MEASUREMENTS OF PROCESSING REPETITION IN INTERACTIVE COMPUTATION*

by

A. S. Noetzel

October 1975 TR-52

Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712

* Research reported in this paper was sponsored by NSF grant GJ-39658

Measurements of Processing Repetition in Interactive Computation

ABSTRACT

The phenomenon of the repetition of resource-demand sequences due to multiple executions of the same or similar programs is investigated. A definition of resource-demand repetition is proposed, and the technique for measuring this property in an instance of an interactive system is described. Results of the measurement of repetition, first at the command level from the system dayfile, and then at the resource-demand level by a detailed trace facility, are described. The results show a significant amount of repetition in the compilers and user-written programs. Implications for operating system design are discussed.

1. Introduction

Operating systems are designed to respond to user program resource requirements that are known in advance only probabilistically, according to the statistical profile of the job class and within the gross upper bounds specified on control cards. However, reflection on the modus operandi of the users of a computing facility suggest that since each program is run many times during its period of development and productive use, the records of the resource demand generated during any run of a program are probably quite similar to those generated during the preceding and succeeding runs of that program. This paper examines the implications of this repetitiveness for the design of computing systems. A definition of resource-demand repetition is proposed, and measurement of this phenomenon in one instance of an interactive system are described. The results show the degree of repetition is probably not significant enough to be used in predictive scheduling, but does indicate possiblities for the design of language processors and monitors.

Part two of this paper discusses repetitiveness within user computation in general terms, and proposes a definition of resource-demand repetitiveness. Part three contains a brief description of the interactive system on which the measurements were made, and presents statistics of the useage and resource requirements of the various software processors in the system. In part four, a description of the resources and allocation methods is presented as a background for the technique of extracting resource-demand characteristics of user programs. The measurements of the degree of resource-demand repetition found in the system are then presented in part five. The implications of these elements for system design are discussed in the conclusion, part six.

2. Repetition of Processing in Computer Systems

Processing sequences representing a program in execution may be considered at the level of individual instructions, or at the level of processing bursts interspersed with I/O and other supervisor requests. We shall call the former the logical level, since it reflects the logic of the computation, and the latter the resource-demand level. Processing sequences that are duplicated at the logical level represent multiple executions of a single routine with identical data. (Only trivial programs have instruction sequences that are not all data-dependent.) This duplication of processing effort represents an inherent inefficiency in a computing system as long as the results of the computation may be stored and retrieved more efficiently than repetitive recalculation. The elimination of redundant effort has not been seriously considered as an operating systmes design approach in most cases because it is assumed that the processing effort that might be saved is not greater than the file operations that would replace it. But no measurements of processing repetition supporting this assumption have been reported, whereas measurements of user behavior (2) indicate that the amount of repetitive effort may be considerable.

The repetition of resource-demand sequences indicates approximate (though possibly exact) repetition of logical level sequences. Typically, repetitions of resource-demand sequences reflect executions of the same program with somewhat differing data values. A great deal of resource-demand repetition in a system strongly indicates a considerable amount of logical repetition. It may be noted that the precision of the resource-demand representation determines the degree to which it may be considered an approximation to the logical sequence. A resource-demand sequence showing only gross memory requirements and the pattern of interspersed I/O and CPU bursts constitutes a relatively poor approximation to the logical sequence. Entirely different sets of routines may be invoked between I/O request in two executions of a program, yet they may produce the same resource-demand sequences. But, if the resource-demand sequence also includes the requests for operating system services by the program, identical resource-demand sequences are more likely to represent identical logical sequences. Finally, if the sequence of page faults is included in the resource-demand sequences of programs run under demand-paging memory allocation, these sequences will be close indicators of the logical sequences.

The presence of resource-demand repetition in a computer system might be exploited in the system design, independently of the consideration of logical repetition. If the actual resource-demand pattern of a program is recorded and stored each time the program is run, it will,in most instances, serve as an exact predictor of the resource requirements in the next run of the program. Schedulers which have precise advance data for resource requirements can achieve much more nearly optimal utilization than schedulers that use probabilistic data, and can allocate resources in sequences that are known to be free of deadlock. Consideration of this technique of scheduling does raise serious questions concerning overhead and the efficacy of schedulers that must operate on the basis of partial predictive data. But first, some indication is needed of the amount of resource-demand repetition that may be expected in a general-purpose processing utility.

The measurements of repetition of resource-demand sequences that are described in this paper were taken from an interactive CDC 6400 system. This is not a virtual memory system. The user program requests for operating system service are included in the resource-demand sequences. Hereafter, repetition of resource-demand sequences will be taken to mean exact processor-burst for processor-burst duplication with exactly the same I/O, memory, or supervisor service requests occurring between the continuous CPU bursts. This definition, with some tolerance, is used in the repetition measurements.

The UT-2D interactive system is half of a dual-processor 6600/6400 system, in which the processors are coupled through the sharing of mass storage devices. The interactive operating system has a great deal of involvement in the management of user programs and files, which enables the collection of data describing the user behavior at several levels. The close interaction of user and system will allow for novel approaches to scheduling, once the user behavior is well understood. The UT-2D interactive system will be briefly described, with a profile of the useage of its various software resources, so that the repetition measurements, and their scope of generality may be properly understood.

3. Interactive Command Usage Characteristcs

The study of processing repetition has been performed on the interactive UT-2D system on the CDC 6400. This interactive system grew out of the file-based batch operating system for the CDC 6000 series machines and retains many of the characteristics of the batch system. It has a simplicity of structure that is an advantage for this particular study, because the various modes of processing can be identified and separated. The results are reported in terms of the commands of this system, but the classes of commands will be seen to characterize user behavior and processing modes in general. After a brief description of the interactive system, profiles of processing time and frequency of use of the interactive commands are presented.

Command Usage Profile

The first versions of the interactive system simply allowed control cards to be entered from an interactive terminal, and although the vocabulary available to the user has now been expanded by many commands that are specifically for interactive purposes, a relatively small subset of the basic control-card commands still represents the major portion of computation in the interactive system.

The file system resides on a hierarchy of storage devices. The storage medium at the lowest level, is magnetic tape. The files on each tape belong to a single user,and are called a permanent file set. When a file in a permanent file set is referenced (as by the interactive command READPF) the entire permanent file set is transferred from the tape to the next level of the hierarchy (if it is not already resident there), which is a set of auxiliary disk units. The required file is read from the auxiliary disk units to the highest level, one of four large system disk units. The requested file is then logically attached to the user job as a local file. Any program requested by the user may reference the local file just as it does the standard local files INPUT and OUTPUT. The modified local file may be restored to the permanent file set by the command SAVEPF. The command EXECPF, with a parameter that specifies a binary program file, performs the same function as READPF, and in addition loads and executes the named file. Various forms of the COPY command write the contents of one file to another.

The interactive system is structured as a command interpreter. Commands entered at terminals may name system routines, or they may name binary files attached to the user job as local files. For a profile of the useage of the interactive system, it is convenient to partition the commands available to users into the following classes:

a) Utility commands. Many of the commands in this class, such as READPF, SAVEPF, COPY, etc., specify various file positioning and transfer operations. Others control file names and job parameters.

b) Editors and interactive compilers. There are several text editors in the system; the one most often used is called EDIT. The BASIC compiler has an interactive mode which can be distinguished at the command level. These processors are characterized by a large amount of interaction with the terminal once the command has been entered.

c) Language Processors and other applications software. FORTRAN and BASIC are the most heavily used languages in this systesm. The most popular FORTRAN compiler is called RUN. The BASIC compiler is included in this class when it is not in the interactive mode.

d) User-written binary program files. Used as a command, a file name causes loading and execution of the program within the file.

Table One shows the frequency of usage of the most heavily used commands of these classes, relative to the usage of the entire class, and the average amount of processing time required for each. The processing requirement is specified in both CPU and peripheral processing unit (PPU) time. Most of the PPU time represents channel queueing and data-transfer time, so the PPU requirement is a gross measure of the I/O requirements of a program.

A combined measure of total resource required is CPU time plus one-sixth PPU time. This will be justified later. The last column of Table One is the ratio of the combined (CPU and PPU) time requirements for the individual command, weighted by the relative frequency of the command's use, to the total processing requirements for the entire class. From this table, it can be seen that the processing for the Utility class is broadly distributed over many commands as opposed to the concentration of processing Edit and Language Processor commands. Also, the average CPU times of the Utility class are much smaller than that of the other classes, and by the high ratio of PPU to CPU processing, it is seen that Utility command processing is I/O bound.

Table Two shows the frequency of use and processing requirements of the command classes. In spite of the low processing requirements of the Utility commands, their high frequency of use makes their total processing requirements in the system comparable to that of the other classes.

Table 1

Profile of Usage Frequency and Resource Requirements of Commands

Command	Frequency	Average CPU Time (sec)	Average PPU Time (sec)	Percent of Class Processing
READPF	15.6	.130	1.99	30.3
REWIND	9.3	.004	.43	2.8
RETURN	6.9	.004	.41	2.1
SAVEPF	6.0	.152	2.98	16.5
SHOW	5.6	.008	.94	3.8
49 others	56.6	.016	1.04	44.5

a) Utility Commands

Command	Frequency	Average CPU Time (sec)	Average PPU Time (sec)	Percent of Class Processing
EDIT	91.5	.8	6.12	93.6
BASIC	6.4	.57	4.37	4.7
TEXEDIT	2.1	.71	4.55	1.7

b) Edit Commands

Command	Frequency	Average CPU Time (sec)	Average PPU Time (sec)	Percent of Class Processing
RUN	38.8	4.7	1.87	41.9
SPSS	12.0	1.1	1.45	3.4
BASIC	9.3	5.3	1.55	11.1
PASCAL	7.7	6.1	2.00	10.7
COBOL	7.1	7.0	3.53	11.9
24 others	25.2	3.6	1.20	21.0

c) Language Processor

Command	Frequency	Average CPU Time (sec)	Average PPU Time (sec)	Percent of Class Processing
User Program	100.	3.1	1.01	100.

d) User Programs

Table 2

Processing Requirements for Command Classes

Command Class	Frequency	Average Processing	Percent of Total
Utility	.744	.23	18.4
Edit	.112	1.78	20.6
Language Processing	.080	4.64	39.2
User Program	.064	3.27	21.8

The figures in Tables One and Two were accumulated from a search of the dayfile for three consecutive days. These dayfiles contained the records of 1203 terminal hours, during which 27,805 commands were entered. The percentages of processing time in these tables are based on the total user-requested processing, exclusive of overhead. No data for overhead occurring for this system has been accumulated, but overhead figures for a similarly-structured system (5) indicate that under heavy load, approximately twenty percent of the total CPU time is spent in routines that perform system overhead functions.

Detailed Repetition of Commands

The system dayfile contains a record of the commands and the parameters of the commands issued by the interactive users. Potentially repetitive program executions will be represented in the dayfile by instance of repeated use of a command with constant parameters. It is considered unlikely that the repeated use of a command with varying file name parameters will produce any significant degree of resource-allocation repetition, hence, this form of command repetition has not been measured. Yet it is clear that the resource-demand sequences of many instance of file-operation commands are identical if their parameters specify files of the same length.

A detailed repetition of a command is said to occur each time an interactive user repeats a command with identical input parameter file names. Parameters that are not file names are not considered in detailed repetition. In searching for detailed repetition, notice is taken and adjustments made for several common file names that often signify different files. One outstanding example is the file LGO (Load and Go), which is the default name chosen by the RUN compiler for the object file of the compilation. Repeated executions of LGO are considered detailed repetitions only if the source inputs to the corresponding RUNs were identical file names. On the other hand, commands using files that have been renamed through the RENAME command, or differently named object files that have been produced from the same source file, are not associated with the commands using their alternate names.

The length of interactive sessions is distributed from a few seconds to many hours; the mean, in this system, is forty minutes, with twenty-seven commands to the interactive system. The probability thtat a command is repeated in the detailed sense increases with the total number of commands issued by the user. Figure One is the cumulative probability distribution for the number of commands issued between occurrences of detailed EDIT and RUN commands. This data was accumulated from the occurrences of EDIT and RUN commands for which at least seventy of the user's preceding commands (possibly crossing session boundaries) were apparent in the dayfile. If the command was a repetition, the inter-reference time is used in computing the probability distribution. If no earlier use of the command was found, the command inter-reference time is greater than seventy and possibly does not exist. The difference between one and the limiting value of this cumulative probability distribution is the probability that the command is not a repetition.

For the EDIT and RUN commands, the limiting value is almost reached after thirty commands. This characteristic is typical of the remainder of the interactive commands as well. The approximate limiting probabilities (probability of detailed repetition) are given in Table Three. These figures represent an upper bound on the degree of repetitiveness of the processing associated with the various commands. It should be noticed that the commands with the highest frequency of use also have the highest probability of detailed repetition. The total number of times a detailed command is used is computed as the reciprocal of the detailed repetition probability.

Figure 1

Probability of Detailed Repetition
of the EDIT and RUN Commands

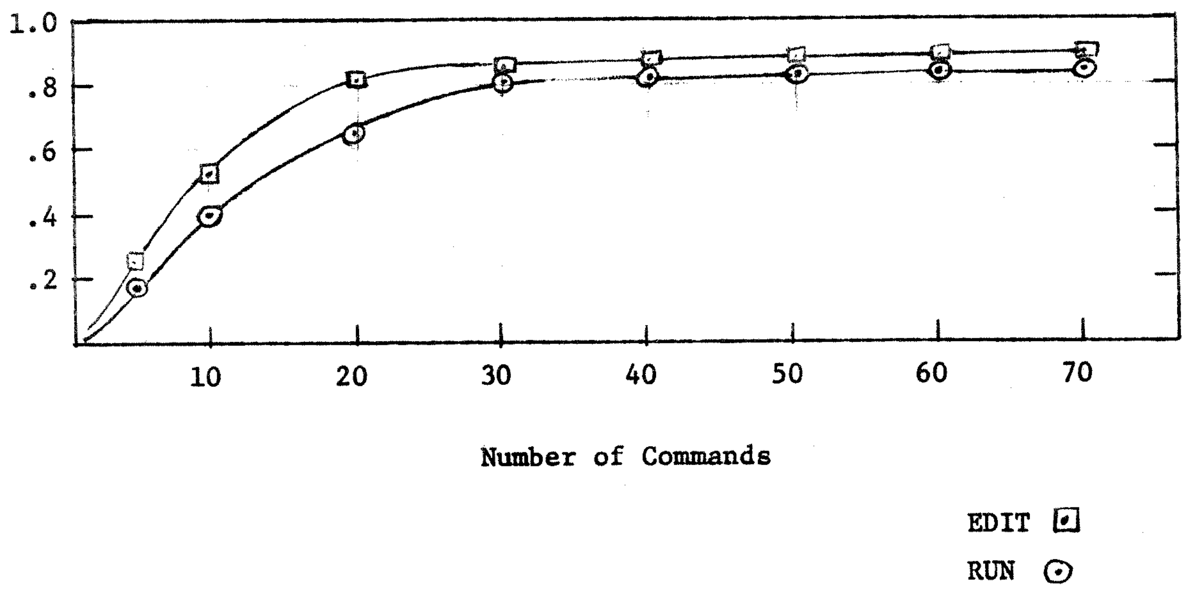


Table 3

Proabability of Detailed Repetition of Commands

READPF	.83
REWIND	.92
RETURN	.42
SAVEPF	.90
SHOW	.67
49 others	.53

a) Utility Commands

EDIT	.92
BASIC	.90
TEXEDIT	.84

b) Editors

RUN	.87
SPSS	.67
BASIC	.84
PASCAL	.82
COBOL	.74
24 others	.69

c) Language Processors

Program Files	.91

d) User-Written Programs

4. Resource-Demand Repetition in the UT-2D Interactive System

Each instance of detailed repetition of a command in the interactive system represents a processing sequence that is potentially identical to a previous processing sequence. The degree of this repetition will be measured in terms of the percentage of the total processing time that is actually close repetition (in terms to be specified precisely in this section) of the preceding sequence, until an irreconcilable difference takes place. In this section, the essential hardware and operating system backgrounds for these measurements are presented, and the method of extracting this data outlined. The results of the measurements are presented in section five.

Hardware Characteristics of the UT-2D Interactive System

User programs in a CDC 6400 system are multiprogrammed in a large (64K sixty-bit words) main memory, and executed by a single fast CPU. A pool of seven peripheral processing units (PPU's), which are twelve-bit minicomputers, each having its own memory unit (4k of twelve-bit words), execute auxiliary functions including I/O for the user programs. Each function performed by a PPU is called a peripheral program (PP). Each PPU can access any of the twelve channels which read and write PPU memory. The PPUs can read and write main memory, and interrupt the central processor. The central processor cannot access PPU memory, nor interrupt a PPU. The PPU's are then ideally suited for control functions, while the CPU is a slave. One PPU, called MTR (monitor), constantly polls the user program active on the CPU to determine if operating system service is needed, and, upon finding a request, assigns the required PP to an available PPU. But not all of the operating system functions are performed in the PPUs. Some auxiliary functions reside in main memory, in a portion of the operating system called CMR (central-memory-resident), and are executed when a PPU interrupts the central processor.

The interface between each user program and the operating system is relative location one of the memory area assigned to the user. The starting address of the active user program is called RA (relocation address), hence the request for operating system service is placed in location RA+1, and is known as an RA+1 call. MTR polls location RA+1 in looking for operating system requests, but, since MTR also polls the other PPU's and performs bookkeeping functions, it may allow the user program to wait as much as two or three milliseconds before responding to an RA+1 call. The record of RA+1 calls by a user program is a record of all its service request, much like the SVC instruction in other computer systems. Typically, after a user program has placed an RA+1 call, it goes into a busy idle loop waiting for the location RA+1 to become zero, indicating that MTR has accepted the request. It will then place an RCL (recall) code into location RA+1; this is effectively the 'wait' semaphore indicating the program will block for a response from the previous request. If the request was for input from an interactive terminal, a special system peripheral program rolls the job out of main memory and places it in 'terminal wait' status.

The hardware resources required by a user program are 1) main memory and 2) CPU processing time. Changes in main memory allocation are requested through an RA+1 call (a system PP requests main memory for a job that is not allocated any main memory), and each user program is allocated CPU time (in quanta) until it signals termination of its processor requirement. A program also requests, via RA+1 call, 3) PPU processing, and, during a PP run to perform I/O, 4) I/O devices and channels. The record of RA+1 calls within the processing time of a job, and the requests made by the corresponding PP runs, reflects the total hardware resource requirements of the user program.

Method of Tracing Resource Demand Characteristics of User Programs

Detailed measurements of the processing activity in the UT-2D system are possible through the use of an on-line trace facility. While the trace is in operation, every communication between the processors of the system is recorded in a buffer in main memory, along with timing data (to a quarter-millisecond precision). This buffer is periodically dumped to tape. The trace session ends when the tape is filled, which under moderate load takes about twenty-five minutes. The actual recording is done by CMR. In order to record interprocessor communications that do not involve CMR, such as an RA+1 call requesting a service function performed by a PPU, MTR interrupts the CPU for a special invocation of CMR.

The following typical sequence of interaction is given as an example of the recording technique.

a) A user job places a request for service in its location RA+1. It is recorded when MTR notices the request in its polling loop and invokes CMR.

b) MTR assigns a PPU to handle the request. This event is recorded as the assigned PPU invokes CMR to locate the PP required for this request.

c) The PP requires assistance from CMR. This event, as above, is recorded after CMR is invoked by interrupt. The PP will wait in a busy idle loop until the service is completed.

d) The peripheral processor requests and releases channels and devices to effect I/O operations. These reservations are handled by MTR, hence MTR invokes CMR to record them.

e) The user job that requested the service issues the RA+1 call RCL, to indicate that it will block until a signal to proceed is received. MTR invokes CMR to record this RA+1 call, and removes this central program from the queue for CPU service.

f) The PP issues the 'awake' signal, called RCP, so that the central program may proceed. This synchronization communication is an indication to MTR to place the central program back on the queue for CPU service. The PP issues this signal by placing it into a main memory location that MTR polls during its main loop. Upon finding the request, MTR invokes CMR to record it.

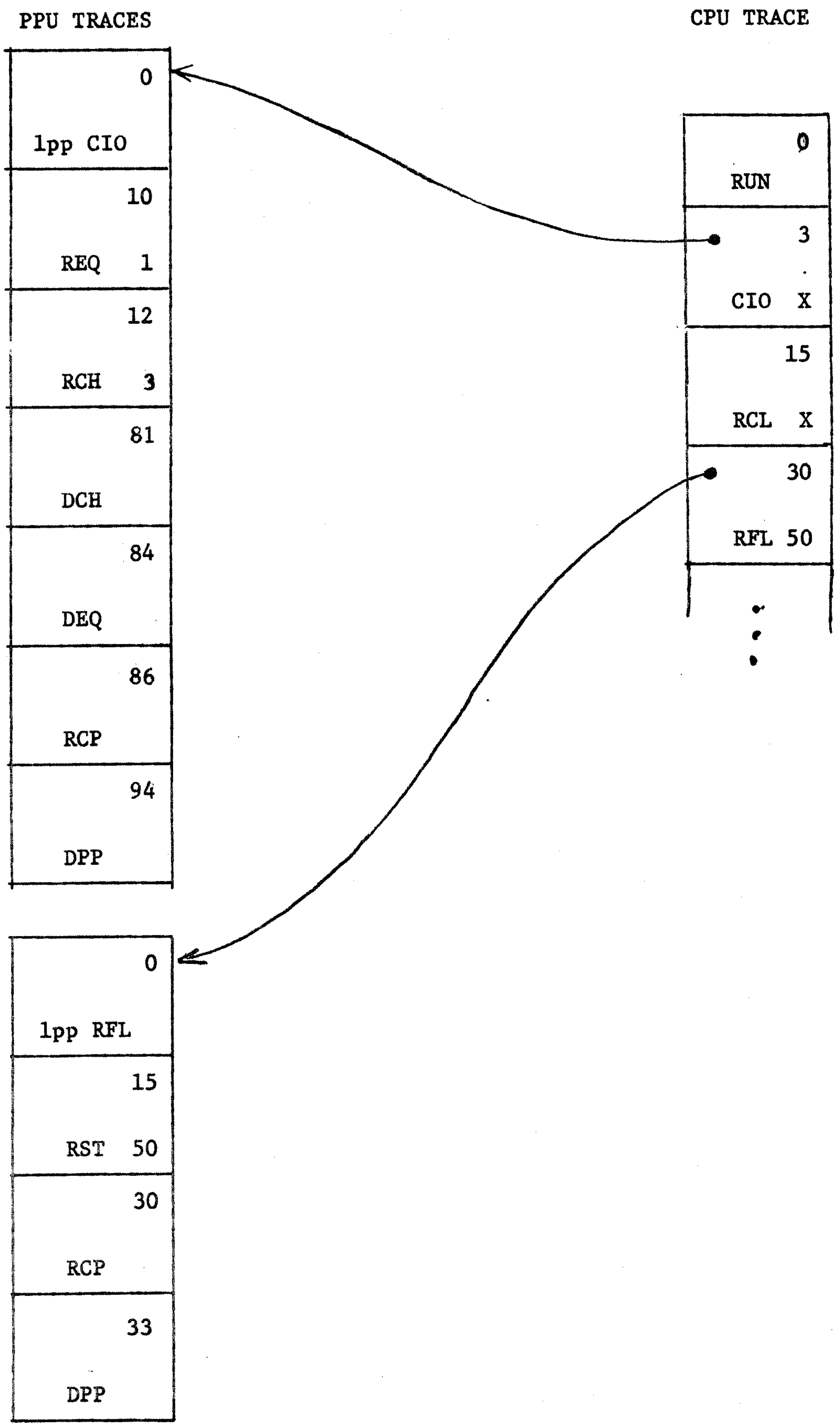
MTR also invokes CMR to record each swapping of the CPU in the round-robin CPU scheduling cycle. Each time it is invoked, CMR records the CPU burst of the last user of the CPU. Memory scheduling activities are effected by special PP's that are not called by means of the RA+1 call. Their activities are recorded through their communications with MTR in the way described above.

The microscopic record of the processing activity provided by the trace facility is amenable to many forms of processing. It is used as the record of the system's activity from which the user program resource-demand sequences are obtained. This technique of recording resource-demand characteristics does involve considerable overhead, but artifact of the additional interrupts due to the trace does not appear in the user program traces extracted from the system trace. A trace decomposiiton program scans the system trace and accumulates the events corresponding to each job. It also adjusts the times of the jobs' events to be relative to the processing time of the job, removing all delays due to the system such as time in queue for the CPU or a PPU, and all system effects such as the memory swap-in and swap-out sequences.

A simplified example of the output of the trace decomposition program is shown in Figure Two. The elements of the trace are (time, event) pairs. In this example the Fortran RUN compiler is executed. It calls for an I/O operation after 3 milliseconds of processing, and then halts for I/O completion after 15 milliseconds of processing. The PP sequence corresponding to the I/O call requests and uses device 3 or channel 2 and then, after 86 milliseconds, recalls the central program. The central program requests a 50k block of memory in the

Figure 2

Example of Resource-Demand Sequences
Extracted from CDC 6400 System



RFL (request field length) call after 30 milliseconds of its processing.

No claims are made for the uniqueness or practicality of this format for the representation of the resource-demand sequence of a user program. The representation has been shown to be stable, in that all the effects of the multiprogramming environment of the user job have been filtered out. With a few minor exceptions (which will be noted below) the job traces obtained from several runs of a single job with constant data values are identical.

Resource-demand Variation in Compilation

It is of some interest to note a few aggregate data values showing the consistency of this representation as the processing for a single job is varied by modifying its input. Table Four shows the results of three compilations of a Fortran program; first with 17 errors, then with five, and then without error. The aggregate resource-requirements of these jobs remain quite constant with this degree of variation of the input program.

The Measurement Precision

The resource demand sequences for extracted multiple runs of a single job with identical data have varied only in the timing of the events in the sequences, and this variation has been bounded. The variations in the CPU burst times result from the following:

a) CPU burst times for the user jobs are summed from the processing periods between interrupts for CMR service and preemptions due to the round-robin scheduling. When the trace is in operation, a job remains on the processor about two milliseconds between interrupts. The precision of the clock used in the event recording is a quarter-millisecond.

b) Each job CPU burst is ended with an RA+1 call. There is an unpredictable delay of about a millisecond in the time at which the event is recorded, due to MTR's polling cycle.

Table 4

Aggregate Resource Requirements for Compilation of a Fortran Program

	Total CPU Time	Total Supervisor Calls	Total PPU Time
Run 1: 17 errors	6.31	152	3.54
Run 2: 5 errors	6.94	171	3.81
Run 3: no errors	6.74	163	3.76

c) The time required for any computation varies with the degree of memory interference from simultaneous I/O operations.

CPU burst times for identical computations taking more than ten milliseconds do not vary by more than twenty percent due to the above reasons. The channel queueing and holding times for repeated instances of a single I/O operation have much greater variability due not only to the upredictable delay for disk arm positioning, but to interference due to I/O operations performed by another processor that shares the system disk units. (The system disk units are shared with the CDC 6600 system processing batch jobs.)

The algorithm used to search for duplicated processing sequences must allow for the limitations of the precision of measurement. Therefore, a resource-allocation sequence is considered to be a duplicate of another if their CPU burst lengths do not differ by more than twenty percent. The PP sequences must be the same, but no limit is placed on the differences in the channel holding times in order for one sequence to be considered a duplicate of another.

Procedure for Searching for Repetition

In observing the resource-demand sequences resulting from several instances of a single command, it has been found that they occasionally differ for some initialization period, and then become identical. But the algorithm that could discover these instances of repetition would be extremely complex, hence, sequences that differ in an initialization period do not represent repetition. On the other hand, some pairs of sequences differ only in the length of one CPU burst, or in one RA+1 call, or have one more or one fewer RA+1 calls. The procedure for searching for resource-demand repetition is as follows:

1) Given two resource-demand sequences that correspond to detailed repetitions of a command, comparison for repetition is initiated with pointers to the first processing bursts of each.

2) If the bursts are within twenty percent of each other, the pointers are

advanced. If the service requests are identical, the pointers are advanced. If nonzero, the error flag for the comparison is decremented by one.

3) If a mismatch is found in step two, comparison continues, but forks into three parallel comparison sequences. The error flag for each of these is set to the current error flag plus two. The three parallel comparisons involve 1) setting the pointer forward one step in each sequence (skipping the mismatch) and 2 & 3) setting the pointer in one sequence forward by three (skipping the mismatch and one burst and RA+1 call) and by one in the other sequence (skipping the mismatch).

4) If the error flag of a comparison ever exceeds three, use of that set of pointers is abandoned. If no comparison possibilities remain, the sequences are no longer repetitive.

This algorithm was designed to overcome obvious transients in sequences that are indeed repetitive, under the constraint that its logic remain simple enough to be used dynamically by a predictive scheduler that uses (and must verify the validity of) the previous resource-demand pattern. There is no theoretical justification of it.

This procedure for determining whether two resource-allocation patterns are duplicates was applied to each pair of processing sequences found during the trace periods that resulted from detailed repetitions of a single command. The degree of repetition is determined as the ratio of the total processing time (the sum of central and peripheral processor times) of the second run that was a repetition of the first, to the total processing time of the second run.

5. Results: Resource-Demand Repetition

Resource-demand sequences of all the interactive jobs in the system were collected by the means outlined in the last section from five trace periods, taken on different days at various times. In all, 57 complete and 107 partial interactive sessions representing 133 different users were measured. The first column of Table Five contains the total number of instances of detailed command repetitions found in the trace periods. Each of these represent a potentially repetitive resource-demand sequence. The second column of Table Five indicates the degree of resource-demand repetition found in the instances of detailed command repetition. This is called the sequence repetition factor. It is computed as the ratio of the total amount of processing time that was repetitive to the total processing time that occurred in the repetitive uses of the commands.

The results show a great degree of repetitiveness within each class of commands except for Text Editor class. Repetition depends on a program being run with essentially constant data. For all commands except the editors, the input data is mostly in the files named as parameters of the commands, with little data being supplied by interaction with the terminal. But examination of a sample of traces of processing within EDIT shows an average of more than ten interactions with the terminal on each EDIT call. Since ten percent of all commands are EDIT calls, this indicates that more commands are directed to EDIT than to the interactive system command interpreter. (This agrees with data shown in (1)). The interactiveness of the text editors causes them to take divergent processing sequences in their various invocations.

The Utility commands show moderate degrees of repetitiveness in the mean, although the repetitiveness factors for the various Utility commands were broadly distributed.

Table 5

Degree of Repetitiveness

Command	Instances	Sequence Repetition Factor (percent)	Normalized Repetitiveness (percent)
READPF	70	85.	21.3
REWIND	44	99.	2.6
RETURN	41	100.	1.9
SAVEPF	42	78.	11.6
SHOW	27	61.	1.6
49 others	223	63.	14.9
TOTALS	447		43.9

a) Utility Commands

Command	Instances	Sequence Repetition Factor (percent)	Normalized Repetitiveness (percent)
EDIT	129	5.2	4.3
BASIC	2	3.1	.04
TEXEDIT	2	3.8	.16
TOTALS	131		4.5

b) Edit Commands

Command	Instances	Sequence Repetition Factor (percent)	Normalized Repetitiveness (percent)
RUN	67	86.	31.4
SPSS	41	71.	1.6
BASIC	33	75.	7.0
PASCAL	9	88.	7.6
COBOL	17	84.	6.2
24 others	20	80.	11.6
TOTALS	187		65.4

c) Language Processor Commands

Command	Instances	Sequence Repetition Factor (percent)	Normalized Repetitiveness (percent)
Program Files	107	89.	80.1

d) User Programs

The most significant result is the high degree of sequence repetition in the Language Processor and User Program classes. In both classes, over ninety percent of the detailed repetitions of a command resulted in resource-demand sequences that were duplicates of their predecessors for at least ninety-five percent of the processing time of the run. For the remainder of the command repetitions, the length of the resource-demand repetition was approximately uniformly distributed in the range of zero to ninety-five percent of the processor time of the run.

The product of command repetition factor (Table 3) and sequence repetition factor is the total repetition factor for the command. Weighting this product by the fraction of processing of the class that the command represents (Table 1) provides the command's contribution to the repetitiveness factor of the class. This, called the normalized repetitiveness, is shown in the third column of Table Five. The sum is the repetitiveness factor for the entire class.

Last, the degree of repetitiveness of all the command classes combined is computed as the sum repetiveness for each class, (Table Five) weighted by the percentage of total processing in the class (Table Two). This figure is fifty-two percent.

6. Conclusions

A primary goal of this study was to determine whether the repetitiveness present in the workload of an interactive system was fruitful source of predictive information for use by the system schedulers. Resource-demand repetitiveness was defined by an algorithm that emulates the operation of a scheduler checking the pattern of resource requirements of a previous run against the immediate requirements of a program, while it simultaneously schedules resources on the basis of future requirements shown in the pattern. If the degree of resource-demand repetitiveness were one hundred precent, the pattern of requests by the previous run would be a perfect predictor, and the scheduler could operate deterministically. But the degree of repetitiveness for all of the command classes combined is only fifty-two percent. With an expenditure of greater computational effort to find repetitive sequences, it may be possible to increase this final figure by a few percent-and it will certainly vary somewhat with the mix of job types on the system, and even more than somewhat if measured in a different interactive system. Nevertheless, certain conclusions can be drawn from even this imprecise result. The first question is that of predictive scheduling.

Schedulers that operate on the basis of partially correct predictive information have been examined in simulation (6). Although the hardware and software assumptions of the simulation model differed from the system of the present study, the results provide evidence of the degree of repetitiveness (or predictability) that such schedulers require in order to be effective. Various straightforward predictive CPU and memory scheduling algorithms, such as shortest burst time, shortest time-to-completion first, etc., were compared with standard algorithms. Not surprisingly, the results showed that properly designed predictive algorithms could achieve higher utilizations and throughputs than standard algorithms. The margins, however, were not more than a few percent, because under proper loading, the simulated system is capable of high CPU utilization (9). When the correctness of the predictive data was degraded to seventy percent, the predictive algorithms lost their advantage over the standard nonpredictive algorithms.

Despite the differences in the assumptions of the two studies, the gap between what seems to be required and what is apparently available is large enough to make this approach to resource scheduling seem not very hopeful.

On the other hand, a positive result for computing systems design can be drawn from this study. It is apparent that the language processors account for a large part of the processing activity in this and most other computing systems. These processors also show a large degree of resource-demand repetition, which suggests that they are performing a great deal of logical repetition. If the compilers were to be redesigned to interface with the text editors in an interactive system so that they might obtain records of the statements that have been modified and recompile only those statements, a significant degree of processing would be eliminated. Using the data for resource-demand repetition presented here, eliminating only half of the repetitive processing of the RUN compiler reduces the total processing workload of the system by over six percent.

The high degree of resource-demand repetition in the user-written programs should motivate a study of their logical repetition. The elimination of logically-repetitive executions would require automatically recording, storing and retrieving the data-transformations effected by various program paths. Compilers could then be designed to position and insert checkpoints automatically. The run-time environment of the program may then monitor these checkpoints, and shortcut execution paths that have already been performed. The practicability of such a scheme cannot be quantified by the study of resource-demand repetition.

REFERENCES

1) Boies, S.J. and Gould, J.D. "User Performance in an Interactive Computer System", Fifth Princeton Conference on Information Sciences and Systems, 1971.

2) Boies, S.J. "User Behavior in an Interactive Computer System", IBM Systems Journal, Vol.13, No.1, 1974.

3) Estrin, G., Muntz, R.R. and Uzgalis, R.C. "Modelling Measurement and Computer Power", Proceedings, S.J.C.C., 1972.

4) Howard, J. "A Large-Scale Dual Operating System", Proceedings, ACM National Conference, 1973.

5) Johnson, D.S. "A Process-Oriented Model of Resource Demands in Large Multiprocessing Computer Utilities", Ph.D. Dissertation, The University of Texas, 1972.

6) Noetzel, A.S. "Simulation Studies of Predictive Scheduling", Technical Report No.37, Department of Computer Sciences, The University of Texas, 1974.

7) Noetzel, A.S. and Haley, J.R. "The Decomposition of a Multiprocessor Event Trace into User Program Resource Demand Patterns", Technical Report No.49, Department of Computer Sciences, The University of Texas, 1975.

8) Sherman, S.W. "Trace Driven Modelling Studies of the Performance of Computer Systems", Ph.D. Dissertation, The University of Texas, 1972.

9) Sherman, S.W., Browne, J.C. and Baskett, F. "Trace Driven Modelling and Analysis of CPU Scheduling in a Multiprogramming System", Proceedings, SIGOPS Workshop on System Performance and Evaluation, Harvard University, 1971.