Proceedings of the 1983
Winter Simulation Conference
S. Roberts, J. Banks, B. Schmeiser (eds.)



# AN OPEN QUEUEING NETWORK VIEW OF COMPUTER SYSTEM PERFORMANCE

Duane R. Ball
FEDSIM

ACMS is a modeling system which translates user-supplied hardware and software descriptions into an activity network and then uses open network queueing theory approximations to estimate the time required for transactions to pass through the network. It is particularly suited for use in estimating the capacity of a computer system, identifying system bottlenecks, evaluating the performance of proposed hardware configuration and software designs, identifying the least-cost set of resources required to achieve a desired performance level, and evaluating the performance impact of both new applications and growth in existing ones.

Features that support the automated sizing of computer systems make ACMS unique among analytical modeling tools. ACMS provides the user with statements to specify device costs and constraints on the number of devices of each type that may be configured. Based on these costs, device performance characteristics, and process completion time goals; ACMS will select, from a menu of possible choices, a set of hardware which minimizes system cost and meets the performance goals.

FEDSIM is a federal center for computer performance analysis and simulation. The primary modeling tool used at FEDSIM has been a discrete event simulation language called ECSS. ECSS was well suited to FEDSIM's needs. A large percentage of the work done at FEDSIM involved the evaluation of special purpose operating systems for existing systems. Because ECSS permits a highly detailed representation of operating system algorithms, it is well suited to evaluations of this type.

As time passed, however, FEDSIM's business shifted from the analysis of existing systems to the analysis of proposed systems. The wealth of measurement data available for existing systems is not available for proposed systems. In addition, the number of design alternatives which must be considered has increased dramatically. Discrete event simulation has proven to be too slow and too costly in this environment. Analysis of actual project cost data indicated that much more time and money was being spent in model development than in model experiments and analysis.

money was being spent in model development than in model experiments and analysis.

To respond to these problems, FEDSIM decided to acquire or develop a new tool more suited to the changes in the business environment. It was decided that the tool must have the following characteristics:

1. permit rapid model development

2. have a low cost per experiment

3. be suited to performance vs. cost trade-off analysis

4. permit experiments to be performed in interactive mode.

One characteristic that ECSS has that was not a requirement for the new tool is the potential for extreme accuracy! Because the systems to be studied are in the design phase, developing extremely accurate models would not be justified by the amount of data available to parameterize the models. The tool would be required to determine which of a number of proposed alternatives is better but not, necessarily, by precisely how much.

The requirements for speed led us to consider a tool based on analytical methods rather than simulation. Reviewing the existing analytical computer modeling tools, we found that they all had one or more of the following shortcomings:

1. the amount of structural data (i.e., number of customer classes, priority groups, etc.) which could be represented was limited,

2. the model definition languages were oriented toward specification of queuing system problems rather than computer performance problems (e.g., requiring specification of $V_i S_i$ products),

3. the user interfaces were very poorly designed.

4. the model CPU time was too high to conduct performance vs. cost trade-off analyses.

We decided at this time that we could either wait for someone to develop a tool which suited our needs, or develop a tool ourselves. Because of the immediacy of our needs, we decided to develop an analytical based modeling tool. The tool which was developed is called A Computer Modeling System (ACMS).

ECSS has a very natural syntax for describing computing hardware and applications routines. At the time ACMS was being designed, approximately fifteen analysts at FEDSIM could describe computer systems using the ECSS syntax. Because of the ECSS experience base, adopting the ECSS syntax as the model specification language for ACMS was a logical choice. In addition, having a common "worldview" for both tools allowed analysts to do preliminary analysis using ACMS and then transfer to ECSS as the specifications for the modeled system developed and both the requirements for accuracy and availability of data increased.

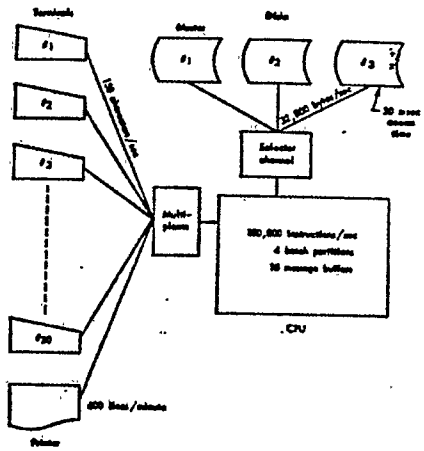
The ECSS/ACMS worldview separates the specification of computer systems into three major components. The first component is the set of resources. Resources are represented as a collection of "things" from which service may be requested. They include both physical resources, such as CPUs, and logical resources, such as disk files. The second component represents the application processes which are ordered sequences of requests for service from the resources. Application processes can represent anything from the execution of routines on a mainframe to the physical manipulation of data outside the bounds of the computer system. The final component is the operating system which is represented in ACMS by specifying each resource's dispatching discipline and overheads for servicing requests. For example, requests for service can be processed first-come, first-served, shortest job first, etc.

Consider the hardware schematic in Figure I(a). It consists of a central computer, two channels, three random-access disks, twenty terminals, and a printer. The ACMS statements which can be used to model the hardware are shown in Figure I(b).

The application process which executes on this hardware represents on-line text-editing by up to twenty users simultaneously. Each terminal user issues a series of requests to the system, which require system resources, and require rapid, interactive responses.

Let us now consider in more detail the behavior of an individual user of the text-editor system. Suppose the user is interested in changing something that was previously written on disk as a text file. First, the user walks into the computer room and sits down at an available CRT terminal. By pressing some buttons and typing in the account number, the user logs into the system. The user then identifies the file to be edited which causes the system to bring part of it into working storage. The user then waits for an acknowledgement that indicates the file is ready to be edited. When the system responds, the user proceeds to issue a series of editing commands, some that modify the file and some that simply display certain sections at the terminal. After issuing each command, the user waits for the system to acknowledge that the operation is complete and then pauses a short time to think about the next command. Finally, the user issues a command to produce a hard-copy listing of the revised file on the printer. For our text-editor, this listing request requires a job to be submitted to the batch job stream. The user then logs off and waits for the listing. Figure II(a) depicts this behavior graphically.

At one level, the above is a description of the behavior of one particular user. However, assuming this is a "typical" user, the same description may apply to several users who all do about the same thing. Though certain parameters



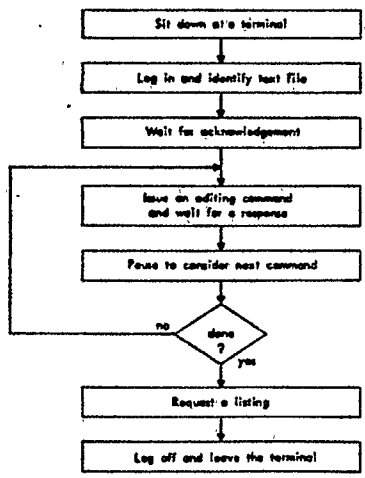

SYSTEM CONFIGURATION FOR AN ON-LINE TEXT-EDITOR

FIGURE I(a)

```
000310 SYSTEM DESCRIPTION
000320
000330 SPECIFY 1 CPU WHICH
000340         EXECUTES 100000 INSTRUCTIONS PER SECOND
000350
000360 SPECIFY 20 A.TERMINALS
000370
000380 SPECIFY MEMORY WHICH STORES 4 PARTITION
000390
000400 SPECIFY ACCTG.FILE
000410
000420 SPECIFY 2 T.DISK WHICH
000430           TRANSMIT 32000 BYTES PER SECOND
000440           AND ABSORB 20 MILLISECONDS PER ACCESS
000450
000460 SPECIFY  BUFFER.POOL WHICH
000470          STORES 10 MESSAGES
000480
000490 SPECIFY 20 T.TERMINALS WHICH
000500           TRANSMIT 150 CHARACTERS PER SECOND
000510
000520 SPECIFY T.PRINTER WHICH
000530         TRANSMITS 10 LINES PER SECOND
000540
000550 SPECIFY CH1 WHICH TRANSMITS 100000 BYTES PER SECOND
000560
000570 SPECIFY CH2 WHICH MULTIPLEXES 32000 BYTES PER SECOND
000580
000590 SPECIFY PATH DISK.PATH WHICH
000600         CONNECTS SPECIFIC T.DISK TO CH1
000610
000620 SPECIFY PATH PR.PATH WHICH
000630         CONNECTS T.PRINTER TO CH2
000640
000650 SPECIFY PATH T.PATH WHICH
000660         CONNECTS SPECIFIC T.TERMINALS TO CH2
000670
000680 END
```

ACMS L1 SYSTEM DESCRIPTION

FIGURE I(b)




TEXT-EDITOR SYSTEM USER BEHAVIOR

FIGURE II(a)

```
000690 WORKLOAD DESCRIPTION
000700 PROCESS USER
000710
000720 ALLOCATE A.TERMINALS
000730
000740 RUN LOGON
000750 DO 4 TIMES
000760       SEND 14 CHARACTERS VIA T.PATH
000770       RUN INTERACTION
000780       WAIT FOR 10 SECONDS
000790 END.DO
000800 RUN LISTOFF BUT CONTINUE
000810 RUN LOGOFF
000820 DEALLOCATE A.TERMINALS
000830
000840 END
000850 STEP LOGON
000860
000870 ALLOCATE ACCTG.FILE
000880 RECEIVE 800 BYTES VIA DISK.PATH
000890 DEALLOCATE ACCTG.FILE
000900
000910 EXECUTE 3500 INSTRUCTIONS ON CPU
000920
000930 SEND 7200 BYTES VIA DISK.PATH
000940
000950 END
```

ACMS L1 WORKLOAD DESCRIPTION

FIGURE II(b)

may differ from user to user, e.g., the length of the file being edited, the disk on which it is stored, or the amount of editing done, the behavior of each user is similar in terms of the nature and sequence of activities during the session.

To represent this kind of behavior in an ACMS model, we use the process construct. Remember a process is a sequence of events that occur over time and have some logical relation to each other. For example, the actions shown in Figure II(a) make up such a sequence: each box describes an event, time may pass between events, and the events are logically related since they describe one user's behavior during a text-editing session. The ACMS statements which may be used to model this process are shown in Figure II(b).

Using an ACMS model, we might like to determine the average time it takes to respond to a user request, the effect of different numbers of terminals on response time, the utilization of each device, the effect of queuing delays, or answer other questions about how the system might perform under various conditions.

Once the static structure of the computer hardware and applications software has been defined, the dynamic workload for the system must be specified. This workload is stated in terms of the mean time between arrivals (MTBA) of events which trigger the execution the of application processes. If no device is 100% busy in the steady state, the rate of initiation of application processes will equal the rate of completion. The concept of "initiation rate = completion rate" is called forced flow. Forced flow and the use of open queuing network analysis techniques permit ACMS to analyze moderately complex situations fairly rapidly.

ACMS combines decomposition and heuristic aggregation in its analysis technique. Active devices (those devices whose service time is a function of request size -- e.g., CPUs) are considered first. Assuming that (1) the events which trigger applications processes are generated by Poisson processes, (2) the distribution of service requests within applications processes are exponentially distributed, and (3) the dispatching discipline for each device is work conserving (never allows the device to go idle when unserviced requests are waiting and does not change the size of the service requests) then the network of active devices may be decomposed and each device analyzed in isolation. Passive devices (those devices which are acquired and held while service is received from other devices) are then evaluated.

Analyzing passive devices is much more difficult than analyzing active devices. While a network composed entirely of active devices can be decomposed relatively easily, networks containing passive resources are highly interdependent. Still, certain observations can be made concerning the behavior of passive devices. The first observation is that a passive device which "surrounds" a single active device will not increase the time to service a request provided the utilization of the passive device is less than 100%. In this case, a passive device only adds a new state

to a request which is waiting for service from the active device (i.e., waiting in line and holding a passive server). The problem becomes more complex, however, when a passive device surrounds two or more active devices. Requests desiring service from an idle active device may be blocked because the passive device is full of requests waiting for, or being served by, another active device. As the number of simultaneous requests a passive device can service and the the number of active devices within it increase, calculating the time a request waits to receive service becomes virtually impossible. To estimate this time, ACMS considers a request time-in-queue to be the sum of two components. The first component arises when lines for one or more active devices extend beyond the bounds of the surrounding passive device. The time for transactions to enter the queues (or servers) within the passive is the sum of these "serialized" times. The second component is the time resulting from transaction blocking (i.e., an active device within the bounds of the passive is idle while a transaction is waiting because the surrounding passive is full). ACMS estimates these components of delay in calculating the total queuing delays for passive devices. This aggregation process is illustrated by Figure III.

Because ACMS uses approximations which cannot be defended formally, it is necessary to determine the degree to which its predictions are valid. Because of the close relationship between ACMS and ECSS, ECSS discrete event simulation models were chosen as "actual systems" to which ACMS predictions could be compared. Since ECSS output statistics under the assumptions of exponential inter-arrival and service times are random variables, the simulations were replicated so that confidence intervals around parameter estimates could be computed. Nine systems were simulated. The experiments were designed so that flaws in the methodology could be detected.

One of the test problems is illustrated in Figure IV(a). In this problem, the modeled system

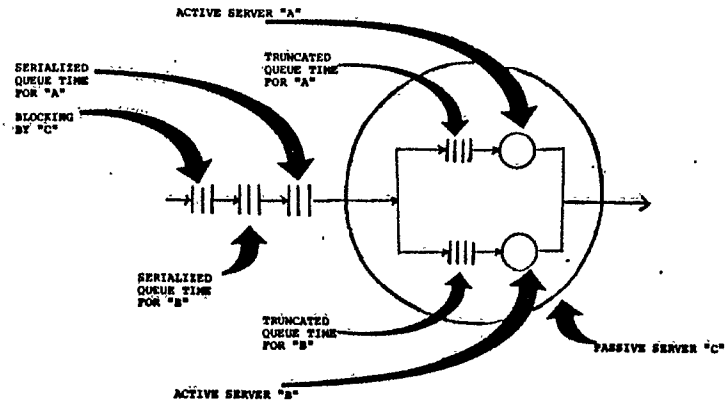
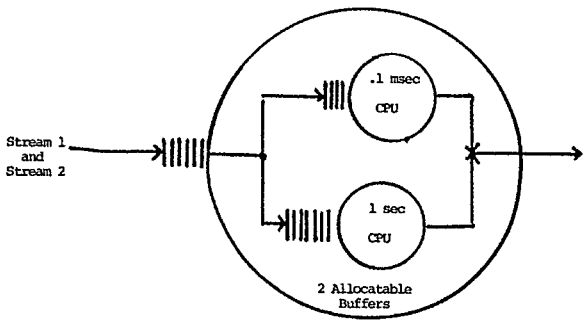



FIGURE III

consists of two job classes. Jobs in the first job class request a buffer from a shared buffer pool and then execute for 0.1 millisecond on a CPU. Jobs in the second job class also request a buffer, but then execute for 1 second on a second CPU. The shared buffer pool consists of two allocatable buffers. Figure IV(b) compares the simulation results to ACMS predictions. In the remaining experiments, ACMS predictions were found to be within 10% to 25% of the ECSS mean values.




SCHEMATIC FOR A VALIDATION EXPERIMENT

FIGURE IV(a)

JOB STREAM 1

| 1/MTBA | ECSS LB | ECSS UB | ACMS | % |
|---|---|---|---|---|
| .5 | .45 | 0.57 | 0.47 | -8 |
| .6 | .86 | 1.06 | 0.95 | -1 |
| .7 | 1.51 | 1.93 | 1.78 | +3 |
| .8 | 2.42 | 3.13 | 3.46 | +24 |
| .9 | 6.06 | 10.02 | 8.48 | +5 |

JOB STREAM 2

| 1/MTBA | ECSS LB | ECSS UB | ACMS | % |
|---|---|---|---|---|
| .5 | 1.92 | 2.07 | 2.00 | +1 |
| .6 | 2.42 | 2.70 | 2.50 | -2 |
| .7 | 3.17 | 3.69 | 3.33 | -2 |
| .8 | 4.11 | 4.90 | 5.00 | +11 |
| .9 | 7.88 | 11.88 | 10.00 | +1 |

FIGURE IV(b)

ACMS has a number of commands which permit the user to request performance statistics for the modeled system. These commands are summarized in Table 1. Two particularly noteworthy commands are CONSTRAIN and RELAX. These commands are used to describe the feasible region for design optimization problems. Because ACMS can solve performance problems very quickly, it is well suited to the iterative solution techniques required to solve these optimization problems.

| COMMAND CATEGORY | COMMAND | FUNCTION |
|---|---|---|
| WORKLOAD SPECIFICATION | THRUPUT | Set Desired Throughput |
| | RATIO | Set Workload Composition |
| | MTBA | Set Mean Time Between Arrivals |
| | LISTR | List the Above Values |
| PERFORMANCE STATISTICS | LISTB | Lists Minimum Job Completion Times |
| | LISTN | Determines Maximum Throughput for This Configuration |
| | LISTS | Report Response Time and Device Utilizations |
| | LISTU | Report Users of a Device or Devices Used by a Process |
| | LISTP | Report Process Service and Queuing Times |
| | LISTD | Report Number, Rate, Mean Service Time, and Request Arrival Rate for Devices |
| | SCALE | Reset Time Units for Reports |
| SYSTEM DESIGN | TFIX | Temporarily Changes Device Characteristics |
| | CONSTRAIN | Set Response Time Performance Goals |
| | RELAX | Specify Device Characteristics That May Be Changed To Meet Performance Goals |
| | SWITCH | Changes the Currently Selected Device in a "CALLED" Group |
| | LISTC | Report Constraints and Relaxed Device Characteristics |
| | LISTO | Lists all Devices in a "CALLED" Group and Indicates Device Which is Currently Selected |
| USER AIDS | COMMENT | Record User Suggestions |
| | HELP | Report Command Format and Function |
| | TUTORIAL | Generate Sample ACMS Session |
| | PRINT | The Command Turns Printing On and Off on the Terminal |
| SESSION TERMINATION | QUIT or END | Exit ACMS |

Summary of ACMS Commands

TABLE I

ACMS can solve problems of the form "minimize total hardware costs subject to application process completion times (response times) and technological constraints." The problems which can be solved include determining the type and number of devices which will minimize total system cost. For example, a solution might be "buy the Model A CPU, 8 Model B disks, 4 Model C tape drives. etc." To solve problems of

this type, ACMS uses the following optimization heuristic:

> Step 0: For each device type, select the least cost device for the initial configuration.
>
> Step 1: Relax the constraint for integer numbers of devices. Use the method of Lagrange multipliers to find the number of devices in the current configuration which minimizes total cost. Round each device quantity to the next higher integer.
>
> Step 2: For each device type, from most to least costly, decrease the number of devices in the configuration until an additional decrease of one device of any type would violate either a reponse time or a technological constraint.
>
> Step 3: Determine if adding a device to the most costly device type, whose quantitiy can be increased, permits enough devices of lower cost to have their quantities decreased so that a lower total system cost results. Repeat this step while such device exchanges are possible.
>
> Step 4: Select the least cost device which has not been evaluated in a configuration and which has the potential to reduce total system cost. If no device which meets these conditions exists, then go to Step 5.
>
> Step 5. STOP.

In theory, this algorithm has many shortcomings. The most significant shortcoming is that the number of possible combinations of devices within device types is severely limited. In practice, however, the algorithm has proven to be extremely effective. FEDSIM has used this feature of ACMS to build a model consisting of over two dozen application processes and a collection of hardware resources which included several CPU models, memory sizes, DASD types, etc. The model included numerous technological constraints (e.g., only certain classes of memory would operate with particular CPUs due to electronic incompatibilities). ACMS converged very quickly to configurations which met the response time and technological constraints and had a total system cost equal (or nearly equal) to the cost found by complete enumeration of the feasible region.

In summary, ACMS has satisfied FEDSIM's original design goals. It is well suited to rapid model development, inexpensive execution, and, in many cases, was more accurate than originally anticipated. One problem with ACMS, however, has been applying it in situations that do not meet the assumptions concerning the interarrival times of events triggering the application processes. ACMS is designed to model systems in which "work" arrives "randomly." Analyzing other arrival patterns with ACMS can produce misleading results. To overcome this problem and extend the usefulness of ACMS, ECSS III is under design at FEDSIM. ECSS III will be a hybrid modeling system which combines the speed of analytical modeling with the robustness of discrete-event