December 1987

UILU-ENG-87-2273
CSG-77

COORDINATED SCIENCE LABORATORY
College of Engineering

Langley GRANT
IN-61-CR
111915
P. 32

PREDICTABILITY OF PROCESS RESOURCE USAGE: A MEASUREMENT-BASED STUDY OF UNIX

Murthy V. Devarakonda
Ravishankar K. Iyer

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION
Unclassified

1b. RESTRICTIVE MARKINGS
None

2a. SECURITY CLASSIFICATION AUTHORITY

2b. DECLASSIFICATION / DOWNGRADING SCHEDULE

3. DISTRIBUTION / AVAILABILITY OF REPORT
Approved for public release;
distribution unlimited

4. PERFORMING ORGANIZATION REPORT NUMBER(S)
UILU-ENG-87-2273 (CSG 77)

5. MONITORING ORGANIZATION REPORT NUMBER(S)

6a. NAME OF PERFORMING ORGANIZATION
Coordinated Science Lab
University of Illinois

6b. OFFICE SYMBOL *(If applicable)*
N/A

7a. NAME OF MONITORING ORGANIZATION
AT&T NASA

6c. ADDRESS *(City, State, and ZIP Code)*
1101 W. Springfield Avenue
Urbana, IL 61801

7b. ADDRESS *(City, State, and ZIP Code)*
AT&T NASA Langley Research Center
Info Systems Hampton, VA 23665
Middletown, N.J. 07738

8a. NAME OF FUNDING / SPONSORING ORGANIZATION
AT&T NASA

8b. OFFICE SYMBOL *(If applicable)*

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
1-5-37411 NASA-NAG-1-613

8c. ADDRESS *(City, State, and ZIP Code)*
See 7b.

10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO.

11. TITLE *(Include Security Classification)*
Predictability of Process Resource Usage: A Measurement-Based Study of Unix

12. PERSONAL AUTHOR(S)
Devarakonda, M. V. and Iyer, R. K.

13a. TYPE OF REPORT
Technical

13b. TIME COVERED
FROM ________ TO ________

14. DATE OF REPORT *(Year, Month, Day)*
1987 December

15. PAGE COUNT
31

16. SUPPLEMENTARY NOTATION

17. COSATI CODES

FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
UNIX, measurement, process resource usage, statistical clustering, probabilistic prediction, resource usage model

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

In this paper, a probabilistic scheme was developed to predict process resource usage in UNIX. Given the identity of the program being run, the scheme predicts CPU time, file I/O, and memory requirements of a process at the beginning of its life. The scheme uses a state-transition model of the program's resource usage in its past executions for prediction. The states of the model are the resource regions obtained from an off-line cluster analysis of processes run on the system. The proposed method is shown to work on data collected from a VAX 11/780 running 4.3 BSD UNIX. The results show that the predicted values correlate well with the actual. The coefficient of correlation between the predicted and actual values of CPU time is 0.84. Errors in prediction are mostly small. About 82% of errors in CPU time prediction are less than 0.5 standard deviations of process CPU time.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS

21. ABSTRACT SECURITY CLASSIFICATION
Unclassified

22a. NAME OF RESPONSIBLE INDIVIDUAL

22b. TELEPHONE *(Include Area Code)*

22c. OFFICE SYMBOL

83 APR edition may be used until exhausted.
All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

PREDICTABILITY OF PROCESS RESOURCE USAGE: A MEASUREMENT-BASED STUDY OF UNIX®

Murthy V. Devarakonda
Ravishankar K. Iyer

November, 1987

Computer Systems Group
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
1101 W. Springfield Avenue
Urbana, IL 61801

ABSTRACT

In this paper, a probabilistic scheme was developed to predict process resource usage in UNIX. Given the identity of the program being run, the scheme predicts CPU time, file I/O, and memory requirements of a process at the beginning of its life. The scheme uses a state-transition model of the program's resource usage in its past executions for prediction. The states of the model are the resource regions obtained from an off-line cluster analysis of processes run on the system. The proposed method is shown to work on data collected from a VAX 11/780 running 4.3 BSD UNIX. The results show that the predicted values correlate well with the actual. The coefficient of correlation between the predicted and actual values of CPU time is 0.84. Errors in prediction are mostly small. About 82% of errors in CPU time prediction are less than 0.5 standard deviations of process CPU time.

1. Introduction

The study reported in this paper addresses two questions: Is it possible to predict resource requirements of a process? And if so, how well can we predict the requirements? Resource usage prediction can be a sound basis for load balancing in a distributed computer system, because costs associated with frequent load information exchange or process migration can be avoided. An additional motivation is in the area of reliable distributed computing— knowledge of resource commitments can be valuable in reorganization of a system under failure.

To our knowledge, there are no empirical studies that predict process resource usage using statistical methods. One relevant study is [Zhou 86b], which concluded that system load cannot be predicted based on load indices. The study, however, does not address predictability of process resource requirements.

Here, we develop a probabilistic scheme for predicting CPU time, file I/O, and memory requirements of a process at the beginning of its life, given the identity of the program being run. The scheme consists of building a state-transition model for each program to represent resource usage of the program in its previous executions, and a procedure for computing resource requirements for the next execution of the program based on this state-transition model. An off-line statistical clustering procedure is used to identify the resource regions where processes are likely to occur. These resource regions are the states of the state-transition model. The prediction scheme is shown to work using process resource usage data that was collected from a VAX® 11/780 running 4.3 BSD UNIX® [Berkeley UNIX 86].

VAX is a registered trademark of Digital Equipment Corporation.
UNIX is a registered trademark of AT&T Bell Laboratories.

We quantified the quality of prediction in two ways: First, statistical correlation between the predicted and actual values are shown. Next, distributions of errors in prediction are plotted and characteristics of these distributions are discussed.

The results of our experiments show that the coefficient of correlation between predicted CPU time requirements and the actual values is 0.84. A perfect prediction would give a result of 1.0. The distributions of prediction errors are heavily skewed towards small values. That is, although there are a few large errors, most errors are small. For example, 82% of errors in CPU time prediction are less than 0.5 standard deviations. When contrasted with the large variability in process CPU times (the difference between 99 and 1 percentiles is about 18 standard deviations), the results are clearly good.

The organization of the remainder of the paper is as follows: Section 2 discusses previous work related to this study. Section 3 describes basic statistics of process resource usage in the measured system. Section 4 describes resource usage modeling. Section 5 describes the prediction scheme in detail and provides error statistics. Section 6 examines issues such as the influence of varying the amount of past used in prediction on prediction error. Section 7 summarizes the paper.

2. Background

In this section, we discuss desirability of resource usage prediction for load balancing purposes. We do that by comparing the resource usage prediction with other empirically observed, process or system, behavior as a basis for load balancing. Many load balancing algorithms have been proposed (for example, [Hwang 82; Bryant and Finkel 81]) and many more simulation studies have been made [Eager 86;

Barak and Litman 85; Wang and Moris 85]. But, only two measurement-based load balancing schemes have appeared so far.

The first of such load balancing schemes [Leland and Ott 85] proposes a heuristic algorithm based on an empirically observed linear relationship between the residual CPU time of a process and its age. The heuristic approximates to a *spiral assignment* of processes. Assuming that the processes are ordered by the their age, the spiral assignment assigns process i to processor $i \ mod \ N$, where N is the total number of processors. Although average residual CPU time requirements of processes can be predicted based on age (as the authors claim), such a prediction may not hold for a single process.

The second load balancing scheme [Zhou 86a] is actually a family of algorithms that gather or propagate (depending on whether the algorithm is centralized or decentralized) load information about a distributed system, and use that information to assign a new job to a processor in such a way that it reduces process response time. In a related study [Zhou 86b], Zhou also showed that process response time strongly depends on processor load, and that the CPU and I/O queue lengths are good indicators of the load.

Using trace-driven simulations, these load balancing schemes were shown to reduce process response times. But, the improvements are sub-optimal. Leland and Ott's load balancing algorithm performs poorly even without process migration. Zhou's algorithms rely on rapid and regular propagation of the global system status to all processors. Since costs associated with frequent exchanges of load information or process migration can be substantial, proper initial placement of processes based on predicted resource requirements of the processes is particularly attractive.

In [Zhou 86b], Zhou considered load indices as predictors of future system load, and he concluded that the future system load cannot be predicted based on the load indices. However, neither he nor any other measurement-based study ever addressed predictability of process resource requirements. This study proposes a probabilistic scheme to predict process resource requirements and shows that the scheme works on a trace collected from a production system.

3. Basic Statistics

In this section, we discuss distributions of process resource usage and inter-arrival times. These statistics characterize the measured system and bring out the variability in process resource usage; the latter shows the inherent difficulty in predicting the process resource usage.

Figures 3.1 through 3.4 show the cumulative distributions of process CPU time, file I/O, memory usage and inter-arrival times. Most processes used only a small amount of CPU time (median 0.24 seconds), but there are processes that used up to 33 minutes of CPU time. This large variability in process CPU times is also apparent from the fact that the standard deviation is over 13 times larger than the mean, and that the mean is larger than the median by a similar ratio.

File I/O distribution, Figure 3.2, shows that about 30% processes have accessed no file bytes at all, and that the distribution has several abrupt slope changes (for example, one such change can be seen just before the 10K bytes mark). As will be seen later, these characteristics make file I/O prediction harder than CPU time prediction.

Memory usage distribution, Figure 3.3, shows that most processes used only a small fraction of memory available on the system (median memory usage is 50K bytes). The

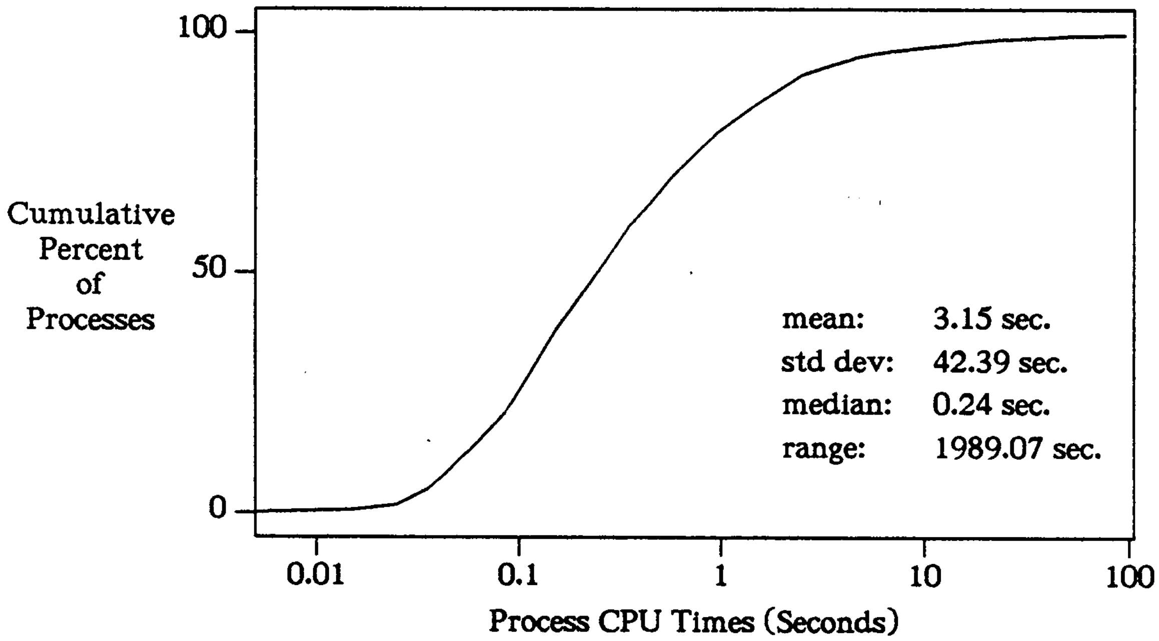

Figure 3.1: Distribution of Process CPU Times

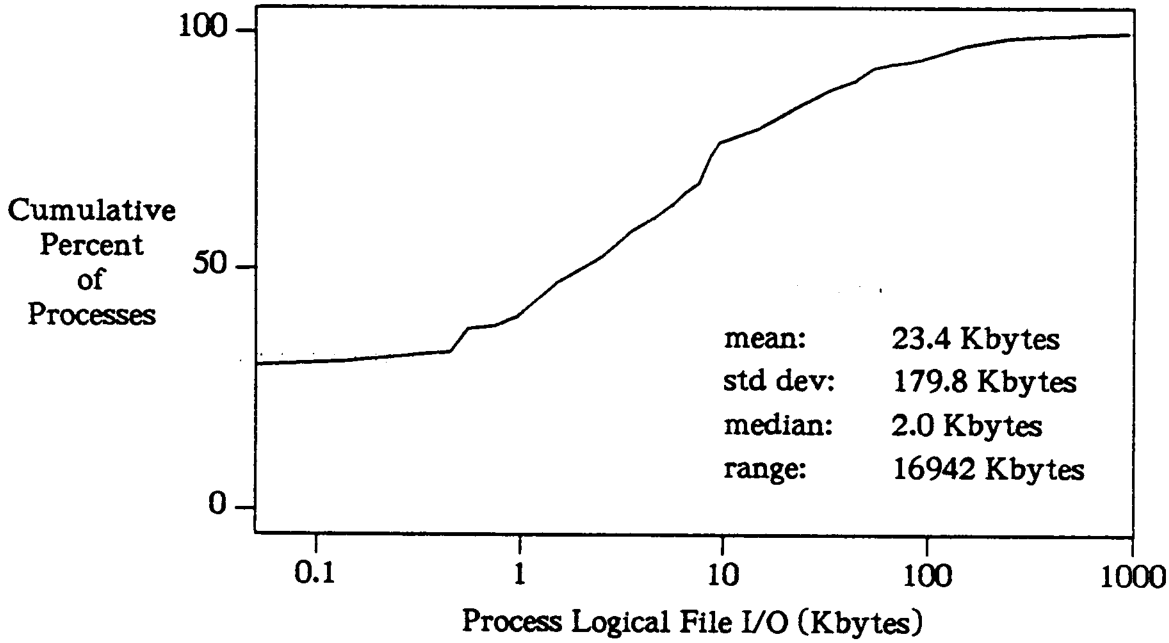

Figure 3.2: Distribution of Process File I/O

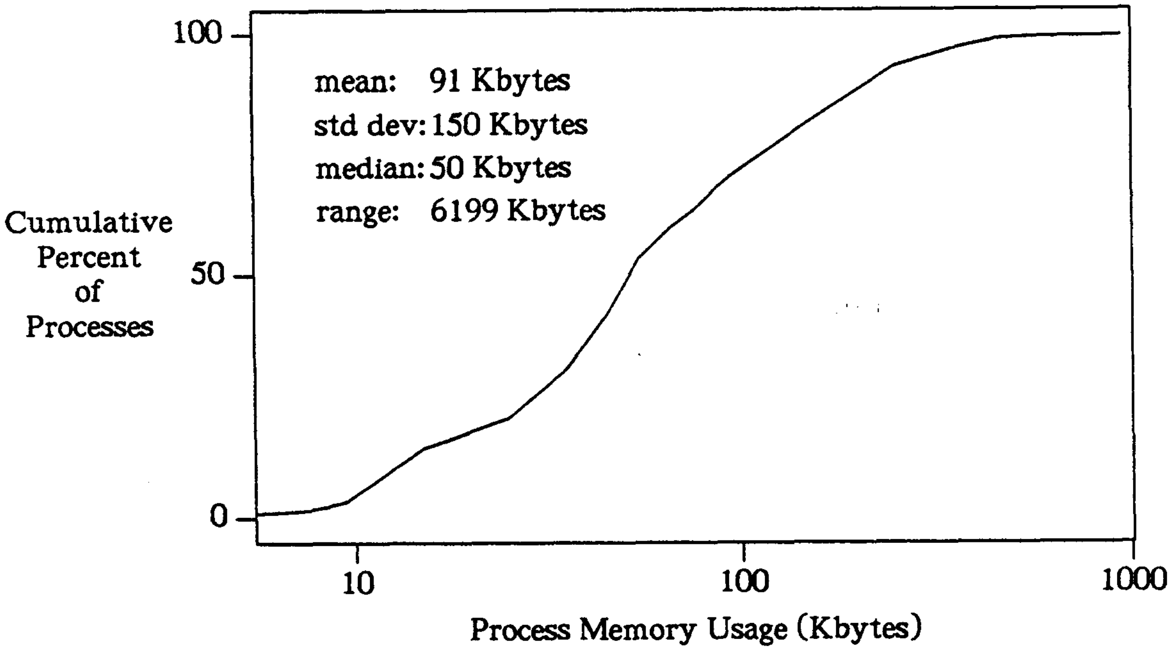

Figure 3.3: Distribution of Process Memory Usage

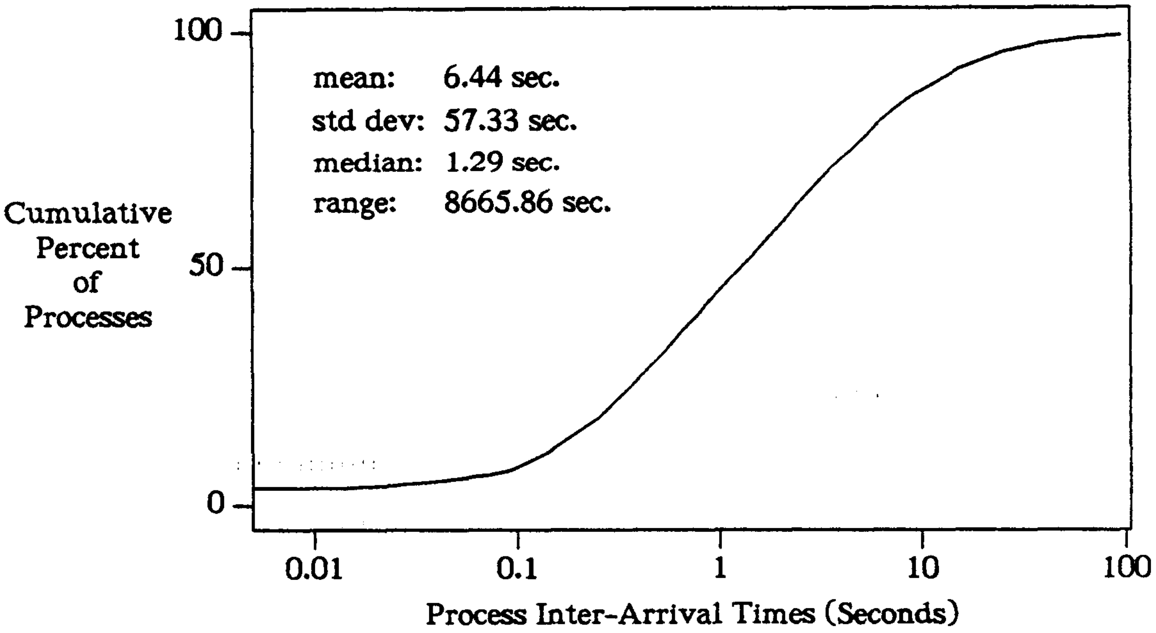

Figure 3.4: Distribution of Process Inter-Arrival Times

distribution also shows the smallest amount of variability. Mean is less than twice as large as median, and the ratio of standard deviation and mean is about the same. These characteristics of the processes make memory usage prediction easier than CPU time prediction.

Even though the process inter-arrival time is of little consequence to the prediction scheme itself, we discuss its distribution to complete the understanding of the measured system. As can be seen from Figure 3.4, mean and median inter-arrival times are larger than the corresponding statistics of process CPU times. It implies that on an average the system utilization is not very high. However, since there are processes requiring large CPU times and small inter-arrival times, the system can be seen to have heavy as well as light usage periods.

In summary, resource usage distributions show that process CPU times have a large variability and that the system had a low as well as a high degree of utilization.

4. Resource Usage Modeling

In this section, we develop a state-transition model to describe dynamics of resource usage in a series of processes. Here, three resource usage parameters — CPU time, file I/O, and memory used — define a 3D resource space, and the processes that ran on the system (during an interval of time) are represented by points in the 3D space. A statistical clustering algorithm is employed to identify the high density clusters in this space. These clusters, defined by their centroids, are taken to be the states for the processes, and appropriate transition probabilities are determined from one state to another. Later, this state-transition model will be used for representing the past resource usage, which in turn will be used to predict the future resource requirements.

4.1. Cluster Analysis

First, each of the three resource usage parameters are normalized so that the values are expressed in standard deviations rather than units specific to a resource. The normalization employed here is called z-transformation:

$$z_i = \frac{x_i}{\sigma_d} \qquad \text{(Eq. 4.1)}$$

where z_i is the normalized value of x_i, and σ_d is standard deviation of the population with the largest d% of samples removed. We used d = 1.5 for CPU and file I/O and d = 0.5 for memory. The removal of the largest d% of samples eliminates the influence of the outliers on the normalization, and such a normalization can be helpful in obtaining well-defined clusters.

The cluster analysis used a k-means algorithm to partition an N-dimensional population into k clusters. Briefly, the algorithm starts with k clusters, each of which consists of a single random point. Each new point is added to the cluster with the closest centroid. After a point is added to a cluster, the mean of that cluster is recalculated to take the new point into account. The process is repeated several times, each time the initial means of k clusters are set to means from the end of the previous iteration, until the changes in the cluster means become negligibly small. Thus at any stage, the k means are in fact the means of the clusters they represent. Therefore, k non-empty clusters, $C_1, C_2, ..., C_k$, are sought such that the sum of squares of the Euclidean distances of the cluster members from their centroids is minimized, i.e.,

$$\textit{minimize} \quad \sum_{i=1}^{k} \sum_{j} |x_{ij} - \bar{x}_i|^2$$

where $x_{ij} \in C_i$ and $\bar{x}_i$ is the centroid of the cluster C_i.

Table 4.1: Cluster statistics.

Cluster Number	Cluster Frequency	Cluster Statistics (median values of the resources)		
		CPU (seconds)	File I/O (Kbytes)	Memory (Kbytes)
1	11.26%	4.62	13.870	194.726
2	2.64%	0.25	0.000	446.461
3	6.43%	0.80	8.486	192.444
4	9.42%	0.25	0.732	117.294
5	29.76%	0.07	0.000	16.000
6	29.69%	0.25	2.000	50.238
7	10.77%	1.54	103.804	134.386

Seven clusters of processes were formed. Table 4.1 shows the cluster statistics and percentage of processes in each cluster. We see from the table that clusters 1 and 7 represent heavy processes. Together they account for 22% of the population. Cluster 1 consists of CPU bound processes, and cluster 7 consists of balanced (CPU as well as I/O) processes. Another interesting class of processes belong to cluster 2: they are memory intensive.

4.2. State-Transition Model

Now that we have the clusters, we can calculate transition probabilities from one cluster to another to build a comprehensive state-transition model. A state-transition model built for a series of processes, taken from the measured data, is shown in Table 4.2 and in Figure 4.1. The processes are executions of a program. The transition probabilities from state i to state j, p_{ij}, were estimated using:

$$p_{ij} = \frac{\textit{observed number of transitions from state } i \textit{ to state } j}{\textit{observed number of transitions from state } i} \qquad \text{(Eq. 4.2)}$$

The state-transition model shows a distinct pattern. Transition probabilities from state 5 to itself (0.576) and from state 7 to itself (0.516), are the largest transition

Table 4.2: A state-transition table for a program.

cluster#	1	2	3	4	5	6	7
1	-	-	-	-	-	-	-
2	-	0.250	-	-	0.250	-	0.500
3	-	-	-	-	-	-	-
4	-	-	-	0.410	0.205	0.154	0.231
5	-	0.003	-	0.038	0.576	0.050	0.333
6	-	0.018	-	0.036	0.382	0.109	0.455
7	-	0.003	-	0.031	0.357	0.093	0.516

probabilities out of states 5 and 7 respectively. Note that the states 5 and 7 also have the highest *visit ratios* (see below). Therefore, from the model it can be concluded that an execution of the program is likely to be in state 5 or 7, and in addition, once an execution occurs in one of the states it tends to remain there. Patterns like these suggest predictability.

For some series of processes, however, transition probabilities out of a state are almost independent of current state. In such cases visit ratios are adequate. A visit ratio is the fraction of times a state occurred in a series of processes. For example, Table 4.3 shows visit ratios for the same series of processes that are used to build the state-transition model of Table 4.2. States 5 and 7 are visited 0.450 and 0.412 fractions of the time, making them the most frequently visited states. As will be seen in the next section, visit ratios, instead of transition probabilities, are used in prediction, when transitions to a state (and hence transitions out of that state) are too few to be

Table 4.3: A visit ratio for a program.

cluster#	1	2	3	4	5	6	7
ratio	-	0.005	-	0.056	0.450	0.077	0.412

statistically significant.

In summary, this section introduced a state-transition model for representing the dynamics of resource usage in a series of processes. The states of the model are the high density regions of a resource space, and they were obtained from a cluster analysis of the processes. We observed that the state-transition model can show interesting resource usage patterns.

5. A Program-Based Resource Prediction Scheme

Now that we have a state-transition model for representing the dynamics of resource usage in a series of processes, we describe how it is used for prediction. The particular scheme described here is a program-based prediction scheme. The scheme predicts resources required for a process at the start of its life, given the identity of the program and resource usage of the program in its past executions. Hence, it is called program-based prediction.

The past executions of a program (for example, that of a LISP compiler) are ordered by the terminating times of processes, where the processes are the executions of the program. From this series a state-transition model, $[p_{ij}]$, $i=1,2,..,N$, $j=1,2,..,N$, is built using Eq. 4.2. Table 4.2 is an example of such a state-transition model.

There is an upper as well as a lower limit on the number of processes used in building the model. The upper limit, enforced via parameter T_1, restricts the amount o past used, and thus makes the model reflect a desired level of dynamic behavior. O course, the exact number of past executions used is $min(m, T_1)$, where m is the nu of past executions of a program that actually took place so far. In the implem discussed here, we used all past executions of a program. The lower limit o

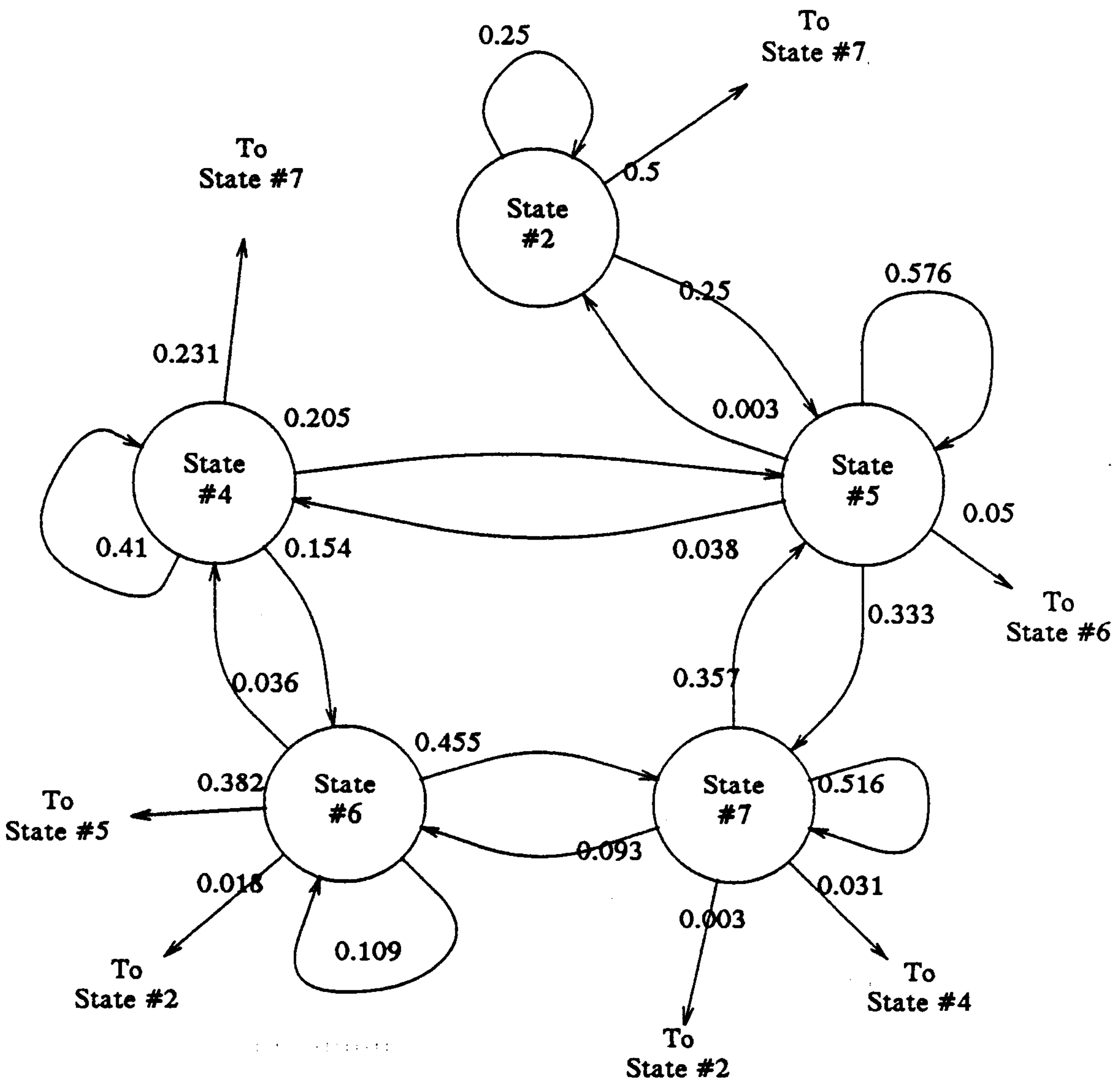

Figure 4.1: State-Transition Diagram for the Model in Table 4.2.

of processes guarantees that the resource usage model is stable enough to make a prediction. Parameter T_2 of the prediction algorithm provides this lower limit.

Assuming that there are enough past executions, $p_{lj}, j = 1,2,...,N$, gives the probability that the next execution will be in cluster j, where l is the (resource usage) state of the program's previous execution. However, these transition probabilities are used in computing resource requirements only if the number of transitions out of the state l satisfy a minimum. Parameter T_3 represents this minimum, and it assures that the state has a statistically significant number of entries and exits. If this parameter is not satisfied, the prediction algorithm uses visit ratios (such as the ones in Table 4.3) for computing resource requirements.

The procedure for computing process resource requirements can be explained as follows. Since we have clustered the environment, each program execution must be in one of the clusters. Within each cluster, however, there is a subcluster that identifies the program. The midpoint of this subcluster is obtained by the most recent executions of the program that belong to the cluster. Then, the process resource requirements are obtained by multiplying the transition probabilities, $p_{lj}, j = 1,2,...,N$, with the midpoints of these subclusters, $d_{jk}, j=1,...,N, k=CPU,I/O,MEM$:

$$r_k = \sum_{j=1}^{N} p_{lj} \times d_{jk}, \qquad k = CPU,I/O,orMEM$$

Note that d_{jk} are specific to a cluster as well as a program. A fourth parameter, T_4, determines the number of past executions used in computing d_{jk}. Also note that T_4 is considerably smaller than T_1. For example, in our implementation T_4=1, whereas T_1 is usually in the hundreds.

Parameters:

T_1 Maximum number of past executions used in building the model (all).

T_2 Minimum number of past executions required to make a prediction (3).

T_3 Minimum number of visits to a state needed, to use the transition probabilities of the state $(max(T_2,\ 5\%\ of\ min(m, T_1)))$.

T_4 Number of past executions used in computing subcluster centroids (1).

Constants:

N Number of clusters (7).

Variables:

l Cluster number to which the previous execution belonged.

m Number of completed executions of the program so far.

Data structures:

$\left[p_{i,j}\right]$ State-transition matrix, $i = 1, ..., N$, and $j = 1, ..., N$.

$\left[v_i\right]$ Visit ratios, $i = 1, ..., N$.

$\left[s_{i,j,k}\right]$ Resources used in previous T_4 executions,
$i = 1, ..., N, j = CPU, I/O, or\ MEM$, and $k = 1, ..., T_4$.

$\left[c_{i,j}\right]$ Cluster medians, $i = 1, ..., N, j = CPU, I/O, or\ MEM$.

Computations:

$$d_{i,j} = \begin{cases} \dfrac{1}{T_4} \sum_{k=1}^{T_4} s_{i,j,k} & \text{if } T_4 > 0 \\ c_{i,j} & \text{if } T_4 = 0 \end{cases} \qquad i = 1, ..., N, \text{ and } j = CPU, I/O, \text{ and } MEM$$

$$r_j = \begin{cases} \sum_{i=1}^{N} [\, p_{l,i} \times d_{i,j} \,] & \text{if } min(m, T_1) \geq T_3 \\ \sum_{i=1}^{N} [\, v_i \times d_{i,j} \,] & \text{if } min(m, T_1) < T_3 \end{cases} \qquad j = CPU, I/O, or\ MEM$$

Figure 5.1: Summary of the Program-Based Prediction Scheme.

The prediction scheme is summarized in Figure 5.1. Parameter values used in our implementation of the scheme are shown in parenthesis. Now that we have described the prediction scheme, we will now proceed to discuss how well the prediction scheme worked on the data collected.

5.1.1. How Good is the Prediction?

In order to determine prediction quality, a trace-driven prediction experiment was conducted. The experiment consisted of predicting process resource requirements using the program-based method, just before the process started its life, and then observing the difference between the predicted and actual resource values after the process terminated. This section discusses results of this experiment.

For some processes prediction could not be made owing to the lack of enough past executions of the program. However, both the percentage of such processes and CPU time used by them are quite small. With T_1=3, less than 4% processes could not be predicted, and these processes used about 8% of CPU time.

We quantified prediction quality in two ways. First, product-moment (Pearson) and rank (Spearman) correlations [Mendenhall and Sincich 84] between the predicted

Table 5.1: Correlations between Actual and Predicted Resource Values.

Resource	Correlation Coefficients	
	Rank (Spearman) Correlation	Product-Moment (Pearson) Correlation
CPU Time	0.8379	0.8406
File I/O	0.8105	0.1974
Memory	0.8925	0.8834

and actual values are considered. The Pearson correlation coefficient measures the strength of the linear relationship between two quantities, and the Spearman's rank correlation measures correlation between ranks of the two quantities. Here, the Spearman's rank correlation is a better indicator than Pearson's because the former does not necessarily look for a linear relationship. Table 5.1 shows that the Pearson correlation coefficient is over 0.84 for CPU time and memory, but it is small (about 0.20) for file I/O. A correlation coefficient of 1.0 implies a perfect prediction. The Spearman correlation coefficient, however, ranges from 0.81 to 0.89 for all the resources. Clearly, quality of prediction is good.

Next, distributions of errors in prediction are considered. An error in prediction is the absolute difference between predicted and actual resource usage. Figure 5.2 shows distributions of prediction errors for CPU time, file I/O, and memory usage. It can be seen that error distributions are highly skewed towards small values. For example, 82% of errors in CPU time prediction are less than 0.5 standard deviations. Also, error in predicting memory usage is the smallest.

Mean and other statistics about prediction errors and actual resource usage values are shown in Table 5.2. The values are in normalized units (standard deviations of the actual) obtained through the application of z-transformation of Eq. 4.1. The table shows that for CPU time the median error is 0.073 standard deviations (about 43% of the actual), and the mean error is 1.224 standard deviations (about 53% of the actual). Since the variability in CPU times is large (about 18 standard deviations), as shown by the difference between 99 percentile and 1 percentile, we believe that these errors are acceptable.

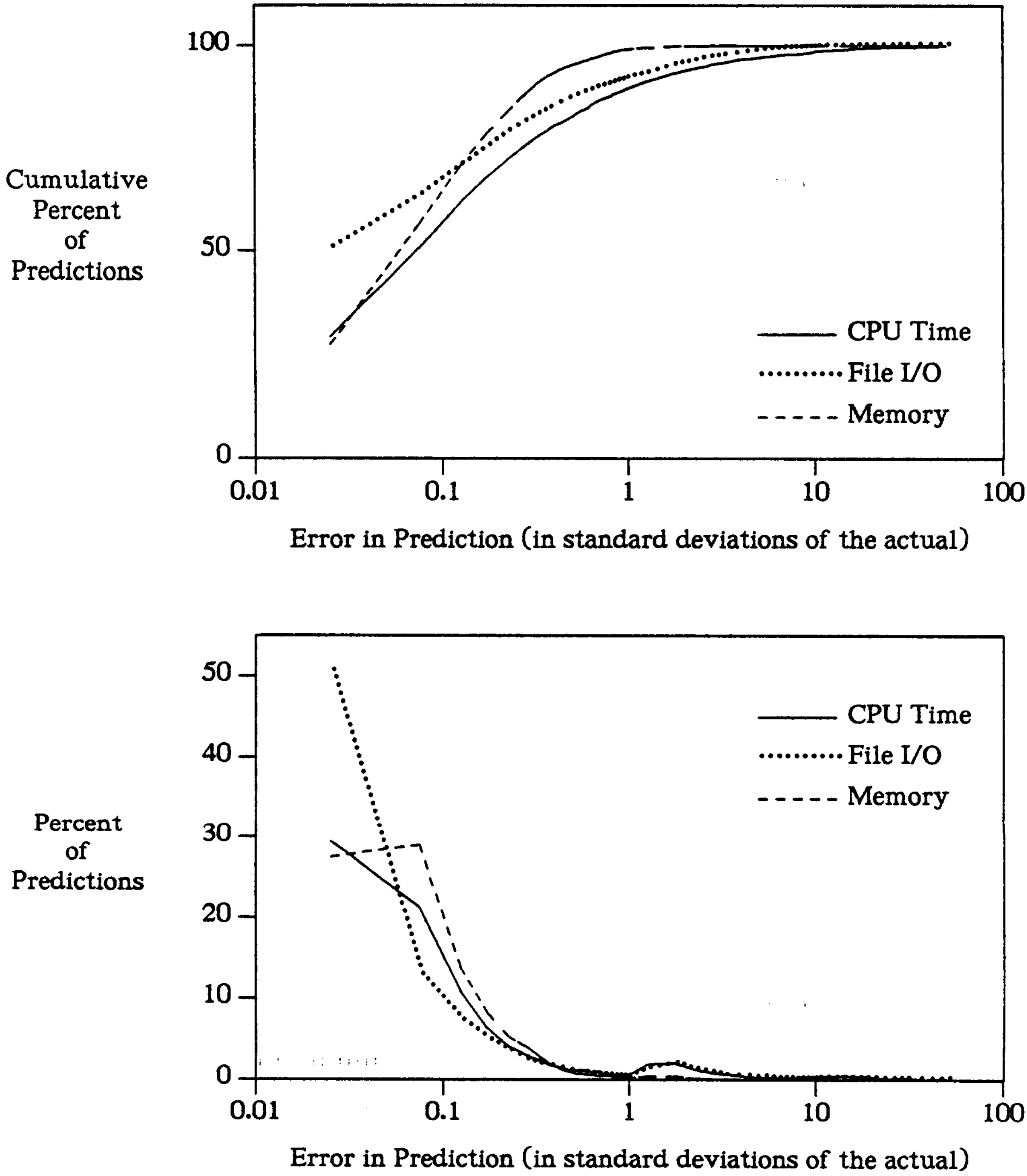

Figure 5.2: Distributions of Prediction Errors

Table 5.2: Statistics of the Prediction Errors and Actual Resource Values.

Resource		Statistic (in units normalized using Eq. 0.4.1)			
		Mean	Std dev	Median	99%-1%
CPU Time	Error	1.224	18.424	0.073	16.24
	Actual	2.230	32.780	0.168	18.23
File I/O	Error	0.485	4.909	0.024	6.13
	Actual	0.601	4.755	0.051	7.26
Memory	Error	0.140	0.560	0.059	0.97
	Actual	0.723	1.181	0.447	3.61

Compared to errors in CPU time prediction, errors in file I/O prediction are larger, but errors in memory usage prediction are smaller. For example, median error in memory usage prediction is about 13% of actual, and mean error is about 19% of actual.

We considered other measures of prediction quality but rejected them on the grounds that they are not suited for the domain we are concerned with. For example, it might seem like a good idea to express the errors as percentages of the actual, and show a distribution of the percentages. However, (since the smallest amount of resource a process can use is 0) when a predicted value is smaller than actual, prediction error can be 0% through 100%, but when a predicted value is larger than actual prediction error is potentially unbounded. This distorted view of error can lead to a misleading perception that a scheme that makes a few large over-estimations is worse than a scheme that consistently underestimates.

We have also compared means and variances of predicted and actual values, and examined correlation between error and actual values. Means and variances of predicted and actual values match very closely. Errors correlate slightly positively (about 0.20) with actual values, implying that large prediction errors (if any) tend to occur only when outliers of process population occurs.

In conclusion, even though the program-based prediction scheme makes a few large errors, errors are mostly small.

6. Additional Implementation Issues

In the previous section, the program-based prediction was described in detail, and using a trace-driven experiment, it was shown that the error in prediction is small. Here, we discuss the following three issues related to the implementation of the prediction scheme.

1. The influence of program execution frequency on prediction quality.
2. The influence of maximum and minimum past used in prediction on prediction quality.
3. The influence of system load on memory usage measurement.

6.1. The Influence of Program Execution Frequency

Each program is categorized as type 0, 1, 2, or 3 based on the total number of executions of the program during the measured period, and using a trace-driven experiment as described in the previous section, prediction quality is quantified for each program type. Results are shown in Table 6.1.

Type 0 consists of programs that are executed three or fewer times in the data, where 3 is the value used for the parameter T_2 (the minimum number of executions required to make a prediction). The remaining three types are defined such that the programs that are executed four (i.e., T_2+1) times or more are equally divided into the three types.

Table 6.1: Dependence of prediction quality on program execution frequency.

Item		Type #0 programs	Type #1 programs	Type #2 programs	Type #3 programs
Number of executions		1 thru 3	4 thru 8	9 thru 45	46 or more
Percent programs		36.4%	21.2%	21.0%	21.4%
Percent processes		2.7%	0.8%	4.4%	92.1%
Correlation of predicted and actual CPU times		-	0.803	0.794	0.879
CPU time	mean	-	19.971	13.629	1.531
statistics	std dev	-	135.785	86.049	24.735
(in norm. units)	median	-	0.488	0.595	0.160
Error in	mean	-	11.766	7.568	0.828
prediction	std dev	-	90.537	54.498	11.935
(in norm. units)	median	-	0.099	0.238	0.069
Error in	mean	-	59%	56%	54%
prediction as	std dev	-	67%	63%	48%
pct of actual	median	-	20%	40%	43%

As can be seen from Table 6.1, about 36% of programs belong to type 0, and about 21% of programs belong to each of the remaining types. However, processes resulting from type 0 programs constitute only 2.7% of total processes. In comparison, processes resulting from type 3 programs are over 92% of the total. Programs of type 2 and 1 programs provide 4.4% and 0.8% processes each. Clearly, a small fraction of programs are executed frequently (e.g., 21% of programs are executed 92% of times).

For type 3 programs, the coefficient of correlation between predicted and actual CPU times is 0.879, and for types 1 and 2, the coefficient is about 0.8. A correlation coefficient of 1.0 implies a perfect prediction. Given that the observed correlations coefficients are above 0.8, prediction quality is quite good for processes produced by programs of any type. The prediction is particularly good for processes produced by type 3 programs, and these processes constitute a major fraction of processes that ran on the system.

Table 6.1 also shows statistics for process CPU times and prediction errors for each category of programs. The CPU times and errors are reported in normalized units obtained through the application of Eq. 4.1, so that these results can be easily compared with those reported in the previous section. The average CPU time used is the largest for processes resulting from type 1 programs, followed by processes resulting from type 2 programs. The average error in prediction follows the CPU time usage pattern. However, when expressed as a percentage of average CPU time used, the prediction error is comparable for all program types, with the error percentage being slightly higher for infrequently executed programs.

In summary, it is shown that the quality of prediction is essentially independent of program execution frequency, except for programs that are executed less than 4 times. These programs constitute about 36% of all executed programs, but produce only 2.7% of all processes. The next section discusses how prediction quality varies when the maximum and minimum past used in prediction is varied.

6.2. The Influence of Maximum and Minimum Past Used

Here, we quantify the influence of maximum and minimum past used in the prediction scheme (parameters T_1 and T_2 of the prediction scheme) on quality of prediction.

A. Maximum Past Used

First, the trace-driven experiment described in the previous section is repeated several times, each time with a different value for the maximum past used in building the resource usage model, while keeping the minimum past fixed at 1. The mean error[1] in CPU time prediction, obtained from these experiments, is shown in Figure 6.1 for the maximum past ranging from 1 through 300.

The figure shows that the mean error decreases as the maximum past is increased. The rate of improvement saturates around a value approximately equal to 100. Note, however, that a change in the maximum past from 1 to 300 brings about a reduction of about 7% in mean error for CPU time prediction.

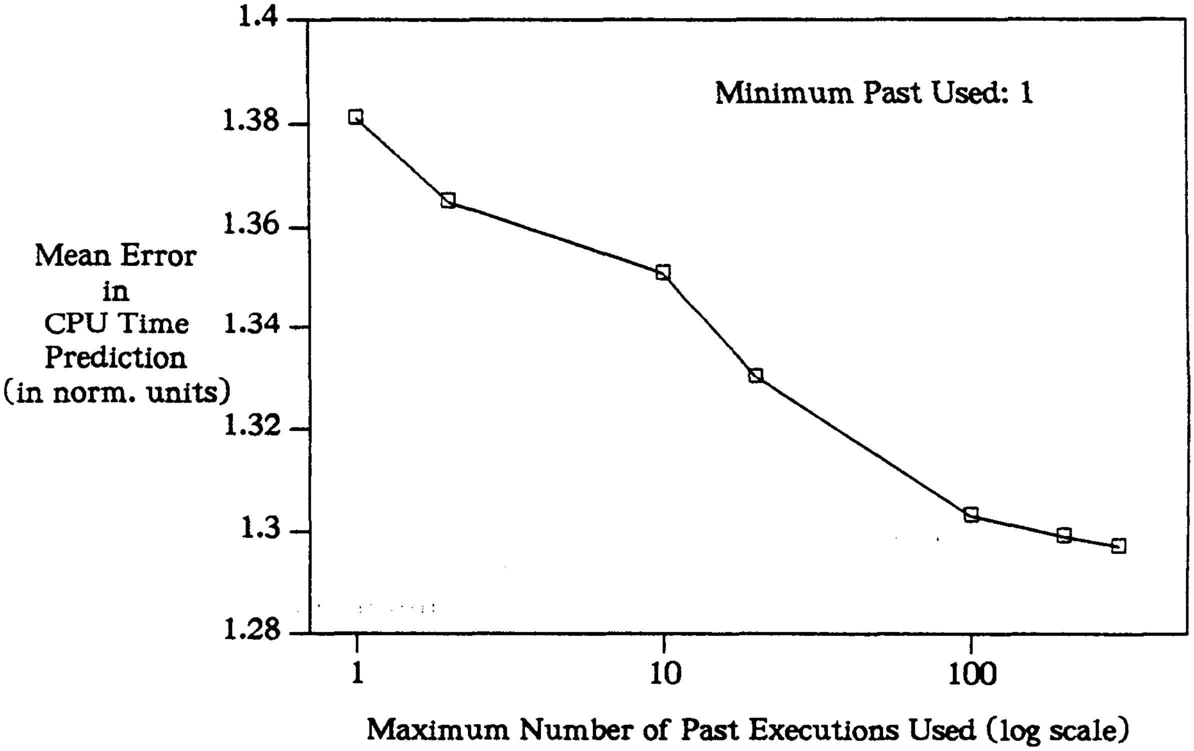

Figure 6.1: Effects of Changing Maximum Past Used in Prediction.

[1] The error is shown in the same normalized units as the actual process CPU time, which is obtained using Eq. 4.1.

An examination of error distributions for different values of maximum past shows that when a small amount of maximum past is used (say $T_1 = 1$), the prediction is overly sensitive to local variations in the resource usage pattern of the predicted program. The error distribution for such a small maximum past (i.e., $T_1 = 1$) is more heavily skewed towards small values and has a longer tail than the error distribution for a large maximum past (say, $T_1 = 300$). Thus, when a large amount of maximum past is used, the prediction errors are evenly distributed while both large as well as small errors decrease. Consequently, using a large amount of maximum past (for example, 300) has a stabilizing effect on prediction, and results in a small average error.

B. Minimum Past Used

Next, the effect of varying the minimum past used, parameter T_2, on prediction quality is examined. The trace-driven experiments are repeated once again with different values of minimum past, while keeping the maximum past fixed at 200. The results of these experiments are shown in Figure 6.2. The mean error[2] in CPU time prediction drops dramatically as the minimum past is increased — the prediction error reduces by about 38% as the minimum past is changed from 1 to 20 executions.

However, unlike the changes in maximum past, increasing the minimum past has a side-effect of decreasing the percentage of predictable processes. More importantly, an increase in the minimum past decreases the percentage of predicted CPU usage by a considerable amount. For example, as the minimum past is raised from 1 to 20, the percentage of predicted processes drops by only 9%, but the percentage of predicted CPU

[2] The error is shown in the same normalized units as the actual process CPU times, which is obtained using Eq. 4.1.

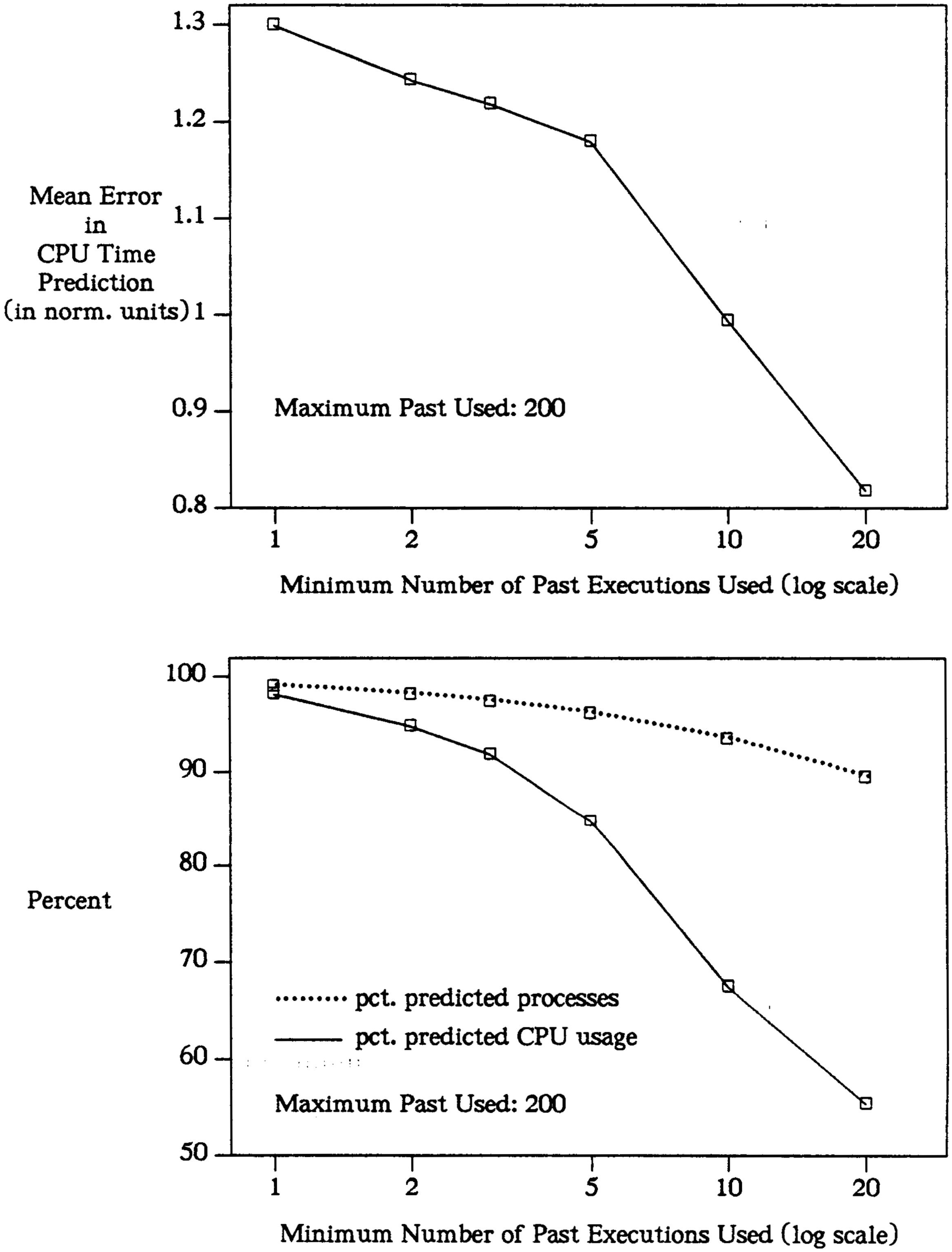

Figure 6.2: Effects of Changing Minimum Past Used in Prediction.

usage drops by 43%. So, a small minimum past, such as 3, is recommended.

6.3. System Load Influence on Memory Usage Measurement

The measured per process memory usage is the average amount of memory allocated to the process by the system. Since this allocation can depend on system *load*, we study the extent of such a dependency in this section. (The system load referred to here is the average number of ready-to-run processes on the system in the last one minute.) In order to do so, four programs, each with a different running time and memory usage pattern, were run on the measured system at regular (about 12 to 15 minutes) intervals for about two days, while the system was in normal use. For each execution of these programs, the system load and resource usages were recorded.

Based on these experimental measurements, we calculate the coefficient of correlation between the system load and memory usage, for each of the four programs. The results are shown in Table 6.2. As the table shows, for a long running program (e.g. 30 secs) having a small working set compared to its address space, the system load has the most prominent effect on the measured memory usage. The correlation coefficient for

Table 6.2: Correlation between system load and process memory usage.

program characteristics		correlation coefficient	Is correlation statistically significant?
running time	memory usage pattern		
large (30 secs)	ws << address space	-0.7824	Yes
small (3 secs)	ws << address space	-0.4809	Yes
large (30 secs)	ws = address space	0.0435	No
small (3 secs)	ws = address space	0.2134	No

this type of program is -0.7824, indicating a negative correlation. However, for a program with a similar memory referencing pattern, but a shorter running time, the effect is not as strong. For this type of program, the coefficient of correlation is only -0.48. Finally, for a program having the working set that is almost equal to its address space, independent of its running time, the system load influence on memory usage measurement is statistically insignificant.

The following, however, should be noted in this regard. Even when measurements are sensitive to system load, the resource usage model can incorporate these influences, and the prediction made using the model is valid if the target processor has a load similar to that of the measured processor. Since, the latter condition is likely to be true in a load balanced system, the influence of system load on memory usage measurement is not a serious problem.

7. Summary

In this paper, we described a probabilistic scheme for predicting CPU time, file I/O, and memory requirements of a process at the beginning of its life. Given the identity of the program being run, this prediction scheme uses a state-transition model of the resource usage in the previous executions of the program. The states of the model are obtained from a statistical cluster analysis of the processes run on the system (in a day). The prediction scheme was shown to work on the measured data using a trace-driven prediction experiment.

The results of the trace driven experiment show that the predicted values correlate well with the actual. The coefficient of correlation between the predicted and actual CPU time is 0.84. Further, the error distributions show that the errors in prediction are mostly small. For example, 82% of errors in CPU time prediction are less than 0.5 standard deviations of process CPU time. These results are particularly interesting since Zhou's study [Zhou 86b] of system load indices as predictors of future load correlated poorly with the actual (correlation coefficients are always less than 0.45). Applications of resource usage prediction in load balancing and in system reorganization under failure are suggested as future work.

ACKNOWLEDGEMENTS

The first author acknowledges the support and encouragement of Professor Roy H. Campbell. We thank Sharon Peterson, Luke Young, and Rick Eichemeyer for a careful proof reading of this paper. This research was supported in part by AT&T Metropolitan Networks Grant Number 1-5-13411, and in part by NASA Grant Number NAG-1-613.

REFERENCES

[Barak and Litman 85] A. Barak, and A. Litman, "A Distributed Load Balancing Policy for a Multicomputer," *Software - Practice and Experience*, 15, 8, Aug. 1985.

[Berkeley UNIX 86] *UNIX Programmer's Manual: Reference Guide*, 4.3 Berkeley Software Distribution, Virtual VAX-11 Version, Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA 94720, 1986.

[Bryant and Finkel 81] R. Bryant, and R. Finkel, "A Stable Distributed Scheduling Algorithm," *Second International Conference in Distributed Computing Systems*, IEEE Computer Society, Los Alamitos, California, April 1981.

[Eager 86] D. Eager, E. Lazowska, and J. Zahorjan, "Dynamic Load Sharing in Homogeneous Distributed Systems," *IEEE Transactions on Software Engineering*, SE-12, 5, May 1986.

[Hwang 82] K. Hwang, W. Croft, G. Goble, B. Wah, F. Briggs, W. Simmons, and C. Coates, "A UNIX-based Local Computer Network with Load Balancing," *IEEE Computer*, 15, 4, April 1982.

[Leland and Ott 85] W. Leland, and T. Ott, "Load-balancing Heuristics and Process Behavior," *Performance '86 and ACM SIGMETRICS Conference*, Raleigh, North Carolina, May 1986.

[Mendenhall and Sincich 84] W. Mendenhall, and T. Sincich, *Statistics for the Engineering and Computer Sciences*, Dellen Publishing Company, San Francisco, California, 1984.

[Wang and Moris 85] Y.-T. Wang, and R. Morris, "Load Sharing in Distributed Systems," *IEEE Transactions on Computers*, C-34, 3, March 1985.

[Zhou 86a] S. Zhou, "A Trace-Driven Simulation Study of Dynamic Load Balancing," *Tech. Report No. UCB/CSD 87/305*, University of California, Berkeley, California, Sept. 1986.

[Zhou 86b] S. Zhou, "An Experimental Assessment of Resource Queue Length as Load Indices," *Tech. Report No. UCB/CSD 86/298*, University of California, Berkeley, California, Sept. 1986.