March 1990

UILU-ENG-90-2207
CSG-121

COORDINATED SCIENCE LABORATORY
College of Engineering

Ames
GRANT
IN-60-CR
271113
45P.

MEASURE: AN INTEGRATED DATA-ANALYSIS AND MODEL IDENTIFICATION FACILITY

Jaidip Singh
Ravi K. Iyer

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

[illegible]

MEASURE: AN INTEGRATED DATA-ANALYSIS AND MODEL-IDENTIFICATION FACILITY

Jaidip Singh   *Ravi K. Iyer*

January 1990

Center for Reliable and High Performance Computing
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
Urbana, IL61801

ABSTRACT

This report describes the first phase of the development of **MEASURE**, an integrated data analysis and model identification facility. The facility takes system activity data as input and produces as output representative behavioral models of the system in near real-time. In addition a wide range of statistical characteristics of the measured system are also available.The usage of the system is illustrated on data collected via software instrumentation of a network of SUN workstations at the University of Illinois. Initially, statistical clustering is used to identify high-density regions of resource-usage in a given environment. The identified regions form the states for building a state-transition model to evaluate system and program performance in real-time. The model is then solved to obtain useful parameters such as the response-time distribution and the mean waiting time in each state. A graphical interface which displays the identified models and their characteristics (with real-time updates) has also been developed. The results provide an understanding of the resource- usage in the system under various workload-conditions. This work is targeted for a testbed of UNIX workstations with the initial phase ported to SUN workstations on the NASA, Ames Research Center Advanced Automation Testbed.

Keywords: performance measurement, data-analysis, real-time modeling, statistical clustering, state-transition model.

ACKNOWLEDGEMENTS

The authors would like to thank Wolfgang Halang (Univ. of Illinois), for helping with the clustering algorithms, and Terry Grant (NASA, Ames) and Wei-Lun Kao (Univ. of Illinois) for many useful discussions. They would also like to thank Gwan Choi (Univ. of Illinois) and Roger Bartlett (NASA, Ames), for helping with the graphical interface and Linda Lin (Univ. of Illinois) for proof-reading this report. This work was supported in part by NASA grant NCA-2-301 and in part by NASA grant NAG-1-613.

Contents

1 Introduction

The goal of this project is to develop an integrated data analysis and model extraction facility. System activity data are collected and analyzed to extract suitable behavioral models of the system under measurement. The development of resource usage models of this type is valuable for several reasons. For example, it can generate performance measures for software developers. Near real-time models can provide instantaneous feedback for system tuning and identification of performance bottlenecks. Furthermore, the impact of abrupt changes on system performability and reliability can be easily quantified.

Although many researchers have addressed the modeling issue and have significantly advanced the state of the art, none have addressed the issue of how to identify the model structure. Further, very few of either the hardware or the software models have been validated with real data. Exceptions are the joint hardware/software model discussed in [2] and a measurement based model of workload dependent failures discussed in [3]. Both, however, describe only the external behavior of the system and thus fail to provide insight into component-level behavior. Much of this project is based on earlier work by M.C. Hsueh [1] in which real data are used to identify suitable models.

System-level activity data for this project was collected via software instrumentation of a network of SUN workstations at the University of Illinois. Initially, statistical clustering is used to identify high-density regions of resource-usage in a given environment. The identified regions form the states for building a state-transition model to evaluate system and program performance in real-time. The model is then solved to obtain useful parameters such as the response-time distribution and the mean waiting time in each state. A graphical interface to display the key models and characteristics (with real-time updates) has also been developed. The results provide an understanding of the resource- usage in the system under various workload-conditions.

The following section gives an overview of the modeling tool. The data gathering procedure is discussed in Section 3. Section 4 deals with the data analysis and model construction. Section 5 explains the model-solution procedures. The results of experimental model-construction in real-time are presented in Section 6. The report concludes with a discussion of the results in Section 7. Possible extensions to the ongoing research are also discussed in this section. These extensions include the incorporation into the modeling facility of a variety of modeling techniques such as time-series analysis. The authors envisage that this tool will provide a spectrum of techniques to the user for modeling and prediction purposes. The code for model construction is contained in Appendix A.

2 Overview of Modeling tool

A simplified block diagram of the analysis tool and graphics package MEASURE is shown in Figure 1. The arrows indicate the flow of data through the system. The dotted lines enclose intended extensions to the tool which will allow the user to perform time-series analysis and make predictions in real-time. The data-gathering module is the interface with the system level instrumentation. The granularity of the collection is specified in this module. The database created by the data-gathering module is used by the clustering module to identify high-density patterns of resource usage. A user-interface allows the user to specify the number of clusters to be formed, the amount of data to be analyzed and the parameters to be analyzed. The calculations of centroids and other cluster parameters are also performed here. The model-identification module takes as input the centroids from the clustering module and creates a state-transition diagram. In the model-solver

BLOCK DIAGRAM OF REAL-TIME MODELING TOOL

(MEASURE)

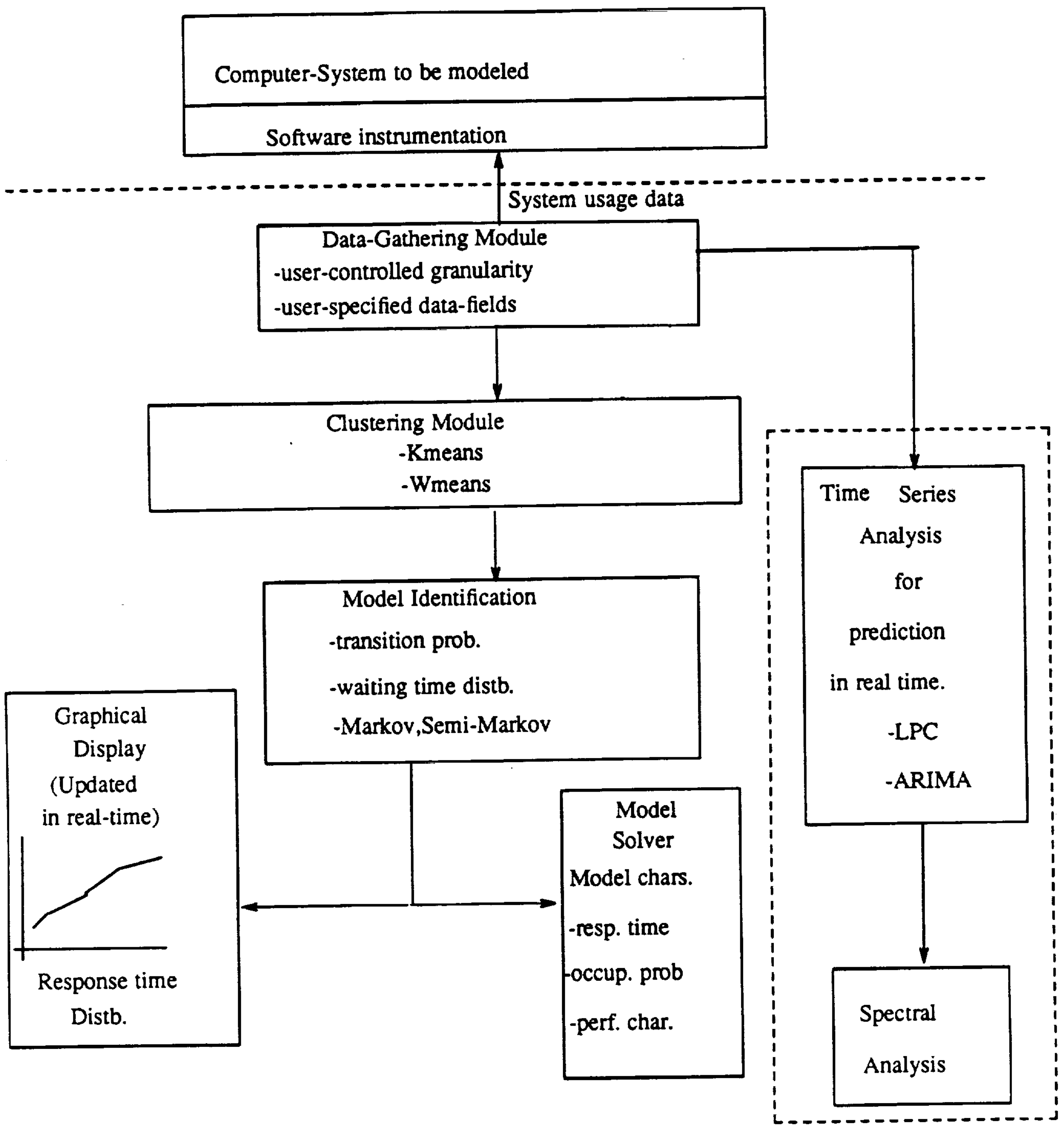


Figure 1: Block Diagram of Modeling Tool

the constructed model is identified (Markov or Semi-Markov) and the state-transition diagram is used to calculate the occupancy probability, the steady-state entry rate and the response-time distribution. A graphical interface is used to display the latest state-transition model and response-time distribution.

3 Data Gathering

Data for this study was collected via software instrumentation of a network of SUN workstations, running SunOS Release 4.0, at the University of Illinois. The network consists of 4 file-servers and 50 diskless SUN workstations. Specifically one of the four file-servers on the network was measured. The data on resource usage was collected using an operating system facility called *vmstat*, This facility collects data on system usage , e.g. the system CPU, number of pageins, size of active virtual memory, the context switch rate etc. by sampling the kernel data tables at periodic user-specified intervals. A typical output from *vmstat* is shown in Figure 1.

The actual statistics gathering is an integrated activity in the operating-system kernel i.e. it is performed by several routines. One of these, the *hardclock()* routine collects statistics at each clock cycle on the CPU mode (system, user or idle) in that cycle. A second, the *pagin()* routine, recalculates paging activity every time a paging request has to be satisfied. The kernel has three types of data structures: *rate*, *sum* and *cnt*. Five-second averages of measured parameters (e.g. CPU-user time percentage) are stored in data structures of the type *rate*. Free-running counters (e.g. number of device interrupts) are stored in structures of the type *sum* and accumulations over one second (e.g. number of context switches) are stored in structures of the type *cnt*. An image of the kernel tables is available in a special file called *kmem*. Vmstat reads *kmem* at user-

```
procs     memory              page                                         faults       cpu
r b w   avm    fre   re  at   pi   po   fr   de   sr  d0  d1  d2  d3   in   sy   cs  us  sy id
0 0 0     0   2808    0   0    0    0    0    0    0   0   0   0   0    4  144   37  63   7 30
0 0 0     0   2672    0   0    0    0    0    0    0   0   0   0   0   16   41   11   2   2 96
1 0 0     0   2616    0   0    0    0    0    0    0   0   0   0   0   12   39   10   4   1 95
0 0 0     0   2600    0   0    0    0    0    0    0   0   0   0   0   13   41   10   1   1 98
0 0 0     0   2600    0   6    0    0    0    0    0   0   0   0   0   13   45   10   2   4 95
0 0 0     0   2600    0   0    0    0    0    0    0   0   0   0   0   40   48   12   8   3 90
0 0 0     0   2600    0   0    0    0    0    0    0   0   0   0   0   16   43   13   6   2 92
0 0 0     0   2600    0   0    0    0    0    0    0   0   0   0   0    4   16    4   0   0100
0 0 0     0   2600    0   0    0    0    0    0    0   0   0   0   0    0    8    1   0   0100
0 0 0     0   2600    0   0    0    0    0    0    0   0   0   0   0    0    4    0   0   1 99
2 0 0     0   2600    0   6    0    0    0    0    0   0   0   0   0    0    3    0   0   3 97
0 0 0     0   2600    0   0    0    0    0    0    0   0   0   0   0    0    1    0   0   0100
0 0 0     0   2600    0   0    0    0    0    0    0   0   0   0   0    0    1    0   0   0100
0 0 0     0   2600    0   0    0    0    0    0    0   0   0   0   0    0    1    0   0   0100
0 0 0     0   2600    0   0    0    0    0    0    0   0   0   0   0    0    1    0   0   0100
0 0 0     0   2600    0   0    0    0    0    0    0   0   0   0   0    0    2    0   0   0100
0 0 0     0   2600    0   8    0    0    0    0    0   0   0   0   0    0    2    1   0   3 97
-------------------------------------------------------------------------------------------------
        Key:
        r: Number of processes in the run queue
        b: Number of processes blocked for resources
        w: Runnable or short sleeper processes
        avm: number of active virtual Kbytes
        fre: size of the free list in Kbytes
        re: number of page reclaims
        at: number of attaches
        pi: kilobytes per second paged in
        po: kilobytes freed per second
        de: anticipated short term memory shortfall in Kbytes
        sr,d0,d1,d2,d3,d4: Disk operation sper second
        faults:
                in: (non clock) device interrupts per second
                sy: system calls per second
                cs: CPU context switch rate per second
        CPU activity distribution in per cent:
                us: user time
                sy: system time
                id: CPU idle time
```

Figure 2: Output from Vmstat

specified intervals and performs simple arithmetic to compute averages and convert from one scale to another. Since vmstat uses some rate-type data, an interval specification of under five seconds can cause erroneous values to be read.

4 Model Construction

The data-analysis facility developed in this project is intended to allow the user to choose the measures to be analyzed. In the experiments conducted to date five parameters provided by vmstat have been analyzed. These are:

1. Non-clock device interrupts.
2. System calls.
3. Context switches.
4. Percentage of CPU time (user).
5. Percentage of CPU time (idle).

Each parameter is treated as a dimension in n-dimensional space, with n=5 in this case. Thus the data samples become five-dimensional vectors. In clustering nomenclature the axes of the space, i.e. the parameters, are called attributes.

The analysis uses statistical clustering to separate the component data into similar classes of resource usage. Similarities or distances are computed between pairs of data items and the clustering algorithm defines rules according to which the data-items are clustered into groups on the basis of inter-item distances or similarities.

A variety of clustering algorithms are being investigated for their suitability. Currently the model-construction code uses a statistical clustering algorithm called K-means which is based on an Euclidean distance measure [4]. The actual code is in Appendix A.

The algorithm partitions data-points into K clusters. K non-empty clusters $C_1, C_2, \ldots, C_k$ are sought such that the sum of squares of the Euclidean distances of the cluster members from their centroids is minimized i.e.

$$\sum_{j=1}^{k} \sum_{i} \| x_i - \overline{x_j} \|^2 \rightarrow minimum$$

where $x_i \in C_j$ and $\overline{x_j}$ is the centroid of cluster C_j. Starting from an arbitrary initial partition every point is transferred experimentally from its cluster to every other cluster and the new sum-of-squares of the Euclidean distances is computed. The point is allotted to that cluster for which the sum-of-squares of the overall system is minimized. This process is repeated until there is no decrease in the sum-of-squares. This implies that a local minimum of the sum-of-squares function has been reached. It is important to note that this algorithm does not guarantee to find the global minimum, since as soon as a local minimum is found no further decreases will occur. Different initial partitions may lead to the discovery of different local minima. Therefore it is prudent to run the program using several different initial partitions and to use the best local minimum thus discovered. The clustering problem is not amenable to exhaustive search techniques for the global minimum since the search-space can be very large. For example there are 10^{68} possible different partitions of 100 objects into 5 clusters.

Once the clusters are identified, the centroid of each cluster (represented by its Euclidean coordinates in n-dimensional space) is defined as a system-state. A transition model is then constructed based on these states. This model is used to evaluate important characteristics of the system such as the state occupancy probabilities, the transition probabilities from one state to another and the mean waiting time in each state.

A common problem that is often encountered in practical situations is that measured parameters (attributes) are usually expressed in non-homogeneous units (e.g CPU usage as a percentage of total time, paging activity in units per second). So as to analyze these parameters on an equal footing a scale change must be performed. Otherwise fields which have large numerical values can mask fields which have smaller values.

We can think of the measurements as constituting a data-matrix. If m measurements have been made the matrix will have m rows and if each measurement has n parameters (attributes) the matrix will have n columns. The measurements are scaled by attribute i.e. each attribute (column) in the scaled data-matrix has a standard deviation of 1 and a mean of 0. That is, each field x_{ik} is transformed to a scaled value y_{ik} such that

$$y_{ik} = \frac{x_{ik} - \overline{x_{i.}}}{S}$$

where $\overline{x_{i.}}$ is the mean value of a particular column, m is the total number of observations and S is the standard deviation of that column, estimated from the data and is given by

$$S = \frac{\sqrt{\sum_{k=1}^{n}(x_{ik} - \overline{x_{i.}})^2}}{\sqrt{m-1}}$$

In practice outliers (e.g. top 1-2 percent of the data) are often excluded in calculating the standard deviation [5]. In effect this prevents the outliers from dominating the other data values. It is important to note that these outliers are not excluded from the clustering process.

The possibility of dispensing with scaling is currently being investigated. This would entail the use of a clustering algorithm which is not susceptible to non-homogeneous data. One such algorithm W-means [4] is being tested for suitability in a real-time environment. Initial experiments show that the run-time of W-means is up to ten times greater than the run-time of K-means.

4.1 State Transition Probabilities

A state-transition model is constructed based on the defined system-states. This entails computing the state-to-state transition probabilities, the mean waiting times and the mean holding times in each state directly from the measured data and from the state definitions. For calculating the state-transition probabilities we use the fact that the data are in time-ordered form and that each data-point is assigned to one state exclusively. From the state-assignments we can calculate the number of transitions from a state i to some specific state j. On dividing this by the total number of transitions out of state i we obtain the transition probability p_{ij}.

There are two notational conventions that can be used to assign transition probabilities to the state diagram. The first convention assumes that transitions occur each time a measurement is taken. If this convention is used, self-transition probabilities (i.e. the probability of transition from some state to the same state) can exist. The second convention does not count self-transition probabilities. In this model construction the second convention is used. That is, the self-transition probabilities p_{ii} are defined to be equal to zero for every state i.

4.2 Waiting and Holding Times

Using this convention mean holding time $\overline{\tau_{ij}}$ for a pair of states i, j is the average time the process spends in state i before it makes a transition to state j. The mean waiting time $\overline{\tau_i}$ for a state i is the average time the process spends in state i before it makes a transition to any other state. The mean waiting times $\overline{\tau_i}$ in each state and the mean holding times from each cluster i to each cluster j, $\overline{\tau_{ij}}$, are also directly computable from the assignments.

5 Model Solving

Now that an appropriate model has been identified conventional solution techniques can be used to solve the model and obtain the key model characteristics such as the steady-state occupancy probability, the steady-state entry rate and the response-time distribution. Many people have discussed these methods and we summarize some key results relevant to our model here.

5.1 Occupancy Probability

If the process has been operating unobserved for some time and if it is known that the process is now making a transition, the probability that the transition is to state j is π_j. If there are n clusters each π_j must satisfy a simultaneous equation of the form

$$\pi_j = \sum_{i=1, i \neq j}^{n} \pi_i p_{ij} \tag{1}$$

(Note that under the convention used $p_{ii} = 0$ for all i.) The above equation is used in conjunction with the constraint

$$\sum_{i=1}^{n} \pi_i = 1 \tag{2}$$

to obtain n linear equations of the form

$$1 = \sum_{i=1}^{n} \pi_i (1 + p_{ij}) \tag{3}$$

The unique solution for each π_i is obtained by solving n equations of this form for each π_i. The steady-state occupancy probability of each state is evaluated from

$$\Phi_j = \Phi_{i,j} = \frac{\pi_j \overline{\tau_j}}{\overline{\tau}} \tag{4}$$

where $\overline{\tau} = \sum_{i=1}^{n} \pi_i \overline{\tau_i}$. These values for the steady-state occupancy probabilities are compared with the actual values and the relative and absolute error percentages are computed. Since data is gathered at regular intervals the actual probabilities can be computed by dividing the number of data-points assigned to each cluster by the total number of data-points. The absolute error percentage is used to validate the use of a stochastic process (Markov or Semi-Markov) to model the system. It is found that a Semi-Markov model is well-suited to model the system. This is borne out by the fact that the absolute error percentage of the computed occupancy probability, is typically less than 2 percent for the examples considered.

5.2 Steady-State entry rate

Another important parameter which the model calculates is the steady-state entry rate. This is the probability that the process is just entering state i at some time instant after the system has attained steady state and is given by

$$e_i = \frac{\pi_j}{\overline{\tau}} \tag{5}$$

5.3 Response-Time distribution

A detailed exposition of the general solution technique for Markov chains with absorbing states may be found in Trivedi [7]. Key results relevant to our model are summarized here.

The response-time distribution of the system for a particular workload is obtained by creating a dummy state i.e. if there are three clusters, there will be a total of four states. This dummy state is designated to be an absorbing state. Once the process enters the absorbing state it is destined to remain in that state. To obtain the response time distribution the model needs to provide the solver with the transition rates λ_{ij} from every state i to every state j. Obviously there will be no transitions and hence no transition rates away from the absorbing state. The transition rates needed can be computed directly from the state assignments.

Let the state occupancy probability of state j at time t be denoted as $P_j(t)$. Then $\sum_j P_j(t) = 1$ for each $t \geq 0$. For each i and $j (j \neq i)$ there is a non-negative continuous function $q_j(t)$ defined by $\lim_{h \to o}(p_{ij}(t, t+h))/h$. Also, $q_j(t) = \lim_{h \to o}(1 - p_{jj}(t, t+h))/h$. In the time-homogeneous situation $q_{ij}(t)$ is independent of t. In the time-homogeneous case the equation

$$\frac{dP_j}{dt} = \sum_{i \neq j} P_i(t) q_{ij} - P_j(t) q_j. \tag{6}$$

holds.

This equation may be used to obtain the distribution of the time taken to reach the absorbing state. If i is the absorbing state and if Y is the time taken to reach the absorbing state, the cumulative distribution function(CDF) of Y is $F_Y(t) = P_i(t)$. The equation must be solved for every state in the system using Laplace transforms.

6 Results

This section illustrates the usage of MEASURE with an analysis of data from a SUN fileserver. A static analysis of 512 data-points gathered at 5-second intervals, from a machine with a load-factor of 18, is compared with two real-time analyses. In each case the data are split into three clusters.

In the first real-time analysis the data are contained in four 640-second size windows i.e. each window contains 128 data-points. In the second case the data are contained in eight windows, each of size 320 seconds (i.e. each window contains 64 points). Data in each window are analyzed independently of data in other windows. No history of previous window-analysis is maintained.

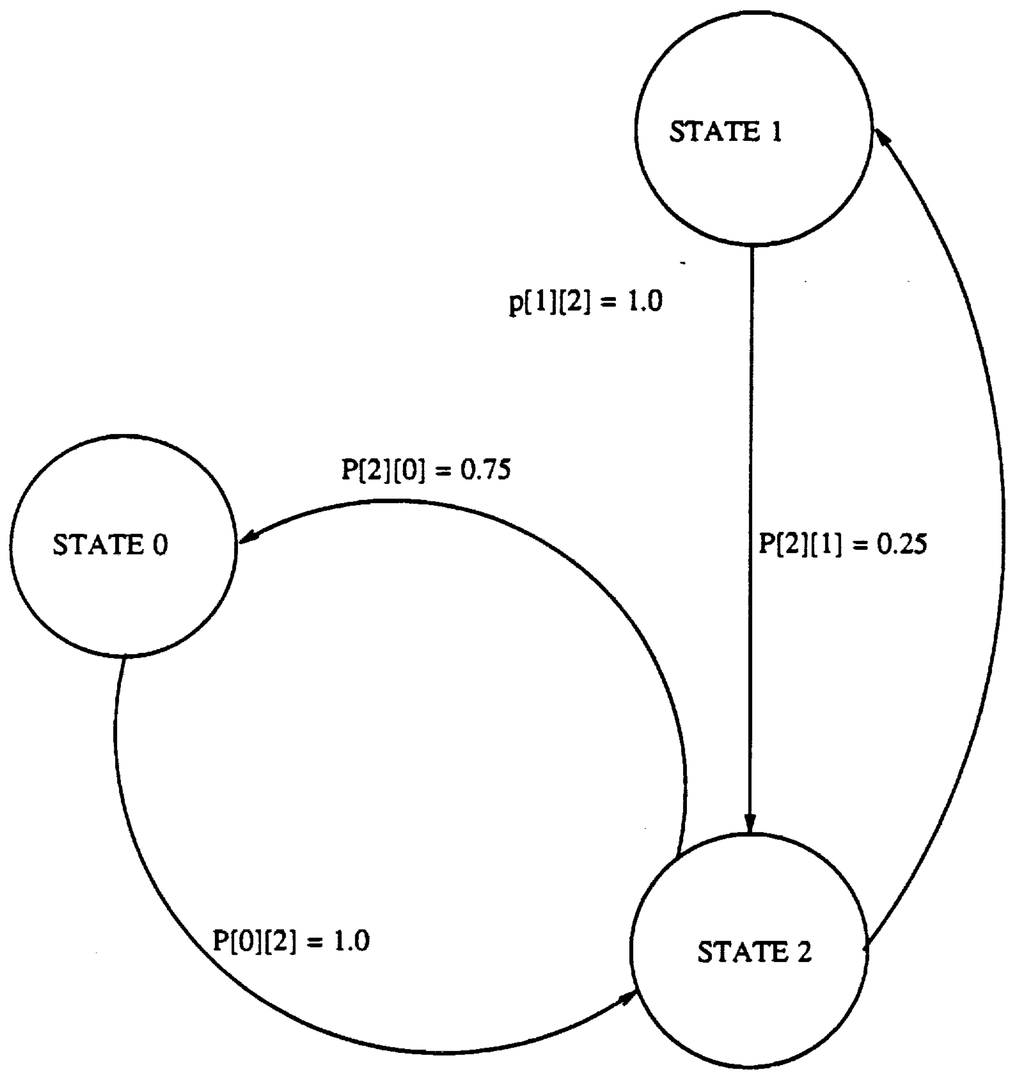
The static-analysis state-transition diagram is shown in Figure 3 and the relevant output from the program in Figure 5. A physical interpretation of this diagram is that the system is running at a high degree of efficiency. The occupancy probability of the inefficient state is very low; the transition probabilities to it are low and the transition probabilities away from it are high.

In the first real-time case an analysis of the third window detects a situation where the system is on the verge of thrashing. There is a high probability of transition to an inefficient state, which has a high occupancy probability. The program output is shown in Figure 6. This example had a load-factor of 18. In such circumstances, there is usually no CPU idle-time.

This unusual resource-usage pattern is also detected in the second case, when the model from the fifth window is analyzed. The state-transition diagram is shown in Figure 4. The effect of the decrease in window size (as compared to the 128-point windows) is to highlight the pattern even further.

There are several methods for the solution of these types of models [7],[10]. For solution purposes we define an absorption state. The system is assumed to go into the absorption state from the exit state. The response time is defined as the time required to transit from a given entry state to the absorption state. Currently we have used a modeling tool called SHARPE [8] for solution purposes. Our final aim is to incorporate the solution procedure into MEASURE. Figures 7 and 8 show the calculation of response-time distributions for a specific benchmark program. The entry-probabilities

STATE TRANSITION DIAGRAM (512 point static analysis)

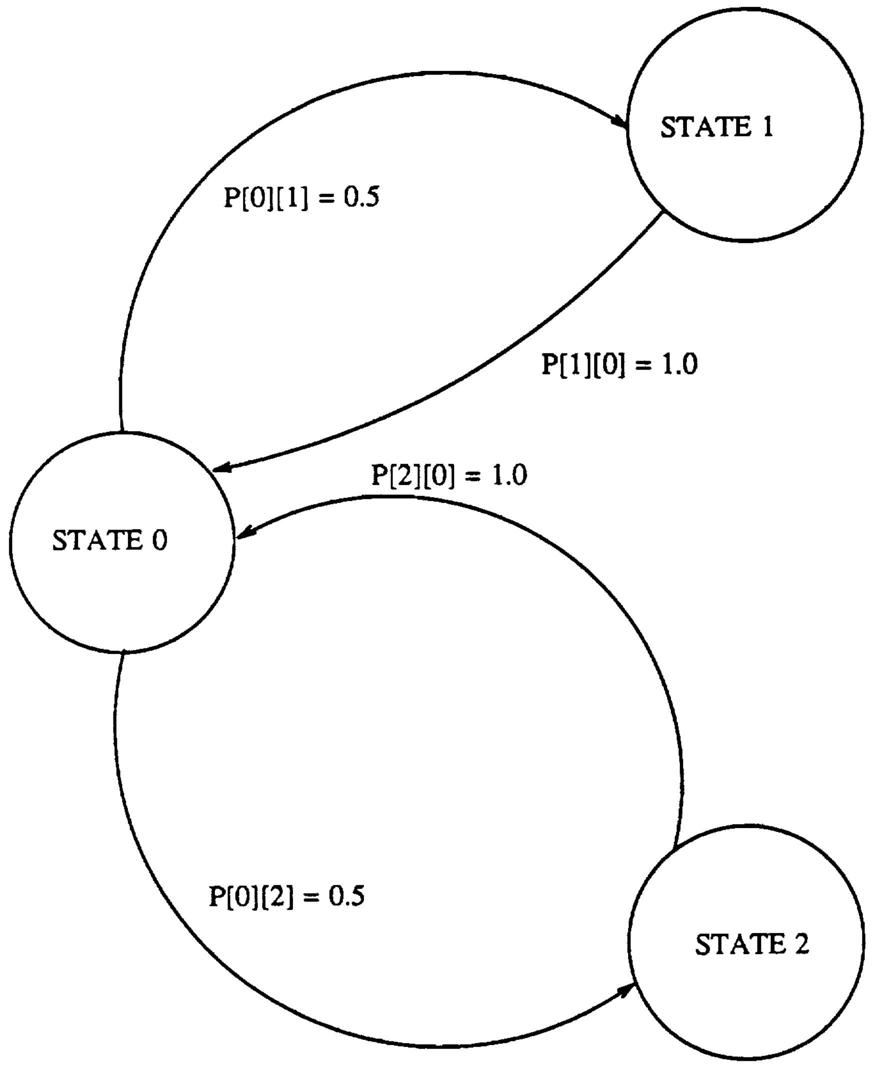


State 0 is a high-usage efficient state
State 1 is an inefficient state
State 2 is high-usage with a higher system CPU-percentage than State 0

State	0	1	2
Actual Occup. Prob.	0.562	0.033203	0.404297
Model Occup. Prob.	0.562	0.033203	0.404297

Figure 3: Transition Diagram for 512 points

STATE TRANSITION DIAGRAM(Dynamic Analysis: 64 point windows)



State 0: High-usage efficient state

State 1: System on verge of thrashing

State 2: High-usage state, higher system CPU-time percentage than State 0.

State	0	1	2
Actual Occup. Prob.	0.734375	0.218750	0.046875
Model Occup. Prob.	0.688645	0.256410	0.054945

Figure 4: Transition Diagram for a 64 point window

```
CLUSTER CENTROIDS

Cluster 0 : number of points = 288
dev.intr    sys.calls      swtchrate      cpu/usr       cpu/idle
per sec.    per sec.       per sec.        (%)           (%)
11.052       4.645         18.711         97.836        0.000

Cluster 1 : number of points = 17
dev.intr    sys.calls      swtchrate      cpu/usr       cpu/idle
per sec.    per sec.       per sec.        (%)           (%)
80.058      49.235         273.529        66.470        0.470

Cluster 2 : number of points = 207
dev.intr    sys.calls      swtchrate      cpu/usr       cpu/idle
per sec.    per sec.       per sec.        (%)           (%)
45.144      56.487         43.396         88.584        0.004

MODEL CHARACTERISTICS:

E[i] is the steady-state entry rate into state i
TAU[i] is the mean waiting time in state i
PHI[i] is the occupancy probability in state i

 E[0]: 0.004
 TAU[0]: 120.000
  Actual PHI[0]: 0.562
  Model  PHI[0]: 0.562
 Absolute Error: 0.000 percent
 Relative Error: 0.000 percent
 E[1]: 0.001
 TAU[1]: 21.250
  Actual PHI[1]: 0.033
  Model  PHI[1]: 0.033
 Absolute Error: 0.000 percent
 Relative Error: 0.000 percent
 E[2]: 0.006
 TAU[2]: 64.687
  Actual PHI[2]: 0.404
  Model  PHI[2]: 0.404
 Absolute Error: 0.000 percent
 Relative Error: 0.000 percent
```

Figure 5: Program Output for 512 points

```
CLUSTER CENTROIDS

Cluster 0 : number of points = 46
dev.intr       sys.calls      swtchrate      cpu/usr      cpu/idle
per sec.       per sec.       per sec.        (%)          (%)
18.869          9.717         18.413         97.108       0.000

Cluster 1 : number of points = 16
dev.intr       sys.calls      swtchrate      cpu/usr      cpu/idle
per sec.       per sec.       per sec.        (%)          (%)
73.250         49.000        282.187         68.187       0.500

Cluster 2 : number of points = 66
dev.intr       sys.calls      swtchrate      cpu/usr      cpu/idle
per sec.       per sec.       per sec.        (%)          (%)
42.196         64.969         39.772         90.090       0.000


MODEL CHARACTERISTICS:

E[i] is the steady-state entry rate into state i
TAU[i] is the mean waiting time in state i
PHI[i] is the occupancy probability in state i

 E[0]: 0.008
 TAU[0]: 38.333
  Actual PHI[0]: 0.359
  Model  PHI[0]: 0.344
 Absolute Error: 1.463percent
 Relative Error: 4.071percent
 E[1]: 0.005
 TAU[1]: 26.666
  Actual PHI[1]: 0.125
  Model  PHI[1]: 0.135
 Absolute Error: -1.089percent
 Relative Error: -8.718percent
 E[2]: 0.0125
 TAU[2]: 41.250
  Actual PHI[2]: 0.515
  Model  PHI[2]: 0.519
 Absolute Error: -0.373percent
 Relative Error: -0.724percent
```

Figure 6: Program Output for a 128 point window

into the states are shown in Figure 7 and the actual distribution is shown in Figure 8. The mean of the cumulative distribution function (CDF) of the response time is the average-response time of the system for the workload which is represented by the data-points.

From Figure 5 and Figure 6 it can be seen that the state-transition diagram is heavily dependent on the size of the window used to analyze the data. Figure 6 highlights the importance of the low-efficiency state by assigning higher transition probabilities to it. Since the window size is smaller this state is also assigned a higher occupancy probability. Therefore we can see that smaller windows can be used to detect unusual behavior patterns as they occur. However, as the window-size is decreased, the values of occupancy-probability calculated by the model become increasingly erroneous. This limits the smallest window-size achievable.

The response-time probability distribution function shown in Figure 4 is a new feature which supplies the user with a quantitative measure of system performance. The response-time information can help the user to predict the amount of time required for a job to complete in a given environment.

7 Conclusions

This report describes the first phase of the development of MEASURE, an integrated data analysis and model identification facility. The facility takes system activity data as input and produces as output representative behavioral models of the system in near real-time. It also enables the measurement of a wide range of statistical characteristics on the system. Initially, statistical clustering is used to identify high-density regions of resource-usage in a given environment. The identified regions form the states for building a state-transition model to evaluate system and program performance

```
markov compsys /*Specification of system to be solved*/
0 1 mu /*mu is the transition rate from state 0 to state 1*/
0 2 nu
0 3 au
1 0 gu
1 2 ru
1 3 bu
2 0 cu
2 1 fu
2 3 hu
end
0 0.9 /*The probability that the system starts up from state 0*/
1 0.1 /*is 0.9 and the probability that it starts from state 1*/
end   /*is 0.1*/

bind
mu 2/45 /*The value of mu is 2/45*/
nu 1/15
au 0
gu 3/70
ru 2/70
bu 0
cu 2/70
fu 3/35
hu 1/35
end

cdf(compsys)
end


CDF for system compsys:

    1.0000e+00 t(  0) exp( 0.0000e+00 t)
 + -1.0129e+00 t(  0) exp(-6.7290e-03 t)
 +  1.2893e-02 t(  0) exp(-1.5933e-01 t) cos 1.4949e-02t
 + -3.1851e-01 t(  0) exp(-1.5933e-01 t) sin 1.4949e-02t

mean: 1.5063e+02
variance: 2.2053e+04
```

Figure 7: Typical SHARPE input and Output

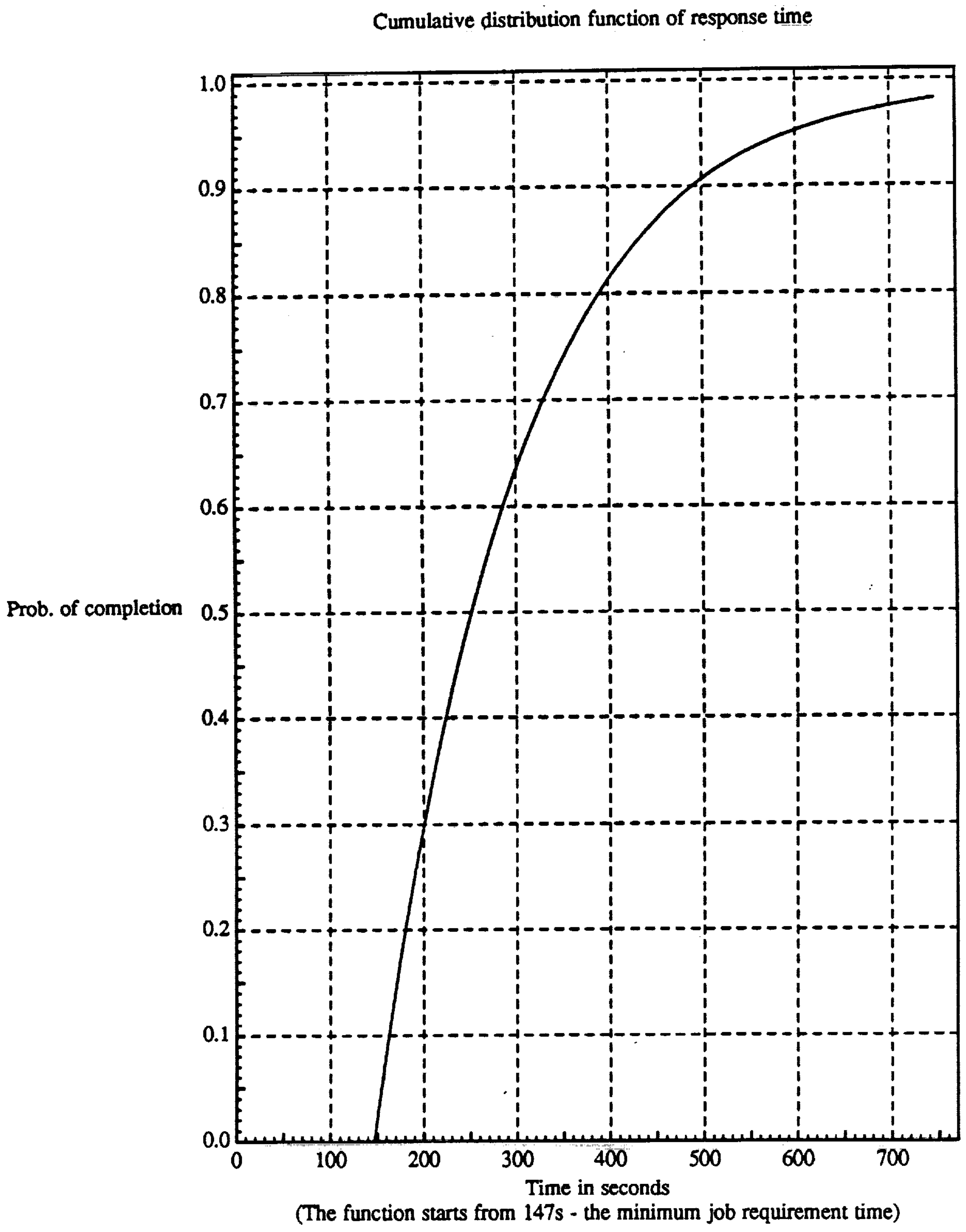


Figure 8: CDF of response time

in real-time. The model is then solved to obtain important parameters such as the response-time distribution and the mean waiting time in each state. The results provide an understanding of the resource-usage in the system under different workloads.

Extensions to the ongoing research include the incorporation of error data into the model and the integration of a model-solving module into the model-construction code. The possibility of using estimation techniques such as Kalman filtering to predict the behavior of the system will also be explored.

Appendix A Modeling Program

The routines freevec.c, lubksb.c,ludcmp.c, vector.c and nrerror.c are taken from Press [6]. The routines freevec.c, vector.c and nrerror.c are general utility routines. The routine ludcmp.c carries out the LU decomposition of a square matrix. The routine lubksb.c is an equation solver which uses the results of ludcmp.c.

```
#
echo "Welcome to MEASURE"
unalias rm
echo -n "Do you want the graph option? (Y or N) "
set q = $<
rm prob >& /dev/null
rm cidle >& /dev/null
echo " "
echo -n "What is the name of  your data-file?   "
set y = $<
cp $y rundata
echo " "
echo -n " In which file do you want the results stored?  "
set x = $<
echo " "
echo -n "How many parameters are to be analyzed?: "
set z = $<
rm trar >& /dev/null
echo " " >> trar
echo "                  CLUSTER CENTROIDS" >> trar
echo -n "                          " >> trar
@ m = $z
while ($m != 0)
echo -n "Input the first column of the parameter and the field width: "
set a = $<
echo -n "Input the parameter name: "
set fav = $<
prh $fav  >> trar
set names = ($a)
@ b = $names[1] - 1
@ c = $names[2] + 1
set tf = tmp$m
colrm 1 $b < rundata | colrm $c > $tf
@ m = $m - 1
end
echo " " >> trar
echo -n "How many points are to be analyzed?: "
set ch = $<
echo -n "Enter display-time in seconds: "
set rip = $<
dum $ch $z
@ ch = $ch + 1
cat temporary main.c > newmain.c
cat temporary proc.c > newproc.c
cat temporary kmeans.c > nkmeans.c
make >& /dev/null
wo:
clus >  $x
cat trar $x > tim
si:
grep -v PRO < tim > /dev/ttyp2
if ($q == Y) then
graph $rip
else sleep $rip
endif
cat blank > /dev/ttyp2
echo -n "Redisplay? (Y or N) "
set bull = $<
if ($bull == Y) then
goto si
endif
rm prob >& /dev/null
rm cidle >& /dev/null
echo -n "Continue with same settings? (Y or N) "
set fa = $<
if ($fa == Y) then
@ m = $z
while ($m != 0)
set tf = tmp$m
tail +$ch $tf > tem
cp tem $tf
@ m = $m - 1
end
goto wo
endif
rm trar
```

```
#include <stdio.h>
#include <malloc.h>

#define MAXLINE 100
main(argc,argv)
int argc;
char *argv[];
{
int dim,i,j,k,q;
char r[20][MAXLINE];
FILE *fopen(),*ifp;
dim = atoi(argv[1]);
printf("dim is %d\n",dim);
printf("Enter the parameter names (upto 20 different parameters) in the ");
printf("order in which they appear in the data file\n\n");
printf("Parameter names should be separated by blanks or carriage-returns\n");
for(i=1; i <=dim ; i++) {
scanf("%s",r[i-1]);
}
printf("How many points do you want to analyze?\n");
scanf("%d",&j);
printf("How many clusters do you want?\n");
scanf("%d",&k);
printf("What is the time granularity in seconds?\n");
scanf("%d",&q);
ifp = fopen("temporary","w");
fprintf(ifp,"# define DIM %d\n",dim);
fprintf(ifp,"# define NPTS %d\n",j);
fprintf(ifp,"# define CLUS %d\n",k);
fprintf(ifp,"# define TIME %d\n",q);
fclose(ifp);
}
```

```
 /* This program clusters input-data into a pre-determined number */
/* of clusters. Cluster information is then used to construct a  */
/* semi-Markov model of the system.                               */
#include <stdio.h>
int p[NPTS],q[12];
float x[NPTS][DIM],y[NPTS][DIM],s[CLUS][DIM],sas[CLUS][DIM],e[CLUS + 1];
                              /*The array s holds the centroid values.    */
main ()
{
        FILE *fopen(), *ip_1, *pr_file, *ip_2, *ip_3,*ip_4, *ip_5;
/*Data is generated by running the vmstat system command. This version */
/*of the program analyses 5 parameters provided by vmstat. These are   */
/*1.) in - (non-clock)device interrupts per second.                    */
/*2.) sy - system calls per second.                                    */
/*3.) cs - cpu context switch rate (switches/sec)                      */
/*4.) us - user time for normal and low priority processes. (% usage)  */
/*5.) id - cpu idle time (%).                                          */
/*From 4.) and 5.) the cpu usage for system activities can also be     */
/*calculated.                                                          */
        extern   float x[ ][DIM];
        int i,i1,i2,i3,i4,i5,n;
        ip_1 = fopen ("tmp1" , "r");  /*The raw data is processed by*/
/*the executable file runs*. It extracts the fields to be analyzed. */
/*The executable code for this program is called from runs*.        */
        if (ip_1 == NULL)
                printf ("***tmp1  could not be opened.\n");
        for (i =1; i < (NPTS + 1); ++i)
        {
                fscanf(ip_1, " %d \n", &i1 );
                x[i-1][0] = i1;
        }
        fclose (ip_1);
        ip_2 = fopen ("tmp3" , "r");
        if (ip_2 == NULL)
                printf ("***tmp3  could not be opened.\n");
        for (i =1; i < (NPTS + 1); ++i)
        {
                fscanf(ip_2, " %d \n", &i2 );
                x[i-1][1] = i2;
        }
    fclose (ip_2);
        ip_3 = fopen ("tmp5" , "r");
        if (ip_3 == NULL)
                printf ("***tmp5  could not be opened.\n");
        for (i =1; i < (NPTS + 1); ++i)
        {
                fscanf(ip_3, " %d \n", &i3 );
                x[i-1][2] = i3;
        }
        fclose (ip_3);
        ip_4 = fopen ("tmp7" , "r");
        if (ip_4 == NULL)
                printf ("***tmp7  could not be opened.\n");
        for (i =1; i < (NPTS + 1); ++i)
        {
                fscanf(ip_4, " %d \n", &i4 );
                x[i-1][3] = i4;
        }
        fclose (ip_4);
        ip_5 = fopen ("tmp8" , "r");
        if (ip_5 == NULL)
                printf ("***tmp8  could not be opened.\n");
        for (i =1; i < (NPTS + 1); ++i)
        {
                fscanf(ip_5, " %d \n", &i5 );
                x[i-1][4] = i5;
        }
        fclose (ip_5);
        transf(NPTS);
        proc(CLUS,NPTS);
}
```

```
 #define DIM 5
#include <math.h>
transf(m)
int m;
{
float t,u,v,q,s;
int i,k;
extern float x[ ][DIM];
extern float y[ ][DIM];
for(k=0; k < DIM; ++k) {
  t = 0.0;
  u = 0.0;
  for(i=0; i < m; ++i) {
    v  = x[i][k];
    t = t + v;
    u = u + v*v;
    }
    q = t/m;
    s = 0;
    if((u - t*q) > 0.0 )
    s = sqrt((m - 1.0)/(u - t*q));
    for(i =0; i < m; ++i)
      y[i][k] = s*(x[i][k] - q);
      }
      return;
}
```

```
 #include <malloc.h>
#include <stdio.h>
#define DIM 5
#define CLUS 3
#define NPTS 500
proc(n,m)
int n,m;
{
        FILE *fopen(),  *pr_file, *iy_file,*ip;
        int i,k,j,jim,jaw,lst,nz,nn,log,cam,indx[12],last_st,fst_st,fndx[12],ch,wu;
        float phi[CLUS + 1],pro[CLUS][CLUS],tw[NPTS],dw,toff,g,h;
         float a[19][38],pi[19],sk[19],tau[CLUS + 1],ct[CLUS + 1],d,f,b[19],**aa;
        float **pr,dummy,tht_sr[CLUS +1][CLUS +1],mtau;
        float mph[CLUS + 1];
        extern int p[ ], q[];
        extern float x[ ][DIM],s[ ][DIM],sas[ ][DIM],e[ ];
        k = 0;
        cam = 1;
        for (i= 1; i<(m +1); ++i)
        {
                tw[i-1] = 5.0*i;
                dw = 5.0;
                ++k ;
                if (k > (NPTS/CLUS)) {
                        k = 0;
                        ++cam;
                        }
                p[i-1] = cam;
        }
        kmeans(NPTS,CLUS);
        ip = fopen("tme","a");
        for(i = 0; i < m; ++i) {
        fprintf(ip,"%f\n",(float)p[i]);
        }
        fclose(ip);
        nz =n;
        for (i=0; i < nz; ++i)
        {       tau[i] = 0.0;
                ct[i] = 0.0;
                phi[i]=0.0;
                mph[i] = 0.0;
                printf(" CENTROIDS \n ");
                printf("dev.intr   sys.calls swtchrate  cpu/usr cpu/idle \n \n");
                for (j=0; j <nz ; ++j){
                        pro[i][j] = 0;
                }
                for (j =0; j < DIM; ++j)
                    printf(" %f ", sas[i][j]);
                printf(" \n");
        }
```

```
    jaw = 1;
         toff= tw[jaw - 1];
         lst = p[jaw - 1];
         toff= toff - dw;
         while(jaw < m) {
                 i = jaw + 1;
                 phi[lst-1] = phi[lst-1] + dw;
                 k = p[i - 1];
                 tau[lst -1] = tau[lst -1] + dw;
                 if(k != lst) {
                 pro[lst-1][k-1] = pro[lst-1][k-1] + 1;
                 ct[lst -1] += 1.0;
                 }
                 lst =k;
                 jaw =i;
         }
         k = p[m-1];
         tau[k-1] += dw;
         ct[k -1] += 1.0;
         phi[k-1] = phi[k-1] + dw;
         h = tw[m-1] - toff;
         if(h <= 0) h= -1;
         for(i=0; i < nz; ++i)
                 phi[i] = phi[i]/h;
         nn = nz + 1;
         for(i=0; i < nz; ++i) {
                 toff = 0.0;
                 for(j=0; j < nz; ++j)
                         toff = toff + pro[i][j];
                 g = g + toff;
                 if(toff == 0.0)
                         toff = 1.0;
                 for(j=0; j < nz; ++j){
                         pro[i][j] = pro[i][j]/toff;
                         printf(" PRO[%d][%d]: %f \n", i,j,pro[i][j]);
                 }
         }
         for(i =1; i <= nz; ++i)
 {
 for(j = 1; j <= nz; ++j) {
                         a[i][j] = pro[j-1][i-1];
                         if(i != j) a[i][j] = a[i][j] + 1.0;
 }
 }

         aa =(float **) malloc((unsigned) 12*sizeof(float*));
         for(i = 1; i <= CLUS; i++)
 { aa[i] = a[i];
   b[i]= 1.0;}
         ludcmp(aa,CLUS,indx,&d);
```

```
        lubksb(aa,n,indx,b);
        for(i=1; i<=CLUS; ++i)
                printf(" PI[%d]: %f \n",i-1, b[i]);
        for(i=0; i < nz; ++i) {
                        tau[i] = tau[i]/ct[i];
        }
        mtau = 0.0;
        for(i=0; i < nz; ++i) {
                     mtau += b[i+1]*tau[i];
        }
        for(i=0; i<nz; ++i) {
                mph[i] = (b[i +1] *tau[i])/mtau;
                e[i] =   (b[i +1])/mtau;
                printf(" E[%d]: %f \n", i, e[i]);
                printf(" TAU[%d]: %f \n", i, tau[i]);
                printf("  Actual PHI[%d]: %f \n", i, phi[i]);
                printf(" Model   PHI[%d]: %f \n", i, mph[i]);
                for(j=0; j<nz; ++j) {
                        printf(" PRO[%d][%d]: %f \n", i,j,pro[i][j]);
                }
        }
        return;
}
```

```
/*The function kmeans actually clusters the given data.   */
/*It identifies the cluster to which an individual point  */
/*belongs. The algorithm uses the "Sum-of-Squared-Distance*/
/*criterion. It seeks to minimize the sum of the squares  */
/*of the distances between members of clusters and their  */
/*centroids. The function computes the final values of the*/
/*centroids and the sums of squares of distances for the  */
/*individual clusters.                                    */
/*kmeans is called from the function proc.                */
#include <stdio.h>
#define DIM 5
kmeans(m,n)   /*m denotes the number of points to be clustered.*/
              /*n denotes the number of clusters.              */
int m,n;

{
        int r,u,v,w,i,it,j,k;
        extern float x[ ][DIM],s[ ][DIM],sas[ ][DIM],e[ ];
        extern float y[ ][DIM];
        extern int p[ ], q[];
        float f,t,h,a,b,d,g;
        for(j=0; j < n; ++j)
        {
                q[j] = 0;
                e[j] = 0;
                for(k=0;k < DIM; ++k)
                        s[j][k]=0;
        }
        for(i=0;i<m; ++i) {
                r = p[i];
                if(r<1 ||r >n)
                        return;
                q[r-1] = q[r-1] + 1;
                for(k=0;k < DIM; ++k)
                        s[r-1][k]=s[r-1][k] + y[i][k];
        }
        for(j=0;j<n; ++j) {
                r = q[j];
                if(r == 0)
                        return;
                f =1.0/(float)r;
                for(k=0;k < DIM; ++k)
                        s[j][k] = s[j][k]*f;
        }
        for(i=0;i<m; ++i){
                r =p[i];
                f =0.0;
                for(k=0;k<DIM; ++k) {
                        t = s[r-1][k] - y[i][k];
                        f = f + t*t;
                }
                e[r-1] = e[r-1] +f;
        }
        d = 0.0;
        for(j=0; j<n; ++j)
                d = d + e[j];
i =0;
        it=0;
        while(it < m) {
        i = i +1;
        if(i>m)
                i = i-m;
        r = p[i-1];
        u = q[r-1];
        if (u <= 1)
                continue;
        h = (float)u;
        h = h/(h-1.0);
        f =0.0;
        for(k=0;k<DIM; ++k) {
                t = s[r-1][k]-y[i-1][k];
                f = f +t*t;
        }
        a = h*f;
        b = 1.0e20;
        j = 0;
        while( j<n)  {
                j = j + 1;
                if(j==r)
                        continue;
                u = q[j-1];
                h = (float)u;
                h = h/(h +1.0);
                f = 0.0;
                for(k=0;k<DIM; ++k)
                {
                        t = s[j-1][k] -y[i-1][k];
                        f = f +t*t;
                }
                f = h*f;
                if (f > b)
                        continue;
                b =f;
                v =j;
                w =u;
        }
        if  (b > a) {
        ++it;
        }
        else {
            it =0;
            e[r-1] = e[r-1] - a;
            e[v-1] = e[v-1] + b;
            d = d - a + b;
            h = (float)q[r-1];
            g = (float)w;
            a = 1.0/(h - 1.0);
            b = 1.0/(g +1.0);
            for(k =0; k < DIM; ++k)
            {
                    f = y[i-1][k];
                    s[r-1][k] = (h*s[r-1][k] - f)*a;
                    s[v-1][k] = (g*s[v-1][k] + f)*b;
            }
       p[i-1] = v;
            q[r-1] = q[r-1] -1;
            q[v-1] = q[v-1] +1;
           }
           }
            for(i =0; i< n; ++i)
            printf("q[%d] is %d\n",i,q[i]);
            for(i=0;i<m; ++i) {
                    r = p[i];
                    if(r<1 ||r >n)
                            return;
                    for(k=0;k < DIM; ++k)
                            sas[r-1][k]=sas[r-1][k] + x[i][k];
            }
            for(j=0;j<n; ++j) {
                    r = q[j];
                    if(r == 0)
                            return;
                    f =1.0/(float)r;
                    for(k=0;k < DIM; ++k)
                            sas[j][k] = sas[j][k]*f;
            }
            return;
    }
```

```
#include <stdio.h>
#include <malloc.h>

void free_vector(v,nl,nh)
float *v;
int nl,nh;
{
free((char*)(v +nl));
}


#include <malloc.h>
#include <stdio.h>
void nrerror(error_text)
char error_text[];
{
void exit();

fprintf(stderr,"%s\n",error_text);
exit(1);
}

#include <malloc.h>
#include <stdio.h>
float *vector(nl,nh)
int nl,nh;
{ float *v;
 v =(float *)malloc((unsigned) (nh -nl+1)*sizeof(float));
if (!v) nrerror("allocation failure");
return (v-nl);
}
```

Appendix B User Guide

The data-analysis facility MEASURE is intended to run on acolor SUN 3/110 or similar machine. It takes as input a file containing system activity measurements such as the CPU usage, the number of context switches per second, the device interrupts per second etc. The file must not contain any non-numerical text. The output consists of a state transition model for the measured system and various system performance parameters obtained by solving the model. Statistical cluster analysis is used to generate the states for building the state-transition model. The state-transition model can be displayed graphically, utilizing a package written in SUN CGI.

Appendix B.1 Set-Up Procedure

The MEASURE tape contains a file '.sunview' and a directory 'demo'. Copy the sunview file into the user's home directory. After copying in the 'demo' directory, type 'suntools -i' from the home directory and wait for the windows to to be displayed. In each of the windows 'cd' to the demo directory. In the rightmost window type 'tty'. The system will respond with '/dev/ttyp2' or '/dev/ttyp3'. One of the files provided in the directory 'demo' is a shell-script called 'sts'. Search for the string 'ttyp' in 'sts' and change the ttyp number to match the tty number in the rightmost window. There are two occurrences of the string; both must be changed. This completes the setup procedure. Prior to running MEASURE, transfer all the data-files to be analyzed to the directory 'demo'. Note that you must 'cd' to the directory 'demo' in all 3 windows before running MEASURE.

Appendix B.2 Running Measure

To start up MEASURE type 'nekey' in the leftmost window. This provides a color key for the graphical display If there is an object code incompatibility the file 'nekey.c' can be recompiled into 'nekey' using the command

```
cc -o nekey nekey.c -lcgi -lsunwindow -lpixrect -lm
```

Currently the key assumes that 5 parameters are being analyzed and that the 5th parameter is the percentage of time that the CPU is idle. To start running MEASURE type in 'sts' in the middle window and hit the carriage-return.

Two sample data files are provided with this tape. The first, 'strip', contains 512 measurements and the second ,'bigdata', contains 3952 measurements. Both were created using the system command 'vmstat' with a measurement interval of 5 seconds. For the data-set 'strip' Figure 9 shows a typical terminal interaction together with the user responses. To start with MEASURE prompts the user to choose between a graphical-textual display and a purely textual display. Type only Y or N (not y or n) in response to the query. Currently the display assumes that 5 clusters are to be constructed. Next the user will be prompted for the name of the data-file. Type the name (in this case strip) and hit the carriage-return. The tool makes its own internal copies of any user-supplied data-file. The user's copy of the data-file is not altered in any way. Next, the name of the file into which numerical results are directed is entered by the user (in this case the file is 'hode'). The user will then be prompted for the number of parameters to be analyzed (enter integers only). The next prompt will be to input the starting column of each parameter and its associated field width. For

```
Welcome to MEASURE
Do you want the graph option? (Y or N) N

What is the name of  your data-file?   strip

 In which file do you want the results stored?  hode

How many parameters are to be analysed?: 5
Input the first column of the parameter and the field width: 58 4
Input the parameter name: dev
Input the first column of the parameter and the field width: 62 4
Input the parameter name: int
Input the first column of the parameter and the field width: 66 4
Input the parameter name: cs
Input the first column of the parameter and the field width: 70 3
Input the parameter name: sys
Input the first column of the parameter and the field width: 76 3
Input the parameter name: use
How many points are to be analysed?: 100
Enter display-time in seconds: 20
No of points is 100
How many clusters do you want?
5
What is the time granularity in seconds?
5
```

Figure 9: Typical MEASURE run

example if a parameter begins in column 42 and has a width of 3, type 42 3 and hit the carriage-return. Then enter the name of the parameter. The parameters can be entered in any order. The interface will then prompt the user for the number of points to be analyzed. The display-time in seconds is entered next; this parameter governs the time for which the results (graphical and numerical) are displayed. The user must also specify the number of clusters to be identified and the time-interval between measurements in the user-supplied data-file (the 'time-granularity'). In the example, 100 measurements separated by 5 second intervals are analyzed for each run i.e., a new model is created and solved every 500 seconds. After displaying the analysis results, the interface will prompt the user to choose between redisplaying the results or carrying out a new analysis. If the next analysis is to be carried out with the same settings for all parameters, the user can select the 'Y" option when prompted by "Continue with same settings? "

References

[1] Hsueh, M.C., *Measurement-Based Models*, Ph.D. thesis, University of Illinois at Urbana-Champaign, 1987.

[2] Castillo, X., *A Comparable Hardware/Software Reliability Prediction Model*, Ph.D. thesis, Carnegie-Mellon University, 1981.

[3] Iyer, R.K., Rossetti, D.J., and Hsueh, M.C., "Measurement and Modeling of Computer Reliability as Affected by System Activity," *ACM Transactions on Computer Systems*, vol. 4. no.3, pp. 214-237, August, 1986.

[4] Spath, Helmuth, *Cluster analysis algorithms*, Ellis Horwood, 1980.

[5] Artis, H.P., "Workload Characterization Using SAS PROC FASTCLUS," *Workload Characterization of Computer Systems and Computer Networks*, North-Holland,1986.

[6] Press, W.H. et. al., *Numerical Recipes in C*, Cambridge University Press, 1988.

[7] Trivedi, K.S., *Probability and Statistics with Reliability, Queuing, and Computer Science Applications*, Prentice-Hall,1982.

[8] R.A. Sahner and K.S. Trivedi. *SHARPE Introduction and Guide for Users*, Dept. of Computer Science, Duke University, 1986.

[9] S. J. Leffler et. al. *The Design and Implementation of the 4.3BSD UNIX Operating System*, Addison-Wesley, 1989.

[10] R.A. Howard, *Dynamic Probabilistic Systems Vols 1. and 2*, 1971.

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS None
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) UILU-ENG-90-2207 CSG-121	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 NASA
6c. ADDRESS *(City, State, and ZIP Code)* 1101 W. Springfield Ave. Urbana, IL 61801	7b. ADDRESS *(City, State, and ZIP Code)* NASA Ames Research Ctr. Moffett Field, CA 94035 NASA Langley Rs.C. Hampton, VA 23665
8a. NAME OF FUNDING/SPONSORING ORGANIZATION NASA 8b. OFFICE SYMBOL *(If applicable)*	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER NASA NCA 2-301 NASA NAG 1-613
8c. ADDRESS *(City, State, and ZIP Code)* NASA Ames Res. Ctr. Moffett Field, CA 94035 NASA Langley Res. Ctr. Hampton, VA 23665	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE *(Include Security Classification)*
MEASURE: An Integrated Data-Analysis and Model-Identification Facility

12. PERSONAL AUTHOR(S)
Jaidip Singh and Ravi K. Iyer

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT *(Year, Month, Day)*	15. PAGE COUNT
Technical	FROM ______ TO ______	1990 March	38

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
FIELD	GROUP	SUB-GROUP	performance measurement, data-analysis, real-time modeling, statistical clustering, state-transition model

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*
This report describes the first phase of the development of MEASURE, an integrated data analysis and model identification facility. The facility takes system activity data as input and produces as output representative behavioral models of the system in near real-time. In addition, a wide range of statistical characteristics of the measured system are also available. The usage of the system is illustrated on data collected via software instrumentation of a network of SUN workstations at the University of Illinois. Initially, statistical clustering is used to identify high-density regions of resource-usage in a given environment. The identified regions form the states for building a state-transition model to evaluate system and program performance in real-time. The model is then solved to obtain useful parameters such as the response-time distribution and the mean waiting time in each state. A graphical interface which displayes the identified models and their characteristics (with real-time updates) has also been developed. The results provide an understanding of the resource-usage in the system under various workload conditions. This work is targeted for a testbed of UNIX workstations with the initial phase ported to SUN workstations on the NASA.

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

19. Abstract, continued

Ames Research Center Advanced Automation Testbed.