A major problem in the evaluation of the performance of a multiprogrammed computer system is the development of an accurate description of its normal workload. This paper formulates the workload characterization problem for a computing environment and presents three types of simplified models for the system workload. The probabilistic models of the workload presented here can be validated with respect to the real workload and are easy to use in a performance evaluation study. The results of a study of the workloads on the Univac 1108 computer at the Computer Science Center of the University of Maryland are also presented.

AN APPROACH TO THE WORKLOAD CHARACTERIZATION PROBLEM

A. K. Agrawala,
J. M. Mohr,
and
R. M. Bryant
University of Maryland

Introduction

Over the years, as the architecture of computer systems has become more complex and system capabilities have improved, so users have become very sophisticated and have learned to apply their systems to a wide variety of tasks.[1] However, little effort has been devoted to designing system workload models for characterizing the user requirements accurately for evaluation, scheduling, or tuning studies. Indeed, only a few techniques are available for workload characterization.[2,3]

The basic methodology used in performance evaluation studies consists of observing (and making measurements on) a system or its model while it is processing a workload.[4] (We call the workload used in any performance evaluation study a "test workload.") The results and conclusions of the study are based not only on the measurements taken but also on the test workload. For the study results to be applicable to the system, the test workload must be "representative" of the system workload.

The form of the test workload depends on the techniques to be used in the evaluation study. When one is studying the actual system using benchmarks or synthetic jobs, the test workload is the set of programs or jobs. In a trace-driven modeling study the test workload consists of the system trace. When analytical models of the system are used,[8] the test workload may be in the form of distributions of arrival time and service times. If a simulation model[9] of the system were to be employed, the test workload might consist of a job script in a form compatible with the simulator used.

Although one can easily decide on the form of the test workload which is compatible with the evaluation techniques used, ascertaining the representativeness of a test workload chosen for a study is not as easy. In many studies this problem is ignored by using a standardized test workload.[10]

An accurate knowledge of the system workload is required before the representativeness of any test workload can be established. In this paper we present a set of models which may be used to characterize the system workload. These models are constructed in a statistical framework which allows us to use statistical techniques to validate a model with respect to a system workload. A validated workload model can be used not only in establishing the representativeness of a given test workload but can also be used in guiding the selection of an appropriate test workload. For example, the parameters for a synthetic job may be derived from a system workload model.[11] If, on the other hand, a monitor is placed on the system while the system is in normal operation, we can determine statistically how representative of the system workload were the jobs run during the observation period.

In this paper, we formulate the workload characterization problem for a general computing environment and present three types of models for the workload. One of these models was used to construct a workload model of the Univac 1108 system at the University of Maryland. The results of this study are also presented.

Problem Formulation

A computer system may be treated as a passive device which carries out data processing tasks in response to the requests made by its users. To meet a user request, a computer system must do processing which may involve the scheduling and the execution of a number of tasks. The workload is considered to be only that processing which is specifically required to satisfy the user request. All other work—e.g., swapping, scheduling, etc.—is considered to be system overhead and hence not a part of the workload.[12]

A computer system consists of a number of hardware and software resources. Each user request* requires a certain amount of these resources in order to be satisfied. A user request R may thus be characterized by a vector X. The jth component X(j) of this vector specifies the amount of service requested from the jth resource to service this request. For example X(1) may be the number of memory blocks required and X(2) the CPU time required to meet this request.

In addition to the vector X, a request has several other characteristics. It is made at a time instant T, and, in a distributed environment, from a location L. The request may also have a flag F associated with it, which may indicate whether it is a timesharing, a batch, or a real-time processing request.

Therefore, any user request R made to a distributed computing system may be characterized by the quadruple (T, L, X, F). Each of these terms has to be specified in appropriate units or codes.

Note that each request or workstep is a complete specification to the computer system of a task to be carried out with no further interference from the user. In other words, once the user has initiated a request, the system can schedule and complete the necessary tasks.

The system workload of a general-purpose computer system usually consists of a large number of worksteps. If a system is handling a repetitive workload in which the same set of requests is made every week or every month, then the problem of workload characterization could be solved simply by examining the set of requests made in one cycle. But in most instances the workload is not repetitive. There may, however, be trends in the workload, which occur due to increasing use of the system or which are periodic. For example, in a university environment it is common to see the load increase during a semester, drop rapidly during the mid-semester break, and then start climbing at the start of the new semester.

In a probabilistic model of the workload, the elements T, L, F are treated as random variables and X is treated as a vector of random variables. A complete description of this model, therefore, consists of a joint probability distribution function defined over the space spanned by the elements of the request R. Since one of the descriptors of R is time, we may treat the model as a stochastic process, and may describe it by the usual characterization of stochastic processes. Clearly such a description is likely to be very complex and may be unmanageable. With this model as the base, let us consider a few simpler types of models.**

Type A Model. In this model, we ignore two of the four characteristics of the request and treat it as R_A = (X, F). Thus, we are ignoring the effects of time and spatial distribution of the requests. To make this type of model useful, we have to talk about a fixed period of time, like a day, and one or all of the nodes of a distributed environment. By eliminating the time dependence, we can characterize the workload by a probability distribution function over (X, F).

Type B Model. In this model, we do take the time characteristics of the requests into account but ignore the source of the requests, L. Thus a request is characterized by a triple as R_B = (T, X, F). Note that these requests can now be treated as a point process (which is a special form of a stochastic process[13,14]). This model allows us to consider scheduling questions for a nondistributed system.

Type C Model. This is the overall model considered above, in which the requests are characterized by the quadruple R. If the location is allowed to have only a few discrete values, then this model may be considered to be a multivariate point process[7] over time and space. This is a rather safe assumption, since a computer network will always have a discrete number of nodes.

The use of probabilistic models to characterize the system workload, as discussed above, provides us with a conceptual framework. For this framework to be useful in practice, we have to consider detailed models of various types. The starting point for such a model will usually be a set of data observed on the real system. This data may consist of the quadruples for each user request. The models may have to be constructed and validated based on this data. A few of these models are discussed below.

*The term "workstep" has also been used[4] for these requests.

**The concept of types of workload models was also used by Hellerman and Conroy,[16] but they formulated only two types of models, which are essentially variants of our type B model.

Type A Model— A Mixture Distribution Approach

Even when we are considering the requests as $R_A = (X, F)$, a probabilistic description of these requests is likely to be very involved. We may, however, observe that while a large variation exists in the request population, this population usually consists of a small number of relatively "homogeneous" classes. For example, if we consider an executive control command as a request, all commands leading to FORTRAN compilations are likely to have distinct similarities and may be considered to form a class. We may, therefore, consider the probability distribution function $P(R_A)$ as a mixture distribution,

$$P(R_A) = \sum_{i=1}^{m} P(R_A/c_i)\, P(c_i) \qquad (1)$$

where c_i, $i = 1, 2, \ldots, m$, are the m classes in the population. The similarities in the requests from a class may imply a unimodal distribution for $P(R_A/c_i)$, the class conditional probability distribution function of R_A for the requests coming from the class c_i.

The advantage of this formulation is that it allows us to represent a very complex probability distribution in terms of several simpler distributions. To carry this out in practice, we have to determine the form and parameters of the conditional distributions, the number of classes, and the class probabilities from the measurements available on a system. Given these measurements, the problem of finding the parameters of the model then becomes that of a decomposition of mixture distributions, for which a number of pattern recognition techniques may be used.[17,18] If we can assume a functional form for the conditional densities, the problem becomes that of function fitting and parameter estimation.

When no reasonable assumptions about the conditional probability distributions can be made, we may use non-parametric learning techniques. For example, a clustering technique[10] may be used to get the clusters from the observed set of requests. We used this technique in a study of the workload of the Univac 1108, Exec 8 system at the University of Maryland. The results of this study are described in the following section.

Workload Model for a Univac 1108: An Example

As an example of using the Type A model proposed here we describe the results of a study made on the Univac 1108 computer installation at the University of Maryland, College Park. As shown in Figure 1, the system employs dual processors with 262K words of memory and a full complement of secondary storage such as disks, drums, and tape drives. It supports four remote batch sites as well

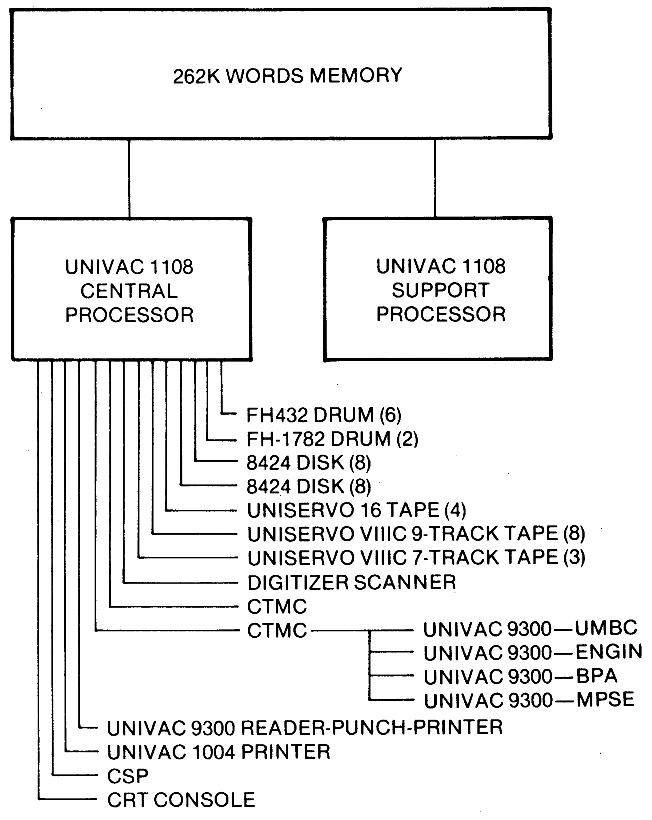


Figure 1. Configuration of the University of Maryland Univac 1108

Table 1. Summary of the information about each job available on the log tapes

1.	RUNID
2.	TYPE OF JOB, BATCH, TIMESHARING, REMOTE JOB, ETC.
3.	CPU TIME (SECONDS)
4.	EXECUTIVE REQUEST AND CONTROL CARD TIME (SECONDS)
5.	NUMBER OF 512 WORD CORE BLOCKS USED
6.	I/O ACCESSES, 432 DRUM
7.	I/O ACCESSES, 1782 DRUM
8.	I/O ACCESSES, 8440 DISK
9.	I/O ACCESSES, 8414-8425 DISK
10.	I/O ACCESSES, UNSERVO-16 9-TRACK TAPE
11.	I/O ACCESSES, F2 DRUM (FASTRAND)
12.	I/O ACCESSES, 8C-8C9 TAPES
13.	NUMBER OF JOB STEPS EXECUTED
14.	RUN LENGTH IN WALL CLOCK TIME (SECONDS)
15.	CORE BLOCKS * (STANDARD UNITS OF PROCESSING)
16.	NUMBER OF ASSIGNMENTS OF TAPES
17.	NUMBER OF ASSIGNMENTS OF FILES

*A Standard Unit of Processing (SUP) is a unit devised to determine as nearly as possible the elapsed time required to perform a unit of work in a serial environment on a unit processor with no overlap of I/O and CPU operations.

as more than 60 timesharing terminals. Used by the students and faculty for education and research, the system typically processes a total of 1000 batch and timesharing jobs a day during the middle of a semester. Towards the end of a semester it may process as many as 1500 jobs a day and maintain timesharing service on all the lines coming into the system.

Since this system is very heavily used most of the time, we needed a data collection technique that would be inexpensive and could be used over long periods of time without degrading system performance. The Univac Exec 8 operating system collects a great deal of information about each job and writes this information on the system log tapes. If we treat a job as a workstep, the workload characterization study may use the system log as the input data, since the log contains the resource utilization information for each user request.[19] Furthermore, we have found that the values recorded on the Univac 1108 log tapes are reasonably accurate and seem to be as valid as those obtainable through the use of a software monitor.[20] A summary of some of the information we obtained about each job from the log tapes is shown in Table 1.

For this study we used the log tapes for three periods—May 27, 1974 (Memorial Day); February 24-26, 1974; and May 1-2, 1974, on which the system was lightly, moderately, and heavily loaded respectively. The gross statistics of jobs in these three periods are shown in Table 2. On May 27, 1974, the university was closed, hence the system was

Table 2. Statistics from tapes

	MEMORIAL DAY MAY 27	END OF SEMESTER MAY 1-2	MID SEMESTER FEB. 24-26
NO. OF JOBS RUN	427	2877	3028
NO. OF JOBS RUN/DAY	427	1438	1009
CPU TIME (MSEC)			
MIN	0	0	0
MAX	3.4×10^7	2.8×10^6	4.3×10^6
MEAN	7.5×10^4	2.2×10^4	3.5×10^4
STVD	4.4×10^5	1.0×10^6	1.6×10^5
CONTROL CARD & EXECUTIVE REQUEST CHARGES			
MIN	0	17	22
MAX	1.1×10^6	3.3×10^6	4.2×10^6
MEAN	4.5×10^4	2.5×10^4	3.2×10^4
STDV	9.4×10^4	9.1×10^4	1.4×10^5
CORE BLOCKS (512 WORDS)			
MIN	0	1	1
MAX	123	176	182
MEAN	29	34	35
STDV	25	27	30
NO. OF JOB STEPS			
MIN	0	0	0
MAX	234	203	186
MEAN	13	7.5	8.3
STDV	25	14	15
WALL CLOCK TIME (SECONDS)			
MIN	1	0	0
MAX	2.5×10^4	1.2×10^4	1.8×10^4
MEAN	1.3×10^3	8.7×10^2	4.6×10^2
STDV	2.4×10^3	1.5×10^3	1.8×10^3

lightly loaded, processing only 427 jobs that day. The period February 24-26, 1974, occurred during the middle of a week in the middle of a semester, hence the system had a moderate load (1009 jobs per day). Because the projects and assignments in a number of courses are due at the end of a semester, the May 1-2, 1974, period shows a very heavy load (1438 jobs per day).

The 17 features about a job shown in Table 1 were extracted and analyzed for the jobs in these three periods. The first step in the data analysis was to plot the histograms for various features. One such histogram is shown in Figure 2. From the histogram we see that very little of the structure of the distribution is visible. More information about the distribution can be obtained from the log histogram shown in Figure 3. These histograms, however, display the distribution of only a single component of the vector x. If we consider two components at a time, we can get two dimensional scatter plots (see sample in Figure 4). The multimodal nature of the distributions is not visible in these figures. Therefore, higher dimensional techniques such as clustering are required.

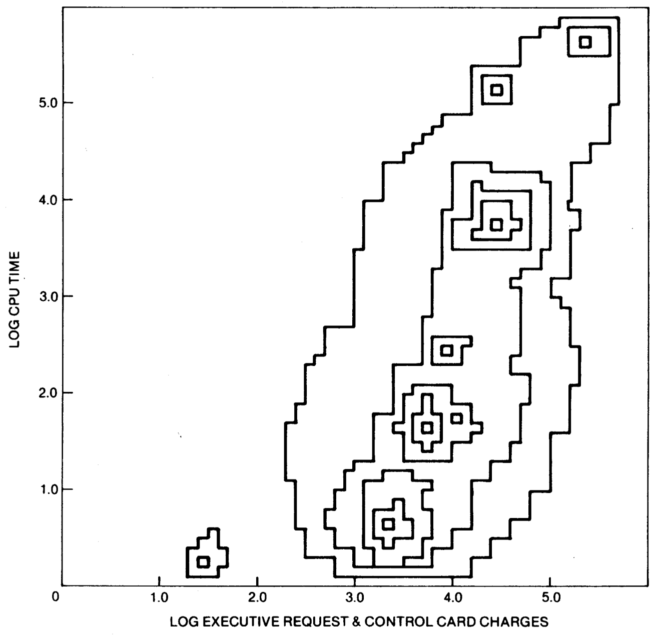


Figure 4. Scatter plot of CPU time vs. executive request and control card charges

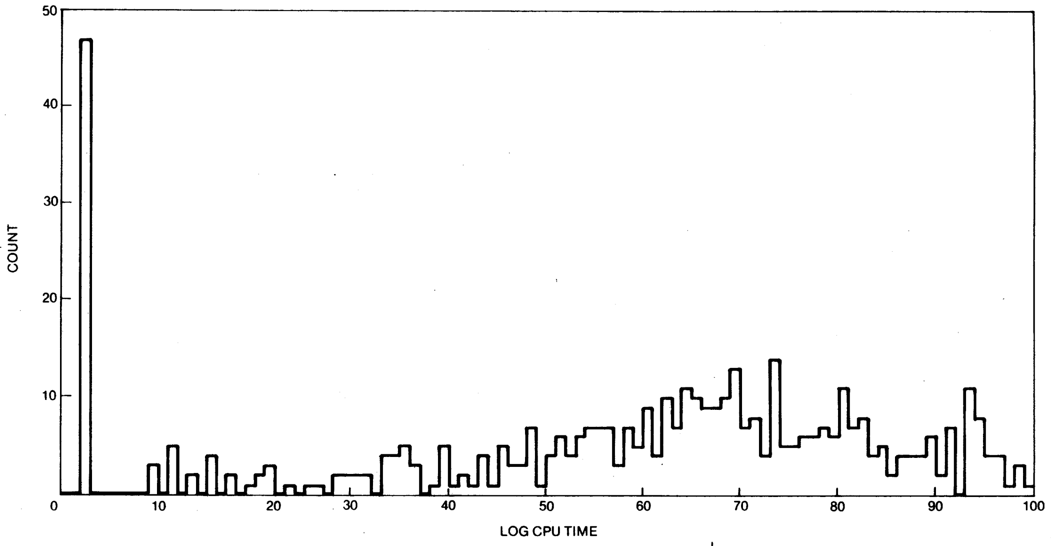


Figure 3. Histogram of log CPU time

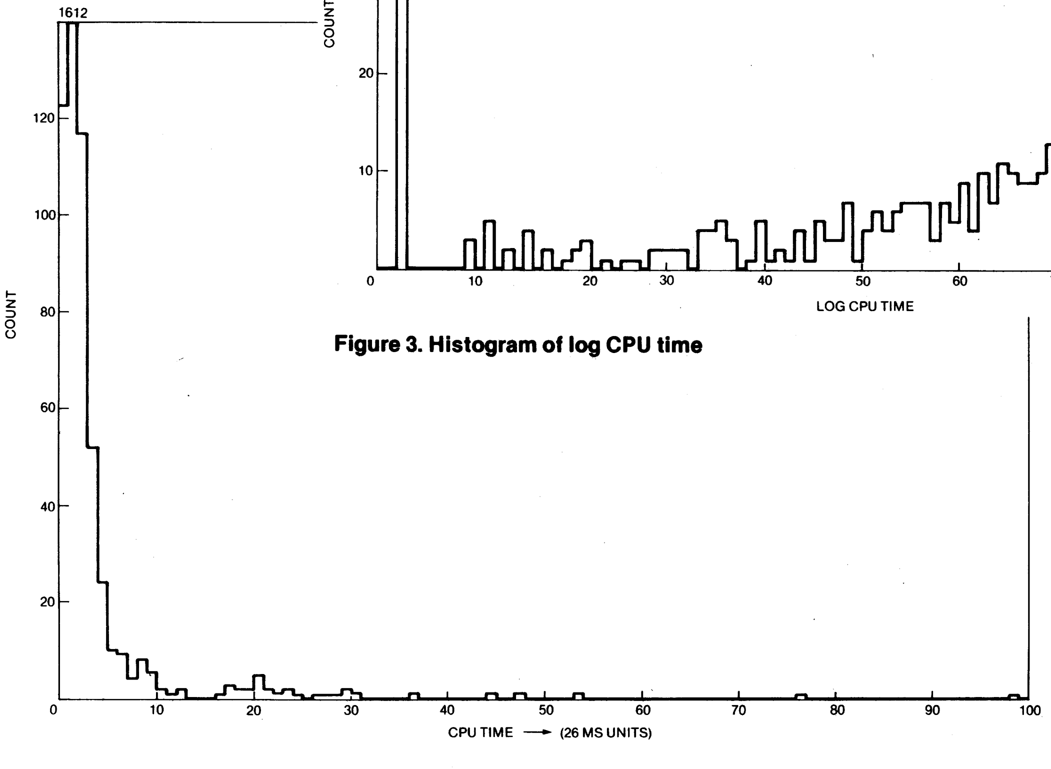


Figure 2. Histogram of CPU time

Mixture Distribution Model—A Clustering Approach. To construct a mixture distribution model from this data we need to specify the number of classes m as well as the conditional density functions $P(x/c_i)i = 1, m$. Since m is unknown, we used the unsupervised learning technique of clustering. In this approach a group of points which are close to each other are assigned to a cluster in the n dimensional space. The clustering approach used was a minor variation on the k-means method presented in Anderberg.[21] Implicit in this approach is the assumption of unimodality of the conditional density function $P(x/c_i)$. Within a few passes, the clustering routines determine from the data the location of all the clusters which meet a clustering criterion.

The following eight features were used in clustering:

(1) CPU time,
(2) executive request & control card (ER/CC) charges,
(3) average number of 512-word core blocks used,
(4) number of job steps (programs) executed,
(5) wall clock time,
(6) I/O to FASTRAND or disk devices,*
(7) I/O to tape,
(8) I/O to high-speed drum devices.

The data for all three periods were collected, jointly scaled, and clustered independently. A weighted Euclidean distance in the scaled space was used. (The details of the scaling method and the clustering method used are presented in Appendix A.)

The means and the standard deviations of the clusters found by the clustering routine are shown in Tables 3-8. The clustering with only these eight

*Although FASTRAND is a drum device because of its speed characteristics, it is treated as a disk in this study.

Table 3. Cluster means: May 1-2 (end of semester)

CLUSTER	CPU TIME	ER & CC	CB SIZE	NO. OF PROGS.	WALL CLOCK	DISK	TAPE I/O	DRUM I/O	NO. OF POINTS	RADIUS
2	6.6	7.1	2.3	0.73	6.6	7.6	7.7	7.5	102	0.83
3	3.4	5.9	1.7	0.37	5.0	0.0	0.0	6.7	244	0.79
4	0.22	0.85	0.12	0.018	1.1	0.0	0.0	0.63	125	0.81
6	5.9	4.8	6.1	0.18	4.5	7.5	1.3	0.0	96	0.11
9	5.9	6.6	3.7	0.64	5.7	7.2	0.0	6.8	1544	0.57
12	7.8	8.8	2.1	3.0	8.5	9.0	8.8	8.6	64	0.86
14	7.6	8.8	2.7	4.7	8.5	8.8	0.0	8.7	549	0.59
15	4.4	10.1	4.0	8.5	9.6	10.4	9.1	9.8	35	0.91
16	9.3	7.9	5.5	1.0	8.2	7.9	9.3	8.3	102	0.84
18	5.3	6.9	1.3	0.80	7.0	0.0	7.4	7.1	15	1.00

Table 4. Cluster means: Feb. 24-26 (mid semester)

CLUSTER	CPU TIME	ER & CC	CB SIZE	NO. OF PROGS.	WALL CLOCK	DISK	TAPE I/O	DRUM I/O	NO. OF POINTS	RADIUS
1	6.3	7.0	2.2	0.82	7.0	0.037	7.7	7.6	153	0.80
2	6.0	7.6	2.1	1.2	7.4	7.5	7.8	7.0	46	1.10
4	8.6	9.6	3.3	5.9	8.9	8.9	8.9	9.7	47	1.00
5	5.0	6.2	3.1	0.67	5.4	0.0	0.0	7.4	1428	1.50
6	5.4	7.1	2.8	0.82	6.3	6.7	0.0	7.7	511	0.57
8	0.044	1.1	0.065	0.0	1.1	0.0	0.0	0.0	199	2.00
10	8.7	10.5	2.8	20.9	9.8	9.3	0.20	10.0	17	0.43
11	6.8	5.8	9.2	0.24	6.0	7.4	0.082	7.9	172	0.72
12	7.6	8.9	2.6	4.6	8.7	7.9	0.0	9.0	379	0.64
13	8.4	10.0	1.6	19.7	9.7	9.3	9.6	9.7	4	1.00
14	9.0	7.8	6.1	0.90	8.0	3.6	9.1	8.9	72	0.75

Table 5. Cluster means: May 27 (Memorial Day)

CLUSTER	CPU TIME	ER & CC	CB SIZE	NO. OF PROGS.	WALL CLOCK	DISK	TAPE I/O	DRUM I/O	NO. OF POINTS	RADIUS
1	7.3	8.3	2.2	2.2	7.9	8.4	7.4	8.1	33	0.66
2	0.0	0.40	0.058	0.0	0.32	0.0	0.0	0.0	10	0.28
4	3.0	6.0	0.93	0.32	4.7	3.7	0.56	6.1	54	0.55
6	6.4	6.9	3.6	0.79	5.8	7.4	0.0	7.2	180	0.54
7	6.9	7.7	1.5	0.55	6.8	0.0	9.0	7.6	5	0.18
8	8.3	7.5	5.7	0.73	6.9	7.9	8.1	8.4	33	0.77
9	7.7	9.0	2.8	4.5	8.0	9.2	0.0	9.0	62	0.61
10	8.6	9.6	2.4	5.8	9.1	9.6	9.5	9.1	31	0.92

Table 6. Cluster standard deviations: May 1-2 (end of semester)

CLUSTER	CPU TIME	ER & CC	CB SIZE	NO. OF PROGS.	WALL CLOCK	DISK	TAPE I/O	DRUM I/O
2	1.3	0.66	1.6	0.46	1.0	0.92	1.6	1.0
3	2.0	6.0	2.2	0.34	1.5	0.0	0.0	1.3
4	0.62	1.0	0.20	0.054	1.3	0.0	0.0	1.8
6	1.9	1.6	2.4	0.032	2.3	0.53	3.1	0.0
9	1.8	0.84	2.7	0.45	1.8	0.95	0.17	2.3
12	1.4	0.63	1.0	1.9	0.79	1.0	1.4	1.0
14	1.2	0.81	1.3	4.4	1.2	0.78	0.0	0.84
15	0.99	1.0	2.2	7.4	0.83	0.82	1.4	0.66
16	0.97	0.73	2.8	0.70	1.0	1.0	1.1	1.2
18	1.7	0.99	0.78	1.2	1.1	0.0	1.9	0.99

features was good in the sense that it is not difficult to assign a common description to the types of jobs falling in a given cluster. The nature of these clusters is well illustrated by their Kiviat Graphs[22] presented in Figure 5.

The model generated using the clustering approach has as many classes as there are significant clusters in the job population. It is encouraging that even in May 1-2 data, only eight such clusters exist. As shown in Tables 4 and 7, there were 11 clusters for February 24-26 data.

The clustering routines that we used supplied a very detailed description of the jobs that formed the various clusters. In addition to supplying us with the standard statistical information about each cluster (e.g., minimum, maximum, standard deviation, number of members), the programs also measured such items as cluster radius (maximum, minimum) and the cluster number of the job closest to that cluster that was a member of another cluster. We also obtained the RUNID and an identification number for each member of the cluster. The identification number allowed us to trace the jobs back to the original log files. On the basis of this information we can easily assign a description to a cluster. The description of a few clusters from the February data follows.

The cluster analysis for the February 24-26 data was based on 3027 different jobs. These jobs were entered from timesharing, batch, and remote batch terminals. Eleven clusters were generated for this data, ranging in size from four members to 1428 members. The scaling used on the features was such that 98% of the values for each feature fall in the range 0-10 (see Appendix A for details).

Cluster 5 has 1428 members, far more than any other cluster. The cluster is composed of approximately equal numbers of batch and timesharing jobs. No member of this cluster performed any tape I/O or disk I/O, although all members of this cluster used moderate amounts of drum I/O. The University of Maryland system is configured so that all catalogued files reside on disk and all temporary files reside on drum. Since no job in this cluster

Table 7. Cluster standard deviations: Feb. 24-26 (mid semester)

CPU CLUSTER	CPU TIME	ER & CC	CB SIZE	NO. OF PROGS.	WALL CLOCK	DISK	TAPE I/O	DRUM I/O
1	2.0	0.88	1.2	0.92	1.2	0.45	1.9	1.1
2	1.4	0.81	1.2	0.85	1.2	1.3	1.3	1.1
4	1.2	0.91	1.9	3.4	0.76	0.94	1.2	0.59
5	2.1	1.0	2.3	0.86	2.0	0.0	0.0	1.1
6	1.3	0.63	1.9	0.57	1.8	1.0	0.0	0.84
8	0.18	1.6	0.045	0.0	1.0	0.0	0.0	0.0
10	0.79	0.37	1.2	7.5	0.62	0.64	0.81	0.35
11	1.3	0.69	1.5	0.19	1.6	0.51	0.53	0.73
12	1.2	0.61	1.1	2.8	0.92	1.1	0.0	0.53
13	0.93	0.56	0.45	2.4	0.79	0.88	0.30	0.50
14	0.70	0.73	2.7	0.24	0.76	4.0	1.1	1.1

Table 8. Cluster standard deviations: May 27 (Memorial Day)

CLUSTER	CPU TIME	ER & CC	CB SIZE	NO. OF PROGS.	WALL CLOCK	DISK	TAPE I/O	DRUM I/O
1	1.3	0.56	1.2	1.3	0.84	0.85	1.4	0.89
2	0.0	.016	.047	0.0	0.39	0.0	0.0	0.0
4	1.4	0.57	0.76	0.14	1.8	3.4	1.8	1.0
6	1.6	0.82	2.4	0.48	2.2	0.92	0.0	2.0
7	0.88	0.93	0.45	0.33	0.90	0.0	0.87	1.6
8	1.6	0.82	2.6	0.49	1.2	0.81	1.7	1.3
9	1.0	0.59	1.4	2.7	1.6	0.81	0.0	0.74
10	1.1	0.56	1.1	4.8	0.71	0.72	1.4	0.88

used the disk but all jobs used drum, it is reasonable to assume that the users in this class used only temporary files. It should be pointed out that a user running on a student account cannot save catalogued files overnight and therefore tends to use temporary files. The user's workspace may be saved, but the system treats this file as a temporary file in that it is stored on drum.

Most of the jobs in this cluster were run on student accounts and consisted of several Fortran compilations, a link edit, and an execution of a user program. The typical job in this class did not make large demands on the system and did not execute a large number of programs, although the standard deviation of the number of programs run was rather large. The large variation in the number of programs executed is reasonable if one considers the type of work a student is likely to perform. If a student submits a batch job, it consists of a relatively small number of compilations, a call to the linkage editor, and a short execution of the program. If a student uses a timesharing terminal, he is likely to use the editor and then compile the program. If the compilation fails he will return to the editor and cycle in this manner until the compilation is successful. He will then use the linkage editor and execute his program. If the program fails in execution, he will return to the beginning of the cycle. This would explain the wide variation in the number of programs executed and amount of resources used. The most important feature in assigning members to this class was the lack of any disk I/O. All other classes of jobs except for class 8 (which used virtually no resources) did I/O to the disk.

Clusters 2 and 4 contained only 46 and 47 members, respectively. Most of the jobs in each of these clusters were run in a timesharing mode. Each cluster also contained a few batch jobs. The jobs in both clusters show very similar usage of various resources.

The basic difference between the two clusters seems to be in the number of programs comprising each job. The jobs in cluster 4 executed more programs than the jobs in cluster 2 and thus

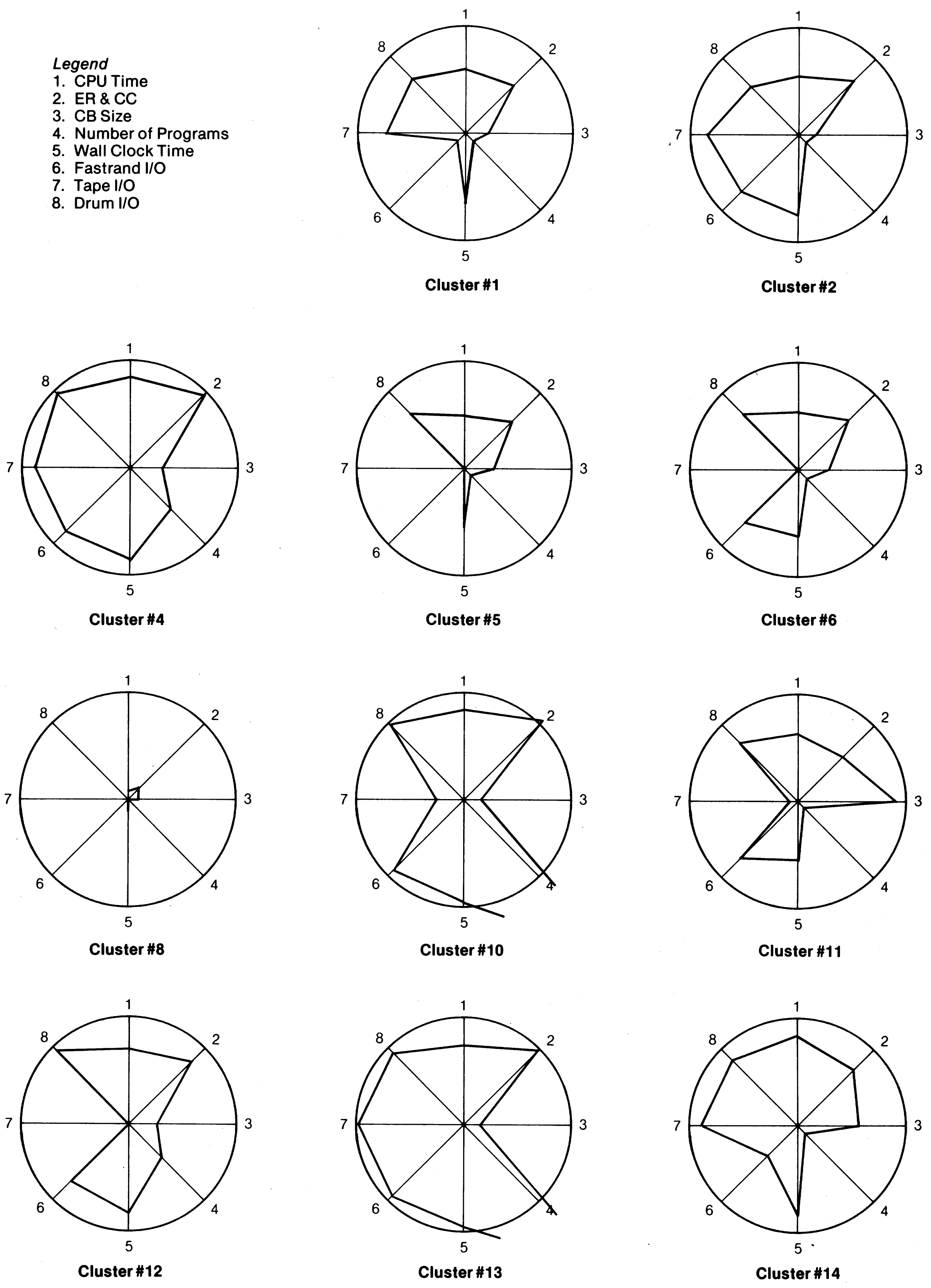


Figure 5. Kiviat graphs for the clusters of Feb. 24-26 data

Note: Due to the scaling technique used only 98% of the values for any parameter will fall in the range 0-10.

used more of each of the various system resources. Therefore, the jobs in the two clusters seem to be separated more on how much of each resource they used rather than the types of resources they used.

Cluster 11 shows some types of extreme behavior. Of the 172 members of this cluster, only nine were demand jobs; of these, seven were submitted by the same user. The members of this cluster executed fewer programs than the members of any other cluster except the members of cluster 8 (which executed no programs). This cluster used more core-blocks than any other cluster. Another interesting point about the jobs in this cluster was that in many cases there were multiple submittals of jobs (i.e., one user submits several copies of the same job using different data). Typically, the jobs in this class signed on, assigned some files, ran one large applications program (in many cases SPSS), perhaps checked the status of his account, did some file processing, and then terminated.

The jobs in cluster 12 were what might be termed large program development jobs. These programs tended to do a large number of compilations, some editing, a link edit, some re-compilations, etc. One characteristic that distinguished these jobs from the student program development was the number of routines that were compiled between edits. It was not unusual to see 10 to 20 compilations, then the editing of several of these routines. There are also a large number of link edits in these jobs which required searches of many of the system libraries. Since most of the jobs in this cluster were involved in program develoment, there were several executions in these jobs. Most of these executions were rather short as the programs were still not totally debugged.

In the above section, we have tried to briefly describe the jobs that were in a few of the clusters. Although these descriptions are generally accurate, we do not mean to imply that every job in a cluster conformed exactly to these descriptions. There was of course a certain amount of variability in the jobs making up each cluster. It is interesting to note that the clusters seem to have some distinctive characteristics, even though they were constructed based strictly on the total amounts of each resource required by a job.

Comparison of Clusters for the Three Periods. It is interesting to compare the clusters obtained for the three periods under study. As noted earlier, all the data from all three periods were jointly scaled. After the common scaling, each set of data was independently clustered. Examining the clusters so obtained (see Tables 3-8) we notice some distinct similarities.

In each data set one cluster contains approximately half of all the jobs. Most jobs in this cluster are student-type jobs. The load on the system increases substantially towards the end of a semester, and a major portion of this increase should be caused by student work. That increase is not directly visible in the ~~clusters, because~~ the resource usage of some student jobs that are run towards the end of the semester may look different. The reason is that towards the end of the semester the typical timesharing student begins to look to the system very much like a research user doing program development. Since he must complete his project the student user may stay on the system for several hours repeatedly compiling, editing, and executing his program. Typical program problems (i.e., infinite loops) cause the student user to use resources in such a manner that the statistics about his job begin to look very much like those that could be expected from a research user.

A number of jobs on the system are run every day regardless of system load. The file and system maintenance programs must be run on a regular basis. In addition, since the systems staff at the university tends to do the system development work very late at night, this work tends to be carried on regardless of the heavy daytime loading of the system. Therefore, clusters containing these types of programs appear in each of the three periods.

A number of research users try to finish their work before the end of the semester either because of graduation, termination of support, or vacation; so even at the end of the semester, when the student load is at its peak, a substantial amount of research work is still being performed.

The similarity in the clusters in these three periods tends to indicate that the clusters arising from the data are a function of the kinds of activities for which the system is being used. It would be interesting to compare the clusters of usage on a different machine or on a different campus to see the extent to which the clusters we obtained are typical of a university. Finally, this similarity indicates to us that the clustering approach is a viable technique of constructing Type A models.

Stochastic Process Workload Models

Type B and Type C models, discussed earlier, result in our treating the workload as a stochastic process. Again, a direct brute force approach is likely to lead to an unmanageable formulation. Therefore, we may consider some simplified models of the stochastic process, such as the time series models and the Markovian models.

A Time Series Model. Since each element of a set of R_B or R_C is tagged with a time, we may treat a sequence of these requests as a time series and apply time series analysis.[15] For example, it has been a common observation that the computational needs of any group of users continue to grow. The rate and order of growth may be determined by a trend analysis of the observed data. Further, it is not uncommon to see some periodic behavior of the load which can be analyzed using time series analysis techniques.

Time series analysis, or course, requires data collected over a long period of time. Inasmuch as time series analysis would be of use in evaluating long-range changes in the workload, we may collect data at a more gross level. For example, the number of jobs processed per day may be adequate to analyze the load on a system over a year.

The time series analysis allows the consideration of the variation of the system load at a very high level. To study the time-dependent nature of the steps in a job, we may use a Markovian model for a job.

A Markovian Model of a Job. Each user job consists of a number of steps of processing, each requiring different resources. To process the job, the system has to handle each step in succession. Therefore, to take into account the time-dependent nature of steps in a job, we need to consider the requests made at the step level and treat a job as a sequence of these steps. A sequence of such requests may be modeled as a Markov chain or a semi-Markov chain,[23] in which the next user request in a job may depend on the current request, or on a few prior requests. For this model to be workable, we need to specify the states of the Markov process. This could be done by considering the various system software resources as states. Therefore, a call to the Fortran compiler may be considered one state, and a call to the linkage editor, another state. The Markovian nature of this process may be characterized by a transition probability matrix.

Since there exists a wide variation of jobs, a single Markovian model for all of the jobs on a system is not likely to be valid.[24] We may combine this approach with that of mixture distributions and separate the jobs into classes. Then, we may create a Markov model for each class. Note that this Markov chain is a model of a job. Because the system may be handling a number of jobs concurrently, a model for each job would be active in generating the requests to the system.

This kind of a model is likely to be very beneficial in the study of the performance of scheduling algorithms. Conceivably one may design schedulers which adapt to the classes of currently active jobs.

Models for a Distributed Environment. At present, very few distributed computing networks are operational. As a result, the models for a distributed workload cannot be made on the basis of the actual system measurements. They may be derived from Types A and B models under certain assumptions; this will allow us to consider some aspects of distributed computing.

A way of introducing the spatial nature of the workload would be to treat L as an independent random variable for which a probability distribution P(L) may be specified. Any Type A or Type B model may now be converted to a Type C model by taking P(L) into account.

If we were to consider the question of connecting a few independent nodes into a network, we could take a Type B model of each node and augment it with a tag T_L identifying that node. This will be a valid model only under the assumption that the interconnection of nodes is not going to change the user characteristics.

In an actual distributed computing environment, more complex models may be required. The techniques of multivariate point process analysis are likely to be useful.

Validation and Use of Workload Models

As mentioned earlier, the purpose of constructing the probabilistic workload models considered here is to be able to construct a "representative" test workload. The approach taken is to create a validated probabilistic workload model first, and then to use that model to construct a test workload.

All of the models considered here are probabilistic models; hence, given a model and a set of data observed from the system, we may test the hypothesis that the data were generated by the model and establish a degree of confidence in the model. The exact hypothesis tests to be used depend on the actual models. Both parametric and nonparametric tests may be applicable.

It should be pointed out that when a model is constructed based on a set of data from the system (the design data set), the techniques used in constructing the model should guarantee it being valid for the design data set. A separate set of data, the test data set, may be required for independent validation. Independent validation may also be obtained by generating models from different sets of data and checking the models for stability and consistency. The experimental study of the Univac 1108 workload model, discussed earlier, used this approach. The models constructed were consistent for the three periods used in that study.

The form of the test workload to be used in a performance evaluation study depends on the technique to be used in the study. These models lend themselves well to constructing analytical test workloads by fitting analytical distribution functions to various parameters. The techniques described by Srinivarsan and Kleinman[11] can be used to construct synthetic jobs based on these models.

If a study requires a benchmark program as the test workload, the "representativeness" of the benchmark with respect to the system workload can be established more easily by using a validated model. If a trace-driven modeling approach is used we may choose to generate a trace based on a validated benchmark of a synthetic job stream. Otherwise the representativeness of a trace taken from the system may be checked with respect to the model.

Conclusions

Within the framework of the workload characterization problem considered here, we have specified three types of models for the workload on an actual system. We have also explored the applicable techniques for constructing validated models of the system workload for each of these types. Such validated models may then be used for generating "representative" test workloads.

Our initial results indicate that clustering may be a viable approach for construction of probabilistic models of workload. The selection and design of specific clustering techniques for workload characterizaton need further examination. All clustering techniques assume that the items to be grouped are specified in terms of certain parameters. Therefore a workload characterization study using the clustering approach can be effected significantly by judicious choice of such parameters. Some of these problems are subjects of ongoing research. ■

Acknowledgements

This research was supported in part by the Mathematics and Information Sciences Directorate, Air Force Office of Scientific Research, Air Force Systems Command, USAF, under grant AFOSR 76-2901, and in part by the Control and Automation Branch, Engineering Divison, National Science Foundation, under grant GK-41062 to the Department of Computer Science of the University of Maryland. We would also like to acknowledge the assistance supplied by the University of Maryland Computer Science Center in obtaining the data used in this work.

Appendix A

Introduction. The purpose of this appendix is to provide a formal description of the clustering algorithm discussed earlier (see p. 23). The algorithm described is a variant of the nearest centroid method described by Anderberg.[21] The distinguishing characteristics of this particular algorithm are that:

1) It uses a scaling procedure which produces an essentially uniform feature space in spite of the presence of outliers.

2) It does not require that a weighting vector for the features be assigned in advance; rather the weighting vector is adaptively created during clustering.

3) It exhibits good stability in the sense that cluster assignments do not change radically with minor variations in the seed points selected, or the order of the input data.

The next four sections of this appendix describe the scaling method, the dissimilarity measure used, the clustering method itself, and the stability results obtained.

Scaling. Let $X_1, \ldots, X_p$ be vectors in R^n and let us suppose that the entries X_{ij} of the ith vector represent the feature values of the ith data point in the data set. In order for our similarity measure to work, it is necessary that each feature be measured in roughly the same type of units, and the relative ranges of the features should be approximately equal. (After examining histograms such as Figures 3 and 4, we felt it necessary to work with the logarithm of some features; for our purposes here we will assume that this logarithmic scaling has already been done.)

One way to obtain this type of feature space is to rescale the data linearly so that the maximum and minimum values of each feature are the same. This method has the disadvantage that a single data point with an unduly large feature value can distort the scaled feature space.

The method we have used is the following:

$$\text{Let } X_j^{\min} = \min_{1 \leq i \leq p} X_{ij}$$

be the minimum observed value of the jth feature. Let α be a number between zero and one; typically we have used α in the interval [0.95, 1.0]. Now select X_j^{α} to be an α-tile of the population $\{X_{ij}\}_{i=1}^{p}$. That is, choose X_j^{α} so that αp of the data points satisfy $X_{ij} < X_j^{\alpha}$. If there is more than one α-tile for the population, choose X_j^{α} to be the smallest one. Having found $X_j^{\min}$ and X_j^{α}, linearly rescale each feature by:

$$X_{ij}^{(s)} = \frac{10(X_{ij} - X_{ij}^{\min})}{(X_j^{\alpha} - X_{ij}^{\min})}$$

After this scaling most of the data will lie in the side-10 hypercube with one corner at the origin. (For the log-tape data with $\alpha = .98$, it is observed that 95% of the data lie in this region.) Thus we have cheaply obtained an essentially uniform feature space which is not subject to distortion due to the presence of outliers.

The Dissimilarity Measure. Recall that the nearest centroid algorithm works in the following way: Initial guesses for m cluster medians $C_1, \ldots, C_m$ are obtained and then a pass through the data points $X_1, \ldots, X_p$ is made. Each point is assigned to that cluster which minimizes the dissimilarity measure $D(C_i, X_j)$. After the last point is assigned, the cluster medians are recalculated in terms of their current membership, and then another pass through the data points is made. This process is repeated until the number of points assigned to different clusters on consecutive data passes falls below some pre-established limit.

In our method there is a different dissimilarity measure for each cluster. Each measure attempts to emphasize those features which are "important" in assigning points to that cluster, and to minimize those features which are not "important" in assigning points to that cluster. The measure of importance we use is the reciprocal of the variance of the feature within the cluster. Basically, if a feature has a large variance within a cluster, then particular values of that feature by themselves are not good indicators of cluster membership. We therefore minimize the importance of such a feature. On the other hand, if the within-cluster variance of a feature is zero, then knowing the value of only that feature could determine whether a given point is a member of that cluster or not. So we increase the importance of features with small variance.

More precisely the ith cluster dissimilarity measure is defined as:

$$D_i(C_i, X_j^{(s)}) = \sqrt{\frac{\sum_{k=1}^{n} w_{ik}(C_{ik} - X_{jk}^{(s)})^2}{\sum_{k=1}^{n} w_{ik}}}$$

where W_i is a weight vector.

Initially each weight vector $W_i = (w_{i1}, \ldots, w_{in})$ is set to the constant value $1/w_s$ where w_s is a small positive number. (We have found $w_s = 0.1$ a good value.) These W_i values are used for cluster assignment during the first data pass. At the end of each data pass the weight vectors to be used during the next data pass are recalculated according to:

$$w_{ij} = \frac{1}{w_s + \sigma_{ij}^2}$$

where σ_{ij}^2 is the variance of the jth feature in the ith cluster. (The factor w_s allows for the case $\sigma_{ij}^2 = 0$.)

The denominator of the fraction which defines D_i serves two purposes. First, it removes the dependence of D_i on the variable n by requiring that the length of the diagonal of the unit hypercube always be one when measured by any D_i. Secondly, it allows the comparison of D_i and D_j in a reasonable manner. The denominator keeps D_i values for clusters with large variances in each feature from becoming too small in relation to other D_j values.

Finally, it should be pointed out that this family of dissimilarity measures creates clusters which grow (from pass to pass) along their major axes and which contract along their minor axes.

The Clustering Algorithm. As previously mentioned, the algorithm used is a variant of the nearest centroid type. The purpose of this section is to describe the differences between the method used and the nearest centroid method as described above.

One difference concerns creation, deletion, and merging of clusters. Creation of new clusters occurs during a data pass when the value

$$d = \min_{1 \leqslant i \leqslant m} D_i(C_i, X_j^{(s)})$$

exceeds a speciified threshold d_{max}. When this happens the number of clusters m is incremented by one, the new cluster median C_m is set to the value $X_j^{(s)}$, and the point j is assigned to the new cluster m. The weight vector W_m is set to the constant value $1/w_s$, and then we proceed to the next data point in the normal way. At the end of each data pass, comparisons are performed to determine if clusters should be merged or deleted. Whenever $D_i(C_i, C_j) < d_{min}$ then clusters i and j will be merged and the cluster descriptions will be updated accordingly. Whenever a cluster has existed for at least three data passes and currently contains fewer than five points, then that cluster will be deleted and its members will be removed from the data set.

The second change from the nearest centroid method concerns the current value of the mean vector during the data pass. In the nearest centroid method, the cluster means are updated only at the end of the data pass. In order to accelerate convergence while maintaining stability we have adopted the following procedure.

At the start of the first data pass we initialize the vectors $S_1, \ldots, S_m$ to the value zero. In addition we set the cluster membership counts $K_1, \ldots, K_m$ to zero. Then during the first data pass, whenever we have decided to assign the point $X_j^{(s)}$ to the cluster a, we perform the following calculations:

```
For i ← 1 step 1 until n do
    $S_{ai} \leftarrow S_{ai} + X_{ji}^{(s)}$;
    $K_a \leftarrow K_a + 1$;
For i ← 1 step 1 until n do
    $C_{ai} \leftarrow$ (if $K_a > S_{min}$ then $S_{ai}/K_a$ else $C_{ai}$);
```

Thus we do not update the cluster mean C_a until a certain number S_{min} of members have been assigned to cluster a. At the end of the first data pass all of the mean vectors are recalculated regardless of whether $K_a > S_{min}$.

On the subsequent data passes we need not reset the values S_i since we know these sums from the last pass. Let us suppose we have examined data point j and we have decided to assign it to the cluster b during the current data pass. Further, suppose that we had assigned it to cluster a on the last data pass. If $a = b$ then C_b will not change and we may proceed directly to the next data point. If $a \neq b$, then we must subtract this point's influence from C_a and add it to C_b.

These calculations are performed as follows:

```
For i ← 1 step 1 until n do begin
    $S_{ai} \leftarrow S_{ai} - X_{ij}^{(s)}$;
    $S_{bi} \leftarrow S_{bi} + X_{ij}^{(s)}$;
end;
$K_a \leftarrow K_a - 1$;
$K_b \leftarrow K_b + 1$;
If $K_a = 0$ then mark $C_b$ as empty cluster else
for i ← 1 step 1 until n do
    $C_{ai} \leftarrow S_{ai}/K_a$;
For i ← 1 step 1 until n do
    $C_{bi} \leftarrow$ ( if $K_b > S_{min}$ then $S_{bi}/K_a$ else $C_{bi}$);
```

Then of course at the end of the data pass we recalculate all cluster means $C_1, \ldots, C_m$.

It is in large part this procedure which gives the algorithm the stability which will be discussed in the next section. The philosophy is that we do not wish to recalculate a cluster mean until we have a sufficient sample of the data points which will eventually be assigned to that cluster. Thus we do not shift the cluster means until we have a reliable indication of where it should be moved.

Finally let us point out that the S_{min} values which we have used are on the order of 5.

Stability of the Method. The purpose of this section is to discuss the results of those stability tests which have been performed on the clustering method. These results apply to the method as stated above. Similar tests on variations of the method (such as recalculating the weighting vectors during the data pass instead of just at the end of the pass) have shown that the method as stated has the best stability among the variations tried. Also these results come from application of the method to the May 27, 1974, log tape data. This is because this was the smallest data set, and so repeated executions on this set would be the easiest.

First we must define the stability measures which we used. As before, let $X_1, \ldots, X_p$ represent the data set and suppose one execution of the clustering algorithm assigns those data points to clusters $a_1, a_2, \ldots, a_p$ respectively where $1 \leqslant a_i \leqslant m_1$, and that another execution produces the cluster assignments $b_1, \ldots, b_p$ where $1 \leqslant b_i \leqslant m_2$. Let us assume that $m_1 \geqslant m_2$. Then we may calculate an m_1 by m_2 confusion matrix M by the following algorithm:

```
M ← 0;
for i ← 1 step 1 until p do
    $M_{a_i b_i} \leftarrow M_{a_i b_i} + 1$;
```

Also let R be a vector of length m_1 which consists of the sum of each row of the M matrix, and calculate a percentage matrix P by the following:

```
for i ← 1 step 1 until $m_1$ do
for j ← 1 step 1 until $m_2$ do
    $P_{ij} \leftarrow M_{ij}/R_i * 100.0$;
```

Now we may define our two stability measures. Consider the P matrix. In each row i there is a maximum entry P_i^{max}. The quantity $100-P_i^{max}$ indicates the percentage of misclassifications in this row. We define the average percent error E_{ave} as

$$E_{ave} = \frac{1}{m_1} \sum_{i=1}^{m_1} (100 - P_i^{max}).$$

Now consider the M matrix. Once again in each row i there is a maximum entry M_i^{max}. The quantity $R_i - M_i^{max}$ indicates the number of misclassifications in the ith row. We define the total percent error E_{tot} as

$$E_{tot} = \frac{100}{p} \sum_{i=1}^{m_1} (R_i - M_i^{max}).$$

Thus E_{ave} represents the average percentage of misclassifications per cluster while E_{tot} gives the percentage of misclassifications in the entire population. The smaller values of E_{ave} and E_{tot} represent better stability.

It should be recalled that $m_1 \geqslant m_2$ by assumption. The reason for this assumption will now be given. If $m_1 < m_2$, then one or more of the clusters from the first execution must have been split into two clusters during the second execution. Clearly any clustering algorithm would not perform well with regard to E_{tot} and E_{ave} in this situation, due to the possibility of a 50-50 split. We prefer not to measure this kind of problem and thus we will assume $m_1 \geqslant m_2$.

The three test situations were the following:

1) $m = 1$ at execution start, i.e. let the algorithm select the number of clusters,

2) $m = 3$ at execution start, i.e. specify some cluster means and let the algorithm proceed,

3) $m = 1$ at execution start but change the order of the input data by moving the first 50 (out of 419) data points to the end of the data set.

In addition three parameter settings were tested:

1) $S_{min} = 1$ $W_s = 1.0$
2) $S_{min} = 5$ $W_s = 1.0$
3) $S_{min} = 5$ $W_s = 0.1$

Table A-1 shows the results of these test situations. As observed previously the values $S_{min} = 5$, $W_s = 0.1$ give the best stability.

Table A-1. Test results

TEST CASE \ PARAMETER VALUES	1		2		3	
	E_{ave}	E_{tot}	E_{ave}	E_{tot}	E_{ave}	E_{tot}
1	12.9	19	4.4	5.9	4.3	6.9
2	16.9	23	7.3	9.9	7.6	4.9
3	13.3	14	4.6	6.4	10.7	6.2

References

1. E. Hunt, G. Diehr, and D. Garnatz, "Who are the Users? An Analysis of Computer Use in a University Computer Center," *AFIPS Conference Proceedings,* Vol. 38, SJCC, 1971, p. 231.

2. H. C. Lucas, Jr., "Performance Evaluation and Monitoring," *Computing Surveys,* Sept. 1971, pp. 79-91.

3. J. M. Metanney, "A Survey of Analytical Timesharing Models," *Computing Surveys,* June 1969, pp. 105-116.

4. D. Ferrari, "Workload Characterization and Selection in Computer Performance Measurement," *Computer,* July/August 1972, pp. 18-24.

5. E. O. Joslin, "Application Benchmarks: The Key to Meaningful Computer Evaluations," *Proc. ACM 20th National Conference,* August 1965, p. 27.

6. W. Bucholz, "A Synthetic Job for Measuring System Performance," *IBM Sys. J.,* Vol. 8, No. 4, 1969, pp. 309-318.

7. S. W. Sherman, "Trace Driven Modeling Studies of the Performance of Computer Systems," Ph.D thesis, University of Texas at Austin, August 1972, TSN-30.

8. E. G. Coffman, Jr., and P. J. Denning, *Operating Systems Theory,* Prentice-Hall, Englewood Cliffs, N.J., 1973.

9. *Symposium on the Simulation of Computer Systems,* National Bureau of Standards, Vols. I-III, 1973-75.

10. Nicholas, Benwell, *Benchmarking—Computer Evaluation and Measurement,* John Wiley and Sons, New York, 1975.

11. K. Sreenivasan, and A. J. Kleinman, "On the Construction of a Representative Synthetic Workload," *CACM,* Vol. 17, No. 3, March 1974, pp. 127-133.

12. M. D. Abrams and I. W. Cotton, "The Service Concept Applied to Computer Networks," *National Bureau of Standards Technical Note 880,* August 1975.

13. P. A. W. Lewis, *Stochastic Point Processes: Statistical Analysis, Theory and Applications,* Wiley-Interscience, 1972.

14. S. K. Srinivasan, *Stochastic Point Processes and their Applications,* Griffin, London, 1974.

15. T. W. Anderson, *The Statistical Analysis of Time Series,* New York, John Wiley, 1971.

16. H. Hellerman and T. F. Conroy, *Computer System Performance,* McGraw-Hill, 1975.

17. L. N. Kanal, "Patterns in Pattern Recognition: 1968-1974," *IEEE Transactions on Information Theory,* Vol. IT-20, No. 6, November 1974, pp. 697-722.

18. Y. C. Ho and A. K. Agrawala, "On Pattern Classification in Algorithms—Introduction and Survey," *Proc. IEEE,* Vol. 56, pp. 2101-2114, December 1968.

19. J. M. Mohr, A. K. Agrawala and J. F. Flanagan, "The EXEC-8 Log System, Part I—Description," University of Maryland Department of Computer Science, Technical Report TR-434, January 1976.

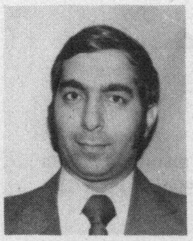
20. ———, "The EXEC-8 Log System, Part II—Validation," University of Maryland Department of Computer Science, Technical Report (in preparation).

21. M. R. Anderberg, *Cluster Analysis for Applications,* New York, Academic Press, 1973.

22. K. W. Kolence, and P. J. Kiviat, "Software Unit Profiles and Kiviat Figures," *PER,* Vol. 2, No. 3, September 1973, pp. 2-12.

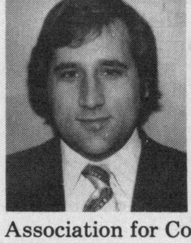
23. R. A. Howard, *Dynamic Probabilistic Systems,* John Wiley and Sons, New York, 1971.

24. A. K. Agrawala, J. M. Mohr, and B. Agarwal, "Application of Markov Models to Workload Characterization," University of Maryland, Department of Computer Science Technical Report (in press).

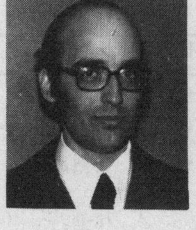


Ashok K. Agrawala has been a member of the computer science faculty of the University of Maryland since 1971. From 1968 to 1970 he was with the Department of Applied Research of Honeywell, Inc., Waltham, Mass., where he worked on optical character reader design and problems with mass storage units. From 1970 to 1971 he was with the OCR Product Development Group of Honeywell Information Systems, Minneapolis, Minn. He received the B.Sc. degree from Agra University, Agra, India, in 1960; the B.E. degree in electrical technology, and the M.E. degree in applied electronics and servomechanisms from the Indian Institute of Science, Bangalore, in 1963 and 1965, respectively. He received the A.M. degree and the Ph.D degree in applied mathematics from Harvard University in 1970.

Dr. Agrawala is a co-author of "Foundations of Microprogramming," and has published many papers in a variety of areas. His current research interests include analysis, modeling and evaluation of computer systems, microprogramming, computer architecture, and pattern recogntion. He is a member of Sigma Xi, ACM, and the Pattern Recognition Society.



Jeffrey M. Mohr is working for his Ph.D degree at the University of Maryland. His special fields of research include workload characterization and computer performance evaluation. He received the BS degree in computer science from the School of Electrical Engineering of Cornell University, Ithaca, N.Y., in 1972 and the MS degree in computer science from the University of Maryland. Mohr is a student member of the Association for Computing Machinery and the IEEE Computer Society.



Raymond M. Bryant is studying for the Ph.D degree in applied mathematics at the University of Maryland. His thesis is concerned with analytical models of computer systems. He received the BS degree in 1971 from South Dakota School of Mines and Technology and the MA degree in 1973 from the University of Maryland, both in mathematics. Bryant is a student member of ACM, the IEEE Computer Society, and ORSA.