

Using Runtime Measurements and Historical Traces for Acquiring Knowledge in Parallel Applications

Luciano José Senger¹, Marcos José Santana², and Regina Helena Carlucci Santana²

¹ Universidade Estadual de Ponta Grossa, Departamento de Informática
Av. Carlos Cavalcanti, 4748 Zip Code 84030-900 Ponta Grossa, PR, Brazil
ljsenger@icmc.usp.br

² Universidade de São Paulo, Instituto de Ciências Matemáticas e de Computação
Av. Trabalhador Saocarlense, 400 PO Box 668 Zip Code 13560-970 São Carlos, SP, Brazil
{mjs, rcs}@icmc.usp.br

Abstract. A new approach for acquiring knowledge of parallel applications regarding resource usage and for searching similarity on workload traces is presented. The main goal is to improve decision making in distributed system software scheduling, towards a better usage of system resources. Resource usage patterns are defined through runtime measurements and a self-organizing neural network architecture, yielding an useful model for classifying parallel applications. By means of an instance-based algorithm, it is produced another model which searches for similarity in workload traces aiming at making predictions about some attribute of a new submitted parallel application, such as run time or memory usage. These models allow effortless knowledge updating at the occurrence of new information. The paper describes these models as well as the results obtained applying these models to acquiring knowledge in both synthetic and real applications traces.

1 Introduction

Advances in hardware and software technology continuously improve performance of parallel systems based on heterogeneous distributed computing. This technological evolution allows heterogeneous systems, which are a group of diverse autonomous computers linked by distinct networks, to support a variety of workloads, including parallel, sequential and interactive jobs [1]. When used to support parallel applications (i.e. computational applications composed of a group of communicating tasks), distributed computers are treated as distributed processing elements (PEs) and compose a distributed memory MIMD (*Multiple Instruction Multiple Data*) parallel architecture [2].

One of the challenges in such systems is to develop scheduling algorithms that assigns the tasks of parallel applications to the heterogeneous machines. Many researchers have demonstrated that using parallel application knowledge may improve the scheduling decisions on multiprogrammed multiprocessor systems [3][4][5][6]. Nevertheless, most of the work has assumed that such knowledge is available *a priori* and does not provide effective indications to obtain it. There are commonly three main sources to obtain knowledge: the description of applications requirements provided by the user (or programmer) who submits the parallel application to the system; historical traces

of all applications executed in a specific system over a time period, and runtime measurements from parallel applications. Among these knowledge sources, historical traces and runtime measurements have demonstrated a great potential to provide information aiming at classifying parallel applications and obtaining knowledge [7][8].

This paper presents two models for knowledge acquisition in parallel applications aiming at improving software scheduling decisions. The first model aims for parallel applications classifying, regarding behavior on resource usage. This model is described in Section 2. The other model presented, described in Section 3, aiming at exploring similarity among applications on workload traces. Compared to previous work, these models have two novel aspects. First, they allow updating of acquired knowledge at the occurrence of new information. Second, they can be used to improve different scheduling policies, not aimed at particular scheduling strategies.

2 Local Knowledge Acquisition from Parallel Applications

The main goal of the model described in this section is the knowledge acquisition from the execution of tasks which compose parallel applications, classifying parallel applications regarding its behavior on resource utilization by classes such as CPU Bound, I/O Bound and Communication Intensive [9]. This model allows the utilization of knowledge acquired in previous runs of the applications and the knowledge updating in presence of new information.

The main idea of the model is the runtime measurements, aiming at identifying the resource usage phases during the parallel application execution. Collected observations are separated by intervals of time T_o . A sample is created for each N_o observations, characterizing an Euclidean vector with dimension d :

$$S^i = [x_1^i \ x_2^i \ x_3^i \ \dots \ x_d^i]^T \quad (1)$$

All samples make up a training set T :

$$T = \{S^1, S^2, \dots, S^n\} \quad (2)$$

The execution samples of the parallel application are grouped in clusters. An important aspect of the model is how to carry out the clustering of the vectors. This clustering can be carried out by statistical techniques, such as the k-means algorithm [10]. Although a good grouping is achieved with this approach, the algorithm needs all training set T before it starts to learn. Another disadvantage is a potentially high execution time and the need for previous definition of the number of clusters to be set. Since an on-line strategy is desired, it is adopted the utilization of self-organizing neural networks [11] as the solution. The ART 2A neural network is used in the knowledge acquisition model, because it allows clustering and classification of a pattern set composed of continuous values (the ART 1 version only accepts binary inputs). Clustering of ART neural networks is not supervised. Learning is incremental (plastic), so that knowledge obtained previously by the model is preserved and updated in the light of new information (stable). Furthermore, input patterns are processed individually rather than in a batch mode. This characteristic is essential to work online with a stream of data [12].

The ART 2A system includes the main components of all ART neural networks, namely the *attentional subsystem*, which contains the input representation field F_1 , the category representation field F_2 , and the *orienting subsystem*, which interacts with the attentional subsystem to carry out the learning process. The two layers (or fields) are linked by both a bottom-up ($F_1 \rightarrow F_2$) adaptive filter and a top-down ($F_2 \rightarrow F_1$) adaptive filter. The path from the i th F_1 neuron to the j th neuron contains a long term memory (LTM), or adaptive weight, z_{ij} . The path from the j th F_2 neuron to the i th F_1 neuron contains a weight z_{ji} . These weights multiply paths signals between fields and are mainly responsible for knowledge storage. The F_2 contains a representation of the input.

The number of clusters, represented by the F_2 neurons, is controlled through the vigilance parameter ρ . Although values from 0 to 1 are allowed, only vigilance values between approximately 0.7 and 1 perform any useful role in controlling the number of clusters [13]. A large vigilance parameter make the ART 2A system to search for new classes in response to small differences between the input feature vector and the LTM traces learning to classify input patterns into a large number of finer categories. Having a small vigilance parameter allows for large differences and more input patterns are classified into the same category.

After ART 2A clustering, a label is assigned for each F_2 neuron. A significance matrix SM [14], which is composed by a set of significance values SV_{ij} , is employed in this step. The significance values are obtained directly from the ART 2A long term memory z : the number of columns in the SM is equal to the number of F_2 neurons (which represent the clusters obtained) and the number of rows is equal to feature vector dimension. The labeling algorithm is built upon the observation that the ART 2A weight vectors resemble as far as possible the corresponding input vector elements of all input elements that are mapped onto a particular cluster. A significance matrix $SM = (SV_{ij})^{7 \times 4}$, for instance, describes a matrix obtained from a neural network which created 4 clusters to represent a parallel application described by a data set with $d = 7$. The significance matrix allows to decide which attributes of the data are significant to characterize each cluster. Initially, the significance values of the attributes are normalized in percentage of the total sum of significance of a cluster. Then, these normalized values are ordering in decreasing order. As significant attributes for cluster description, the attributes in the ordered sequence are taken until the cumulative percentage equals or exceeds a given threshold value (the threshold value used is 51%). This algorithm is performed for all clusters and all attributes and gives for each cluster the set of significant attributes to be used in a meaningful cluster description.

2.1 Model Evaluation

The model was evaluated using an input feature vector with dimension $d = 7$ (Table 1). The input vector contains information related to a parallel program execution: CPU user time; memory usage; read and write file operations and send and receive network operations. Such information is obtained by instrumenting the Linux kernel (kernel version 2.4.5) and reading the /proc pseudo-file system.

The collected features compose the training set T . This set is introduced to ART 2 neural network as the input vector I . The vigilance parameter is set to $\rho = 0.9$ and others parameters are set according to [15]. These settings produced a good clustering of all training set considered.

2.2 Experimental Results

The model was tested on two parallel applications. The first one, namely **A**, is a synthetic application implemented using the master/slave computing model. The slave tasks contain four distinct phases: data receiving, CPU bound computing, I/O bound computing and data sending (to the master). The second one, namely **B**, is the PSTSWM (*Parallel Spectral Transform Shallow Water Model*) [16]. The PSTSWM is a message-passing benchmark code that solves the nonlinear shallow water equations on a rotating sphere using the spectral transform method. This application was developed to evaluate parallel algorithms for the spectral transform method as it is used in global atmospheric circulation models. The PSTSWM is a representative compact application code and currently is included in the ParkBench benchmark suite. All applications are implemented using the PVM message passing system [17] and executed in a Linux computing network (Ethernet 100 Mbits/s).

Table 1. Input feature vector

Attribute	Description
1	CPU time (user)
2	I/O (Bytes read)
3	I/O (Bytes write)
4	Network (TCP bytes read)
5	Network (TCP bytes write)
6	Network (UDP bytes read)
7	Network (UDP bytes write)

The Table 2 shows the obtained results using the model on **A**. The model parameters were $T_o = 200ms$ and $N_o = 1$. This parameterization gave a training set T with size (n) equal to 340 and produced a cluster set equals to 4. The time spent by the parallel application on each phase is given by the accumulate frequency in clusters and the significance matrix allows the cluster interpretation. Thus, it can be observed that the application **A** spent nearly 86% (cluster $k = 2$) of its execution time performing CPU Bound work. The I/O Bound phase is represented by cluster $k = 3$ and accounts for 11% of all application time.

The application **A** had its code modified (**A'**) to observe the plasticity behavior of the model. This modification drastically cut down the CPU bound phase. Table 2 shows the results using the neural network previously trained with the original application (**A**). The model stability and plasticity were observed from the results: clusters previously created were preserved and the internal frequency on each cluster was updated. A change (increase) in relative frequency of cluster $k = 3$ occurred (nearly 74%), due to CPU bound computing reduction. Moreover, a good classification performance of the model was observed, which captures modifications in applications behavior regarding resource usage.

Table 2. Applications **A** and **A'**

Attribute	ART-2A LTM traces Cluster(k)				Significance Matrix			
	1	2	3	4	1	2	3	4
1	0.00	1.00	0.05	0.00	0.00%	100.00%	4.76%	0.00%
2	0.00	0.00	0.00	0.00	0.00%	0.00%	0.00%	0.00%
3	0.00	0.00	1.00	0.00	0.00%	0.00%	95.23%	0.00%
4	1.00	0.00	0.00	0.00	100.00%	0.00%	0.00%	0.00%
5	0.00	0.00	0.00	1.00	0.00%	0.00%	0.00%	100.00%
6	0.00	0.00	0.00	0.00	0.00%	0.00%	0.00%	0.00%
7	0.00	0.00	0.00	0.00	0.00%	0.00%	0.00%	0.00%
Frequency (A)	01.76%	85.58%	11.47%	01.17%				
Frequency (A')	11.11%	05.55%	74.07%	09.25%				

The application **B** was executed with the default configuration provided by the ParkBench suite. This application is more resource consuming than the other previously considered (**A**). Model parameters were $T_o = 1000ms$ and $N_o = 1$ and the results obtained are illustrated in Table 3. The majority of application phases (about 56%) were classified as communication intensive (cluster $k = 1$). The other phases were classified as communication intensive ($k = 2$ and $k = 4$) and CPU-Bound (cluster $k = 3$).

Table 3. Application **B**

Attribute	ART-2A LTM traces Cluster(k)				Significance Matrix			
	1	2	3	4	1	2	3	4
1	0.003	0.12	1.00	0.09	0.30%	10.94%	100.00%	6.26%
2	0.00	0.00	0.00	0.00	0.00%	0.00%	0.00%	0.00%
3	0.00	0.00	0.00	0.00	0.00%	0.00%	0.00%	0.00%
4	1.00	0.00	0.00	0.70	99.70%	0.00%	0.00%	46.87%
5	0.00	0.99	0.00	0.70	0.00%	89.06%	0.00%	46.87%
6	0.00	0.00	0.00	0.00	0.00%	0.00%	0.00%	0.00%
7	0.00	0.00	0.00	0.00	0.00%	0.00%	0.00%	0.00%
Frequency	56.07%	16.58%	14.79%	12.56%				

2.3 System load influence

The application **A** was executed in loaded PEs to observe model stability in presence of external load. This load was generated through a new computer program, called *parasite*, which is composed of a mix of CPU bound operations. These operations increase the CPU utilization on each PE to 100% in a short time interval. A larger granularity of sampling was also considered, with $T_o = 500ms$. Tables 4 and 5 show

the results using respectively 1 and 3 parasite programs. Changes in the external load as well as the larger sampling interval did not depreciate classification performance, when compared to obtained using dedicated PEs (Table 2).

Table 4. Load Influence (1 *parasite* and $T_a = 200ms$)

Cluster(k)	Frequency	(%)
1	12	01.75
2	582	84.47
3	88	12.77
4	7	01.01

Table 5. Load Influence (3 *parasites* and $T_a = 500ms$)

Cluster(k)	Frequency	(%)
1	5	0.90
2	476	85.92
3	71	12.81
4	2	0.36

3 Global Knowledge Acquisition

The local knowledge acquisition model presented in the previous section aiming at knowledge acquiring of a particular application resource usage. Scheduling algorithms often need to know some application attribute in advance of its execution (e.g. *run time*, *memory usage* and *cpu time*). A global knowledge acquisition model which searches for similar applications in a database of experiences and make predictions about some parallel application attribute can be used for this purpose. The workload traces can be treated as a database of previous experiences (*experience base*) about parallel applications execution and depending on the scheduling algorithm requirements, some attribute, like *run time* or *memory usage*, can be considered as the attribute to be predicted. Thus, the knowledge acquisition approach must search for similarity among applications recorded in an experience base and, by some approximation, compute an estimate of some attribute value. Again, the model must allow knowledge updating aiming at adapting to workload changes. The global knowledge acquisition model described in this section is constructed using instance-based learning.

Instance-based learning (IBL) is an approach which find similar instances in an experience base aiming at approximating real-valued or discrete-valued target functions [18][19]. Learning consists of simply storing the presented training data, which are composed of instances and each instance is composed of a set of input and output attributes. Input attributes describe the conditions under which an experience was observed and output attributes describe what happened under those conditions. IBL algorithms compute the similarity between a new query instance and the experience base instances, returning a set of related instances as output. Only the relevant instances are used to classify the query instance. These algorithms can construct a different approximation to the target function for each distinct query instance that must be classified. This has significant advantages when the target function is very complex, but can be described by collection of less complex local approximations.

A common IBL algorithm is the k-Nearest Neighbor Learning. This method assume that all instances in experience base correspond to points in the n -dimensional space \mathcal{R}^n

and a distance function is used to choose the relevant points, close to a query point. The query point nearest neighbors are defined in terms of the standard Euclidean distance.

To approximate a real-valued function $f : \mathbb{R}^n \rightarrow \mathbb{R}$, weighting the contribution of each K relevant neighbors according to their distance to the query point x_q , the following equation can be used:

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^k w_i f(x_i)}{w_i} \quad (3)$$

The weight function w_i should approach a constant value as the distance goes to 0 and should approach 0 as the distance goes to infinity. Locally weighted regression (LWR) is the method used to weight each relevant instance. This method constructs an approximation to f over a local region surrounding x_q , by some function. Although any function can be used, a Gaussian function, centered at the point x_q , with some variance σ_u^2 , is a common choice:

$$K_u(d(x_q, x)) = e^{-\frac{d^2(x_q, x)}{2\sigma_u^2}} \quad (4)$$

where $d^2(x_q, x)$ is the Euclidean distance between x_q and x . The LWR has the advantage of searching for similar parallel applications working with real-valued attributes (such as *number of processing elements used*, *memory used* and *cpu time*) and discrete-valued attributes (such as *user id* or *group id* who submits the application and *scheduling queue number*).

3.1 Model Evaluation

The IBL algorithm using locally weighted regression was implemented in C++ classes and the experience base was implemented through MySQL server tables. Each instance in experience base corresponds to a submitted parallel application and has the same attributes defined in the Feiltelson's workload format ³. Each experience base record has an additional field responsible for temporally storing its distance to query point. The MySQL data indexing mechanism are used to access the experience database faster. All experience base instances are read during the distance computing phase, and the K neighbors of x_q are quickly retrieved using MySQL indices.

The output attribute considered is the parallel application run time, although any attribute can be used as output in our IBL algorithm implementation. Many authors have demonstrated that parallel applications run time knowledge can be very useful to space-sharing scheduling algorithms [20][3]. Parallel application traces recorded in two computing centers, namely SDSC (*San Diego SuperComputer Center*) and CTC (*Cornell Theory Center*), are used to evaluate the IBL model. Only the CTC workload is heterogeneous in the sense that not all 512 PEs are identical, but have distinct amounts of memory. Applications either not complete or with missing attributes are ignored. Details of each workload used, including the system platform, time period, total number of requests and number of ignored applications, are presented in Table 6. Although any feature about parallel application execution can be used as input attribute, the

³ www.cs.huji.ac.il/labs/parallel/workload/

evaluation was restricted to features recorded in workload traces. The most relevant input attributes, selected according to [8], are listed in Table 7 (*user id* and *group id* attributes were not recorded into SDSC95 and SDSC96, and *queue number* and *user id* attributes were not recorded into CTC).

Table 6. Workload traces

Workload	System	Number of PEs	Time Period	Number of Requests	Ignored
SDSC95	Intel Paragon	416	1995	76,872	13,328
SDSC96	Intel Paragon	416	1996	38,719	825
SDSC2000	IBM SP2	128	05/1998 to 04/2000	67,667	11,175
CTC	IBM SP2	512	07/1996 to 05/1997	79,302	16,672

In the experiments, each workload trace was organized through a number of disjoint sets. Each of these sets were partitioned considering 2/3 as experience base and 1/3 for testing, and a partial mean prediction error was computed for each set. This validation strategy is called *holdout*. This strategy was chosen aiming at preserving the data locality in the workloads. The final prediction error for each workload is the mean of these partial errors. Details of each experiment, including the number of applications in experience base, number of applications in test set and the number of disjointed sets (samples) are listed in Table 8. The main IBL algorithm parameters are the neighborhood size K , which defines the number of relevant instances used for computing the estimate, and the σ^2 , which defines the Gaussian function variance and consequently the neighbor weighting values. The values experimented for K were 5, 10, 25 and 50. For σ^2 , the values were 0.125, 0.250, 0.500, 1.000 and 2.000. The combination of these values forms 20 distinct parameters groups. The prediction mean errors of these groups were evaluated using *ANOVA* and compared using the Tukey’s extended test at a 95% confidence interval [21].

Table 7. Workload traces attributes

Attribute	SDSC2000	SDSC95	SDSC96	CTC
number of allocated processors	•	•	•	•
user id	•	•	•	•
group id	•			
executable number	•			•
queue number	•	•	•	

3.2 Experimental Results

Our IBL approach achieves mean prediction errors that are between 50 and 58 percent of mean application run times (Table 8). For all workload traces, the best K value founded was 5 and there was not statistical difference among the prediction errors obtained by the experimented σ values. Nevertheless, it was observed a tendency for more higher prediction errors as the σ^2 value increases.

The Table 9 shows the best IBL model prediction error obtained compared to five previous approaches (the mean absolute error is used in order to compare the results with previous published work). Downey [22] categorizes the applications in a workload using the *queue number* attribute by cumulative distribution functions to predict application run times. He defines two different techniques to produce predictions of run times. The first one uses the median application lifetime and the second one uses the conditional average lifetime. Gibbons [7] uses fixed templates and Smith [8] uses

Table 8. Experiments details

Workload	Experience base size	Test size	Samples	Percentage of Mean Runtime Error
SDSC95	716	369	70	57.46±1.06
SDSC96	421	210	60	55.40±1.18
SDSC2000	621	320	60	50.05±1.59
CTC	688	355	60	50.45±1.29

adaptive templates to make a prediction. The Smith’s adaptive templates are defined by greedy first and genetic algorithm searches. The results obtained by these researchers were extracted from [8] (Smith used the prediction software source code provided by Downey and Gibbons to predict applications run times using SDSC95, SDSC96 and CTC workload traces). The IBL model achieves predictions that are respectively 43.03% and 21.78% better than the best results previously achieved by other researchers using SDSC95 and SDSC96 workload traces. The IBL model achieved a prediction error only 2.18% worse than the best result previously achieved by other researchers using the CTC workload trace.

Table 9. IBL algorithm results compared to previous work

Workload	Downey		Gibbons	Smith		IBL
	Median Lifetime	Average Lifetime	Fixed Templates	Greedy Search	Genetic Search	
SDSC95	82.44	171.00	74.05	67.63	59.65	33.98
SDSC96	102.04	168.24	122.55	76.20	74.56	58.32
CTC	179.46	201.34	124.06	118.05	106.73	109.06

4 Conclusions

This paper presented two models for acquiring knowledge in parallel applications. The aim of these models is to improve scheduling decisions, supporting the scheduler with knowledge about the resource usage patterns of parallel applications. The models can be used in scheduling policies which require on-line and off-line knowledge acquisition in their decisions. Moreover, these models can be used to define workload models, aiming at improving parallel networks and file systems, and to construct other performance models, such as a runtime prediction framework.

Through the experiments, the local knowledge acquisition model presented a good classification performance among all parallel applications considered. Another aspect observed is the robustness of this model at the different computational loads and processing elements configurations [9].

The global knowledge acquisition model presented a great potential to define similarity among parallel applications, weighting the more relevant instances in an experience base to generate an output attribute estimate. Through the experiments, the IBL approach achieved a good prediction performance when compared to both static and adaptive templates prediction approaches. Although the IBL algorithm was not used in workloads originated from highly heterogeneous systems, it can be observed a great potential to be explored, through distinct distance functions which can be used to

represent the computing power difference among distinct processing elements. Another possibility is the utilization of different output attributes such as memory usage or CPU usage, depending on scheduling algorithm requirements. There is work in progress aiming at supporting these models to scheduling software, as well as making suitable the IBL algorithm to highly heterogeneous systems.

5 Acknowledgments

This project is supported by CAPES/PICDT program. The authors would like to thank Reagan Moore, Allen Downey, Victor Hazelwood (San Diego Supercomputer Center) and the Cornell Theory Center for graciously providing the workload traces used in this work. Particular thanks to Warren Smith, for providing his prediction software source code.

References

1. Anderson, T., Culler, D., Patterson, D.: A Case for NOW (Networks of Workstations). *IEEE Micro* **15** (1995) 54–64
2. Flynn, M.J., Rudd, K.W.: Parallel architectures. *ACM Computing Surveys* **28** (1996) 68–70
3. Harchol-Balter, M., Downey, A.B.: Exploiting Process Lifetimes Distributions for Dynamic Load Balancing. *ACM Transactions on Computer Systems* **15** (1997) 253–285
4. Brecht, T., Guha, K.: Using Parallel Program Characteristics in Dynamic Processor Allocation Policies. *Performance Evaluation* **27/28** (1996) 519–539
5. Naik, V.K., Setia, S.K., Squillante, M.S.: Processor Allocation in Multiprogrammed Distributed-memory Parallel Computer Systems. *Journal of Parallel and Distributed Computing* **47** (1997) 28–47
6. Sevcik, K.C.: Characterizations of Parallelism in Applications and their use in Scheduling. *Performance Evaluation Review* **17** (1989) 171–180
7. Gibbons, R.: A Historical Application Profiler for Use by Parallel Schedulers. In: *Job Scheduling Strategies for Parallel Processing*. Springer Verlag (1997) 58–77
8. Smith, W., Foster, I.T., Taylor, V.E.: Predicting Application Run Times Using Historical Information. In: *JSSPP*. (1998) 122–142
9. Senger, L.J., Santana, M.J., Santana, R.H.C.: A new approach fo acquiring knowledge of resource usage in parallel applications. In: *Proceedings of International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS'2003)*. (2003) 607–614
10. Devarakonda, M.V., Iyer, R.K.: Predictability of Process Resource Usage: A Measurement-based Study on UNIX. *IEEE Transactions on Software Engineering* **15** (1989) 1579–1586
11. de P. Braga, A., Ludermir, T.B., Carvalho, A.C.P.L.F.: *Redes Neurais Artificiais: Teoria e Aplicações*. LTC (2000)
12. Whiteley, J.R., Davis, J.F.: Observations and problems applying ART2 for dynamic sensor pattern interpretation. *IEEE Transactions on Systems, Man and Cybernetics-Part A: Systems and Humans* **26** (1996) 423–437
13. Fausett, L.: *Fundamentals of Neural Networks*. Prentice Hall (1994)
14. Ultsch, A.: Self-organising neural networks for monitoring and knowledge acquisition of a chemical process. In: *Proceedings of ICANN-93*. (1993) 864–867
15. Carpenter, G.A., Grossberg, S., Rosen, D.B.: ART 2-A: An Adaptive Resonance Algorithm for Rapid Category Learning and Recognition. *Neural Networks* **4** (1991) 494–504

16. Foster, I.T., Worley, P.H.: Parallel algorithms for the spectral transform method. *SIAM J. Sci. Stat. Comput.* **3** (1997) 806–837
17. Beguelin, A., Gueist, A., Dongarra, J., Jiang, W., Manchek, R., Sunderam, V.: *PVM: Parallel Virtual Machine: User's Guide and tutorial for Networked Parallel Computing*. MIT Press (1994)
18. Aha, D.W., Kibler, D., Albert, M.K.: Instance-based learning algorithms. *Machine Learning* (1991) 37–66
19. Mitchell, T.M.: *Machine Learning*. McGraw-Hill (1997)
20. Feitelson, D.G., Rudolph, L., Schwiegelshohn, U., Sevcik, K.C., Wong, P.: Theory and Practice in Parallel Job Scheduling. In: *Job Scheduling Strategies for Parallel Processing*. Volume 1291. Springer Verlag (1997) 1–34 *Lect. Notes Comput. Sci.* vol. 1291.
21. W.C.Shefler: *Statistics: Concepts and Applications*. The Benjamin/Cummings (1988)
22. Downey, A.: Predicting queue times on space-sharing parallel computers. In: *Proceedings of International Parallel Processing Symposium*. (1997)