Characterizing Process Execution Behavior Using Machine Learning Techniques

Kishore Kumar P. Atul Negi*
Department Of Computer and Information Sciences
University of Hyderabad, Hyderabad-500046, India
atulcs@uohyd.ernet.in, kishoregupta_os@yahoo.com
*all correspondence to atulcs@uohyd.ernet.in

Abstract

A deep knowledge of process execution behavior is very useful in formulating better scheduling techniques. In this paper we study the approaches to the process characterization problem. Also we propose a novel method to characterize and categorize process execution behaviors by using machine learning techniques that learn from previous execution instances of programs. The "Waikato Environment for Knowledge Analysis (Weka)", an open source machine-learning tool is used for training and performance evaluation. We hope this study shall ultimately help in the design of an advanced knowledge-based process scheduler. Finally we propose an architecture that may improve the PBS scheduler in grid systems.

Keywords: Process Scheduling, Process Characterization, Machine Learning.

1. Introduction

In high performance environments, a request to execute a program is not serviced immediately but instead serviced only when resources are available. Many grid applications have varying resource requirements and the problem with these applications is that current scheduling systems (as in PBS scheduling for grids) [1] rely on user's estimates of the resource requirements. Scheduling in such systems could benefit by scheduling of resources using the knowledge of previous process execution behavior. Thus there is a need to *characterize* the process execution behavior and resource utilization (like total CPU time). Such knowledge could help to improve the scheduling policy, guide the resource selection and balance the workload distribution. In this paper we present an approach to characterize the process execution behavior and thereby predict the required resources for processes using data from previous executions of programs.

Our approach to characterize process execution behavior combines the program intrinsic behavior and also the execution history of the programs [2]. The success of prediction based on process characterization and categorization hinges on a characterization method, which

determines similarity [3] between two processes. Mainly we use two broad categories within which we compare programs and find similarities. These are the *interactive* and *non-interactive* classes of programs. Since we would like to study the program characterization in general, we include the interactive class of programs also although the practical utility of such programs in batch systems may not be meaningful. In each category, we run the programs with different input sizes [4]. By using these run traces, we collect data such as system time, user time etc., and prepare a knowledge base. Applying machine-learning techniques on this knowledge base, we extract the most effective parameters that can characterize the process execution behavior [6]. These isolated parameters were used to predict the resource requirements of similar process.

The remainder of this paper is organized as follows: first, in section 2, we describe the related work, section 3 describes the detailed methodology and section 4 shows experimental results and our proposed model to improve the PBS scheduler.

2.Related Work

In the paper by Smith et al. [3], the authors use genetic algorithms for identifying "good templates" for a particular workload. They use Genetic Algorithm (GA) search techniques to determine those application characteristics that yield the best definition of similarity for making predictions. GAs are probabilistic techniques and are well known for exploring large search spaces. However from a machine learning perspective, GAs are expensive in terms of computation and also their results are considered fragile. That is, essentially due to their probabilistic nature, it is not clear how well the end results from GAs can generalize to data outside the training set. In the thesis [5], the author used Statistical Regression methods for prediction. However regression methods have limitations; since they work well for numeric data and cannot readily be used for nominal data. Also in this work, we note that the template for defining similarity between applications was "predefined" and not arrived at by some criterion.

The work [7] by Suranauwarat and Taniguchi, presents an approach to remember the previous execution behavior of certain well-known programs. They study the process times of these programs in various states. The knowledge of the *program flow sequence* (PFS) is used to extend the CPU time slice of a process. PFS of a process is computed from its past execution history. PFS characterizes the process execution behavior and is used to decide whether the program executing currently needs additional time. They set a threshold T_m called as maximum

dispatch delay time, which determines the time limit for context switching; and is a multiple of the delay for minimum process switching time. They control the CPU time of a process T_p, by either reducing or increasing a scaling feature. The information for computing the PFS and T_p adjustment thresholds are based on observations from the 4 phases:

1) when a process is created,
2) when a process uses up its time-slice,
3) when a process blocks itself, and
4) when a process terminates.

They conclude from experimental results that overall processing time is reduced for known programs. To schedule a process they search for its name in the PFS knowledge base and thus improve its behavior.

From this brief sample of literature, we see that it is really very difficult to derive clear directions as to the specific technique used to learn or characterize program behavior. In our study we use varied machine learning techniques like *Decision Trees*, *K-NN* and *Decision Tables* for finding robust and accurate predictions. The techniques we use are more generic and efficient. Another important difference in our work is that we did not rely merely on a label as done previously [7], but we try to go deeper and analyze dynamic process attributes in predicting the process execution behavior. We find that prediction of the total execution time (and process execution behavior) of a process is possible to a large extent.

3. Experimental Methodology

In our study to make for a larger variety of programs to be studied (merely for the purpose of prediction), we divide programs to be either *interactive* or *non-interactive* programs. For the non- interactive programs, we have taken as representative examples: gunzip, matrix multiplication, sorting and searching programs and for the interactive programs, we have taken the examples, acroread and ghostview, which are Linux benchmark programs [4]. The experimental procedure is divided into two phases. In the first phase, we create the data set from the above program's run traces. In the second phase, we train and classify this data according to the fitness function, “total execution time” by using machine-learning techniques. While other fitness functions could have been used we considered this to be the most important for scheduling performance.

Since any machine learning technique requires fully labeled (categorized) data for

training we categorized our data into twenty classes and trained the classifiers [8]. The trained classifiers are then used on a different data set called as a *test data set*. Often due to limitations in size of the data, we train classifiers using a *leave-one-out technique* [9] which is a standard technique.

3.1 Creating the Dataset

To characterize the process execution behavior, we need to find the attributes (the characteristics or parameters), which most affect the prediction of the process execution behavior (here we take *total execution time*). So In this first phase, we used *gprof, readelf and size* commands to get the following attributes as shown in Table 1.

Attribute	**Definition (the values of the attributes are in terms of bytes)**
. bss	This section holds uninitialized data that contribute to the program's memory image.
. data	This section holds initialized data that contribute to the program's memory Image.
. comment	This section holds version control information.
. debug	This section holds information for symbolic debugging.
. dynamic	This section holds dynamic linking information
. dynstr	This section holds strings needed for dynamic linking, most commonly the strings that represent the names associated with symbol table entries.
.dynsym	This section holds the dynamic linking symbol table.
. fini	This section holds executable instructions that contribute to the process termination code.
.got	This section holds the global offset table.
.hash	This section holds a symbol hash table.
.init	This section holds executable instructions that contribute to the process initialization code.
.interp	This section holds the path name of a program interpreter.
.line	This section holds line number information for symbolic for symbolic debugging, which describes the correspondence between the source program and the machine code.
.plt	This section holds the procedure linkage table.
.rodata	This section holds read-only data that typically contribute to a non-writable segment in the process image.
.text	This section holds the "text" or "executable instructions", of a program.
.strtab	This section holds strings, most commonly the strings that represent the names associated with symbol table entries.

Table 1. Attribute Template for Computing Similarity Between Processes

In addition, we use three other attributes: *input size* (in Kbytes), *number of page reclaims* and *number of page faults*. The last two and the *user_time* and *system_time* are found using the system calls *getrusage* [8], *profil* [10] and *times* [11]. Totally we collected 24 attributes (characteristics) and we use these to predict the attribute *total_execution_time* (user_time+system_time). We have taken the (non-interactive) programs: *gzip, gunzip* and two

other test programs with loops and array operations and collected the sets of training examples of sizes 400 and 800 for the next phase (as shown in Tables 3 and 4).

3.1.1 An Example of A Training Example (or Instance)

The following is an example of data gathered for "gunzip" instance (Table 2):

Attribute	**The value**
Total_execution time	Class t3 (20001—30000 microseconds)
Input size	1000kilo bytes
Minimum Page reclaims	36 bytes
Maximum Page faults	83 bytes
.dynsym	912 bytes
.dynstr	525 bytes
.gnu.version	114 bytes
.gnu.version_r	80 bytes
.rel.dyn	48 bytes
.rel.plt	384 bytes
.init	23 bytes
.plt	784 bytes
.text	38804 bytes
.fini	27 bytes
.rodata	5700 bytes
.dynamic	200 bytes
.ctors	8 bytes
.dtors	8 bytes
.dynbss	20 bytes
.bss	333372 bytes
.got	208 bytes
.jcr	4 bytes
.data	2784 bytes
.eh_frame	92 bytes
.eh_frame_hdr	28 bytes
Total	386160 bytes

Table 2. Template Data for the program "gunzip"

We used the same method for data collection as shown in the example for gunzip program. Data was collected and made into 20 categories based on the attribute *total execution time,* with each class having an interval of 10000 microseconds. This data set was gathered for a system with GNU/Linux on Intelx86 platform. Following the methodology shown in [4], we collect the data for the acororead and ghostview programs. That is, these two programs are interactive programs, so we used the run traces of these programs when displaying successive pages of the pdf and ps files.

3.2. The Machine Learning Techniques

In this section we bring in some background information about the machine learning techniques so that our work may be better appreciated, readers familiar with this material could skip ahead to section 3.3.

3.2.1. The Important of Machine Learning

The field of machine learning (ML) is concerned with the question of how to construct computer programs that automatically improve with experience. Machine Learning Theory [8] attempts to answer questions such as *"How does learning performance vary with the number of training examples presented" and "which algorithms are most appropriate for various types of learning tasks?"* Overtime several ML techniques have been proposed in the literature. In recent years many successful machine-learning applications have been developed, ranging from data-mining methods that learn detection of fraudulent credit card transactions, to information-filtering systems that learn user's reading preferences etc.

3.2.2. The Learning Algorithms (Classifiers)

For our experiments, we selected a representative set of standard machine learning algorithms with different model classes. All of these are available in the "Waikato Environment for Knowledge Analysis (Weka) [6], an open source machine learning tool. The selected learner types (classifier types) were: "*Trees, Lazy, Rules,* which are listed below:

3.2.2.1. Trees

C4.5 (or J48 in Weka [6]) is a Reduced-Pruned Decision Tree Learner. Decision Tree learning is one of the most widely used and practical methods for inductive inference. It is a method for approximating discrete-valued functions, and is robust to noisy data and capable of learning disjunctive expressions.

3.2.2.2. Lazy (or Instance Based Learner)

In contrast to learning methods that construct a general, explicit description of the target function when training examples are provided, instance-based learning methods [6] simply store the training examples. Generalizing beyond these examples is performed until a new instance must be classified. Each time a new query instance is encountered, its relationships to the previously stored examples is examined in order to assign a target function value for the new instance. In this category, we use the IB family.

IB k Learner (K-NN):

This assumes all instances are points in an *n*-dimensional space. The nearest neighbors (NN) of an instance are defined in terms of the standard Euclidean Distance. The class label of a new instance is found from the K instances nearest to it by assigning to it the majority label of the K-NN [6].

3.2.2.3. Rules

One of the most expressive and human readable representations for learned hypotheses is a set of *if-then* rules. In this category, we studied the following:

Decision Table Learner:

This produces rules formatted as a table from a selected set of attributes (following a wrapper-type feature selection prior to the training phase).

3.3. Training and Testing Methodology

We performed two types of tests on the training examples with all the above learners on the data sets collected in the first phase. The tests are:

Use Training Set: The classifier is evaluated on how well it predicts the class of the instance it was trained on.

Cross-Validation: The classifier is evaluated by cross-validation, using the number of folds that are entered in the *Folds* text field (Weka). Recognition accuracy was tested via cross-validation. In this test, the training examples are divided into 10 parts and the classifier classifies by taking one part as a test set and other 9 parts as training set. Likewise, we continue for all parts.

4. Experimental Results

The results of the "Cross-Validation" and "Use Training Set" experiments are summarized in the following tables, which give the percentage of correct prediction achieved by the individual classifiers on different sizes of the set of training examples.

Classifier Type	**Classifier**	**%Classification 10 Fold Cross-Validation**		**%Classification Use Training Set**	
		Non-Interactive Programs	**Interactive Programs**	**Non-Interactive Programs**	**Interactive Programs**
Trees	C4.5	94.8670	92.6532	95.2135	98.0769
Lazy	IB K (K-NN)	94.9550	94.2308	95.3153	98.0769
Rules	Decision Table	91.53	91.4320	95.5	99.1250

Table-3: Classification for 400 Training Examples (Instances)

Classifier Type	Classifier	%Classification 10 Fold Cross-Validation		%Classification Use Training Set	
		Non-Interactive programs	Interactive Programs	Non-Interactive Programs	Interactive Programs
Trees	C4.5	96.8563	93.0230	96.8563	98.1459
Lazy	IB K(K-NN)	96.6250	94.2430	96.7813	98.1459
Rules	Decision Table	95.8745	92.2640	96.3135	99.6750

Table-4 Classification For 800 Training Examples (Instances)

Looking at the tables (3-4) in terms of learning algorithms first, we observe that all algorithms are giving significant recognition rates (*%prediction*). Some learners may be better suited to this task than others and the overall prediction accuracy varies between 91.4% and 99.7%. A closer look on the above result is very encouraging – most results are significantly near to the perfect recognition rate (prediction) of 100%. The results also show that the prediction is better with the larger number of training instances. The following charts show the above results.

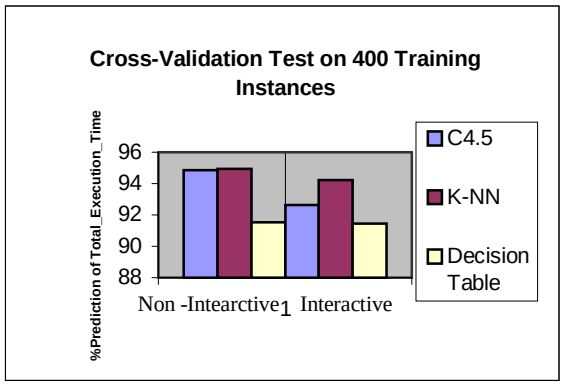


Chart (a)

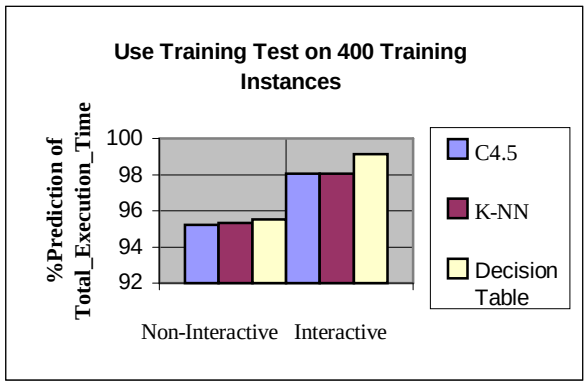
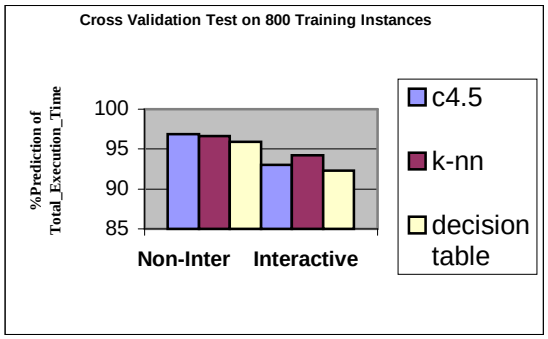


Chart (b)



Chart(c)

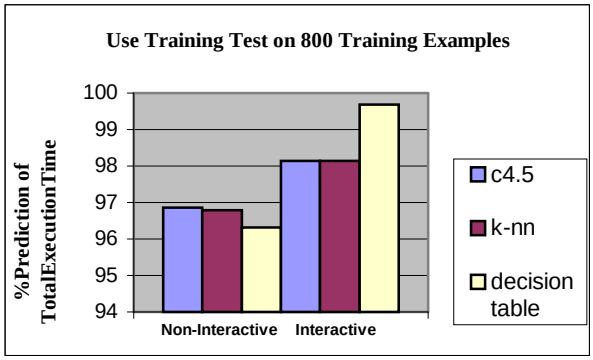


Chart (d)

4.3.1. Extracting the Best Attributes

For this we used "Waikato Environment for Knowledge Analysis (Weka) as follows**:** Attributes selection involves searching through all possible combinations of attributes in the data to find which subset of attributes works best for predicting the program execution behavior. Here our goal is to predict "the total execution time (this is one aspect of characterizing the process execution behavior)". To do this, two objects must be set up; "an attribute evaluator "and "a search method". The evaluator determines what method is used to assign a value (weight or worth) to each subset of attributes. The search method determines what style of search is performed. The attribute selection mode has two options:

Use full training set: The worth of the attribute subset is determined using the full set of training data.

Cross-validation: a process of cross-validation determines the worth of the attributes subset. The Fold and Seed field set the number of folds to use and the random seed used when shuffling the data.

We applied the search methods such as *Genetic Search* [6], *Best First Search* [6] and *Rank Search* [6] and the evaluation methods such as *CfsSubsetEval* [6] and *Consistency SubsetEval* for both the above attribute selection mode options. We found the below attributes as "good attributes", that is those attributes which effectively characterize the process execution behavior.

From the two searches *genetic* and *bestfirst* and the above two evaluation methods, the best attribute found is *input size* and *page reclaims* is the second best. Other good attributes are *.text,. data, .rel.plt, .dynamic,page faults* and *.plt*.

By applying *rank search* [6] with *CfsSubsetEval* [6] the ranking of the above good attributes is summarized in the below table.

Attribute	Input size	Page Reclaims	Page faults	.dynamic	.text	.data	.rel.plt	.plt
Rank	1	2	3	4	5	6	7	8

Table 4: The Effectiveness of attributes and their Ranks

4.4. Improved PBS Scheduler**:**

From our study above now we attempt to propose a scheme for PBS scheduling. In the PBS scheduling of grid systems [1], users need to give the resource requirements along with the job for execution. Then the PBS executes them in the future, according to the site policy, to best satisfy the site production needs.

As shown in our proposed model, which is shown in Figure 1, the advanced knowledge module predicts the resources required by the submitted job using machine-learning techniques. We maintain a knowledge base of the previous program run traces (execution behaviors). The advanced knowledge module updates the knowledge base when a new program comes. The advanced resource manager takes the information from the advanced knowledge module and compares the current system resources with the predicted resources of the submitted program. If the resources are available, it instructs the local process scheduler to schedule the submitted job.

This proposed model may remove the burden of the users when they are submitting the jobs to PBS, since the advanced resource manager would predict their job' s resource requirements. Optionally, the predictions about resource requirements are used as suggestions by the users who may modify their job submissions, to give an improved performance. Hence our proposed model of advanced knowledge based PBS may improve the scheduling process of grid systems.

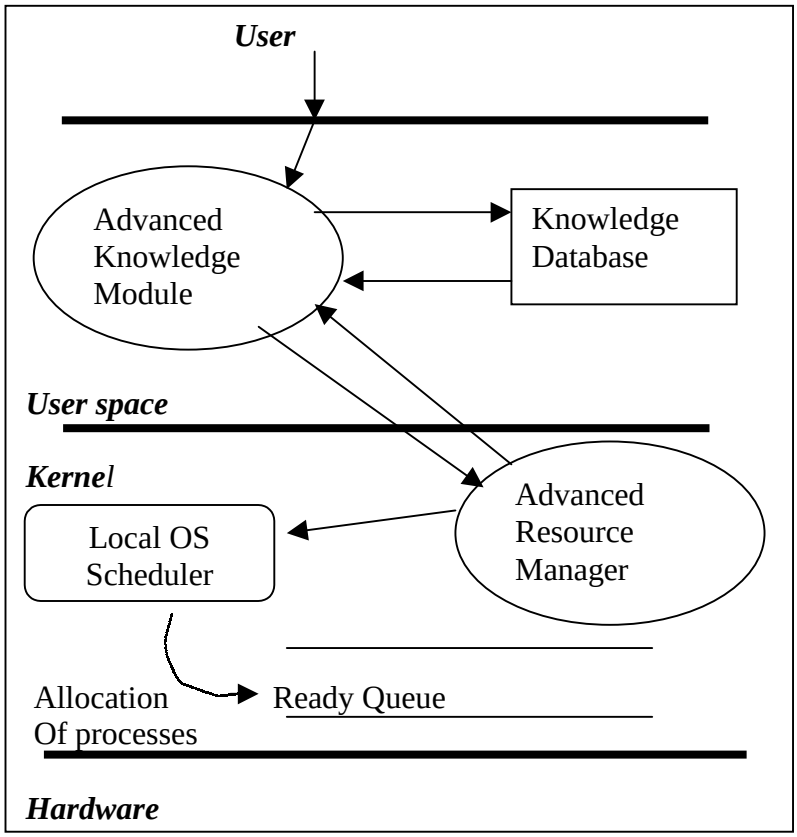


Figure 1: Advanced Knowledge Based PBS Scheduler

5. Conclusions and Future Work

This paper has presented experimental results to show that machine learning techniques can successfully characterize process execution behavior. The machine learning algorithms achieved good prediction (91.4%-99.7%), which indicates that when suitable attributes are used, that a certain amount of predictability does exist for known programs. A more comprehensive study of High Performance Application characteristics and execution behavior is being conducted on an 8-node Linux cluster. We are also exploring the portability of the characteristic set of parameters for parallel applications on this cluster. A more detailed study on AIX UNIX systems with different processors is also underway.

Acknowledgments: We would like to thank Sasi Kanth.A and Dr.C.R.Rao for their valuable suggestions. We also thank the referees for improving the paper with their most useful comments.

References

[1] Albeaus Bayacan, Portable Batch System, *Administrator Guide, NASA Ames Research Center, Release: 1.1.12,* August 7, 1998.

[2] F.Vraalsen, R.A. Aydt, C.L. Mendes and D.A. Reed, "Performance Contracts: Predicting and monitoring grid application behavior", in Lee, Craig A. (Ed.), *Grid Computing – Grid 2001: Proceedings of the 2nd International Workshop on Grid computing*, Denver, CO, USA, November 12, 2001, *LNCS 2242: Lecture Notes in Computer Science*, Springer, 2001

[3] Warren Smith, Valerie Taylor, Ian Foster, "Predicting Application Run-Times Using Historical Information", *In Job Scheduling Strategies for Parallel Processing, IPPS/SPDP'98 Workshop*, Orlando, Florida, USA, March 30, 1998 *Lecture Notes in Computer Science, LNCS-1459: 122 – 142,* Springer 1998.

[4] Krisztian Flaunter, Rich Uhlig, Steve Reinhardt, Trevor Mudge, "Thread Level Parallelism and Interactive Performance of Desktop Applications", *Architectural Support for Programming Languages and Operating Systems, ASPLOS-IX proceedings. Pp.129-138, Cambridge, MA ,USA.*, Dec.2000

[5] Hui Li. Master's Thesis. Predicting Job Start Times on Clusters. *Internal Report 03-11, Leiden Institute of Advanced Computer Science, Leiden University,* 2003.

[6] Garner, S.R., "WEKA: The Waikato Environment for Knowledge Analysis". *In Proc. of the New Zealand Computer Science Research Students Conference, pp 57-64.,* 1995.

[7] Surkanya Suranauwarat and Hide Taniguchi, "The Design, Implementation and Initial Evaluation of An Advanced Knowledge -based Process Scheduler", *ACM SIGOPS Operating Systems Review, Volume 35, pp. 61-81. Issue 4,* October 2001.

[8] Tom Mitchell, *Machine Learning., pp.: 55-56,230-244 and 274-304.,* The Mc-Graw Hill Company.Inc.International Edition, 1997.

[9] R.O.Duda and P.E.Hart, *Pattern Classification, Wiley, NewYork NY.* 1973.

[10] Maurice J Bach, *The Design of Unix Operating System*, Prentice Hall, *pp. 260-266,* 1983.

[11] Richard Stevens, *Advanced Programming in the Unix Environment*, Pearson Education, *pp. 232-235,* 2004 (Reprint*).*