Computer System Simulation: An Introduction

M. H. MacDOUGALL

Control Data Corporation, Palo Alto, California

This is a tutorial paper describing the construction of a basic simulation model for a disk-based multiprogrammed computer system. First, the job processing sequence of the system to be simulated is described and techniques for generating the simulated job mix are discussed. The structure of simulation models, events, and events scheduled are then introduced. A simulation model for the multiprogrammed system is presented and the event routines comprising the model are discussed. The use of GPSS, SIMSCRIPT, and FORTRAN in constructing a simulation model is reviewed.

Key words and phrases: simulation, multiprogramming, computer system analysis

CR categories: 4.30, 4.32

INTRODUCTION

This paper is an introduction to methods of developing simulation models and simulators for computer systems. Experience indicates that the most effective way to introduce these methods is by actually going through the development of a simulation model for a simple system. The system selected as a basis for this approach is composed of the basic elements of a disk-based multiprogramming system; in keeping with the national pastime of the computing profession, this system is called BASYS.

The techniques employed in constructing a simulation model for BASYS are readily extended to more complex problems; in fact, such a model is a useful foundation for the development of simulators for several contemporary multiprogramming systems. A BASYS simulator also can be a useful tool in its own right in developing the analyst's understanding of some of the basic interactions in system behavior.

This discussion is not oriented toward any particular simulation language. The characteristics of several of the most widely used simulation languages are reviewed and, as we shall see, a BASYS simulator can be implemented easily in FORTRAN.

BASYS DESCRIPTION

The components of BASYS are shown in Figure 1. The system is comprised of a central processor (CPU), a central memory (CM), and a movable-head disk. It is assumed that the purpose of the simulation analysis is to determine the relative merits, insofar as the utilization of the central processor is concerned, of (1) adding additional central memory, or (2) replacing the disk with a faster model. It is further assumed that the nature of the job mix is such that card-to-disk, disk-to-printer, and disk-to-CM (i.e. program loading) operations have little effect on the parameter of interest; the combined I/O rate resulting from these operations is much less than that generated by central processor programs.

The portion of the job processing sequence to be simulated is described below and illustrated in Figure 2:

1. A job arrives at the system.

CONTENTS

2. Central memory space is requested for the job. If space is available, it is allocated to the job; if space is not available, the job is entered in a queue.
3. The central processor is requested; if it is free, the job is assigned to it and executes until an I/O request is encountered or until execution completes. If the central processor is busy, the job is entered in a queue.
4. When a job issues an I/O request, it releases the central processor. (If there is another job waiting in the CPU queue, it now is assigned to the central processor.) If the disk is free, it is assigned to the job to process the I/O request; if the disk is busy, the request is entered in a queue.
5. On completion of processing of an I/O request, the disk is released and the central processor requested once again. (When the disk is released, the disk queue is checked; if there is a waiting request, it is assigned to the disk.)
6. When a job completes execution, it releases the central processor and its central memory space is released. (The CM queue is checked to determine if there is a waiting job to which space now can be assigned, and the CPU queue checked to determine if there is a waiting job which now can be assigned to the central processor.)
7. The job leaves the system.

This is a comparatively simple process; however, its simulation employs most of the techniques required for more complex processes and can be extended readily to incorporate other parts of the job processing sequence, such as the card-to-disk operation.

JOB GENERATION

An examination of the job processing steps described in the preceding section shows several job characteristics which must be specified in constructing a simulation model of this process. These characteristics (numbered in accordance with the appropriate steps in Figure 2) are as follows:

1. Job interarrival time—the interval between the arrival of successive jobs at the system.
2. Central memory space requirement.

3,4. Interrequest time—the interval between a job's assignment to the central processor and its issue of an I/O request. In BASYS, a job releases the CPU upon issuing an I/O request; the total CPU time used by a job is divided by I/O requests into a number of intervals called interrequest times.

3,4. Number of I/O requests (which also determines the number of interrequest times, and hence the job's total CPU time).

4,5. I/O record length (this, in BASYS, determines the I/O request processing time).

The "arrival" of a job in the simulation model is marked by the generation of a set of values for these characteristics. To provide the variability of job characteristics which occurs in a real-world job mix, values for these characteristics are assigned by generating random samples from appropriate distributions. We shall assume that a study of the actual system in operation has provided data on the job characteristics so that distributions of job interarrival times, CM space requirements, etc., have been determined.

In some cases it is possible to fit a known probability distribution function to the empirical data and generate a sample from that distribution using analytical methods. In other cases, no convenient fit can be found and so samples are generated directly from the data. Suppose job CM requirements are distributed as shown in Figure 3. A random sample from this distribution is obtained by generating a random number p uniformly distributed in the range 0–1, and using as the CM space requirement sample the space value which corresponds to p. Suppose a p value of 0.4 is generated; the CM space requirement would then be taken as 35K (using the midpoint of the class interval). This process could be implemented in the simulation program by a simple table look-up procedure.

It is possible, of course, to tabulate

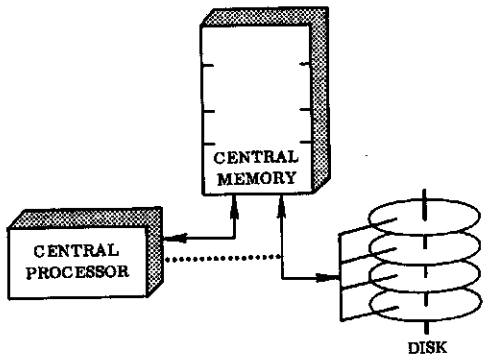

FIG. 1. BASYS components

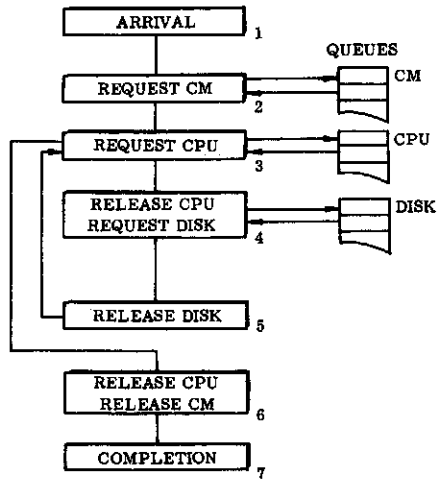

FIG. 2. Job processing steps in BASYS

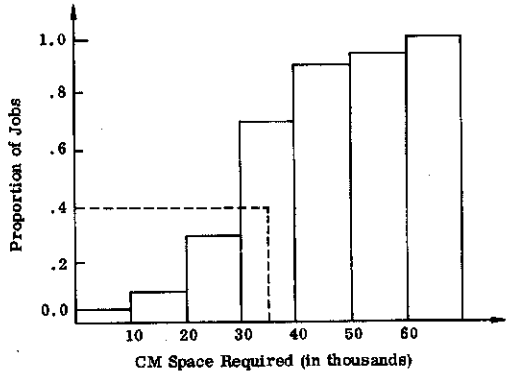

FIG. 3. Typical CM space distribution

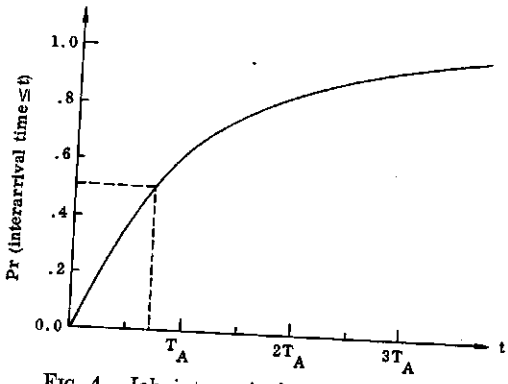

FIG. 4. Job interarrival time distribution

values and use a table look-up procedure to generate random samples from known probability distributions. However, samples from several commonly used distributions may be generated by using the inverse form of the distribution function. Suppose the distribution of job interarrival times is found to follow a negative exponential distribution with mean T_A, as shown in Figure 4. The distribution function for the negative exponential distribution is

$$Pr(x \leq t) = 1 - e^{-t/T_A}.$$

Since Pr is uniform on the interval 0, 1, a random sample from this distribution can be obtained by generating a random number uniformly distributed in the range 0–1 and using as the sample the corresponding value of t, as illustrated in Figure 4. This value of t can be computed using the inverse form of the distribution function,

$$t = -T_A \log_e (1 - p),$$

or, since $(1 - p)$ is distributed identically to p,

$$t = -T_A \log_e p.$$

Most program libraries provide uniform random number generators, and the above sampling process can be implemented easily in FORTRAN using such a generator; for example,

```
T = -TA*LOG(RANDOM(1))
```

The inverse of many distribution functions is difficult or impossible to obtain; for these, an approximation to the inverse may be found or some other sampling method employed. Sampling techniques are discussed at length by Gordon [2], Naylor et al. [5], and Tocher [7].

As a practical matter, it is exceedingly difficult to obtain statistics on the interrequest interval distribution and much less difficult to obtain data on total job CPU time requirements. Accordingly, interrequest intervals often are determined by generating a sample from the distribution of job CPU times, generating a sample from the distribution of the number of records per job, and dividing the former by the latter to obtain the mean interrequest interval for that job. Then, each time a job is assigned to the CPU, the duration of that assignment (the interval until an I/O request is issued) is determined by generating a sample from, say, the negative exponential distribution, using the computed mean interval as the distribution mean.

SIMULATOR STRUCTURE

Events and the Event List

In the simulator, a job is represented by an entry in a job table. This entry contains the characteristics established for this particular job as well as various counters for the accumulation of job-related statistics. As the job moves through the system—enters queues, is assigned to the central processor, etc.—its movement is reflected by moving a pointer to this job table entry, rather than by moving the entry itself.

The progress of the job through the system is marked by the occurrence of a series of *events*. These events correspond to transition points between operations, or activities; they represent a change of state. Some events of significance might be:

- the assignment of a job to the CPU (the transition between the "wait" and "execute" states);
- the release of the CPU to wait for

the completion of an I/O request (the transition between the "execute" and "wait" states).

It is convenient to develop the simulation program so that there is a one-to-one correspondence between the events of the simulation model and the simulator routines. With this philosophy, each simulator "event routine" essentially does two things: it performs the operations whose initiation corresponds to the occurrence of this event and it predicts, for the job for which the operation was performed, which event is to occur next and at what time it is to occur. For example, the basic steps in an event routine "Assign CPU" might be as follows:

- set a flag marking the assignment of the CPU to this job;
- predict the time at which this job will resease the CPU and determine if the event corresponding to the release is to be "end execution" or "issue I/O request".

As each simulated event occurs, then, the time and identity of the next event to occur for that particular job are determined. In the simulator, job processing is a quasi-parallel operation, reflecting the true simultaneity of the system being simulated. The predicted duration of an operation is added to the value of simulated time at which the operation was initiated to obtain the point in simulated time at which an event (the end of that particular operation) is to occur. Thus, if we were to obtain a snapshot of the clock (the current value of the variable representing simulated time) and a tabulation of the jobs in the system together with the next event identity and next event time for these jobs, at some randomly selected point during the simulation, the results might appear as follows:

TIME = 108075

JOB (J)	NEXT EVENT (E)	EVENT TIME (T)
63	I/O ISSUE	110042
22	RELEASE CPU	110121
88	JOB ARRIVAL	124003

This reflects the simulation of several parallel activities. However, the simulation program is concerned only with the start and end of these activities; these points comprise a sequence of events, the ordering of which is determined by the times at which these events are to occur.

Establishing this event sequence ("scheduling" the events) is facilitated by use of an *event list*, although there are other ways to accomplish this. In the BASYS simulator, it is assumed that the event list is a linked list with four-element entries. One element of an entry is an event identifier, a second element is an event time, the third element is a pointer to a job table entry, and the fourth element is a link. Whenever an entry is made in the event list, its position is determined by the relative value of the event time; the list is ordered in ascending values of event times. The structure of this list is illustrated in Figure 5.

All the event routines in the BASYS simulator make entries in the event list, but only one routine (the Scheduler) removes entries from this list. All transfers to event routines are from the Scheduler and all exits from event routines are to the Scheduler. The basic steps in event scheduling are as follows:

1. The Scheduler removes the entry at the head of the event list. This entry specifies an event time T, an event identifier E, and a job table pointer J.
2. The Scheduler advances the clock (TIME) to the event time T specified in the first step, because this time represents the earliest of all the scheduled events to occur.
3. The Scheduler transfers control to the event routine designated by the event identifier.
4. The event routine performs the required processing for the job, determines its next event and event time, and inserts the event identifier, event time, and job table pointer in the event list. It then returns control to the Scheduler.

An event scheduling mechanism such as that described can be easily implemented in FORTRAN; contemporary event-oriented simulation languages provide similar facilities.

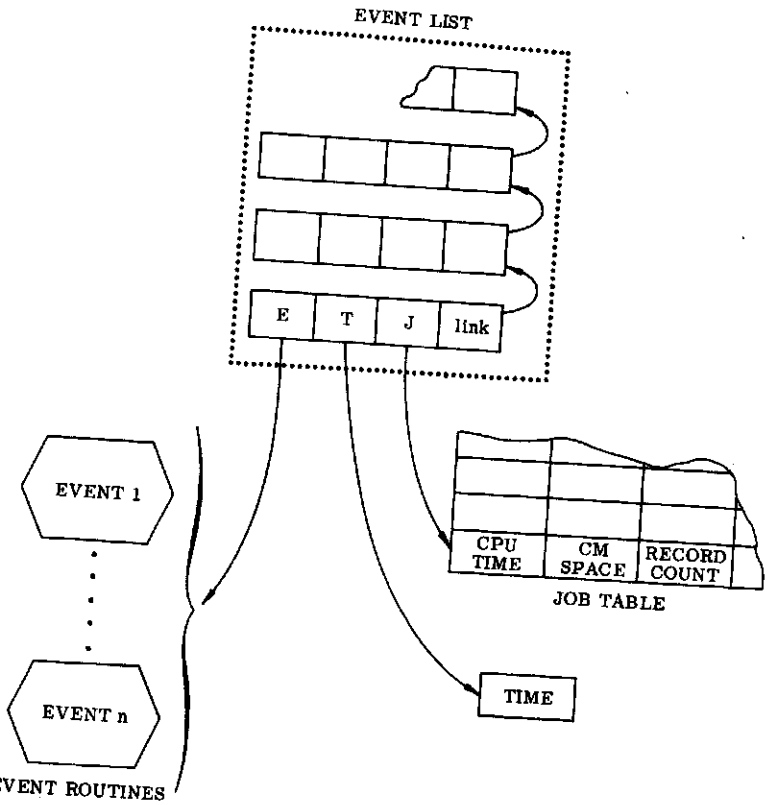

FIG. 5. Event list structure: E, event identifier; T, event time; J, job table entry pointer

There are instances where the next event time for a job cannot be predicted and so no entry for this job appears in the event list. This occurs, for example, when a job finds a facility busy and enters a queue. At the time at which this event occurs, it is impossible to predict the time at which this job will leave the queue and be assigned to the facility. When the facility is released, a job is selected from the queue according to the appropriate queue discipline and scheduled for the event which will result in its assignment to the facility. In the BASYS simulator, every job in the system is represented by the appearance of its job table entry pointer either in the event list or in a queue, but not both. This is not a general rule; for example, suppose a job did not release the CPU upon issuing an I/O request. In this case, its job table entry pointer might appear in the event list twice (once for the completion of the current I/O request and once for the issue of the next I/O request) or it might appear both in the event list and in a queue.

Queues

Another use of linked lists in the simulator is the maintenance of queues. In its most elementary form, a simple first-in, first-out queue might be mechanized as a linked list with two elements per entry, one element containing the job table pointer and the other a link. The queueing of a job would then be represented by adding an entry at the tail of the list; the removal of a job from the queue would be represented by removing the entry at the head of the list.

A more useful mechanism would provide a convenient way of handling queue disciplines in addition to first-in, first-out, and might also provide facilities for collecting statistics on queue behavior. One possible mechanism is illustrated in Figure 6. This employs the same four-element list structure used for the event list, with

TABLE I

Insert entry in queue	*Remove entry from queue*
1. $[J, Pr, \text{TIME}] \rightarrow$ list	1. Unlink entry at head of list
2. $\sum T_Q + (\text{TIME} - T_{last})\ Q \rightarrow \sum T_Q$	2. $\sum T_Q + (\text{TIME} - T_{last})\ Q \rightarrow \sum T_Q$
3. $Q + 1 \rightarrow Q$	3. $Q - 1 \rightarrow Q$
4. $\text{MAX}\ [Q, Q_{max}] \rightarrow Q_{max}$	4. $\sum T_W + (\text{TIME} - T_{in}) \rightarrow \sum T_W$
5. $N + 1 \rightarrow N$	5. $\text{MAX}\ [W_{max}, \text{TIME} - T_{in}] \rightarrow W_{max}$
6. $\text{TIME} \rightarrow T_{last}$	6. $\text{TIME} \rightarrow T_{last}$

the addition of a list header for each queue. Mean and maximum queue length and waiting time statistics are kept, and any queue discipline ordered on a single attribute may be employed. Queue entry and removal algorithms are outlined in Table I, using the notation of Figure 6.

Entries are inserted in the queue immediately behind the entry nearest the tail of the queue which has a Pr value equal to or less than that of the entry to be inserted. The list is then ordered in ascending values of Pr; should all entries have equal Pr values, the queue discipline is first-in, first-out. Ordering in this way lets us use the same list processing routines for queues as for the event list, but causes increasing priorities to correspond to decreasing Pr values.

Maintained in the list header are the current queue length, the time the queue last changed in length, the total number of queue entries, and a length-time product accumulator. Each time an entry is inserted in or removed from the queue, the product of the old length of the queue and the interval of time over which this length existed is accumulated (step 2, above). At the end of the simulation, the mean queue length can be obtained by dividing this quantity by the elapsed time:

$$\text{mean queue length} = \frac{\sum T_Q}{N}.$$

Each time an entry is inserted in the queue, the time of entry is recorded by setting $T_{in} = \text{TIME}$. When the entry is removed, the interval of time spent in the queue is accumulated (step 4 under "Remove entry from queue" above). The mean waiting time is obtained by dividing this sum by the number of queue entries:

$$\text{mean waiting time} = \frac{\sum T_W}{N}.$$

Note that, unless all jobs pass through the queue, this mean waiting time is conditional on a job's entering the queue.

Extension of this structure to accommodate ordering on more than a single attribute or to provide for the collection of additional statistics is straightforward. For example, the queue length distribution

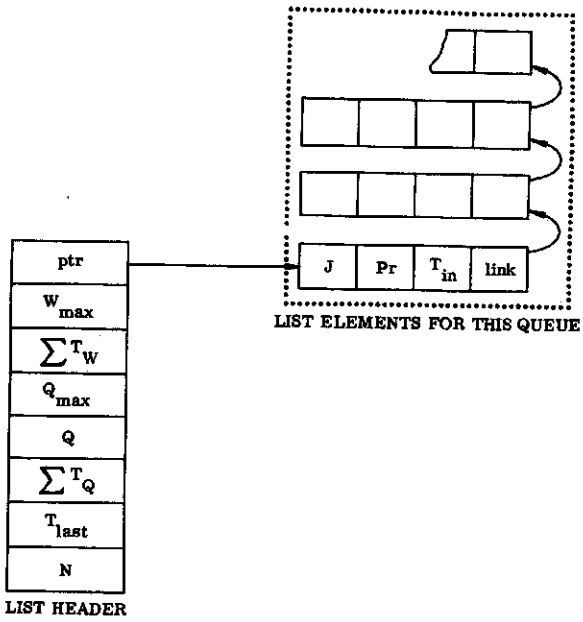

FIG. 6. Queue structure example:

ptr	pointer to head of list
W_{max}	maximum waiting time
ΣT_W	waiting time accumulator
Q_{max}	maximum queue length
Q	current queue length
ΣT_Q	length · time product accumulator
T_{last}	time of last entry/removal
J	job table entry pointer
Pr	priority or other ranking attribute
T_{in}	time of entry in queue
N	entry count

can be obtained by recording in a table the total length of time in which the queue was 1, 2, . . . , n entries in length and, at the end of the run, dividing each total by the elapsed time.

Similar queue processing facilities are available in most simulation languages, although the extent to which queue statistics are maintained and reported varies. This mechanism, like the event list, can easily be implemented in the form of FORTRAN subroutines, using a few list processing routines as a base.

THE BASYS SIMULATOR

The event routines which, together with an initialization routine and scheduling routine, compose the BASYS simulator are illustrated in Figure 7 and described below. The scheduling routine functions in the manner previously described; the initialization routine established system parameters (central memory size, disk transfer rate, etc.) and job mix parameters, and triggers the simulation by scheduling the arrival of the first job.

Event 1 marks the arrival of a job in the system. The characteristics of this job are determined by sampling the appropriate distributions and stored in the job table entry assigned for the job. (Another use of lists in the simulator is the recording of available job table entries.) For the BASYS simulation, the following job characteristics are generated:

- CM space required,
- number of records read/written,
- CPU time required,
- mean interrequest interval (computed by dividing the job's CPU time by the number of records),
- record size (assumed to be constant for a given job).

The next event for this job is scheduled by inserting the job table pointer (J), the event designator (E), and the event time in the event list. In our basic model, there is no delay between the time at which the job arrives and the time at which it requests central memory. Therefore, it would be possible to transfer directly to the event 2 routine or combine the event 1 and event 2 routines. However, the organization described makes it easier to expand the simulator to include, for example, the card-to-disk operation.

Event 2 is the occurrence of a job's request for central memory space. The CM requirement of the job (from the job table entry) is compared with the available CM space. If sufficient space is available, it is allocated to the job; otherwise, the job is entered in the CM queue. It is assumed that allocation of CM space to a job may require relocation of other jobs to provide sufficient contiguous CM space. The next event for this job (event 3) is scheduled to occur at a time equal to the current time plus the time required to relocate CM. (Although, in an actual system, this relocation may require interrupting the CPU, no provision is made to represent this in the basic model. Incorporation of interrupt processing into the model will be discussed later.)

Event 3 corresponds to a job's requesting the central processor. If the CPU is free, it is reserved for the requesting job; otherwise, the job is entered in the CPU queue. The number of records to be read or written and the mean interval between I/O requests were established when the job "arrived." If the record count has not been reduced to zero, a sample from a distribution of the appropriate form with this mean value is used to determine the time at which the next I/O request is issued. The record count is decremented and event 4 (release CPU) is scheduled for this time. If the record count has been reduced to zero, event 7 (job completion) is scheduled.

Event 4 is the release of the CPU by a job issuing an I/O request. Upon the requestor's release of the CPU, the CPU queue is examined; if there are jobs waiting, one is selected from the queue and scheduled for event 3 (request CPU). The job releasing the CPU is scheduled for event 5 (process I/O request). Both events are scheduled to occur at a time equal to the current time plus an overhead time (T_0) required to process the request. Note that if accounting for this overhead time is not required, events 4 and 5 could be combined.

Event 5 marks the initiation of processing for an I/O request. If the disk is busy, the job is entered in the disk queue. If the disk is free, it is reserved for the job and the request processing time is computed. For example, if positioning is required, the processing time might be computed as

$$T_D = T_P + T_L + T_T \times R,$$

where T_P is the positioning time, T_L is the latency time (a randomly selected fraction of a revolution time), T_T is the disk transfer time per word, and R is the record size in words. The next event

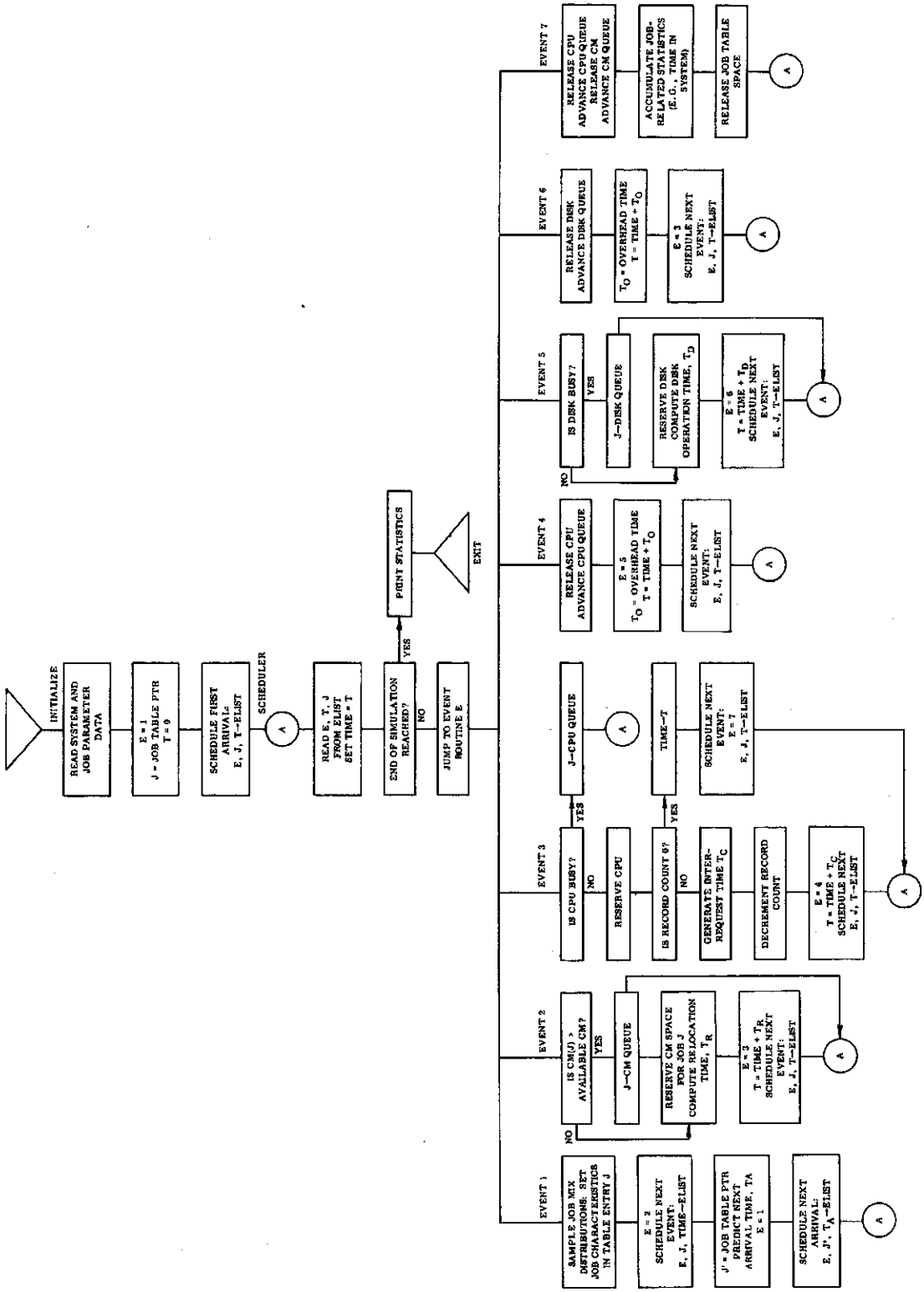

Fig. 7. The BASYS simulator

(event 6, release disk) is scheduled to occur at a time equal to the current time plus T_D.

Event 6 is the completion of processing for an I/O request. The disk is released and the disk queue checked for waiting requests; if there are waiting requests, one is removed from the queue and scheduled for event 5 (process I/O request). The job releasing the disk is scheduled for event 3 (request CPU). Here again, overhead time accounting is easily incorporated if desired.

Event 7 marks the completion of a job. The CPU is released and, if there are jobs in the CPU queue, one is removed and scheduled for event 3. The CM space allocated to the job is now released. If there is a job in the CM queue which now can be assigned CM space, it is removed from the queue and scheduled for event 2.

In some areas, such as in the accounting for overhead and relocation times, the foregoing description has been intended to be descriptive rather than complete. These and other features (e.g. disk-to-printer operations) may be incorporated into the simulation model in a number of ways.

INTERRUPT PROCESSING

Expansion of the basic simulation model will require incorporation of interrupt processing. There are innumerable types and levels of interrupt systems which may be encountered in practice. In this discussion, we shall be content with a single example: the interruption of a job executing on the central processor by a higher priority job. Many other types of interrupts can be handled in a similar if not identical manner.

Let's return for a moment to the description of the event 3 routine. This routine processed the assignment of a job to the central processor; upon returning control to the scheduling routine, it had

- entered the time of the next event, the event designator, and the job pointer in the event list, and
- modified the appropriate quantities in the job table entry (e.g. the record count) to reflect the processing to be performed during this interval.

Suppose the next event for this job is to occur at some time T_s, but the central processor is preempted by another job at some earlier time T_p. What do we have to do to properly reflect the interruption and eventual resumption of this job? (The preempting job is to be processed in the usual manner and presents no special problems.)

As a result of the preemption of the central processor, the event list now contains an invalid entry; the event scheduled for the interrupted job is not going to occur at the time specified and so one requirement is to cancel this event. The entry corresponding to this event is located in and removed from the event list (or the event identifier changed to indicate a null event). The exact method employed will depend upon the simulation language used. In general, locating the proper entry in the event list requires knowledge of both the event identifier and the job identification. Thus, when interrupt situations are to be simulated, the simulator should keep track of the user of a facility.

In order to keep job time accounting straight, the remaining CPU time in this interval ($T_s - T_p$) must be saved; when the interrupted job is restarted, it will be assigned to the central processor for this period. It is also necessary to save the event designator for the event which was cancelled. The interrupted job may be placed in an interrupt list or may be entered in the CPU queue, depending upon the design of the system being simulated. If the job is entered in the CPU queue, it must be flagged so that the event routine can distinguish between interrupted jobs on one hand and new jobs or jobs returning from an I/O wait on the other hand. The event 3 routine, in processing an interrupted job, performs only the following steps for this job:

- clear interrupt flag in job table entry;
- obtain remaining compute time and event identifier from job table entry;
- reschedule event to occur at current time plus remaining compute time.

Figure 8 shows how the event 3 routine

may be extended to provide interrupt processing.

IMPLEMENTING A BASYS SIMULATOR: SIMULATION LANGUAGES

There are many simulation languages, embodying several basic conceptual approaches to simulation modeling, in existence today. An excellent (although no longer current) survey of simulation languages has been prepared by Teichroew and Lubin [56]. The most widely available simulation languages are GPSS and SIMSCRIPT; these two, together with FORTRAN, are also (or, perhaps, therefore) the languages most often used in simulation modeling.

When a choice exists, there are several factors to consider in selecting a simulation language. Among these are ease of learning, expressiveness (the ease with which the model can be described in the language), compilation and execution speeds, reporting facilities, general computation capability, and execution time facilities (trace, display of contents of the event list, queues, etc.). The relative importance of these considerations depends upon the problems at hand. If the requirement is to build a number of different small- to medium-scale simulation models, GPSS may prove most efficient; for large models or models in which much general computation is required, SIMSCRIPT may be preferred.

In discussing the construction of the BASYS simulator, details of some of the major functions (event scheduling, queue maintenance) were described in sufficient detail to provide a basis for their implementation in FORTRAN. It is possible to construct a set of routines which, together with FORTRAN, constitute a simulation language of moderate power and relatively high compilation speed. One such FORTRAN-embedded language is GASP [45].

SIMSCRIPT [53] is an event-oriented simulation language somewhat resembling in form, and providing general computation facilities similar to, FORTRAN. Originally developed at the RAND Corporation in 1961–62, it has come to be widely used and is now available on most large computers (IBM 7090/94 and S/360; CDC 36/3800 and 64/6600; Philco 211/212; UNIVAC 1107/8; GE 625/35). A more advanced version, SIMSCRIPT II, has been developed by RAND and Simulation Associates [40].

SIMSCRIPT deals with *entities* described by parameters called *attributes*, with *sets*, which are collections of related entities, and with *events*. Entities may be *permanent*, existing for the duration of the simulation, or *temporary*, arriving and leaving during the course of the simulation. In a BASYS simulator, permanent

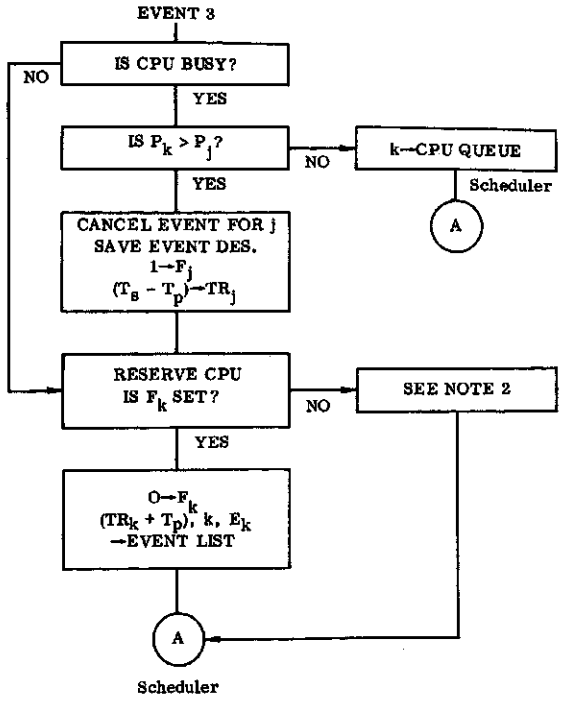

FIG. 8. Example of CPU assignment with priority interrupt

Note 1.
j pointer for job in execution
k pointer for job preempting CPU
P_i priority of job i
T_s next event time for job j
T_p current time
F_i interrupt flag for job i

Note 2. The processing performed here is essentially that discussed in the event 3 routine description.

entities would be system resources such as the central processor and central memory with attributes such as "busy" and "available space." Jobs would be temporary entities, possessing attributes such as number of I/O requests, CM space required, etc. The queues in the system would be defined as sets. SIMSCRIPT permits the elements of a set to be ordered on a first-in, first-out or a last-in, first-out basis, or to be ordered on the basis of a single attribute in either an ascending or descending sequence. Permanent and temporary entities, together with their attributes, and sets are specified on a special definition form and constitute a preamble to the program itself.

A SIMSCRIPT simulation program takes the form of a set of event routines. An event routine is analogous to a subroutine; it begins with and is named by an EVENT statement and is terminated with an END statement. Events are scheduled by a statement of the form

CAUSE *event-name* AT *event-time*

and may be cancelled via a CANCEL statement (as required for example, in simulating CPU interrupt processing).

The arrival of a job in the system would be effected via a CREATE statement, which generates a single instance of a temporary entity. The attributes of that job then would be assigned via a LET statement, which is an extended form of the FORTRAN replacement statement. Entering jobs in and removing jobs from queues would be accomplished by FILE and REMOVE statements.

Other features of SIMSCRIPT include statistics-gathering statements such as COMPUTE (mean, variance, etc.) and ACCUMULATE, and a report generator facility. Most FORTRAN statements (DO, READ, WRITE, IF, etc.) have similar but not identical equivalents in SIMSCRIPT.

GPSS (General Purpose Systems Simulator) is a transaction-oriented interpretive simulation language developed at IBM. A first version appeared in 1961; a second version, GPSS II [41], appeared in 1962 and has since received widespread use. The current version, GPSS III [42], is an extension of GPSS II and provides some new facilities and greater ease of use. With but one exception, GPSS II for the UNIVAC 1107/8, GPSS is generally available only on IBM systems. (However, a number of versions and variants for other systems have been developed by individual installations.)

In GPSS, a simulation model is described in the form of a block diagram, using a number of special block types. GPSS deals with entities called *facilities, storages, queues*, and *transactions*. Transactions are characterized by their apparent motion through the system. Facilities are entities which can be occupied by only one transaction at a time, while storages are entities which may be occupied by more than one transaction at a time. A GPSS version of the BASYS simulator would treat the central processor and disk as facilities, central memory as a storage, and jobs as transactions.

The development of a GPSS simulation model is done from the point of view of the transaction. The operations performed for a transaction are specified by appropriate block types; the flow of transactions is specified by the manner in which blocks are connected. SEIZE and RELEASE (or PREEMPT and RETURN) blocks are used to reserve and release facilities; the interval for which the facility is reserved is specified by an ADVANCE block. ENTER and LEAVE blocks are used to reserve and release storage space, and QUEUE and DEPART blocks used to enter transactions in and remove transactions from queues. The arrival of a transaction is effected by a GENERATE block, parameters established for that transaction by one or more ASSIGN blocks, and the eventual removal of the transaction from the system accomplished by a TERMINATE block. Other block types, such as TEST, LOOP, and TRANSFER, provide means of controlling the flow of transactions.

Figure 9 shows part of a GPSS block

diagram for a simple BASYS simulator. Jobs "arrive" when generated by a GENERATE block; also, each time a job is generated, the arrival of the next job is scheduled by this block. A number of ASSIGN blocks are used to establish the various job characteristics, such as CM space required, number of I/O requests, etc. The job then enters the CM queue (queue 1) and requests CM space via an ENTER block. If sufficient space is not available, the job remains in the queue. If space is available, it is reserved and the job leaves the CM queue via a DEPART block. The TEST block tests the I/O request count for this job and causes the job to terminate if it is zero. If the count is nonzero, the job enters the CPU queue (queue 2). If the CPU is free, it is reserved for the job at the SEIZE block and the job removed from the queue via a DEPART block. The length of time for which the CPU will be reserved for this job is specified by the ADVANCE block; upon occurrence of an I/O request, the CPU would be released via a RELEASE block (not shown). A similar block sequence (QUEUE → SEIZE → DEPART → ADVANCE → RELEASE) might be used to represent the processing of a disk request. When the job's execution is completed, its storage space is released via a LEAVE block and the job removed from the system by the TERMINATE block.

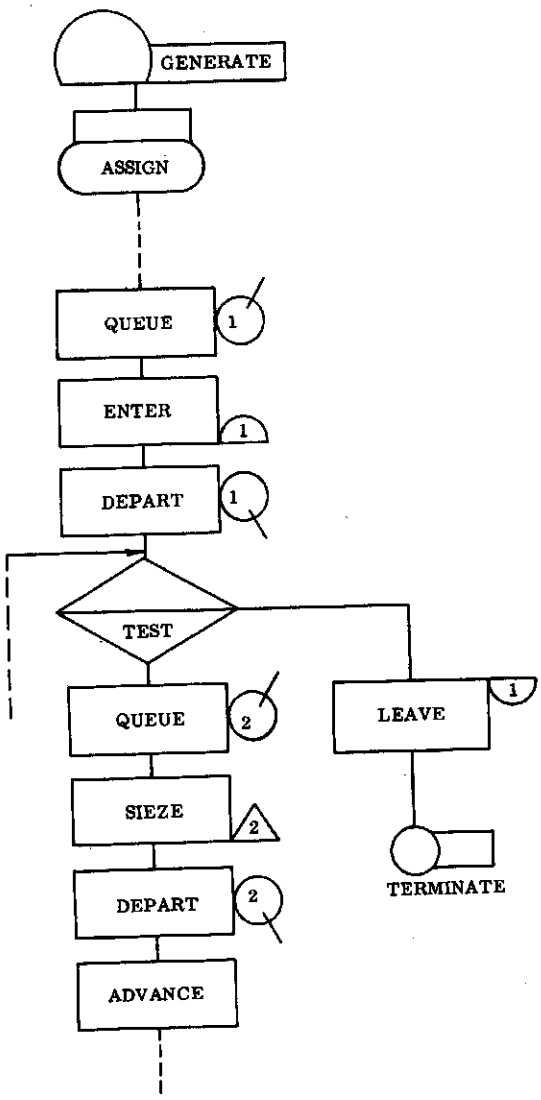

FIG. 9. GPSS simulation model example

GPSS provides comprehensive built-in statistics-gathering facilities. The output of a GPSS simulation includes statistics on the utilization of all facilities, storages, and queues in the system. Queue length distributions may be obtained by use of a QTABLE card (which establishes a table for data collection and defines the number of frequency classes desired), and statistics on any other variable of interest easily gathered by use of a TABLE card and the TABULATE block.

SUMMARY

The development of the simulation model is only a part of the analysis process. Subsequent steps in this process include testing and validation of the model, design of experiments, and the analysis and interpretation of results.

Since the simulation model developed in the preceding pages was assumed to be a model of an existing system, it can be validated by comparison of simulation results to actual system performance. When no such comparison is possible, other methods of validation must be employed. These methods might include the substitution of fixed values for job characteristics, rather than random samples, to permit comparison of simulation results with cal-

culated values, or the manual verification of model behavior through the examination of a step-by-step trace produced by the simulation program.

Our objective was to determine the effect on central processor utilization of increasing the size of central memory or substituting a faster disk. Thus, the experimentation process requires three versions of the simulation model: one representing the original system, one the system with added central memory, and one the system with the faster disk. One or more simulation runs are made for each of these. Each run provides an estimate of central processor utilization (as measured, say, by recording the amount of time during which the central processor was assigned to jobs and, at the end of the run, dividing that amount by the elapsed time). It is also desirable to record queue statistics, disk and central memory utilization, the time spent by jobs in the system, and so forth; such data may be useful in identifying system bottlenecks. The estimates produced by the set of runs for a given version are combined to provide a single performance estimate for that version, and results for the three versions are compared. Methods for comparison of simulation experiment results are discussed and reviewed by Naylor et al. [65].

The determination of the number and length of runs required to obtain a given accuracy presents complex statistical problems. These and related problems have been surveyed by Fishman and Kiviat [62] and Burdick and Naylor [60]; these surveys provide an excellent starting point for familiarization with the nature of the problems and methods of solution.

BIBLIOGRAPHY

Part I. Texts and Bibliographies

1. Evans, G., Wallace, G. F., and Sutherland, G. L. *Simulation Using Digital Computers.* Prentice-Hall, Englewood Cliffs, N. J., 1967.
2. Gordon, G. *System Simulation.* Prentice-Hall, Englewood Cliffs, N. J., 1969.
   The simulation of both discrete and continuous systems is discussed. The first two chapters deal with the nature and concepts of systems modeling and simulation, and are followed by three chapters on continuous systems simulation. A chapter introducing discrete system simulation describes a telephone system model; subsequent chapters introduce simulation with Fortran, GPSS (the author is the originator of GPSS), and Simscript, and show the application of these languages to this model. Other chapters discuss the generation of random variables and the verification of simulation results.
3. Martin, F. F. *Computer Modeling and Simulation.* Wiley, New York, 1968.
4. Mize, J. H., and Cox, J. G. *Essentials of Simulation.* Prentice-Hall, Englewood Cliffs, N. J., 1968.
5. Naylor, T. H., Balinty, J. L., Burdick, D. S., and Chu, K. *Computer Simulation Techniques.* Wiley, New York, 1966.
   Recommended as one of the best available texts on simulation. The two introductory chapters are followed by chapters on random number generation and generation of samples from continuous and discrete probability distributions. The next two chapters present queueing, inventory, economic, and industrial models. Chapter 7 discusses discrete simulation languages and incorporates well-written articles from the *IBM Systems Journal* on GPSS II and Simscript, as well as a discussion of Gasp, Dynamo, and others. The next chapter touches on the problem of verification, and the final chapter is a good survey of some of the problems of experimental design. Each chapter is accompanied by references.
6. ——. Bibliography 19. Simulation and gaming. *Comput. Rev. 10,* 1 (Jan. 1969), 61–69.
   This is a bibliography of 440 entries on simulation and gaming. It is not annotated but, perhaps even better, the *Computing Reviews* review number is cited.
7. Tocher, K. D. *The Art of Simulation.* Van Nostrand, Princeton, N. J., 1963.
   This book, the first published on simulation, has several chapters dealing with the generation of random variables. Three chapters present simulation models of queueing and industrial systems. These models are presented in flowchart form and fully discussed in the text.
8. Bibliography on simulation. Rep. No. 320-0924-0, IBM Corp., White Plains, N. Y., 1966.
   This is an extensive bibliography on simulation in many fields: economics, industry, management games, etc. It is indexed by title, author, and subject.

Part II. Simulation of Computer Systems

9. Cheng, P. S. Trace-driven system modeling. *IBM Syst. J. 8,* 4 (1969), 280–289.
   A trace-driven approach to computer system modeling is described. The workload of interest is first executed on a real system, and trace data obtained by monitoring significant events in the course of this execution. The resultant data is then used as input to the

desired simulation model, which may reflect changes in the configuration or operating system of the real system.

10. FINE, G. H., AND MCISAAC, P. V. Simulation of a time-sharing system. *Manag. Sci. 12,* 6 (Feb. 1966), 180–194.

    This paper discusses (in little detail) the simulation of the SDC time-sharing system. Simulation results for various scheduling algorithms are presented. The most interesting part of the paper is the description of the use of the direct search technique in simulation optimization.

11. FOX, D., AND KESSLER, J. L. Experiments in software modeling. Proc. AFIPS 1967 Fall Joint Comput. Conf., Vol. 31, pp. 429–436 (Thompson Book Co., Washington, D. C.).

    Requirements and design criteria for modeling a total hardware/software system configuration are discussed, past work is briefly reviewed, and a mechanism for simulating such systems is described. This mechanism permits designers of system software such as compilers to simulate the functional execution of programs at the design stage.

12. GLINKA, L. R., BRUSH, R. M., AND UNGAR, A. J. Design, thru simulation, of a multiple-access information system. Proc. AFIPS 1967 Fall Joint Comput. Conf., Vol. 31, pp. 437–447 (Thompson Book Co., Washington, D. C.).

    The modeling and simulation of a simple information retrieval system are discussed. Performance measures are defined, and data base, job types, and program characteristics described. A number of simulation results are presented. No description of the simulator is given.

13. HERTEL, H. F., AND STANLEY, W. I. Statistics gathering and simulation for the Apollo real-time operating system. *IBM Syst. J. 7,* 2 (1968), 85–102.

    The Real Time Computer Complex (RTCC) is presented, followed by a description of the measures of interest and the techniques employed for gathering statistics. Mean execution times for a number of OS/360 system functions are tabulated, and a brief description of the modeling of the RTCC is given.

14. HUESMAN, L. R., AND GOLDBERG, R. P. Evaluating computer systems through simulation. *Comput. J. 10,* 2 (Aug. 1967), 150–156.

    This is a survey article on work done in the simulation of multiprogramming, multiprocessing, and time-sharing systems; it has 46 references.

15. HUTCHINSON, G. K. A computer center simulation project. *Comm. ACM 8,* 9 (Sept. 1965), 559–568.

    This paper describes a macroscopic, job-shop-like simulation of a computer center. The center's operations are described and performance measures are defined. Simulation experiments in job scheduling are described, and the results of these experiments are presented. No description of the simulator is given.

16. —— AND MAGUIRE, J. N. Computer systems design and analysis through simulation. Proc. AFIPS 1965 Fall Joint Comput. Conf., Vol. 27, pp. 161–167 (Spartan Books, Washington, D. C.).

    The simulation of a UNIVAC 1107 system is described in this paper. The purpose of this simulation was to determine the effectiveness of various peripheral devices and buffering techniques. The paper discusses model validation and system measurements, and describes how the results of the simulation were used to plan for a new system.

17. ——. Some problems in the simulation of multiprocessor computer systems. In Buxton, J. N., ed., *Simulation Programming Languages,* North-Holland, Amsterdam, 1968, pp. 305–318.

    Similarities and differences between job shop and multiprocessing systems are discussed in the paper's introduction. Next, problems of event time differentiation and simultaneous event occurrence are described. In the last section of the paper, a description of the Program Description Language (PDL) is given. PDL was designed to permit the programs of a multiprocessor system to be described in a manner suitable for simulation.

18. KATZ, J. H. Simulation of a multiprocessor computer system. Proc. AFIPS 1966 Spring Joint Comput. Conf., Vol. 28, pp. 127–139 (Spartan Books, Washington, D. C.).

    This paper describes the simulation of the IBM 7040–7090 direct-coupled system. The parameters produced by the job generator, the system parameters, and the simulator output are described. A tabulation of the various states of the system is presented and discussed. The simulator was written in SIMSCRIPT, and the paper contains a brief review of SIMSCRIPT concepts as they relate to the simulator structure.

19. ——. An experimental model of system/360. *Comm. ACM 10,* 11 (Nov. 1967), 694–702.

    The simulation of System/360 at a macroscopic level is described in this paper. After a brief review of system concepts, job generation, simulation parameters, and simulator output are described. The structure of the simulator is described briefly using SIMSCRIPT terminology.

20. LEHMAN, M. M., AND ROSENFELD, J. L. Performance of a simulated multiprogramming system. Proc. AFIPS 1968 Fall Joint Comput. Conf., Vol. 33, pp. 1431–1442 (Thompson Book Co., Washington, D. C.).

    Simulation results from experiments with a simulation model of the multiprogramming variant (MVT) of OS/360 (on a 360/65 system) are presented. No description of the simulator is given.

21. MACDOUGALL, M. H. Simulation of an ECS-based operating system. Proc. AFIPS 1967 Spring Joint Comput. Conf., Vol. 30, pp. 735–741 (Thompson Book Co., Washington, D. C.).

    This paper describes the simulation of a CDC 6000 system with Extended Core Storage. The simulator is described in some detail and some simulation results are presented.

22. MERIKALLIO, R. A., AND HOLLAND, F. C. Simulation design of a multiprocessing system. Proc. AFIPS 1968 Fall Joint Comput. Conf., Vol. 33, pp. 1399–1410 (Thompson Book Co., Washington, D. C.).

The simulation of a multiprocessing system designed for air traffic control is discussed in this paper. The system and its requirements are described, and the advantages of and problems presented by multiprocessing are discussed. A brief description of the simulation models is given and some results of the simulation are presented.

23. NIELSON, N. R. The simulation of time sharing systems. *Comm. ACM 10,* 7 (July 1967), 397–412.

The simulation of the IBM 360/67 time-sharing system is described in this paper. Considerations in the development of the model and construction of the simulator are discussed at length, and the results of a number of simulation experiments are described.

24. ——. An approach to the simulation of a time-sharing system. Proc. AFIPS 1967 Fall Joint Comput. Conf., Vol. 31, pp. 419–428 (Thompson Book Co., Washington, D. C.).

A detailed description of techniques for simulating memory paging processes and task execution sequences is presented in this paper.

25. ——. Computer simulation of computer system performance. Proc. 22nd Nat. Conf. ACM, 1967, ACM Pub. P-67, pp. 581–590 (Thompson Book Co., Washington, D. C.).

This paper reviews the history of computer systems performance prediction and describes some of the problems encountered in the simulation of multiprogramming, multiprocessing, and time-sharing systems. Measures of system performance are discussed.

26. ——. ECSS: An Extendable Computer System Simulator. Proc. Third Conf. on Applications of Simulation, Los Angeles, 1969, pp. 114–129 (ACM/AIIE/IEEE/SHARE/SCi/TIMS).

ECSS provides a higher level computer system simulation language which is an extension of, and written in, SIMSCRIPT II. ECSS statements provide for the definition of system components, the characteristics of jobs to be simulated, etc. This paper describes and illustrates some of the features of ECSS, and discusses its objectives and design considerations.

27. REHMAN, S. L., AND GANGWERE, S. G. A simulation study of resource management in a time-sharing system. Proc. AFIPS 1968 Fall Joint Comput. Conf., Vol. 33, pp. 1411–1430 (Thompson Book Co., Washington, D. C.).

A simple simulation model of a multiprogramming system is described, and the results of experiments with various time slice and job characteristic values are presented.

28. SCHERR, A. L. An analysis of time-shared computer systems. Res. Monograph No. 36, MIT Press, Cambridge, Mass., 1967.

This is a study of the MIT Compatible Time Sharing System (CTSS). Statistics on CTSS job behavior are presented and the simulation model described in detail. The simulator employed the CTSS scheduling algorithm, and MAD listings for this algorithm as well as the simulator itself are given in appendices. Markov models of the system are developed, and the system performance as predicted by the simulation model and Markovian models is compared with actual system performance.

29. SEAMAN, P. H., AND SOUCY, R. C. Simulating operating systems. *IBM Syst. J. 8,* 4 (1969), 264–279.

The Computer System Simulator (CSS), which provides a language and a structure specifically designed for modeling computer systems, is described in this paper. The CSS language is a higher level language in which the subject system can be described in terms of its configuration, operating system programs, and job environment.

30. ——. On teleprocessing system design. Part VI. The role of digital simulation. *IBM Syst. J. 5,* 3 (1966), 175–189.

Considerations in undertaking a computer system simulation study are discussed in this paper. Some of the modeling techniques required for such a study are presented and illustrated via GPSS.

31. YOUCHAH, M. I., RUDIE, D. D., AND JOHNSON, E. J. The Data Processing System Simulator (DPSS). Proc. AFIPS 1964 Fall Joint Comput. Conf., Vol. 26, Pt. 1, pp. 251–276 (Spartan Books, Baltimore, Md.).

This paper describes a general-purpose computer system simulator which can be used for evaluating a variety of system configurations and job processing rules by specifying system behavior in a higher order macro language.

Part III. Simulation Languages

32. ARMSTRONG, J., ULFERS, H., MILLER, D. J., AND PAGE, H. C. SOLPASS, A Simulation Oriented Language Programming and Simulation System. Proc. Third Conf. on Applications of Simulation, Los Angeles, 1969, pp. 24–37 (ACM/AIIE/IEEE/SHARE/SCi/TIMS).

SOLPASS is an ALGOL-based simulation and programming language for the Burroughs B5500 computer. It is based on SOL, but provides extensions which facilitate simulation of large communication networks.

33. BLUNDEN, G. P., AND KRASNOW, H. S. The process concept as a basis for simulation modeling. *Simulation 9,* 2 (Aug. 1967) 89–93.

Several recent simulation languages (e.g. SIMULA) are based on the "process" concept, as opposed to the "event" concept (SIMSCRIPT) or the activity concept (CSL). This paper discusses the dynamic and static characteristics of systems, and suggests that the process concept provides a unified way of treating these characteristics.

34. BUXTON, J. N., AND LASKI, J. G. Control and Simulation Language. *Comput. J. 5,* 3 (Oct. 1962), 194–200.

This paper describes the activity-based Control and Simulation Language (CSL) and describes the compiling technique employed (FORTRAN is used as an intermediate language). CSL provides facilities for simulation somewhat similar to those in SIMSCRIPT.

35. DAHL, O.-J., AND NYGAARD, K. SIMULA—An ALGOL-based simulation language. *Comm. ACM 9*, 9 (Sept. 1966), 671–678.

SIMULA, an extension of ALGOL 60, is a process-based simulation language developed for the UNIVAC 1107/1108. This paper gives a formal description of the extensions to ALGOL and illustrates the features of the language with several examples. An extended version of the language described in this paper (SIMULA 67) has been developed for the 1107/1108 systems and the CDC 3300 and 64/6600 systems.

36. DIMSDALE, B., AND MARKOWITZ, H. M. A description of the SIMSCRIPT language. *IBM Syst. J. 3*, 1 (1964), 57–67.

This paper describes the basic features of SIMSCRIPT by demonstrating their use in the simulation of a supermarket. It also is incorporated into [5, Ch. 7].

37. EFRON, R., AND GORDON, G. A general purpose digital simulator and examples of its application. Part I—Description of the simulator. *IBM Syst. J. 3*, 1 (1964), 22–34.

This paper gives a very good introduction to the concepts and facilities of GPSS II, using a simple real-time data processing system as an example. Since GPSS III retains most of the GPSS II facilities, this paper is recommended reading for anyone interested in either version of GPSS. This paper also appears as part of [5, Ch. 7], where it is accompanied by another example provided by one of the authors.

38. FREEMAN, D. E. Discrete systems simulation. *Simulation 7*, 3 (Sept. 1966), 142–148.

A brief introduction to the nature of discrete system simulation is given, followed by an example of the application of GPSS to the simulation of a gas station.

39. GREENBERGER, M., JONES, M. M., MORRIS, J. R., AND NESS, D. N. *On-Line Computation and Simulation: The OPS-3 System.* MIT Press, Cambridge, Mass., 1965.

The OPS-3 system is a general-purpose time-sharing language developed for use with the MIT time-sharing system. This manual describes the facilities and features of the language. OPS-3 contains a number of special operators (e.g. the AGENDA operator for scheduling activities) for simulation.

40. GEISLER, M. A., AND MARKOWITZ, H. M. A brief review of SIMSCRIPT as a simulating technique. RM-3778-PR, RAND Corp., Santa Monica, Calif., 1963 (AD 411-324).

This paper provides a readily assimilated review of the basic concepts and facilities of SIMSCRIPT.

41. GENERAL PURPOSE SYSTEMS SIMULATOR II. Form B20-6346, IBM Corp., White Plains, N. Y., 1963.

42. GENERAL PURPOSE SYSTEMS SIMULATOR III. Form B20-0001, IBM Corp., White Plains, N. Y., 1965.

This manual (as well as the GPSS II manual cited in the previous reference) is a readable description of the features and facilities of GPSS. More detailed information is available in the user's manuals.

43. HERSCOVITCH, H., AND SCHNEIDER, T. H. GPSS III—An expanded general purpose simulator. *IBM Syst. J. 4*, 3 (1965), 174–183.

GPSS III is an extension of GPSS II. This paper reviews these extensions. A knowledge of GPSS II is assumed.

44. JONES, M. M. On-line simulation. Proc. 22nd Nat. Conf. ACM, ACM Pub. P-67, pp. 591–599. (Thompson Book Co., Washington, D. C.).

Following a brief review of OPS-3 and a discussion of facilities provided by Multics and PL/I, some problems in modeling are discussed. The features of OPS-4, a PL/I-like language designed for simulation in a time-sharing environment, are described.

45. KIVIAT, P. J. GASP—A General Activity Simulation Program. Project No. 90.17-019(2), Appl. Res. Lab., U. S. Steel, Monroeville, Pa., 1963.

This report describes GASP, a FORTRAN-imbedded, event-oriented, simulation language. Facilities for performing such simulation functions as scheduling of events, manipulating queues, etc., are provided in the form of FORTRAN subroutines.

46. ——. Simulation language report generators. P-3349, RAND Corp., Santa Monica, Calif., April 1966 (AD 631-940).

In this interesting paper, the author proposes simulation language facilities for debugging, report generation, collection of statistics, and dynamic display of system behavior.

47. ——. Development of new digital simulation languages. *J. Ind. Eng. 17* (Nov. 1966), 604–608. (Also P-3348, RAND Corp., Santa Monica, Calif., April 1966.)

Event, activity, and process concepts are discussed, followed by a commentary on current simulation languages. The bulk of the paper is devoted to a discussion of SIMSCRIPT II. There are 20 references.

48. ——. Development of discrete digital simulation languages. *Simulation 8*, 2 (Feb. 1967), 65–70.

This paper gives a brief review of the development of simulation languages, and discusses the conceptual differences and principal features. There are 36 references.

49. ——. Digital computer simulation: Computer programming languages. RM-5883-PR, RAND Corp., Santa Monica, Calif., Jan. 1969.

The conceptual approaches in the development of various simulation languages are reviewed, and four languages (GPSS, SIMSCRIPT, SIMULA, and CSL), are compared and discussed. Sample programs in each language are provided. Current work and future trends in simulation languages are described. This is

a well-written paper and recommended reading.

50. ——, VILLANUEVA, R., AND MARKOWITZ, H. M. The *SIMSCRIPT II Programming Language*. Prentice-Hall, Englewood Cliffs, N. J., 1969.

The features and facilities of SIMSCRIPT II are presented in this reference manual. While SIMSCRIPT II commands are, in general, similar to those of SIMSCRIPT, many have expanded facilities. The language is now free-form, and the data definition form has been done away with; data definition is now accomplished by statements.

51. KNUTH, D. E., AND MCNELEY, J. L. SOL—A symbolic language for general-purpose systems simulation. *IEEE Trans. EC-13* (Aug. 1964), 401–408.

SOL, an ALGOL-like simulation language developed for the Burroughs B-5000, is described in this paper by applying it to the simulation of a multiconsole computer system. The complete simulation program is given, followed by a clear explanation of the purpose and function of each statement.

52. —— AND ——. A formal definition of SOL. *IEEE Trans. EC-13* (Aug. 1964), 409–414.

This paper briefly introduces the basic concepts of SOL and gives a formal, metalinguistic description of the language. While SOL has not come into widespread use, it has provided many ideas and much stimulus for current simulation language development.

53. MARKOWITZ, H. M., HAUSNER, B., AND KARR, H. W. *SIMSCRIPT—A Simulation Language*. Prentice-Hall, Inc., Englewood Cliffs, N. J., 1963.

This is a reference manual describing the SIMSCRIPT language.

54. MCNELEY, J. L. Simulation languages. *Simulation 9, 2* (Aug. 1967), 95–98.

This paper introduces some of the features of SIMULA via a store checkout counter simulation example.

55. PARENTE, R. J., AND KRASNOW, H. S. A language for modeling and simulating dynamic systems. *Comm. ACM 10,* 9 (Sept. 1967), 559–566.

This paper describes a PL/I-like, process-oriented, simulation language. Following a discussion of the general approach taken in developing the language, a scheduling problem is used to illustrate its features. A subsequent section discusses highlights of the language in some detail, and a semiformal description of part of the language is given in an appendix.

56. TEICHROEW, D., AND LUBIN, J. F. Computer simulation—discussion of the technique and comparison of languages. *Comm. ACM 9,* 10 (Oct. 1966), 723–741. (Also, *Simulation 9,* 4 (Oct. 1967), 181–190, with additions by T. D. Truitt.)

This excellent survey paper gives a detailed comparison of six simulation languages: SIMSCRIPT, CLP, CSL, GASP, GPSS, and SOL. The features of each language are compared in a series of tables, and the comparison expanded upon in the text. The systems for which these languages have been implemented are given. In addition to the six languages compared in detail, a number of other simulation languages, both discrete and continuous, are also listed. There are 89 references. (*Note:* This paper is available as an ACM reprint.)

57. ULRICH, E. G. Serial/parallel event scheduling for the simulation of large systems. Proc. 23rd Nat. Conf. ACM, 1968, ACM Pub. P-68, pp. 279–287 (Brandon/Systems Press, Inc., Princeton, N. J., 1968).

Event-oriented simulations generally employ a list of events, ordered on an event time basis, as a mechanism for event scheduling. If the number of scheduled events is very large, other methods of scheduling become more efficient. This paper describes a scheduling technique called "time mapping" and compares it with the next event technique.

58. WEAMER, D. G. QUICKSIM—A block structured simulation language written in SIMSCRIPT. Proc. Third Conf. on Applications of Simulation, Los Angeles, 1969, pp. 1–11 (ACM/AIIE/IEEE/SHARE/SCi/TIMS).

QUICKSIM, a simulation language written for the NCR 315, provides a block structure closely resembling that of GPSS, but with facilities for a user to easily insert FORTRAN- or SIMSCRIPT-coded blocks of his own.

59. WEINERT, A. E. A SIMSCRIPT-FORTRAN case study. *Comm. ACM 10,* 12 (Dec. 1967), 784–792.

A vehicle dispatching simulation was programmed in both SIMSCRIPT and FORTRAN. This paper describes the simulation model, compares the programs (the FORTRAN program executed faster and required less memory, but the SIMSCRIPT program was easier to modify), and discusses some of the implications of the comparison.

Part IV. Statistical Aspects of Simulation

60. BURDICK, D. S., AND NAYLOR, T. H. Design of computer simulation experiments for industrial systems. *Comm. ACM 9,* 5 (May 1966), 329–339.

This survey article provides information on the literature on experimental design techniques and relates this material to digital computer simulation experiments. Major emphasis is on analysis of variance, but multiple ranking, sequential sampling, and spectral analysis techniques are also considered. There are 78 references.

61. FISHMAN, G. S., AND KIVIAT, P. J. Spectral analysis of time series generated by simulation models. RM-4393-PR, RAND Corp., Santa Monica, Calif., Feb. 1965 (AD 612-281).

The application of spectral analysis techniques to the statistical analysis of the output of simulation experiments is described in this interesting report. The theory underlying

these techniques is reviewed and illustrated by application to a single-server queueing problem. The approach provides a method for determining the experiment run time required to generate the equivalent of an independent observation.

62. —— AND ——. The statistics of discrete-event simulation. *Simulation 10,* 4 (April 1966), 185–195.

A brief resume of the statistical aspects of simulation experiments is presented in this paper. Problems in random variable generation, model structure verification and validation, run length determination, and experimental design are considered. There are 26 references.

63. ——. Problems in the statistical analysis of simulation experiments: The comparison of means and the length of sample records. *Comm. ACM 10,* 2 (Feb. 1967), 94–99.

This paper summarizes and extends the work reported in [61].

64. ——. The allocation of computer time in comparing simulation experiments. *Oper. Res. 16,* 2 (March–April 1968), 280–295.

This paper describes a rule for choosing sample sizes—and therefore computer running time—so as to obtain a specified reliability in the estimate of the sample mean. When the objective is a comparison of means resulting from two experiments, it is shown that, for a given accuracy, selecting sample sizes according to the rule rather than employing equal sample sizes may reduce significantly the computer time required.

65. NAYLOR, T. H., WERTZ, K., AND WONNACOTT, T. H. Methods for analyzing data from computer simulation experiments. *Comm. ACM 10,* 11 (Nov. 1967), 703–710.

An economic system model is used to demonstrate three methods of analyzing data generated by computer simulation experiments. The three methods are the *F*-test, multiple comparison, and multiple ranking. The assumptions under which these techniques can be employed and considerations in selecting a technique are discussed. There are 50 references.