to work well. Same characteristics apply for queue wait times. For predictions in these situations, Instance Based Learning (IBL) [1] that exploits local models on relevant data become a more viable solution. We introduce our IBL-based technique and the genetic search algorithm as follows.

4.1 Instance Based Learning

IBL techniques typically store training data in a historical database, and make predictions for a particular query by applying an induction model on its "nearby" data entries. Two main components of IBL techniques, namely the distance function to measure "nearness" and the induction models, are defined for our algorithm.

4.1.1 The Distance Function

We employ the Heterogeneous Euclidean-Overlap Metric (HEOM) [13] as the distance function. This distance function can be used on nominal and numeric scalar attributes, and we extend it so that it can also be used for numeric vector attributes. The extended HEOM distance function defines distance between two values x and y of a given attribute a as:

$$d_a(x,y) = \begin{cases} \mathit{overlap}(x,y), & \text{if } a \text{ is nominal,} \\ \mathit{ns\_diff}_a(x,y), & \text{if } a \text{ is numeric-s,} \\ \mathit{nv\_diff}_a(x,y), & \text{if } a \text{ is numeric-V,} \\ 1, & \text{otherwise.} \end{cases}$$

The function *overlap* for nominal scalar values is defined as:

$$\mathit{overlap}(x,y) = \begin{cases} 0 & \text{if x = y,} \\ 1 & \text{otherwise.} \end{cases}$$

The function *ns_diff* for numeric scalar values is defined as:

$$\mathit{ns\_diff}_a(x,y) = \frac{|x-y|}{max_a - min_a},$$

where max_a and min_a are the maximum and minimum values observed in the training data for attribute a.

The function $\mathit{nv\_diff}_a(x,y)$ for numeric vector value is defined as:

$$\mathit{nv\_diff}_a(x,y) = \frac{\sum_{i=1}^{N_a}(x_i - y_i)}{\sum_{i=1}^{N_a}(x_i + y_i)},$$

where i is the ith category and N_a is the total number of categories of x and y. This numeric vector distance is the ratio of sum of per-category differences to the total amount of values in all categories.

Unknown attribute values are handled by returning an attribute distance of 1, i.e., the maximal distance. The above definition for d_a returns a value in the range 0..1. The overal distance between two input vector of attributes **x** and **y** is given by the Heterogeneous Euclidean-Overlap Metric function $D(\mathbf{x},\mathbf{y})$:

$$D(\mathbf{x},\mathbf{y}) = \sqrt{\frac{\sum_{a=1}^{m} w_a * d_a(x_a,y_a)^2}{\sum_{a=1}^{m} w_a}},$$

where attribute weights w_a (of binary values 0 or 1) enables attribute selection in the genetic search.

4.1.2 The Induction Models

Two induction models, namely 1-Nearest-Neighbor (1-NN) and n-Weighted-Average (n-WA) are considered in our algorithm.

1-NN: 1-Nearest-Neighbor predicts the target value of the query point using that of the nearest neighbor according to the distances.

n-WA: n-Weighted-Average makes predictions using the weighted average of the nearest n neighbors. It is defined as:

$$P(q) = \frac{\sum_{i=1}^{n} W_i * Val(e_i)}{\sum_{i=1}^{n} W_i},$$

$$W_i = K(D(q,e_i)), \text{ and } K(d) = e^{-d^2},$$

where q is the query point, $P(q)$ is the target value to be predicted, e_i is the ith nearest neighbors and $Val(e_i)$ is its target value, K is the kernel function and D is the distance function mentioned above.

4.2 Genetic Search

There are many parameters to be tuned for the basic IBL algorithm. For instance, it is important to select only relevant attributes in the distance function for better "nearness" measurement. In other words, attribute weights w_a have to be determined. The best policy attribute set has to be discovered for reflecting the local scheduling policies. Proper induction models as well as history database size have to be set for a practically useful algorithm. We exploit a genetic search to optimize these parameters by minimizing the average prediction error on the training data set. Our genetic algorithm is designed using standard operators such as selection, mutation, and crossover [3]. Chromosomes are structured to match the different objectives (application run time or queue wait time).

4.2.1 Application Run Time

For predicting application run times the job attributes listed in Table 1 are used in the distance function. The chromosome is encoded in binary bit format and it is structured in three building blocks as follows:

$$\{(w_1, w_2, ..., w_{N_a}), (m_1, m_2, ..., m_{N_m}), (h_1, h_2, ..., h_{N_h})\},$$

where N_a is the number of job attributes, N_m is the number of bits used for representing induction models, and N_h is the number of bits used for representing history database size. In our experiment we set $N_m = 3$ and select 8 candidate induction models for evaluation (1-NN, 3-WA, 5-WA, 10-WA, 15-WA, 20-WA, 30-WA, 50-WA). We set N_h also to 3 so 8 possible history database sizes are evaluated $(1000, 2000, ..., 8000)$. Larger history size than this set results in significantly longer lookup time while not much improvement in prediction accuracy is gained. The genetic search is performed on the training set and attempts to minimize the average prediction error for application run times. The optimized parameters after evaluation are used for run time predictions on the test set.

4.2.2 Queue Wait Time

For queue wait time predictions the chromosome is structured as follows:

$$\{(wp_1, wp_2, ..., wp_{N_p}), (ws_1, ws_2, ..., ws_{N_s}),$$
$$(wa_1, wa_2, ..., wa_{N_a}), (WA_1, WA_2),$$
$$(m_1, m_2, ..., m_{N_m}), (h_1, h_2, ..., h_{N_h})\},$$

where wp, ws, and wa are weights for job, policy, and resource state attributes, N_p, N_s, and N_a are their numbers, respectively. Induction models and history database size share the same structure as for application run times. Since both resource state and application similarities are concerned in queue wait times, we introduce a new building block (WA_1, WA_2) (binary values) to describe the relative significance of application similarity against resource state similarity. The distance function for queue wait times can be formulated by combining application distance and resource state distance:

$$D'(\mathbf{x}, \mathbf{y}) = WA * D_a(\mathbf{x}, \mathbf{y}) + (1 - WA) * D_s(\mathbf{x}, \mathbf{y}),$$

where $D_a(\mathbf{x}, \mathbf{y})$ and $D_s(\mathbf{x}, \mathbf{y})$ are distances of applications and resource states, respectively. *WA* is one of the four values $(0.25, 0.5, 0.75, 1)$ with respect to WA_1 and WA_2. *WA* defines the proportion of application and resource state in the overall distance function for queue wait times. For example, different jobs under similar states may have totally different waiting times. One may consider that application distance is more important than resource state by setting $WA = 0.75$, or vise versa using $WA = 0.25$. During the test stage, two resource state attributes, namely VRunRemain and VQueueDemand, are calculated using the estimated application run times. Predictions for response times is obtained by combining application run time and queue wait time estimates.

5 Experimental Results

We empirically evaluate our algorithm using eight-month (from June 2004 to January 2005) workload traces on the LCG production cluster at NIKHEF. This resource has the following characteristics:

1. **Architecture** The NIKHEF cluster is built using popular COTS (Commodity Of The Shelf) components and it consists of a mix of Intel and AMD CPUs, 512MB to 1GB memory, and Gigabit Ethernet connections. The total number of CPUs change in the range of 244 and 288 as nodes are added or removed.

2. **Workloads** The workload traces are recorded in the openPBS accounting logs. The cluster is heavily used in production as there are more than 130,000 job entries within eight-month time. Most of the jobs come from Virtual Organizations on the LCG testbed.

3. **Scheduling Policies** The cluster is running Maui [7] as the local scheduling engine. Firstfit backfilling is employed in the scheduling algorithm. Priority/fairshare/throttling policies are enforced for different groups and users according to their QoS requirements. Multiple queues are defined for jobs with different requested run times.

Two kinds of metrics are used for measuring the performance of our algorithm. *Prediction accuracy* is measured by the average absolute prediction error as well as by two types of relative prediction errors. The first type (referred as "absolute relative error") is calculated by the average absolute prediction error divided by the average run time (or wait time, or response time), as is used in Table 4, 5, and 6. The second type (referred as "relative error") is the ratio of differences between prediction and actual values against their sums. It is formulated as $r_e = (t_{est} - t_{real})/(t_{est} + t_{real})$, where t_{est} and t_{real} are predicted and actual values, respectively. Figure 1 illustrates the relative errors for application run time and queue wait time predictions.

Prediction time (Time per Prediction) is the average execution time in milliseconds (ms) per prediction. Our evaluation is carried out on a Intel Xeon machine with four 2.8GHz CPUs and 3GB shared memory. The workload dataset is divided into training and test sets throughout the evaluation. Performance are evaluated on the test trace data

Period	# Jobs	Best Parameters by GS	Abs. Error	Rel. Error	Time per Prediction
Aug '04	16,917	(1, 1, 0, 0, 1, 0) (15-WA) (3000)	468.5 min	0.67	11.5 ms
Sep '04	21,297	(0, 1, 0, 0, 1, 1) (10-WA) (5000)	253.4 min	0.63	19.8 ms
Oct '04	18,167	(1, 0, 1, 0, 1, 1) (10-WA) (6000)	409.5 min	0.73	24.9 ms
Nov '04	18,020	(0, 1, 0, 0, 1, 0) (10-WA) (5000)	210.5 min	0.43	17.1 ms
Dec '04	11,598	(1, 0, 1, 0, 1, 0) (15-WA) (5000)	577.1 min	0.63	18.1 ms
Jan '05	7,561	(0, 1, 0, 1, 1, 0) (15-WA) (6000)	542.1 min	0.72	17.7 ms

Table 4. Performance of the IBL algorithm in predicting application run times. Parameters: (g, u, e, n, r, TOD) (model) (history size).

Period	Best Parameters by GS	Wait Time			Response Time	
		Abs. Error	Rel. Error	T. p. P.	Abs. Error	Rel. Error
Aug '04	(1, 1, 0) (1, 1, 1, 0, 1, 0, 0) (0, 1, 1, 0, 0, 1) (0.5) (5-WA) (8000)	294.7 min	0.82	236 ms	659.0 min	0.62
Sep '04	(1, 0, 1) (0, 1, 1, 0, 1, 1, 1) (1, 0, 1, 1, 0, 1) (0.5) (10-WA) (5000)	184.0 min	0.83	141 ms	366.0 min	0.61
Oct '04	(0, 1, 0) (1, 0, 0, 1, 0, 0, 1) (1, 1, 1, 1, 1, 1) (0.75) (1-NN) (5000)	619.4 min	0.81	94 ms	946.6 min	0.70
Nov '04	(1, 0, 0) (0, 1, 0, 1, 0, 0, 1) (1, 1, 1, 1, 1, 0) (0.5) (10-WA) (5000)	192.4 min	0.39	112 ms	327.5 min	0.35
Dec '04	(1, 1, 1) (0, 0, 0, 1, 0, 0, 1) (1, 1, 0, 0, 0, 1) (0.5) (1-NN) (5000)	132.3 min	0.80	42 ms	716.4 min	0.64
Jan '05	(1, 0, 0) (0, 1, 0, 1, 0, 0, 1) (1, 1, 0, 0, 0, 0) (0.5) (10-WA) (6000)	54.4 min	0.99	56 ms	583.4 min	0.69

Table 5. Performance of the IBL algorithm in predicting queue wait times and response times. Parameters: (g, u, q) (freeCPUs, VRunJobs, VQueueJobs, VAlreadyRun, VRunRemain, VAlreadyQueue, VQueueDemand) (g, u, e, n, r, TOD) (WA) (model) (history size).

of every one month, with the parameters found by the genetic search on the preceding two-month traces. The use of independent test sets enables a more objective evaluation process and a more accurate reflection of how search and prediction would be performed in practice.

Table 4 shows the performance of application run time predictions in consecutive months. The absolute average errors range from 210.5 to 577.1 minutes, and the absolute relative errors are between 0.43 and 0.73. As is shown in Figure 1(a), we can see that the relative errors are centered at zero and a majority of jobs fall in between -0.5 and 0.5. this means the algorithm performs quite well for run time predictions. In Figure 1(c), we observe that more than 70% of underestimation ($r < 0$) comes from long-running jobs (with runtime larger than 10^4 seconds), which contribute largely in the above absolute error measurements. The algorithm suffers also from the "lagging" problem. Since a lot of long production jobs are submitted in "bags" (many almost identical jobs in very short time interval), the same instances in history are selected by IBL for predictions for all jobs in the bag. If the selected instances turn out to be "bad", large prediction errors are made for every job in the bag. In situations where short test jobs (appears similar in terms of features) exist between long production jobs the algorithm performs poorly. Possible improvements which are not investigated in this paper include adding more useful job attributes (such as executable arguments), applying locally weighted regression as the induction model, or taking application age (already elapsed run time) into account.

Table 5 shows the performance of the IBL algorithm in predicting queue wait times as well as response times. For queue wait times, the absolute average errors range from 54.4 to 619.4 minutes and the absolute relative errors are between 0.39 and 0.99. The relative error shown in Figure 1(b) and (d) indicates that the IBL algorithm is less effective in predicting wait times than run times. The underestimation for longer-waiting jobs still exists, but not as much as for run times. The algorithm suffers similarly from the "lagging" problem as mentioned above. The accuracy of wait time predictions is also related to the quality of run time predictions, as resource state attributes "VRunRemain" and "VQueueDemand" are calculated using run time

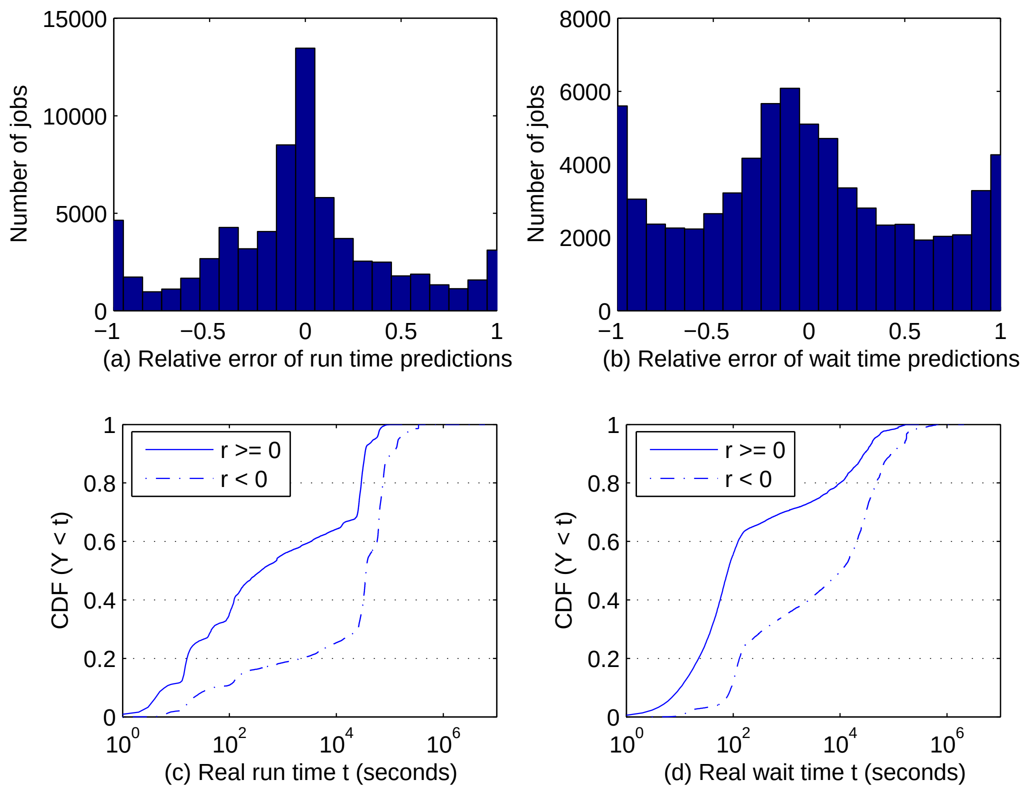

Figure 1. Histograms and cumulative distribution functions (CDFs) for relative errors.

estimates. The policy attributes identified by genetic search (in Table 5) correctly reflect the local scheduling policies on the NIKHEF LCG production cluster, in which group name is the most important attribute for policy expression (fine-grained policies are also specified for certain users). In terms of prediction time our algorithm performs well, with time per prediction less than 300 milliseconds. This is one major advantage compared with scheduler simulation, in which the prediction times are in the order of seconds or even minutes [6]. The prediction performance for response times are shown in Table 5. The absolute relative errors are between 0.35 and 0.70.

We further explore the internal workload structure by analyzing group behavior. Table 6 shows the prediction performance for 9 most active groups on the cluster. We can see that most of the "short" jobs are from group 5 (16.7%), which has an average running time in the order of minutes. For this group our prediction algorithm is able to perform fairly well, with an average absolute error of 12.1 minutes for response times. As the high priority for this group as well as backfilling policies are captured by IBL, good estimations can be made even in the states which have hundreds of other jobs waiting in the queue. We also notice that a majority of groups have relatively "long" jobs with average running times of hundreds of minutes. For these long-running jobs the prediction performance varies from group to group. On many groups such as the most active one - group 2 (36.5%), our algorithm performs relatively good, with an absolute relative error for response time of 0.45. However, poor performance is observed on groups such as group 7. One reason is that not enough similar job instances with similar resource states could be found in the limited history traces. On the other hand, even if similar jobs are available, the waiting time itself could not be determined merely by the job's current resource state. For instance, future job arrivals may cause queued jobs to wait longer and it results in more unpredictable wait times for jobs from groups with a low priority and strict throttling policies (like group 7). However, this is generally difficult for most prediction problems. Although our algorithm tends to underestimate for long-runing or waiting jobs, it is able to approximate the response time scales for different policy groups. This will provide useful information for the resource brokers to make decisions.

Group ID	Job Percent (%)	Run Time		Wait Time		Response Time	
		Abs. Error	Rel. Error	Abs. Error	Rel. Error	Abs. Error	Rel. Error
1	2.7	206.6 min	0.76	82.8 min	1.12	249.9 min	0.72
2	36.5	306.8 min	0.46	316.6 min	0.64	527.3 min	0.45
3	12.1	782.4 min	0.83	53.9 min	1.13	817.5 min	0.82
4	2.6	1218.3 min	0.75	33.1 min	1.51	1221.3 min	0.74
5	16.7	0.77 min	1.22	11.6 min	1.12	12.1 min	1.10
6	3.4	892.2 min	0.74	327.6 min	0.49	1037.1 min	0.55
7	4.0	350.1 min	1.65	2458.4 min	0.98	2625.8 min	0.97
8	2.4	34.3 min	1.75	607.3 min	0.29	601.5 min	0.28
9	17.9	403.3 min	0.59	120.5 min	0.92	471.5 min	0.58

Table 6. Prediction performance for 9 most active groups on the NIKHEF LCG production cluster.

6 Conclusions and Future Work

In this paper we propose an Instance Based Learning technique to predict application response times using historical workload traces on clusters. We define a new type of attributes (numeric vector) to represent a resource state, and introduce a distance function that can measure distances between these attributes. Local scheduling policies are embedded in the resource state attributes and they can be automatically discovered using a genetic search on historical data. Empirical evaluation is conducted using 8 month traces recorded on the NIKHEF LCG production cluster. The absolute relative prediction errors for response times are between 0.35 and 0.70. The average time per prediction is in the order of milliseconds. These yield a practical solution for response time predictions, which feed useful information in the resource brokering process.

The solution presented in this paper is our first attempt in deriving response time predictions using solely historical information and the results are preliminary rather than complete. We are evaluating and validating the technique on more clusters and public domain workload traces. More comparison study with the scheduler simulation approach is being conducted. We are also improving our technique in several directions. Firstly, the basic IBL algorithm can be further improved. We are experimenting with instance pruning techniques to reduce irrelevant instances for better performance. Secondly, we are improving the genetic algorithm to enable a more efficient search and reduce possibilities to be trapped in local minima. Thirdly, we are investigating metrics to reflect prediction inaccuracies and algorithms for resource brokers that can incorporate inaccuracies when making decisions.

Acknowledgments

We thank Jeroen Eggermont, Peter v.d. Putten (LIACS), and Jeff Templon (NIKHEF) for their insightful discussions. We also want to express our gratitude to reviewers for their suggestions that improved the quality of this paper.

References

[1] C. G. Atkeson, A. W. Moore, and S. Schaal. Locally weighted learning. *Artificial Intelligence Review*, 11(1-5):11–73, 1997.

[2] D. G. Feitelson. Workload modeling for performance evaluation. In *Job Scheduling Strategies for Parallel Processing*, pages 114–141. Springer Verlag, 2002.

[3] D. E. Goldenberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Inc., 1989.

[4] N. H. Kapadia, J. A. B. Fortes, and C. E. Brodley. Predictive application-performance modeling in a computational grid environment. In *IEEE International Symposium for High Performance Distributed Computing (HPDC)*, 1999.

[5] The lhc computing grid (lcg) project. `http://lcg.web.cern.ch.LCG/`.

[6] H. Li, D. Groep, J. Templon, and L. Wolters. Predicting job start times on clusters. In *proceedings of 4th IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2004.

[7] Maui scheduler. `http://supercluster.org/`.

[8] National institute for nuclear and high energy physics (nikhef), physics data processing and grid. `http://www.nikhef.nl/grid`.

[9] W. Smith. *Resource Management in Metacomputing Environments*. PhD thesis, Northwestern University, 1999.

[10] W. Smith, I. Foster, and V. Taylor. Predicting application run times using historical information. *Lecture Notes in Computer Science*, 1459:122–136, 1998.

[11] W. Smith, V. Taylor, and I. Foster. Using run-time predictions to estimate queue wait times and improve scheduler performance. In *Job Scheduling Strategies for Parallel Processing*, pages 202–219. Springer Verlag, 1999.

[12] W. Smith and P. Wong. Resource selection using execution and queue wait time predictions. Technical Report NAS-02-003, NASA Ames Research Center, 2002.

[13] D. R. Wilson and T. R. Martinez. Improved heterogeneous distance functions. *Journal of Artificial Intelligence Research*, 6:1–34, 1997.