Improving a Local Learning Technique for Queue Wait Time Predictions

Hui Li* Juan Chen* Ying Tao* 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

Abstract

Local learning has been proposed as a common framework to predict both application run times and queue wait times based on workload traces. The queue wait time is shown to be more difficult and expensive to predict because its distance calculations typically involve not only job attributes but also resource states. In this paper methods and algorithms are investigated to improve prediction accuracy and prediction performance for queue wait times. Firstly, the so-called "local tuning" is adopted to tune parameters for each training subset divided by a pivot attribute (e.g., group or queue name). Bias-variance analysis of error is conducted on local tuning and its global counterparts - tuning parameters on the whole training set. A method is then developed to select tuning type adaptively based on the generalization error and bias-variance decomposition. Secondly, an efficient search tree structure called "M-Tree" is integrated into our algorithm to speed up k-nearest neighbor search. Experimental studies are conducted to evaluate the proposed methods and algorithms using real-world workload traces, which are collected from the NIKHEF production cluster on the LHC Computer Grid and Blue Horizon in the San Diego Supercomputer Center (SDSC). The results show that adaptive tuning can reduce the average prediction error by 3 to 10 percents compared to global tuning, and that the M-Tree nearest neighbor search is up to 8 times faster than the original sequential search.

1 Introduction

Job response times on computing resources are important information for supporting metascheduling decisions in large-scale Grids. In [8] local learning, or instance based learning, has been proposed to predict both application run times and queue wait times, from which response times can be derived. On one hand, the basic local learning technique proves to be quite effective in predicting run times, where good prediction accuracy and performance can be achieved. On the other hand, however, it is shown that the queue wait times are more difficult and expensive to predict. This is because the distance calculations of queue wait times involve not only job attributes but also resource state attributes. *Resource state* is defined as the pool of running and queued jobs on the resource at the time to make a prediction. Several attributes are defined to represent a resource state from different views, such as the sum of run time multiplied by the number of CPUs of queued jobs. Policy attributes are further introduced to categorize resource state attributes in a more fined-grained level for similarity comparison. The complexity brought by the resource states makes it much more challenging for queue wait time predictions.

Nevertheless, the queue wait time is the key component in the response time and the local learning technique can be called effective only when it also performs well for wait times. In this paper solutions are proposed to improve the queue wait time predictions under the local learning framework. Firstly, we investigate methods that can improve prediction accuracy. A "local tuning" method is introduced to tune parameters for each training subset divided by a pivot attribute. For instance, *group* is a good candidate to partition the whole training data into subsets. Bias-variance analysis of error is then conducted for both global and local tuning to gain insights on how each tuning type performs. Based on this we further develop a method that can select the tuning type adaptively. Secondly, we adopt a search tree structure called "M-Tree" for k nearest neighbor search. Compared with the original sequential search in our algorithm, the "M-Tree" is intrinsically much more efficient in searching neighbors for predictions. Finally, we evaluate the proposed methods and algorithms on real-world workload traces from NIKHEF and SDSC.

The contributions of this paper are three-folds. Firstly, we conduct bias-variance analysis of error for global and local tuning and develop an adaptive selection method that can effectively detect overfitting by local tuning while keep its advantages. Secondly, the "M-Tree" structure is success-

fully integrated in our prediction algorithm to improve performance of nearest neighbor search. Compared with previous search optimization methods like caching and instance editing [5], to our best knowledge this is the first time that a tree-structured access method is investigated in the Grid performance predictions context. Thirdly, compared with our previous paper [8], we conduct a more extensive empirical study using different workloads with diverse characteristics to validating our algorithm.

The rest of the paper is organized as follows. Section 2 briefly describes our basic prediction algorithm. Section 3 introduces global tuning and local tuning, bias-variance analysis, and the adaptive method for tuning selection. Section 4 describes the M-Tree nearest neighbor search algorithm. Section 5 presents the experimental results and analysis using NIKHEF and SDSC traces. Conclusions and future work are discussed in the final section.

2 The Basic Prediction Algorithm

The basic idea of our approach is to derive queue wait time predictions from similar historical observations in workload traces. From this perspective, we assume that "similar" jobs under "similar" resource states would most likely have similar waiting times, given that the scheduling algorithm and policies remain unchanged for a reasonable amount of time. The key problem here is how to properly define similarity to compare jobs as well as resource states. Job comparisons can be done via their attributes recorded in the traces. For instance, "user name", "number of CPUs" and "application name" are some important attributes that characterize a job. For resource states we define the following new attributes: *RunJobs* (the number of running jobs), *QueueJobs* (the number of queued jobs), *AlreadyRun* (the sum of already past run times multiplied by #CPUs of running jobs), *RunRemain* (the sum of remaining run times multiplied by #CPUs of running jobs), *AlreadyQueue* (the sum of already past queue times multiplied with #CPUs of queued jobs), and *QueueDemand* (the sum of run times multiplied by #CPUs of queue jobs). These derived attributes characterize a resource state from different angles. Furthermore, we introduce policy attributes that reflect local scheduling policies to categorize resource state attributes. Therefore the resource state comparisons can be done on a more fine-grained level. Once the attributes are defined, the local learning technique can be applied for predictions [1]. We employ an extended Heterogeneous Euclidean-Overlap Metric (HEOM) as the distance function to handle both job and resource state attributes. Weighted Average (WA) and Locally Weighted Linear Regression (LLWR) are used as the candidate induction models for predictions. We refer to [9] for details and formulations of the basic prediction algorithm.

3 Parameter Tuning

There are quite a lot parameters to be tuned for the basic local learning technique. For instance, it is important to select only relevant attributes in the distance function for better "nearness" measurement. In other words, attribute weights have to be determined. The policy attribute set has to be discovered to reflect the local scheduling policies. The induction model, history size, neighbor size, and kernel bandwidth have to be set for a practically useful algorithm. We designed a genetic algorithm to optimize these parameters by minimizing the average prediction error on the training data set. Here we refer it as "global tuning" since the evaluation is done on the whole training data. The optimized parameters are then used for predictions on the test data set.

One observation based on many Grid sites is that resources typically have one major attribute defined for scheduling policy expression. On many clusters in the LHC Computing Grid (such as NIKHEF) this attribute is *group* (or Virtual Organization). On supercomputers such as Blue Horizon at SDSC the priority factor in the scheduler is based on *queue*. Of course sites may have multiple attributes combined to define scheduling rules, however, there is usually one which has a dominant impact. More interestingly, the number of values for this attribute is typically quite small. This is because policies are made by site administrators or investors and it is natural to keep it in a manageable fashion. For example, on Blue Horizon there are six function or policy queues defined, namely *interactive, express, high, normal, low* and *diagnosis*. On NIKHEF there are around fifteen groups, of which five or six are regularly active and contribute to a major fraction of jobs. One idea based on this observation is to introduce the policy attribute as a pivot to partition training data into subsets and tune parameters for each subset. This so-called "local tuning" method may improve the prediction accuracy as the parameters are optimized for every targeted policy group. Moreover, as the number of subspaces is small, we are quite confident that there is enough data in each set for training separate learners.

However, the problem of local tuning is that it has a higher probability of suffering from overfitting because of less data. It is useful to examine the prediction error under the framework of bias-variance decomposition, which will be discussed in the following section.

3.1 Bias-Variance Analysis

In many real-life learning problems it is somewhat counter intuitive that simple methods are often competitive and sometimes superior to more complex ones for estimation. This is because the bias and variance components of

the estimation error have impacts in a different way and usually there is so called *bias/variance dilemma* [4]. In our case the locally tuned learner may be less biased, but it can introduce more variance during generalization. Therefore it is important to decompose the bias and variance components of error and analyze their individual influence.

The following is a paraphrase of Geman et al. [4] on bias/variance decomposition of the mean-squared error. Suppose the regression problem is to construct a function $f(\mathbf{x}; D)$ based on a training set $D = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)\}$. The mean-squared error of f as an estimator of the regression $E[y|\mathbf{x}]$ can be written as

$$E_D[(f(\mathbf{x}; D) - E[y|\mathbf{x}])^2] = (E_D[(f(\mathbf{x}; D)] - E[y|\mathbf{x}])^2 + E_D[(f(\mathbf{x}; D) - E_D[f(\mathbf{x}; D)])^2]$$

The first component on the right of the equation is "bias" and the second one is "variance". As we can see, both bias and variance can contribute to the mean-squared error. An unbiased estimator may still have poor performance if the variance is large. There is often a tradeoff between the bias and variance contributions to the estimation error. Given this tradeoff, we develop a method that can adaptively determine when it make sense to use locally tuned models as compared with the global model.

3.2 Adaptive Selection

There are several design principles for the adaptive method. Firstly, there must be enough data in the training subset for obtaining statistically significant results. Secondly, local tuning must have comparable or smaller bias than global tuning otherwise we are most likely fitting noise rather than signal. Thirdly, local tuning must produce smaller or comparable average prediction error on the training set and the last observed generalization set. Based on these principles, our adaptive method combines the generalization error, training data size and bias/variance analysis to make an educated selection of tuning.

Algorithm 3.2 shows the pseudo-code of the adaptive selection method. In essence the algorithm consists of a set of "filters" that implement the design principles. We define *Error* as the average prediction error normalized by the average real value. Line 3 of Algorithm 3.2 says that the training subset size (N_{train}) must be large enough and local tuning should perform better or comparable on the training subset. Line 4 indicates that local tuning should have a comparable or smaller bias on the previous generalization set. ε_{err}, ε_{num}, and ε_{bias} are threshold values for comparing error, data size and bias. If the above conditions are met, we proceed to examine the errors on the previous generalization set. If local tuning has a smaller generalization error

Algorithm 1 Adaptive Selection of Tuning

```
1:  set the pivot attribute
2:  for each training subset divided by the pivot attribute do
3:    if TrainError_local − TrainError_global < ε_err and N_train > ε_num then
4:      if GenBias_local/GenBias_global < ε_bias then
5:        if GenError_global − GenError_local > ε_err then
6:          return LOCAL
7:        end if
8:        if |GenError_global − GenError_local| < ε_err then
9:          if (GenVar_local + GenBias_local) < (GenVar_global + GenBias_global) then
10:           return LOCAL
11:         end if
12:       end if
13:     end if
14:   end if
15:   return GLOBAL
16: end for
```

(line 5), we use locally tuned estimators. If the generalization error is comparable for global and local tuning (line 8), we select local tuning only when it also has a smaller mean-squared error (bias + variance). In other situations we use the globally tuned model. As we will see in the experimental studies this method is effective in detecting overfitting caused by local models and combining the advantages of both global and local tuning.

4 Nearest Neighbor Search

The nearest neighbor predictor is practically useful and fast compared to the scheduler simulation counterparts [7]. However, the learning process (optimizing parameters) is shown to be very slow if the data size becomes large. In the basic prediction algorithm we implement the k nearest neighbor search sequentially with some small improvements. The sequential search is relatively slow as it has to calculate distances with all entries in the history base. Since it involves resource state attributes, the distance calculations for queue wait times are much more expensive and it cannot employ caching like run times without compromising accuracy [9]. To improve performance a different access structure is needed and we investigate M-Tree in this context.

The M-Tree [3] is a search tree structure to organize and access large data sets from a generic metric space. In the metric space object proximity is only defined by a dis-

tance function satisfying positivity, symmetry, and triangle inequality postulates [2], on which our distance function for queue wait times hold. M-Tree is a tree where a set of representatives are chosen at each node and the elements closer to each representative are organized into a subtree. Each representative stores its *covering radius* r and all objects in the subtree are within the distance r from the representative. At query time, the query is compared against all the representatives of the node and enters recursively into all those that cannot be discarded using the covering radius criterion. The k nearest neighbor search in the M-Tree uses a branch-and-bound technique. The query radius is firstly assumed positive infinite and it is dynamiclly decreased to the distance to the k-th nearest neighbor. The M-Tree structure is intrinsically much more effective in reducing the number of distance calculations compared with the sequential search. Comprehensive descriptions and formulations of the algorithm are out of scope of this paper and we refer to [3] for details. We implement the M-Tree k nearest neighbor query based on the XXL library [12].

5 Empirical Evaluation

We empirically evaluate the proposed methods using workload traces with diverse characteristics. The NIKHEF production cluster is a representative site in the LHC Computing Grid [6], which is primarily used for physics data processing. It consists of around 300 Intel and AMD CPUs, up to 4GB memory per node, and Ethernet connections among nodes. The cluster runs Maui as the local scheduling engine. Firstfit backfilling is employed in the scheduling algorithm and policies are enforced for groups and users according to their QoS requirements. The SDSC Blue Horizon is a terascale IBM SP supercomputer with 1152 CPUs [11]. The scheduler on this machine, Catalina, uses one priority execution queue, performs backfilling, and supports reservations. Multiple submission queues are defined with different priorities. The selected traces are suitable for experimental study because of the diversities in application types, machine architecture, and scheduling policies.

Prediction accuracy and *prediction time* naturally are the two metrics for measuring the performance of our methods. Prediction accuracy is measured by the average absolute prediction error and the relative error, which is the average absolute error divided by the average real value. Prediction time is measured as the average execution time in milliseconds per prediction/query. The evaluation is carried out on multiple Intel Xeon machines with four 2.8 GHz CPUs and 3 GB shared memory. The workload dataset is divided into training and test sets. On NIKHEF, we test the performance on trace data of one month of consecutive months, with parameters trained on the preceding two-month traces. On the SDSC traces, testing is done on data of every three months because of less jobs. The parameters are optimized using the preceding six-month traces. This evaluation process is considered more objective as it resembles how learning and prediction would be performed in practice.

Our experimental study is conducted to answer the following questions:

1. How does local tuning perform compared to its global counterparts?
2. How effective is the proposed adaptive method in detecting overfitting and reducing the generalization error?
3. How much performance gain can be obtained by M-Tree compared with the sequential search?

Figure 1 illustrates the performance of global and local tuning on the training and generalization sets, respectively. The comparison is done on a per group/queue basis. The first two columns of each period in the subfigures are the relative absolute prediction errors on the training set. As we can see, local tuning achieves comparable or better accuracy than global tuning in most cases during training. It is intuitive that locally optimized parameters for one subset generally fit that dataset better than global parameters. The relative generalization errors are shown as the third/fourth columns in each period and the results are diverse from period to period. On some groups/queues like group *dzero* and queue *express*, local tuning is able to produce smaller prediction errors than global tuning in consecutive months. However, local tuning performs even worse on periods like December, 2004 of group *lhcb* and October to December, 2001 of queue *high*, despite that it is superior on the training set. This is commonly due to overfitting and it indicates that we cannot make decisions on whether to use local models only based on the training error. More sophisticated analysis like bias-variance decomposition is needed for a sound selection of tuning methods.

Table 1 and 2 shows the comparisons of global, local, and adaptive tuning on SDSC and NIKHEF, respectively. On each trace, the results for the most representative groups or queues are listed. The generalization error and its bias/variance decomposition are shown for consecutive periods for each group/queue. The training period of the first generalization is also listed since the adaptive method is applied on the training set if no previous generalization is available. The three parameters for adaptive selection are set as $\varepsilon_{err} = 0.05$, $\varepsilon_{num} = 1000$, and $\varepsilon_{bias} = 2$. In other words, the relative error is considered *comparable* if the absolute difference is below 5 percent. The training set is large enough if there are more than 1000 jobs. The bias for local tuning is considered comparable or smaller than the global bias if the local bias is smaller than two times of the global bias. Firstly we examine the results on SDSC. We can see

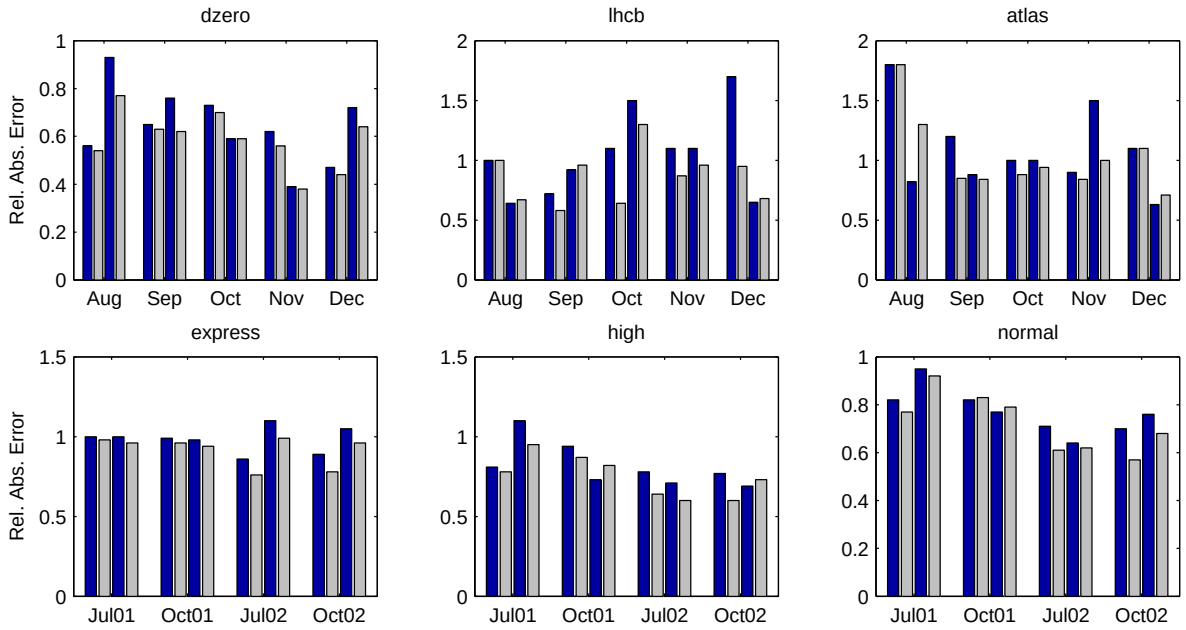

Figure 1. Comparisons of training (column 1, 2) and generalization errors (column 3, 4) of global (dark color) and local models (light color) for selected groups (NIKHEF) and queues (SDSC).

Queue	Metric	Jan-Jun-01	Jul-Sep-01	Oct-Dec-01	Overall	Jul-Sep-02	Oct-Dec-02	Overall
express	**global err**	1.0 | 44.8	1.0 | 72.4	0.98 | 176	0.99 | 127	1.1 | 91	1.1 | 114	1.06 | 103
	global var	3.6×10^7	1.0×10^8	9.2×10^7		1.8×10^8	1.3×10^8	
	global bias	6.3×10^5	2.0×10^6	3.0×10^6		3.5×10^6	1.3×10^7	
	local err	0.98 | 42.3	0.96 | 67.2	0.94 | 169	0.94 | 121	0.99 | 83	0.96 | 105	0.97 | 94
	local var	3.4×10^6	7.4×10^6	2.0×10^7		5.8×10^7	1.4×10^8	
	local bias	4.0×10^6	1.2×10^7	7.8×10^7		2.8×10^6	1.2×10^7	
	adaptive		global	global	0.99 | 127	local	local	0.97 | 94
	size		7366	8308	15674	9205	9707	18912
high	**global err**	0.81 | 387	1.1 | 720	0.73 | 829	0.95 | 750	0.71 | 1138	0.69 | 1439	0.70 | 1294
	global var	2.0×10^9	4.5×10^9	3.0×10^9		1.1×10^{10}	3.5×10^{10}	
	global bias	4.9×10^7	1.7×10^8	7.0×10^8		5.6×10^8	2.0×10^9	
	local err	0.78 | 373	0.95 | 629	0.82 | 930	0.90 | 711	0.60 | 970	0.73 | 1530	0.68 | 1262
	local var	1.5×10^9	1.4×10^9	3.6×10^9		7.5×10^9	1.7×10^{10}	
	local bias	5.4×10^7	3.9×10^8	1.7×10^9		9.4×10^7	2.1×10^9	
	adaptive		local	global	0.86 | 683	local	local	0.68 | 1262
	size		1910	706	2616	1062	1152	2214
normal	**global err**	0.82 | 745	0.95 | 955	0.77 | 1008	0.86 | 977	0.64 | 2370	0.76 | 1221	0.67 | 1876
	global var	8.5×10^9	1.3×10^{10}	6.7×10^9		3.4×10^{10}	1.7×10^{10}	
	global bias	2.1×10^8	3.3×10^8	5.7×10^8		1.1×10^{10}	1.5×10^9	
	local err	0.77 | 703	0.92 | 923	0.79 | 1036	0.86 | 970	0.62 | 2284	0.68 | 1098	0.64 | 1774
	local var	6.0×10^9	1.3×10^{10}	1.4×10^{10}		6.5×10^{10}	2.0×10^{10}	
	local bias	5.0×10^8	5.5×10^8	8.7×10^8		6.5×10^9	5.0×10^8	
	adaptive		global	global	0.86 | 977	local	global	0.65 | 1827
	size		5147	3657	8804	4778	3607	8385

Table 1. Comparisons of three tuning methods for selected queues on SDSC. Error: relative error | absolute error (minutes). Adaptive tuning parameters: $\varepsilon_{err} = 0.05$**,** $\varepsilon_{num} = 1000$**,** $\varepsilon_{bias} = 2$**.**

Group	Metric	Jun-Jul	Aug	Sep	Oct	Nov	Dec	Overall
dzero	**global err**	0.56 | 506	0.93 | 1332	0.76 | 220	0.59 | 348	0.39 | 196	0.72 | 243	0.59 | 302
	global var	1.2×10^9	3.0×10^9	1.9×10^8	5.0×10^8	5.0×10^8	1.0×10^8	
	global bias	3.9×10^8	2.0×10^9	5.6×10^7	1.0×10^8	1.6×10^7	2.0×10^8	
	local err	0.54 | 490	0.77 | 1099	0.62 | 180	0.59 | 349	0.38 | 193	0.64 | 216	0.55 | 282
	local var	1.0×10^9	7.2×10^8	2.5×10^8	8.3×10^8	4.3×10^8	1.4×10^8	
	local bias	2.9×10^8	2.8×10^9	3.5×10^7	7.2×10^7	2.0×10^7	1.6×10^8	
	adaptive		local	local	local	global	local	0.55 | 283
	size		1099	7535	10056	8743	691	28124
lhcb	**global err**	1.0 | 208	0.64 | 156	0.92 | 3.5	1.5 | 9.6	1.1 | 25	0.65 | 133	0.65 | 91
	global var	8.4×10^7	2.5×10^8	7.7×10^4	8.8×10^5	1.5×10^6	1.3×10^8	
	global bias	1.2×10^8	1.9×10^7	1.2×10^4	1.3×10^4	6.7×10^5	7.0×10^6	
	local err	1.0 | 210	0.67 | 163	0.96 | 3.6	1.3 | 8.6	0.96 | 21	0.68 | 141	0.68 | 96
	local var	1.7×10^8	2.4×10^8	1.6×10^5	1.4×10^6	1.4×10^5	1.3×10^8	
	local bias	1.0×10^8	4.2×10^7	9.7×10^3	3.9×10^3	1.4×10^6	1.3×10^7	
	adaptive		global	global	global	local	global	0.65 | 91
	size		2498	3122	703	471	4753	11551
atlas	**global err**	1.8 | 0.56	0.82 | 35	0.88 | 12	1.0 | 25	1.5 | 14	0.63 | 57	0.78 | 32
	global var	8.0×10^3	7.0×10^6	1.6×10^5	4.9×10^5	4.7×10^7	3.3×10^7	
	global bias	8.8×10^1	1.3×10^6	2.4×10^5	6.0×10^5	3.3×10^2	6.6×10^6	
	local err	1.8 | 0.55	1.3 | 55	0.84 | 11	0.94 | 23	1.0 | 9	0.71 | 64	0.90 | 37
	local var	8.6×10^3	2.0×10^7	7.4×10^4	6.8×10^5	7.9×10^5	2.2×10^7	
	local bias	1.5×10^2	1.3×10^3	2.6×10^5	7.3×10^5	6.7×10^4	8.9×10^6	
	adaptive		global	global	local	local	global	0.73 | 29
	size		1927	260	658	2813	1049	7707
theory	**global err**	0.74 | 1346	0.86 | 2634	0.98 | 1577	1.3 | 3477			1.14 | 2685
	global var	1.8×10^9	4.4×10^8	6.0×10^6	3.9×10^{10}			
	global bias	4.5×10^9	2.5×10^{10}	9.0×10^9	4.0×10^9			
	local err	0.83 | 1515	0.95 | 2925	1.0 | 1643	0.86 | 2382			0.91 | 2145
	local var	7.9×10^8	1.5×10^9	9.5×10^8	5.0×10^{10}			
	local bias	5.4×10^9	1.9×10^{10}	6.7×10^9	3.2×10^9			
	adaptive		global	global	local			0.89 | 2096
	size		255	1187	1675	no data	no data	3117

Table 2. Comparisons of three tuning methods for selected groups on NIKHEF. Error: relative error | absolute error (minutes). Adaptive tuning parameters: $\varepsilon_{err} = 0.05$, $\varepsilon_{num} = 1000$, $\varepsilon_{bias} = 2$.

Trace	Size	Method	Rel. Error	Abs. Error	Variance	Bias
NIKHEF	62129	global	0.77	294 min	2.0×10^9	5.0×10^7
		local	0.68	259 min	3.3×10^9	2.0×10^7
		adaptive	0.67	255 min	3.3×10^9	2.0×10^7
SDSC01	40295	global	0.89	379 min	3.5×10^9	5.6×10^7
		local	0.87	370 min	3.8×10^9	1.3×10^8
		adaptive	0.86	368 min	3.1×10^9	7.8×10^7
SDSC02	29511	global	0.70	696 min	1.1×10^{10}	7.0×10^8
		local	0.66	659 min	1.7×10^{10}	3.8×10^8
		adaptive	0.67	674 min	1.6×10^9	4.7×10^8

Table 3. Comparisons of overall generalization errors and bias/variance for global, local, and adaptive tuning on NIKHEF and SDSC.

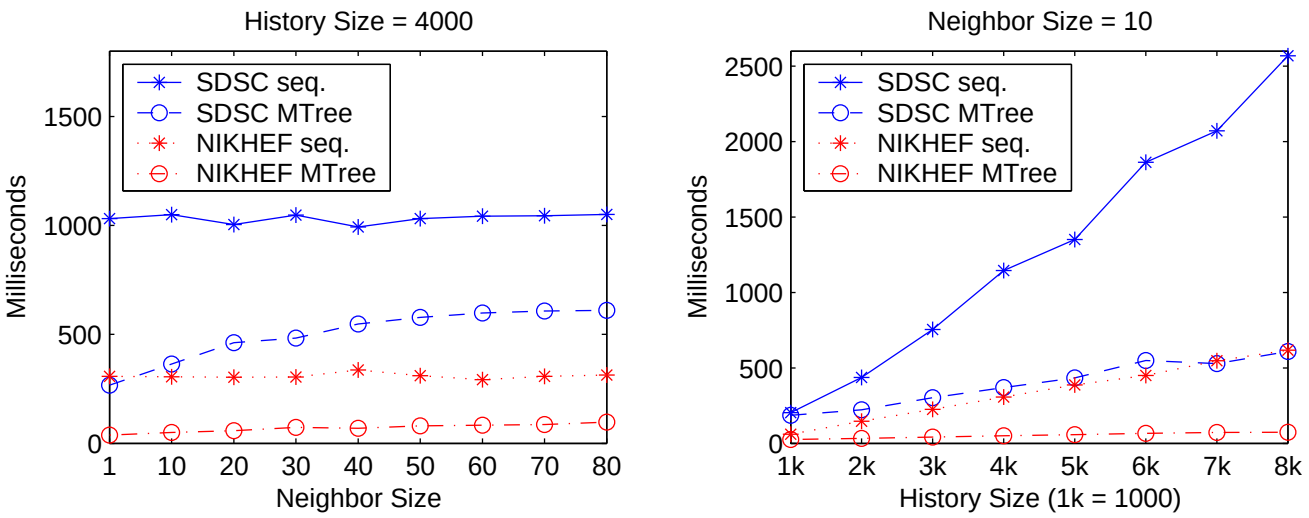

Figure 2. Performance comparisons of the sequential search and the M-Tree nearest neighbor search. Experiments are carried out on a Intel Xeon machine with four 2.8 GHz CPUs and 3 GB shared memory.

that the local models outperform global models for most queues. The adaptive method is clearly seen in action on the period from October to December, 2001 on queue *high*. On the previous generalization set local tuning is superior to global tuning in terms of relative error. However, the adaptive method detects that there is a considerably larger bias by local tuning so it switches to global tuning, resulting in an improved overall accuracy. Nevertheless, there are also periods on queue *express* and *normal* where the strict rules of the adaptive method filter out local tuning even if it does perform slightly better during generalization.

We now look at the results on NIKHEF in Table 2. The overfitting effects by local tuning are clearly observed on groups *lhcb* and *atlas*, for which its overall performance is noticeably worse than global tuning. The benefits of adaptive selection can also be clearly observed, especially on group *atlas*. On group *theory* we can see that the average absolute errors are significantly larger than others. This is mainly due to the unpredictability of this particular group. Lower priority and strict processor throttling policies make the waiting times for jobs from this group very sensitive to current and future coming jobs. Bias becomes the dominant factor in the mean-squared error and locally tuned models with smaller bias will surely be preferable. On the most active group *dzero* local tuning is constantly producing comparable or better results. In its case, large number of training data and high scheduling priority are the reasons why locally optimized parameters are better than global ones.

Table 3 shows the overall generalization errors for the three tuning methods. We can see that local tuning outperforms global tuning, and the adaptive method can further improve the prediction accuracy. Although the overall errors are mostly contributed by those with large values, the benefits of adaptive tuning are evident in the above detailed analysis. Furthermore, the results shown here are from those groups or queues whose number of jobs is large enough for local tuning. For other "smaller" groups global parameters have to be used so in any case adaption is needed for better accuracy. Bias and variance are also shown in the table. We can see that in general our prediction problem is more about variance than bias therefore we should effectively avoid overfitting caused by local tuning. Based on the group-wise analysis of queue wait time predictions, another observation is that different groups/queues have very different scales of errors. For instance, the average absolute errors increase from less than 100 minutes on queue *express* to over 1000 minutes on queue *normal*. If we associate corresponding confidence levels to different queues or groups, it will be more realistic and useful predictions for use by Grid level schedulers.

The execution time per prediction is mainly decided by the search time for nearest neighbors. Figure 2 shows the performance comparisons of the sequential search and the M-Tree search for k nearest neighbor queries. During the experiment we turn on all attributes so the most expensive distance calculations are anticipated. Firstly we investigate how the search times of the two methods scale with the neighbor size given a fixed history size. As we can see in the left figure in 2, the sequential search time remains roughly the same for the same history size while the M-Tree search time increases slowly along with the growing neighbor size. This is because with a fixed data size more or less the same

amount of distance calculations are performed by the sequential search but for M-Tree more comparisons need to be done for bigger neighbor sizes. Secondly, we study how search times scale with the history size given a fixed neighbor size. On the right of figure 2, we can see that as expected the sequential search time increases linearly with the history size. The M-Tree search, however, scales much better than the sequential search because substantially fewer distance comparisons are required in a tree-based structure. Overall the M-Tree search is 2 to 8 times faster than the sequential search. One observation is that queries on SDSC take quite a lot more time than on NIKHEF. This is because the job compositions on SDSC are more diverse than those on NIKHEF. If the policy attributes are all turned on, the resource state attributes would be partitioned into many categories, resulting in a more expensive distance comparison. We also notice that the performance gain on SDSC is not as much as that on NIKHEF. One reason is that on SDSC traces there are many instances with "similar" distances in one node, which results in slower converge time for query. This problem can be solved by using "approximate" search rather than "exact" search, where the full potential of M-Tree can be exploited.

6 Conclusions and Future Work

In this paper we propose and evaluate several methods that improve a local learning technique for queue wait time predictions, both in terms of prediction accuracy and prediction performance. We introduce a pivot attribute that partitions the training set into subsets and tune parameters for each subset. Bias-variance analysis of error is carried out for both global and local tuning and a method is developed to select the tuning type adaptively based on the generalization error and bias-variance decomposition. Experimental results on two real-world traces show that with adaptive tuning the average prediction error can be reduced by 3 to 10 percents compared with the global model. We also employ an efficient tree-based structure called M-Tree that speeds up k nearest neighbor search 2 to 8 times compared with the original sequential search. With these improvements our prediction algorithm based on local learning stands as an effective and practically useful technique in predicting queue wait times on clusters and space-shared supercomputers.

The improvements come with limitations and prices as well. Firstly a good pivot attribute is essential for successful local tuning, in which expert knowledge plays a central role. We are investigating automatic techniques that manipulate the training space, where ensemble methods in machine learning point out an interesting direction. The M-Tree structure excels in search but involves extra costs in maintaining and updating the tree. Therefore more studies are needed for examining its dynamic properties and working on a deployable version. We are developing a toolkit called PDM (Performance Data Miner), which implements all the ideas and algorithms in this paper and is made available in [10]. Finally, future work also includes research on how effective the predictions would be for Grid scheduling in a real environment.

Acknowledgments

We thank Peter v.d. Putten (LIACS) for his many insightful suggestions and discussions on topics in data mining. We are grateful to Jeff Templon (NIKHEF) for his help in experimental studies. We also want to express our gratitude to the Parallel Workload Archive through which the SDSC traces are made publicly available.

References

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

[2] E. Chavez, G. Navarro, R. Baeza-Yates, and J. L. Marroquin. Searching in metric spaces. *ACM Computing Surveys*, 33(3):273–321, 2001.

[3] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB'97)*, pages 426–435. Morgan Kaufmann Publishers, Inc., 1997.

[4] S. Geman, E. Bienenstock, and R. Doursat. Neural networks and the bias/variance dilemma. *Neural Computation*, 4:1–58, 1992.

[5] 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.

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

[7] 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 (CCGrid'04)*, 2004.

[8] H. Li, D. Groep, and L. Wolters. Efficient response time prediction by exploiting application and resource state similarities. In *proceedings of 4th IEEE/ACM International Workshop on Grid Computing (Grid'05)*, 2005.

[9] H. Li, D. Groep, and L. Wolters. Mining performance data for metascheduling decision support in the grid. Technical Report LIACS-2005-07, Leiden Institute of Advanced Computer Science, Leiden University, 2005.

[10] Pdm: A toolkit for mining performance data in the grid. `http://www.liacs.nl/home/hli/pdm/`.

[11] Parallel workload archive. `http://www.cs.huji.ac.il/labs/parallel/workload/`.

[12] The extensible and flexible library. `http://dbs.mathematik.uni-marburg.de/research/projects/xxl/xxl.html`.