Run 2

Function	Number of References	Mean Service Time*	(Coefficient of Variances)^2
RSY	5373	77.7231	5.3346
RCH	16365	23.8348	17.1703
DCH	38039	0.0000	0.0000
CCH	0	0.0000	0.0000
RMS	292	1.0890	.1377
RHT	15033	1.6876	.7339
DHT	4752	1.7298	.1774
RST	8317	36.5118	10.7557
PFR	772	20.5207	7.2305
DPP	60259	.4879	1.0893
ABT	19	.3684	1.7143
RCL	51026	0.0000	0.0000
RCP	12037	0.0000	0.0000
DCP	2396	0.0000	0.0000
REC	4618	5.4073	.5041
WEC	3689	6.6208	.2991
CES	133	1.0075	.1407
REQ	5862	.9795	.1692
DEQ	2605	1.0192	.1630
PDE	0	0.0000	0.0000
RPP	81	.1481	5.7500
LPP	103826	1.0189	.1234
edr	9663	0.0000	0.0000
CEF	173	0.0000	0.0000
RGF	0	0.0000	0.0000
ACB	2256	.9734	.1583
RWD	26564	0.0000	0.0000
RLF	39239	1.0229	.1529
SCH	5633	0.0000	0.0000
SCP	161	0.0000	0.0000
REV	0	0.0000	0.0000
STM	0	0.0000	0.0000
DTL	0	0.0000	0.0000
CLS	4183	1.1265	.1526
FJN	2163	1.0388	.1597
RTP	12132	3.4074	13.8957
RES	191	0.0000	0.0000
RCX	1902	7.8586	8.6452
RCS	2180	83.1569	4.9883

*Time in milleseconds

Table 4.4

Function Statistics

functions. A peripheral process always blocks after issuing a function until a response is received from MTR. Due to the polling nature of MTR, some time elapses between the issuance of the function request and MTR's noticing it. As the event is recorded by the software probe in MTR, this time is seen as part of the active state for a process prior to the request. This polling interval averages on the order of two milleseconds.

In certain instances, however, this time can be of a much larger magnitude. CPM runs at the command of MTR for those reasons as stated in section 3.1.2.6. Whenever MTR invokes CPM, it blocks until CPM is finished. During this blocked interval, of course, MTR cannot notice nor record function requests. When CPM is running to process a function, its active period averages around 1 millesecond. For the job scheduler, on the other hand, the average is around 20 milleseconds and can be as long as 50 milleseconds. The corresponding statistics for a central relocation are 9 milleseconds and 80 milleseconds. To decrease the effect of these perturbations, any request recorded within 2 milleseconds after a job scheduler or central relocation execution does not have that particular instance of apparent peripheral processor active time included in computing the timing statistics for that arc.

As the model reintroduced similar perturbations, the overall timing statistics for each process included these perturbed times for validation purposes.

4.1.2.1 SPG Builder for Peripheral Processes

A program was written to sequentially analyze the trace data and construct and parameterize the SPG's for peripheral processes. The program did not do any automatic construction of loops as no scheme could be found which would allow loop construction and result in graphs free of spurious paths. The graph builder, of course, had to be able to follow pre-existing paths in graphs, producing a new arc only when a new node was encountered as a successor. This allowed a pre-constructed graph skeleton structure to be input as data to the graph builder. If the graphs had no loops, their size became unmanageable. The fact that loops were necessary and that the builder could accept a pre-constructed graph structure as data led to an iterative methodology being employed to construct the SPG's.

The graph builder, on its initial execution, might be supplied with only a root node as a graph skeleton. The graph builder would then make a short run down the trace tape. The resulting graphs would be examined closely in association, possibly, with the source code for the corresponding processes. Real loops would be carefully determined and a more elaborate graph skeleton produced for the next execution. This would be repeated until the graph builder constructed no new paths, that is, the graph skeleton included all paths encountered during the system run being emulated. Figure 4.1 is a block diagram illustrating the methodology employed in construction of SPG's for system processes.

Figures 4.2, 4.3, 4.4, 4.5 and 4.6 present the SPG's for some of the more regularly used peripheral processes. It should be remembered that these SPG's only reflect those paths through the various

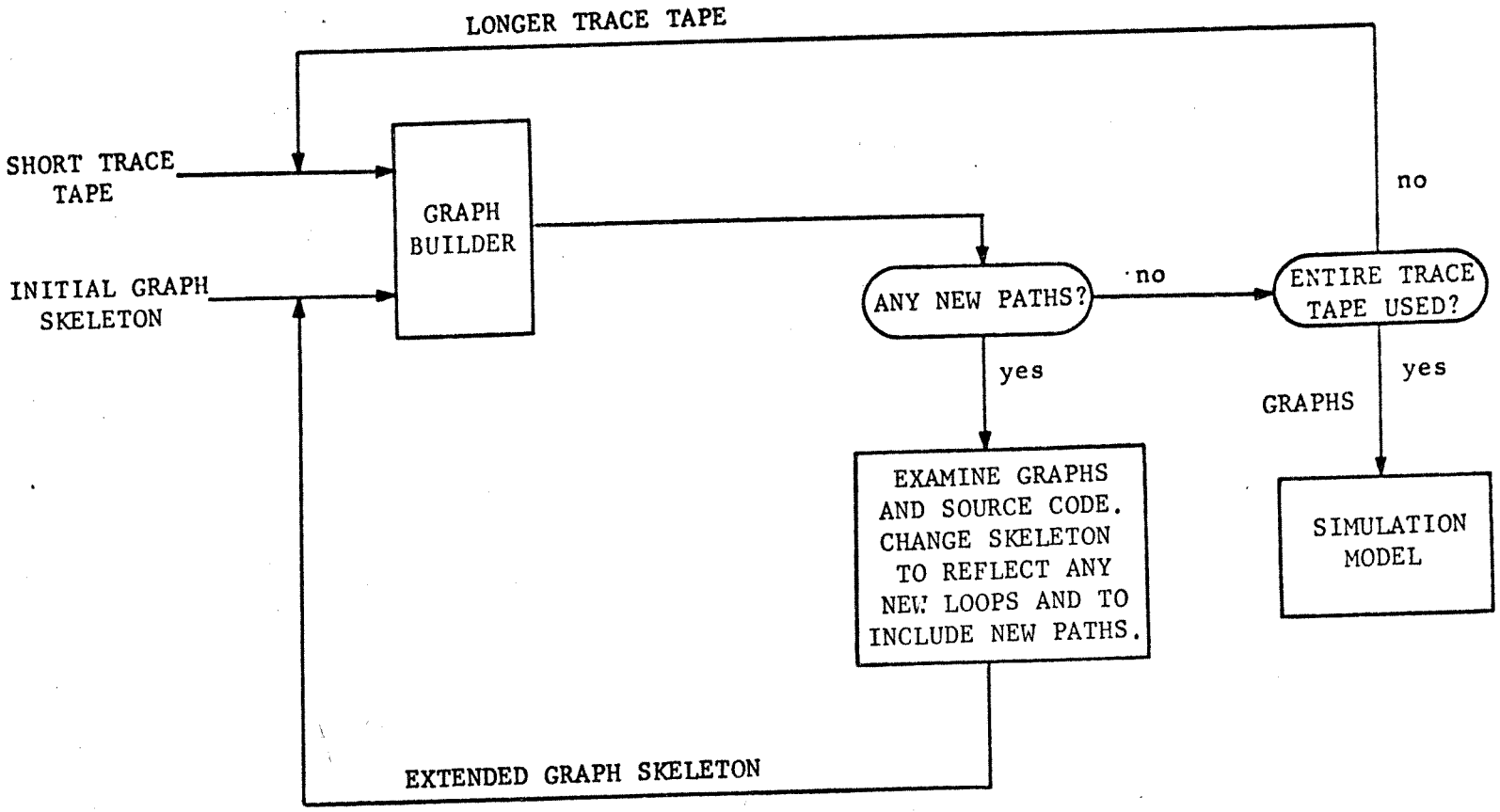

Figure 4.1

Methodology for Constructing SPG's for Peripheral Processes

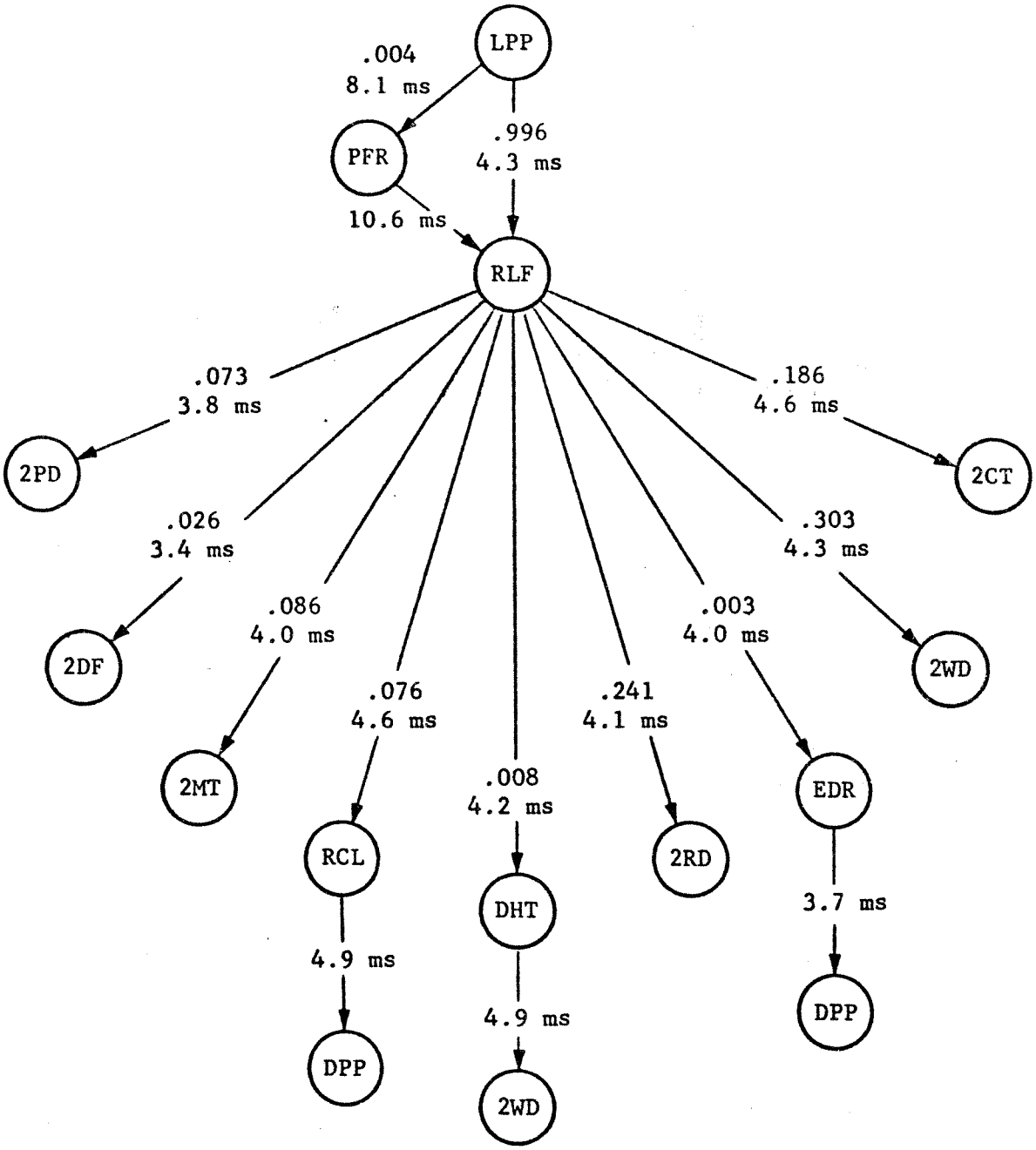

Figure 4.2
CIO

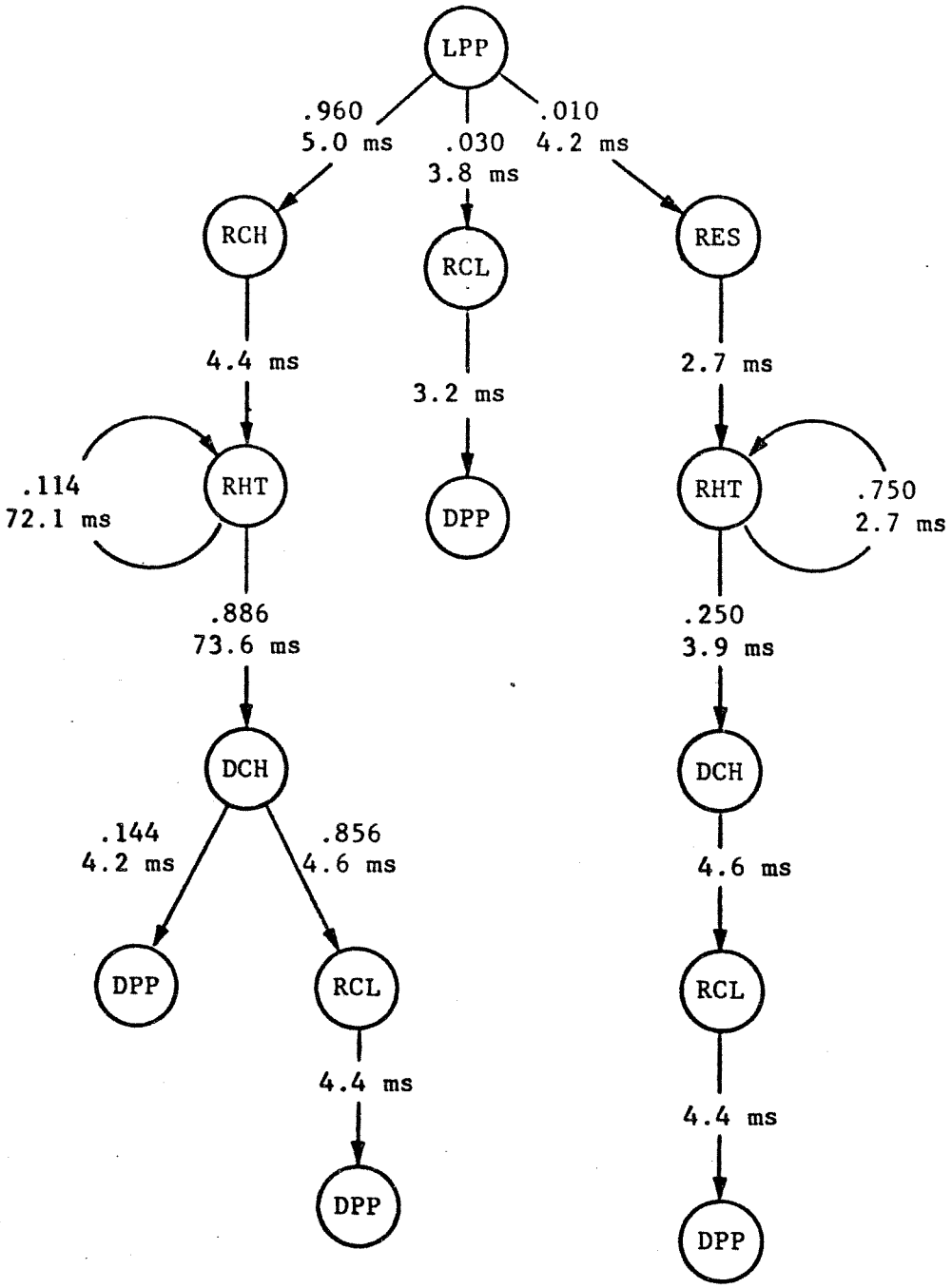

Figure 4.3
2RD

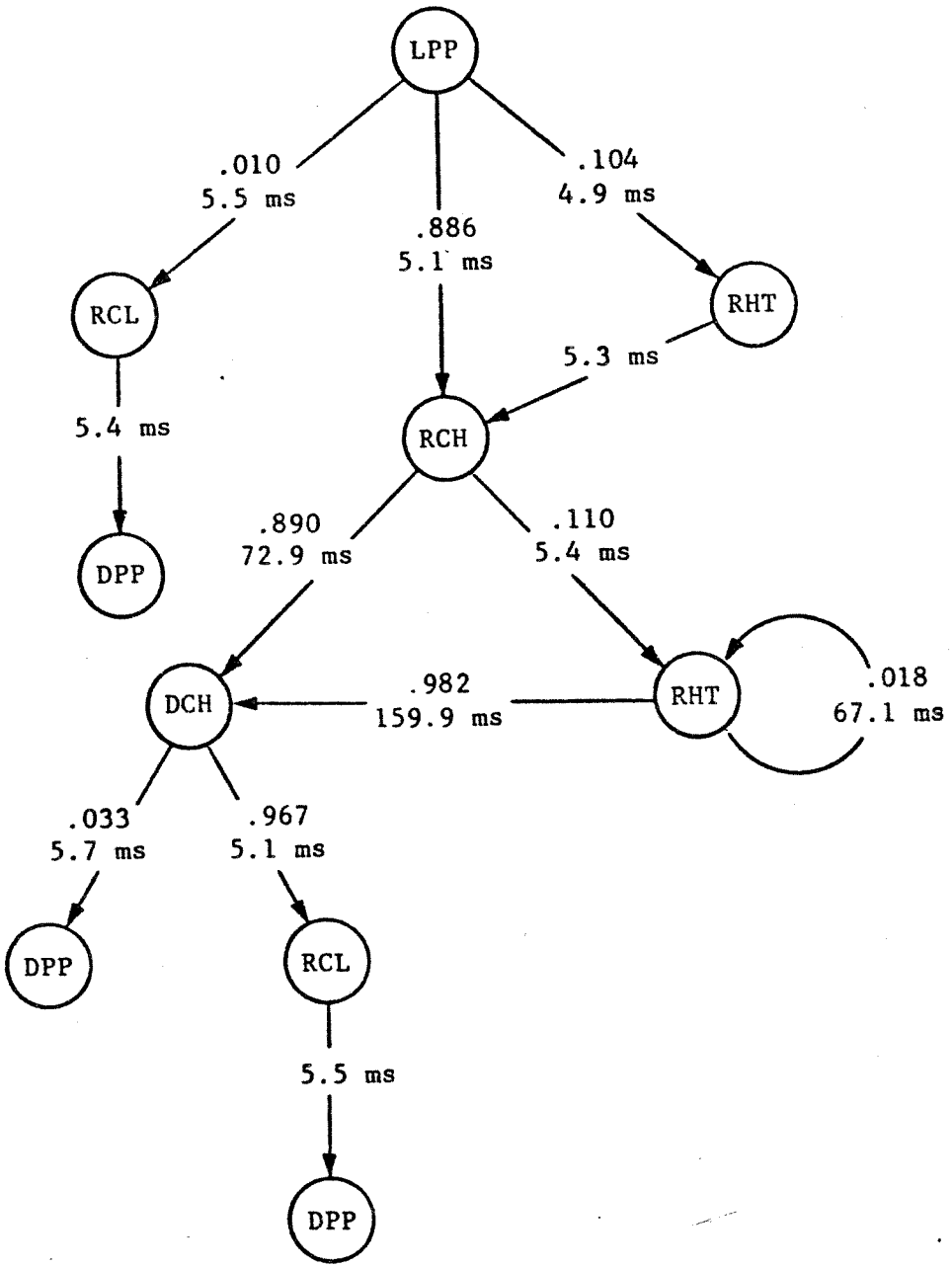

Figure 4.4
2WD

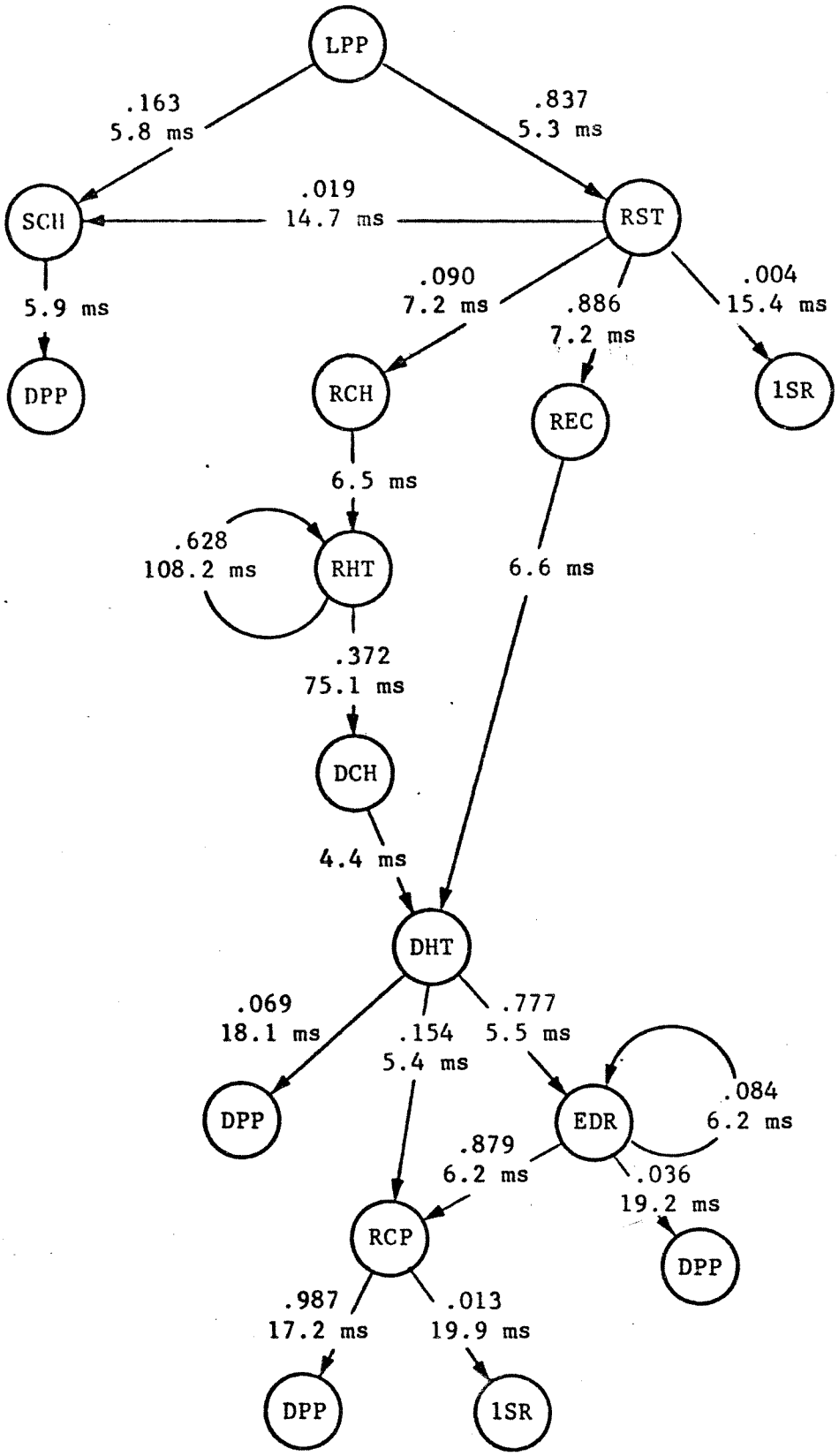

Figure 4.5
1RJ

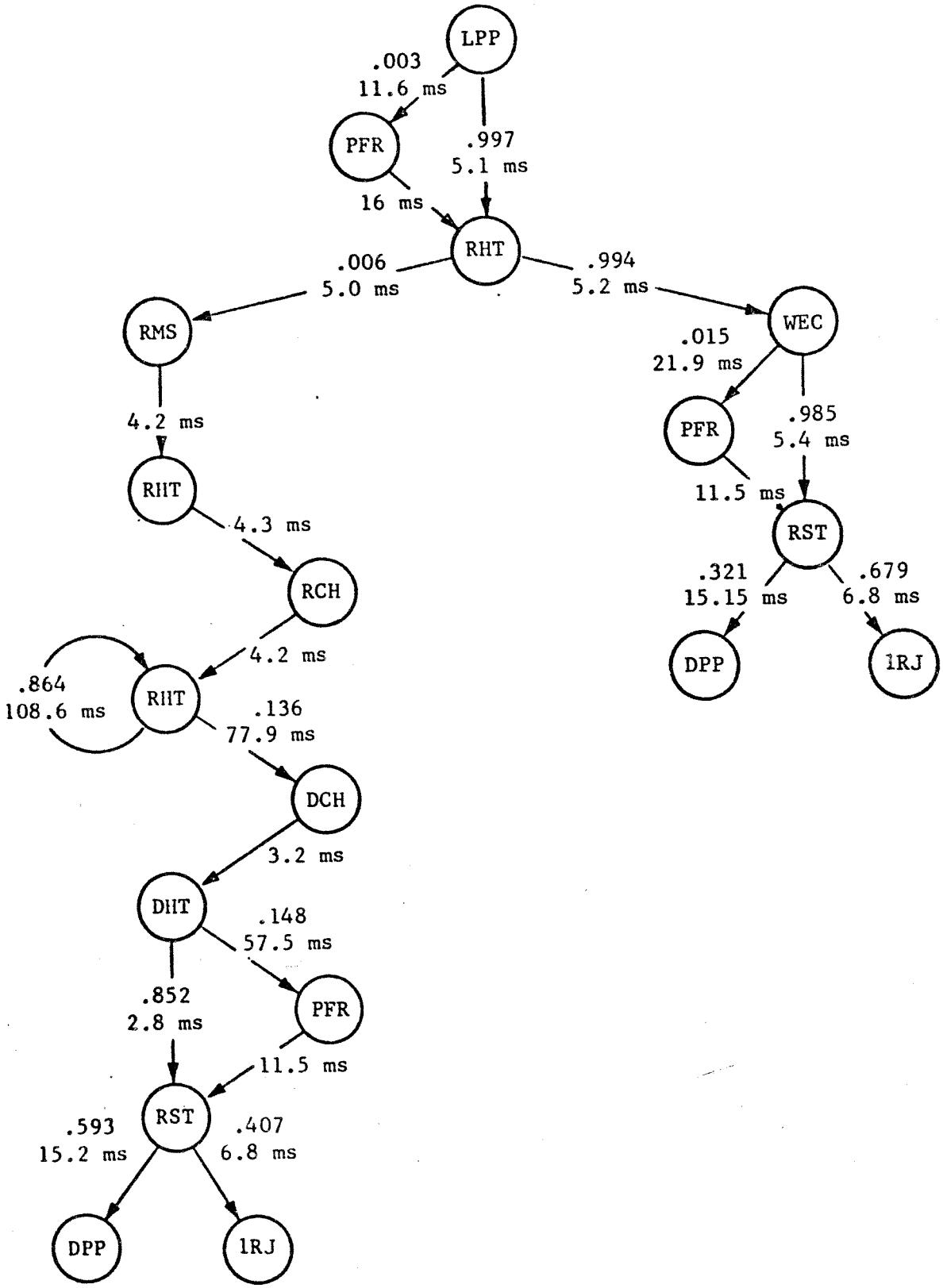

Figure 4.6
1SJ

process state graphs that were observed during the monitored period of system operation.

4.1.3 Central Processes

SPG's were constructed for central processes at the job level. No system nor library processes were isolated into their separate SPG's. This was done for several reasons. First, it reduces the complexity of the model. Also, only a few of the central processes, specifically the central loader, Fortran compiler, interactive control card processor and a few file managing processes, ran often enough to provide sufficient performance statistics to construct the SPG's from analysis of the trace data. It was also difficult to determine when the central loader transferred control to the process it had loaded, that is, it would be hard to separate the central loader SPG from the SPG for the loaded process.

4.1.3.1 Construction of Job SPG's

A program was also written to separate the resource patterns into individual job graphs. A central process consumes resources in three ways: it uses time on the CPU; it occupies space in central memory; and it creates dependent processes. It was this information that needed to be recorded and preserved in the SPG.

The SPG's for jobs are linear, the nodes of the graphs are RA+1 calls which are serviced by MTR and often create dependent peripheral processes. The arcs are tagged with the active CPU time between RA+1 calls. In order to keep track of these memory requirements for each job, when a peripheral process (running for some job) issued an RST

request, the field length requested in the RST was attached to the preceding node in the SPG for that job.

The program had to isolate each user job and remove all system interference. The resulting graphs reflect only the active periods of the central processes.

As many of the jobs were existing prior to initiating the software monitor and many continued to exist after the trace finished, many of the central SPG's do not reflect complete jobs.

The graph builder also kept certain statistics on central processes, particularly, the total active CPU time listed by each job, the degree of multiprogramming and the mean core occupancy.

4.2 Operation of the Model

The software half of the model, the SPG's being the other half, consists of routines simulating those processes not represented by SPG's, specifically MTR and CPM, as well as path selection and graph traversal routines, routines for gathering model performance statistics, and other I/O, bookkeeping and functional routines.

4.2.1 Degree of Resolution

By having SPG's down to the level of functions and by simulating closely those processes not represented by SPG's, a highly detailed model can be realized. How accurately the model realizes the system it is simulating is controlled by several real restrictions and various pragmatic concerns.

First of all, consider the simulation of non-SPG processes. In order for the simulation to faithfully reproduce the decisions and actions of the real processes, it must have access to the same data used by the actual processes. Quite often this data is incomplete or totally missing from the trace tapes. For instance, the job scheduler selects jobs for residence in central memory based on their field length and remaining service time requirements. The simulated job scheduler in most cases has a relatively accurate representation of the field length requirements for each job, but has only the observed processor time used by each job as obtained from the trace tape. A long job might only receive a few cycles of CPU service during the actual system run and a relatively short job might receive a much larger amount of CPU service. As the simulated job scheduler makes decisions on the

observed times, the long job would receive priority over the short.

To truly reflect the execution of the system, all paths through the SPG's should be deterministic or environment dependent. Again, in many cases, the information on which to base the selections deterministically is not available. In other cases some or all of the necessary information is available but its use would increase the complexity of the model (the SPG's, software or both) beyond its worth. As an instance of this, consider the RWD (reserve word) function. This function is used as an interlock to insure that only one PPU at a time is referencing a given word in central memory. That portion of an SPG containing a RWD node might typically appear as in Figure 4.7.

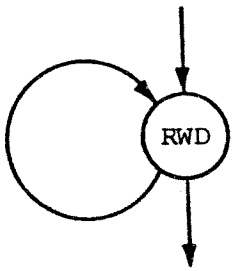

Figure 4.7

Typical "RWD" Node

A process requesting access to a particular word generally loops on the request as long as MTR indicates some other process owns the access rights to that word. The word referenced by each RWD is recorded by

the software probe. To provide deterministic path selection an association would have to be made between each RWD and its designated word. Two approaches readily come to mind: (1) as with the RCH function, extend RWD to a set of functions, one for each unique word referenced by an RWD during the system run being emulated; or (2) encode the path selection process so that it can determine (by the process issuing function, the parent job number, *etc.*) the word to be accessed. Both of these schemes complicate the model to a substantial extent. Further, the model would also have to contain additional necessary bookkeeping routines with queues and "busy" indicators for each word. All this added complexity has really gained is that the model can now determine whether the loop path will be taken after a RWD node or not.

The selection of a particular arc is environment dependent in this model when reflecting system access decisions for resources being studied or upon the invoking of processes whose behavior is of critical interest or importance.

The non-SPG processes are modeled as closely as possible after those of the actual system, particularly when controlling access of important resources or invoking interesting processes.

4.2.2 CPM and MTR Processes

As previously indicated, routines simulating the various algorithms of CPM and MTR constitute a large portion of the model software. The CPU scheduler, PPU scheduler, job scheduler and several of the routines handling the requesting and relinquishing of interesting resources and processes are simulated in some detail.

4.2.2.1 CPU Scheduling

The CPU scheduler, as described in Chapter 2, is located in MTR and is basically a round robin scheduler selecting among waiting processes resident in central memory.

In the model, the selection of a process to become active results in one or more events being placed on the event list which drives the simulation. A chronological traversal of the SPG for the active central process will determine the events to be entered. If no interesting events, *i.e.*, RA+1 calls, occur during the next quantum for the process, the only event entered on the list is one indicating when the CPU scheduler is to run once more. Any RA+1 call occurring during the next quantum would of course be entered in the event list. A RCL function or RA+1 call with the recall flag set automatically terminates the traversal of the SPG for this period and, when such an event is removed from the list, the CPU scheduler will again run.

The events for an active process will be chronologically perturbed each time that CPM is exchanged on to the CPU.

4.2.2.2 PPU Scheduling

During active periods, central processes can issue RA+1 calls which result in the creation of dependent peripheral processes. Peripheral processes can themselves, by issuing EDR (enter delayed request) functions or LPP (locate peripheral process) functions, cause other peripheral processes to be executed. In many instances and for various reasons, peripheral processes issue EDR's for themselves to be run at later times. There also exists a RPP (request peripheral process)

function by which one peripheral process can cause another peripheral process to be executed if a PPU is available at the time, but does not result in any queuing for PPU's. MTR creates peripheral processes, lSJ (suspend job) and lRJ (resume job), to carry out the decisions of the job scheduler. A peripheral process by definition consists of a peripheral program being executed on a peripheral processor. As there are only a limited number of these processors, there has to be a scheduler. The PPU scheduler resides in MTR and assigns PPU's, basically, on a first come first serve basis, differing from a strict implementation of this discipline as detailed in section 3.1.2.6.

Requests for peripheral processes, when encountered during execution of the simulator, are entered into the event list. The case of RA+1 calls was covered in the previous section. EDR's are entered into the event list, delayed appropriately, when they are encountered in the traversal of the SPG for an active peripheral process. The others, lSJ and lRJ, are entered when the job scheduler is run.

When a peripheral process request is the next chronological event in the list, the PPU's are checked to see if one is free. If so, the peripheral process is invoked and traversal for its SPG is initiated by entering its first function request, LPP, into the event list. If no PPU is currently free, the request enters the PPU queue with lSJ's and lRJ's going to the head of the queue.

Each ABT or DPP (abort or drop peripheral process) function causes the queue to be checked for waiting requests and the next request, if there is one, to be processed.

4.2.2.3 Job Scheduler

In section 4.2.1 it was mentioned that the job scheduler had to select jobs for residency in central memory using incomplete information, specifically the only central processor service time requirement available for each job was the total of its observed active periods extracted from the trace tape. This only presents a problem when computing the cost of batch jobs. For interactive jobs, the timing factor of the cost formula is based on the CPU service time used since the last interaction and this information is supplied by analysis of the trace tape.

To make the competition of batch jobs with interactive jobs more in line with that of the modeled system, short batch jobs had their costs computed using a time factor greater than observed. This will be discussed in more detail in the sections on validation.

As the job scheduler is in CPM, each time the scheduler was run in the model it was necessary to compute the processing time required, again based on data from the trace, and to perturb the event list to reflect this overhead.

4.2.2.4 Servicing of RA+1 Calls

When MTR encounters a RA+1 call from a central process, it either services the call itself or, when the call is a request for a peripheral process, initiates the creation of that process. In either instance, the execution time required by MTR to act upon the request cannot be determined by analysis of the trace tape. However, the time is quite small and is reflected in a slower polling rate for MTR, *i.e.*,

it might make the active times for both central and peripheral processes between successive communications with MTR appear slightly longer than is actually the case.

Table 4.5 lists the RA+1 calls recorded on the trace tape. The first fourteen are requests for peripheral processes resulting in the execution of the associated transient peripheral process in the actual system and in the invoking and traversal of the associated SPG in the model. RA+1 calls for MSG are sometimes requests for the MSG peripheral process and at other times are requests for service from MTR alone. This is reflected in the model by having an appropriate fraction of the MSG calls result in the invoking of the SPG for MSG and treating the other MSG calls as requests for no service. It should be noted that if the auto-recall flag is set, that is, the RA+1 call has the form XXXP, the process issuing the request goes into the recall state, *i.e.*, the CP is removed from the CPU queue, until the termination of the process created by the call or until the issuance of a RCL function by that process.

In the actual system TIM and GSN calls are handled solely by MTR and would be treated, essentially, as requests for no service in the model. However, to reduce the size of the SPG's for user jobs, these calls were simply not included.

END and ABT notify the system that the current central process running for a job has terminated. This results in the execution of the transient peripheral process, lAJ (advance job), as soon as all dependent activity has terminated for the control point. In the model, under like conditions, the SPG for lAJ is invoked and traversed.

1. CIO - Standard I/O
2. OPE - Open file
3. CLO - Close file
4. RSF - Release system file
5. SNP - Snapshot of central memory
6. DMP - Dump central memory
7. INT - Interrupt central program
8. RFL - Request field length
9. PCC - Process Control Card
10. RCC - Read control cards
11. CPU - CP utility services
12. LDR - PP loader
13. EPR - TAURUS service routine
14. PFM - Permanent file manager
15. MSG - Issue message
16. TIM - Get time, clock or date
17. GSN - Get job name
18. END - End program
19. ABT - End program and abort job
20. RCL - Enter standard CP recall
21. RCLP - Enter auto-recall

Table 4.3

RA+1 Calls on the Trace Tape

The RA+1 call, RCL, presents a somewhat more difficult problem in the model. Typically, a RA+1 call for RCL follows one or more RA+1 calls for peripheral processes. An RCL causes the CP to go into recall status for a number of milleseconds as specified in the call. At the end of that period, MTR returns the CP to the queue. Immediately upon becoming active, the process at that CP checks to see if the previously created peripheral processes have terminated or reached some desired state. If not, the process again issues an RCL request. This procedure might be repeated several times, the actual number depending on the state of the system. Although the time required to check the state of the peripheral processes might be measured in microseconds, due to the polling nature of MTR, the CP typically holds the CPU for at least one millesecond waiting upon MTR to service its new RCL. Multiple RCL's constitute a form of system interference. As mentioned in section 4.1.3.1, it is desirable to eliminate, as near as possible, all system interference from the SPG's for jobs.

When constructing the SPG's for jobs, only the first RCL in each series was recorded and included as a node. The active time used by a job in the issuance of other RCL's in the series was included in the time associated with the arc between the last RCL and the next RA+1 call. When an RCL RA+1 call was encountered during execution of the model, the CP was first removed from the CPU queue. A check was then made to determine if the central process concerned had any requests for peripheral processes in the PPU queue or any dependent peripheral processes currently active. If there were no requests or dependent processes, the CP was returned to the CPU queue and the

execution of the simulation continued. If there were requests or dependent processes, the once associated with the most recent RA+1 call had its auto-recall flag set. Until this dependent process terminates or issues an RCL function, the CP containing the parent central process cannot enter the CPU queue.

A simpler alternate approach to handling the RCL problem would have been to, during construction of the SPG's, set the auto-recall flag on the node preceding the RCL call and to discard all RCL's otherwise. This would have eliminated, however, some of the possible overlap between central and peripheral processing for a given job.

It should be noted that two central processes were not included in the special handling of RCL's, neither in the construction of their SPG's nor processing during execution of the simulation. All RCL's for TAURUS and PISCES were retained in their SPG's as they were to actually remove their CP's from the CPU queue for specified time periods. The CP for TAURUS was reentered into the CPU queue periodically (every 90 to 100 milleseconds) by a RCL function from 1ED (the remote entry driver).

4.2.2.5 Processing of Functions

Those algorithms in MTR and CPM which process functions that request or relinquish central memory, processors and channels (for disk, tapes, and ECS) are represented with a high degree of resolution by routines in the simulation. Those algorithms processing other functions are represented in much less detail, some only to the extent that the PPU waits after issuing the request for the time associated with the arc of the function's SPG to elapse.

It should be noted that queues for resources requested by functions are each serviced by a cyclic scheduler in the UT-2 system. MTR polls cyclically the communication cells in CM associated with the PPU's. PPU's that have issued function requests are waiting for MTR to respond. Each time MTR encounters a request, it checks to see if it can process that request. If not, it continues on leaving the request word in CM unchanged.

In this fashion a PPU would wait, say, on channel #1 to become available. It would be possible for a PPU to come along at a later time, also request channel #1, and have its request satisfied before or after the one already waiting.

In the model, the queues representing the polling queues are all serviced by a FCFS scheduler. Specifically, such queues are kept for the ECS pseudo-channel, all disk channels, the tape channel, and all requests for increases in the amount of central memory allocated to a control point.

The detailed servicing of particular functions will be discussed in greater detail as they relate to the traversal of the SPG's for peripheral processes or to the various experiments conducted using the simulation model.

4.2.2.6 Traversals of SPG's

The SPG's for central processes are traversed as outlined in section 4.2.2.1. As each node has at most one successor, traversal is rather simple, progress depending only upon the process being in the active state.

The traversals of SPG's for peripheral processes is somewhat more complex. The invoking of these SPG's and the initiation of their traversal is covered in section 4.2.2.2. When the next chronological event is a reply to a function issued by a peripheral process, the simulation selects the next node in the SPG for that process. There are two basic methods in the simulation for selecting the next node. One is strictly statistical in nature and the other method is environment dependent or deterministic in nature.

As each arc is tagged with its observed relative frequency of traversal, statistical selection is quite simple.

When the selection of the successor node is to be environment dependent, the present node has a pointer to a subroutine which makes this selection. Prior to being input to the simulation, these pointers to subroutines consist of the names of the subroutines only. At compile time a table relating the names and entry points of path selecting subroutines is constructed by an assembly language subroutine. As the SPG's for peripheral processes are read, the subroutine pointers attached to the various nodes are compared with the names of the subroutines in the table. If a match is found, the name is replaced by the address in the pointer. If not, the pointer is cleared. During traversal, a non-zero pointer indicates the path selection is to be made by a subroutine. The subroutines which construct the table, set the pointers and make the subroutine calls during traversal are written in Compass, the assembly language for the CDC 6600. All other code in the model was written in Fortran.

As an example of selecting a path based on environmental conditions, consider the first few nodes of the SPG for 1RJ (Figure 4.5). After the LPP function node, the successor node will be either an RST (request for memory) or an SCH (request for the job scheduler to run). The subroutine associated with the LPP would check to see if enough free memory is available to satisfy the RST request. If so, the RST node is selected; if not, the SCH node.

Sometimes a mixture of the two methods is used. Again look at the SPG for 1RJ. Suppose that the RST node had been selected. Due to the queuing of RST's requesting increases in memory, it is possible for another RST to be satisfied first reducing the free memory to a level less than that required. Should this be the case, the subroutine associated with the RST would force the selection of the SCH node as successor. If not, one of the other four successors would be selected statistically. This statistical selection is itself a special case of deterministic selection by the subroutine as the SCH node is not allowed to be a successor.

Once a node has been selected, the time required to traverse the arc linking the two nodes would be computed. The request for the function would then be inserted in the event list along with the clock time for the request.

When a control point has been selected for relocation, it must wait until all dependent peripheral processes have either terminated or indicated that they are ready for the move to occur by issuing a PFR (pause for relocation) function. Many of the peripheral processes can issue this function at various times during their execution. It is a

conditional function, however, being issued only when the move flag is set for the CP for which the PPU is running. The PFR nodes in the SPG's are never selected statistically, but only when their CP is to be moved. A problem arises, however, in that as soon as a node is selected, the corresponding function request is put on the event list to occur at the proper future time. In the actual system a peripheral process might, between the issuing of two functions, check the move flag for its CP several times and issue the PFR function if it were to be set during execution. In the model, the PFR node would be selected only if the move flag was set when the previous function response were received. To alleviate this problem each time a PFR node was a sibling of the selected node, a record was kept of this fact. Then, when a CP was selected to be moved, a check was made of all active dependent processes and, if a PFR node were a sibling as mentioned above, the event in the list was changed to a PFR request and the SPG pointer and bookkeeping records were changed accordingly.

Another problem can arise when the statistical selection of nodes is the sole control over the number of times the SPG for a particular process is invoked, particularly when that process ran only a few times during the run being simulated but consumed a large quantity of resources each time it ran. Consider for instance the peripheral process, 1XX. 1SS calls 1XX whenever ECS becomes seven-eights full. Each execution 1XX removes one job from ECS and transfers it to disk. 1XX repeatedly runs, by issuing an EDR for itself, until ECS is less than three-quarters full. During one of the system runs used in this study, 1SS called 1XX only nine times. By issuing EDR's, 1XX ran a total of

35 times. Further, 1XX occupied a PPU for an average of over one and a half seconds each time it ran, holding a channel for a not inconsiderable fraction of this time. Due to the observed relative infrequency of execution, statistical fluctuation in the simulation could cause the number of times that the SPG for 1XX was invoked to vary considerably. This, coupled with the amount of resources consumed by that process, could result in severe perturbations in the system performance measures. Even though the relative fullness of ECS is not computed in the simulation, the selection of the nodes resulting in the invoking of 1XX cannot be statistical. In these instances, the selection could be rated, that is, force the invoking of the process when its use of resources falls below a predetermined level and prevent its invoking if its use of resources exceeds that level. Again, for 1XX, its selection was based on the amount of PPU time it was observed to consume.

4.2.3 Validation

The importance of validation in trace-driven modeling has been stressed [1,17]. In order to have confidence in the results of studies utilizing a trace-driven simulation model, it is first necessary to determine that the simulation does in fact accurately emulate the computer system concerned.

Validation is a long and complicated process due primarily to the large number of variables involved. The more complicated the model, the more difficult the validation.

The actual methodology used in validation can be considered an iterative process consisting of two basic steps:

(1) Remove all known logical errors from the program and verify that the various parts of the program execute as planned. In this particular model, it is also necessary to verify that the SPG's adequately represent the resource utilization patterns of their associated processes and that all known spurious paths have been eliminated.

(2) Execute the simulation and compare the system performance measures obtained with those of the system modeled. If the measures agree sufficiently, the validation is done. If not, the areas of disagreement would indicate where changes in the model might be required. The changes would be made in the model and step (1) would again be performed.

4.2.3.1 Job Mix

In section 4.2.2.3 the problem of scheduling jobs on the basis of incomplete data was presented. A related problem, perhaps the most difficult one encountered during validation and perhaps the one most responsible for variations in performance measures, was that of providing the simulated job scheduler with a representative job mix from which to make its selections. It is possible that, for the duration of the actual system run, most of the jobs were available to the job scheduler, indeed there were probably jobs available which did not appear in the trace because of their "expense." Clearly, if all the jobs observed during the run were made available at the beginning of the simulation, based upon their observed execution statistics, all

"cheap" appearing jobs would run first and all "expensive" appearing jobs run last, severely perturbing model performance.

Another approach would be to make the jobs available to the simulator based upon the times of their first appearance in the trace. This approach was not taken, however, as it would present restrictions on the system performance during experimentation. For instance, experiments which increased the throughput of the model would have fewer jobs to select from and those that decreased the same measure would have more. In either instance, the resulting effects on model performance would be moderated or perturbed.

The approach taken was one which attempted to minimize the effects mentioned in the first approach and, at the same time, not impose any artificial restrictions. The number of batch and interactive jobs in the available job mix was determined by their total CPU time requirements. The CPU time required to complete all batch jobs in the job mix was insured to represent some fraction of all CPU time consumed by batch jobs during the run and similarly for the interactive jobs. The total CPU time represented by the two types of jobs was sampled periodically and, if below the threshold for either type, new jobs of the required type would be added to the mix.

Secondary controls were also enforced. The maximum number of interactive jobs in the mix was held to 40. Further, the simulations were terminated when the remaining jobs in the job mix no longer constituted a representative sample.

To make competition between batch jobs and interactive jobs in the mix more in accordance with the actual system, batch jobs were

considered, initially, to require a minimum of 4 seconds of CPU time.

4.2.3.2 Comparison of Performance Measures

In this study, the trace tapes produced during two production runs of the UT-2 system, referred to as Run 1 and Run 2, were used to parameterize the model and provide the SPG's for inputs. System performance statistics were also obtained from analysis of these tapes. Table 4.4 presents a comparison of the basic processor and memory utilization statistics for each of the two runs and their corresponding simulations.

MEASURE	RUN 1	SIM 1	RUN 2	SIM 2
Elapsed Time (seconds)	1691.232	1541.406	1309.594	1238.074
CPU Utilization				
Control Points	.3613	.3959	.6420	.6788
CPM	.2386	.2620	.2397	.2506
Mean Core Occupancy (Percent)	.80	.90	.89	.91
Degree of Multiprogramming	11.28	12.77	9.18	9.34
PPU Utilization	.9493	.9955	.8418	.8772

Table 4.4

Comparison of Basic Performance Statistics

As stated earlier, validation for a complex model is a formidable task. That this model is complex was realized early in the study and that ideal agreement between the performance measures of the system

and the model might be different or impossible to obtain. For this reason, two extremely dissimilar runs were selected for this study. Validation criteria were selected, allowing more tolerance in the agreement of individual measures, but stressing that the model closely reflect the system's behavior under diverse conditions.

Run 1 had a heavy interactive load, severe PPU saturation and was on the edge of thrashing. Run 2 had a somewhat lighter interactive load and represented a rather balanced system. The differences in the two simulations consisted of the job SPG's, the parameterization of the SPG's for peripheral processes and functions, the parameters relating to the execution time requirements for the job scheduler and the central memory relocation algorithm, the disk channel selection probabilities and the CPU time fractions used in determining the job mix.

For both simulations, the CPU utilization was higher than that for the system run and, therefore, the elapsed time was shorter. These descrepancies result primarily from the job mix problem.

The scheduling of jobs on incomplete data manifested itself in yet another measure. The effect of the "time" factor, due to the method employed in its determination, was moderated when computing the cost of batch jobs by the job scheduler. This caused more emphasis than usual to be placed on the central memory requirements, resulting in an increased tendency to select small jobs. The degree of multiprogramming and the fraction of central memory occupied, therefore, was high in both simulations. The effect was more noticeable in Run 1 where the system was near thrashing.

The higher CPU utilization by control points results in an

increased rate of RA+1 calls. This is reflected in higher PPU utilization. The PPU utilization would be higher, however, even if the CPU utilization were not. The attempt to remove interference due to CPM execution from the arcs of the SPG's for peripheral processes could only be partially successful. Any interference not removed would result in longer traversal time requirements as the simulator reintroduces its own system interference. Consider the CIO process. The only function it issues which would require a variable service time would be a PFR. In both the simulations and the actual runs, a PFR was issued in less than 0.6% of the executions of CIO. Table 4.5 7 compares the execution statistics for CIO indicating the effect of the interference left in the SPG.

MEASURE	Run 1	SIM 1	Run 2	SIM 2
Number of Executions	28345	27476	25148	26445
Mean Execution Time (ms)	14.042	15.624	9.507	10.394
Coefficient of Variance	.894	.597	.898	.928

Table 4.5 7

Execution Statistics for CIO

Tables 4.6 8 and 4.7 9 present the disk channel utilization statistics. Again the higher CPU utilization is reflected in a higher demand for channels. During Run 1 there were 13.51 requests for disk channels per second. The rate for SIM 1 was 14.56 per second. The corresponding rates for Run 2 and SIM 2 were 14.5 and 16.7 requests per second.

Channel Number	0	1	2	3
Run 1				
Count	4669	8933	5751	3511
Mean Holding Time	103.549	127.132	95.531	95.936
Mean Waiting Time	43.377	139.885	37.040	26.447
Utilization	.2859	.6715	.3248	.1991
Balance	.7048	.4761	.7206	.7839

Non-System Disk Totals	
Count	13930
Mean Holding Time	98.316
Mean Waiting Time	36.495
Utilization	.2699
Balance	.7293

Channel Number	0	1	2	3
SIM 1				
Count	4601	8728	5663	3451
Mean Holding Time	99.621	129.439	102.258	98.887
Mean Waiting Time	33.895	144.618	46.660	21.383
Utilization	.2974	.7329	.3757	.2214
Balance	.7461	.4728	.6867	.8222

Non-System Disk Totals	
Count	13717
Mean Holding Time	100.525
Mean Waiting Time	36.343
Utilization	.2931
Balance	.7362

*All times are in milleseconds

*The statistics for Channel 1 include both RCS and RSY functions

Table 4.6

Disk Channel Statistics for Run 1 and SIM 1

Channel Number	0	1	2	3
Run 2				
Count	4922	5962	4869	3302
Mean Holding Time	79.321	102.916	75.837	67.346
Mean Waiting Time	35.459	79.312	21.382	10.124
Utilization	.2981	.4686	.2819	.1698
Balance	.6910	.5648	.7800	.8693

Non-System Disk Totals	
Count	13093
Mean Holding Time	75.005
Mean Waiting Time	23.835
Utilization	.2500
Balance	.7588

Channel Number	0	1	2	3
SIM 2				
Count	5306	6462	5376	3629
Mean Holding Time	76.24	103.051	74.372	78.841
Mean Waiting Time	28.462	88.2614	26.209	22.971
Utilization	.3267	.5378	.3229	.2311
Balance	.7282	.5386	.7394	.7744

Non-System Disk Totals	
Count	14313
Mean Holding Time	76.324
Mean Waiting Time	26.490
Utilization	.2936
Balance	.7439

Table 4.7 [9]

Disk Channel Statistics for Run 2 and SIM 2

4.2.3.3 Validation Summary

The reasonable, while not exact, agreements of the performance measures for each of the two runs and its corresponding simulation satisfy the validation criteria as stated in the preceeding section. Confidence, therefore, can be placed in the results of the experimental studies using the model.

CHAPTER 5

EXPERIMENTAL APPLICATIONS

5.0 Introduction

The goal of this research was to develop a model of an operating system, retaining the resource utilization patterns of various processes with directed graphs, and to demonstrate the utility of this model in the evaluation of suggested modifications to the actual system. This chapter presents eight experimental studies where changes were made in the model and its resulting behavior analyzed.

These experiments, while of interest in themselves, were selected primarily to demonstrate the versatility of the model in the investigation of the effects of both large and small scale modifications. Particular emphasis is placed on the model's utility in the study and isolation of precisely defined functional changes.

Each experiment was conducted twice, once on the model using the SPG's and other parameters from Run 1 and again using those of Run 2.

5.1 Experiment A - First Fit Memory Allocation

This experiment represents an attempt to reduce the time required to roll a job into central memory by modifying the algorithm which first allocates memory to its control point.

5.1.1 Memory Allocation

This section presents a description of the memory allocation scheme used in the UT-2 system.

All memory allocation to control points is controlled through MTR's processing of RST functions. These functions are issued by peripheral processes running for those control points. The RST function may be requesting more memory (RST "up") or requesting that its portion of memory be decreased (RST "down").

An RST "down" can be served as soon as its control point is idle, that is, all peripheral processes have terminated or paused (by issuing the PFR function).

An RST "up" is more complicated to service. MTR will first insure that the request can be granted and, if not, will reply accordingly to the calling PPU. MTR will only process one RST "up" at a time. It creates a gap adjacent to the requesting control point large enough to satisfy the new storage requirement by moving the other control points one at a time. When servicing a RST "up" request, it picks the next control point to be moved accordingly to the algorithm depicted in Figure 5.1. Before a control point can be moved, it must be idle as above. If the move is into a gap above the control point, it is moved all the way to the top of the gap. If the move is into a gap below, it

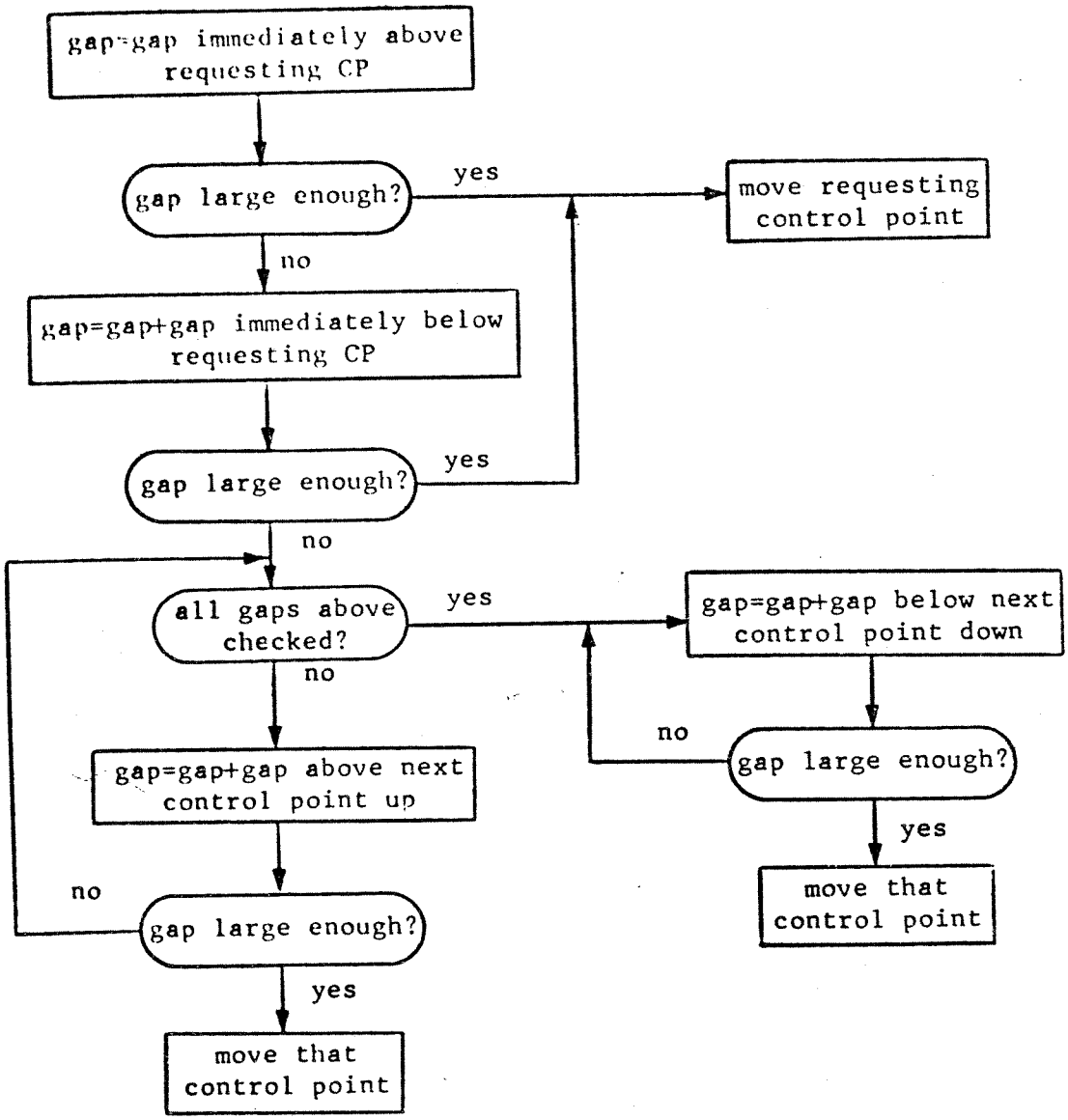

Figure 5.1

Selection of Moving Control Point

only moves down far enough so that the sum of all gaps above is just sufficient to satisfy the request.

Each time a control point is to be moved or its field length changed, MTR invokes CPM to run the storage move algorithm. One RST "up" request can result in several control points being moved. Decreasing the number of storage moves required would, of course, decrease the associated CPM overhead, but it also might decrease the time wasted by control points during the idling down period, by PPU's waiting on the servicing of RST "up" requests and, again, by PPU's pausing for the move to occur.

When a job is to be "rolled in" to a control point, the control point is originally considered to have zero length and is located just above and adjacent to the TAURUS control point (Figure 5.2). This is true even though the "roll in" is the second half of a "swap," that is, one job is "rolled out" and another "rolled in" to the same control point. 1RJ will issue a RST "up" for the necessary field length and the function will be serviced as before.

It should be noted that the algorithm for selecting control points to be moved and the distance the control points are moved was deliberately designed so that control points are packed toward the upper end of memory. This tends to produce gaps in the lower end. This is no guarantee, however, that a gap will exist next to TAURUS large enough for the newly occupied control point.

5.1.2 Experiment Description

The model followed the actual storage allocation scheme quite

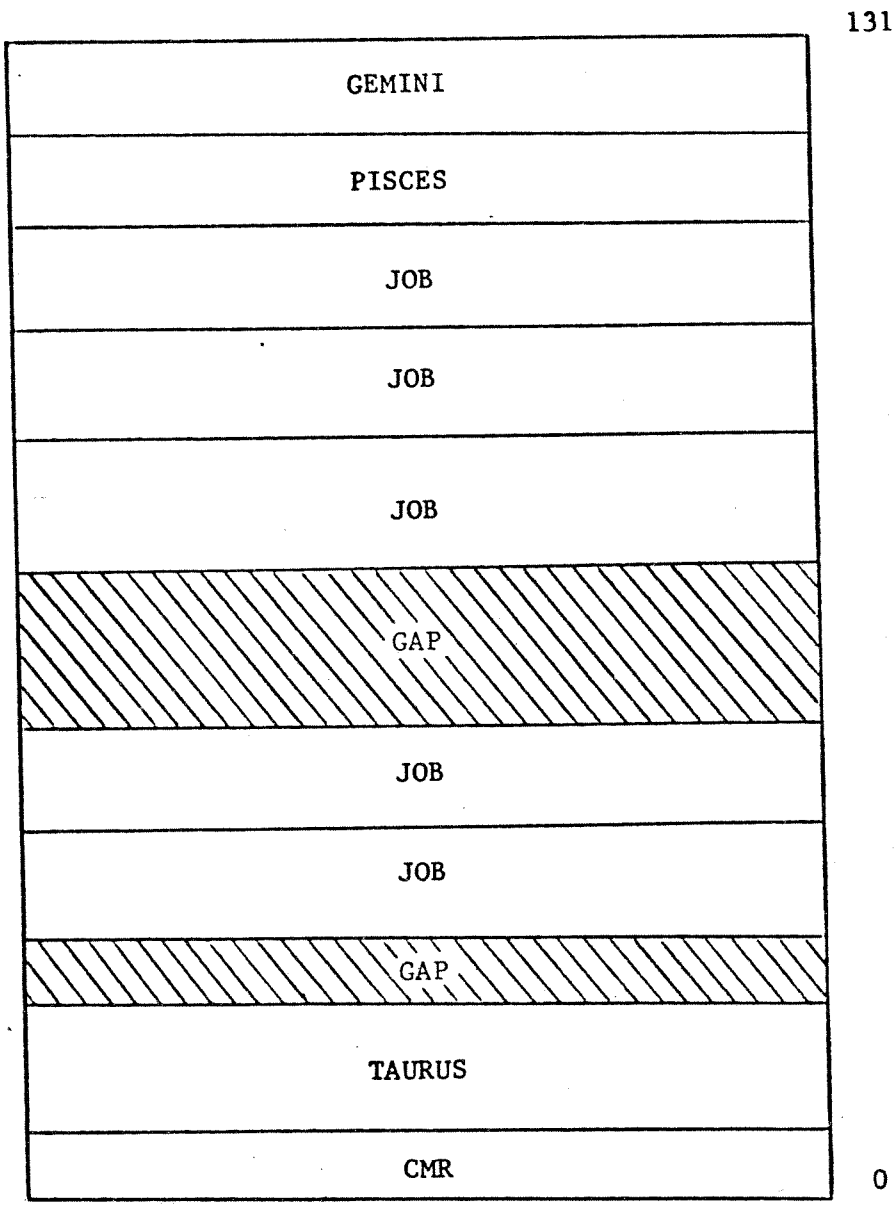

Figure 5.2

Central Memory

closely. The modification in this experiment consisted of searching the existing gaps each time a job was to be "rolled in." The control point would be located in the lowest gap large enough to satisfy the memory requirements of the job. If no such gap was found, the control point would be placed in the largest existing gap. The RST requests would then be handled as before.

5.1.3 Comparison of Results

Table 5.1 compares some of the basic performance measures of the modified model with those of it unmodified. The performance measures for SIM 1 showed very little change other than a 2.29% decrease in CPM utilization of the CPU. This was expected as the PPU's are the major bottleneck in SIM 1, not the CPU. SIM 2 showed some improvement when modified as the reduction in CPM utilization was coupled with a slight increase in CPU utilization.

MEASURE	SIM 1	EXP. A-1	SIM 2	EXP. A-2
Elapsed Time (seconds)	1541.406	1548.577	1238.074	1223.192
Utilization				
CPU				
Control Points	.3959	.3941	.6788	.6871
CPM	.2620	.2391	.2506	.2334
PPU	.9955	.9951	.8772	.8631
Central Memory	.90	.92	.91	.92
Degree of Multiprogramming	12.7	12.7	9.3	9.4

Experiment A - First Fit Memory Allocation

Table 5.1

Comparison of Performance Measures

Table 5.2 demonstrates this model's ability to isolate the results of small functional changes. The primary goal was to reduce the time required to roll a job in. This was clearly done. Also, in both cases, the number of storage moves was reduced significantly. Another observation is that the job scheduler ran more often, the increases being similar for both models. This reflects the fact that the job scheduler, after being requested through a SCH function, cannot run until all decisions from the previous scheduler execution have been completed. The decrease in the time required to roll a job in would tend to lessen the time required to carry out the decisions of the job scheduler, removing this inhibition a little more quickly.

As the number of storage moves decreased, so did the service time required for RST functions and the number of PFR's. Note, however, that the service time for PFR's increased considerable indicating that the eliminated PFR's had relatively small service times. The service time for a PFR denotes the time a PPU waits after issuing the function until the storage move is completed. This time, of course, is affected by having to wait for other PPU's to terminate or also issue a PFR. The number of executions of the storage move algorithm for the PISCES and GEMINI control points will be affected only slightly, if at all, by the first fit modification. The same follows for the number of PFR's issued by their dependent peripheral processes. This means that a larger percentage of the storage moves, and PFR's are attributed to these control points. Both PISCES and GEMINI often have more than one dependent process active at a time. This is especially true of GEMINI which generally has several. This explains, at least in part,

MEASURE	SIM 1	EXP. A-1	SIM 2	EXP. A-2
Function				
RST				
Number	7751	7769	5440	5390
Mean Service Time (ms)	59.05	33.42	48.28	25.55
PFR				
Number	910	475	561	303
Mean Service Time (ms)	24.47	39.819	26.13	36.129
Overlay				
1RJ				
Number	4294	4324	2999	2973
Mean Processing Time (ms)	183.50	135.95	122.27	82.80
Storage Move				
Number	13763	11662	8369	7349
Rate (per second)	8.9	7.5	6.7	6.0
Job Scheduler				
Number	4629	4824	3660	3787
Rate (per second)	3.0	3.1	2.9	3.1
1RJ Punts				
Number	702	735	516	504
Rate (per second)	.455	.474	.417	.412

Experiment A - First Fit Memory Allocation

Table 5.2

Detailed Comparison

the longer PFR service times seen.

5.1.4 Summary

This experiment indicates that some improvement in system performance might be realized by this modification, particularly if the CPU were heavily utilized.

5.2 Experiment B - Reduce Punts

Experiment B represents an attempt to reduce the number of abortive "roll in" attempts (punts) by 1RJ, thus reducing the number of times the job scheduler runs and the amount of CPU overhead associated with CPM.

The job scheduler selects those jobs to be resident at control points on a least cost basis, restricted by their ability to fit into the available allocatable central memory. Each time a job is selected for residency, the amount of available allocatable memory will be reduced by an amount equal to the memory requirement for that job. Any job currently at a control point and not selected for residency must be rolled out and its assigned memory freed. 1RJ's and 1SJ's are then invoked to perform the actual "roll in" or "roll out" as necessary. In order for a job to be rolled in successfully, there must be sufficient free memory when 1RJ is executed for that job. Two factors can prevent this. First, some other job already at a control point could have had more memory allocated since the job scheduler made its decisions and, secondly, insufficient "roll out's" may have occurred to free the necessary space. This second factor could be considered as an ordering problem. The job scheduler attempts a partial solution by combining "roll in's" and "roll out's" into "swaps" whenever possible, that is, 1SJ runs for a job at a control point and then invokes 1RJ to run for the same control point. No consideration, however, is given to the relative memory requirements of the two jobs involved in each case.

At two points 1RJ can abort (punt) an attempted "roll in" by issuing a SCH function requesting execution of the job scheduler and then a DPP function relinquishing the PPU (see Figure 4.5). Immediately after the LPP function, 1RJ checks to see if sufficient free memory exists to accomodate the job being "rolled in." If not, 1RJ punts; if so, it issues an RST function requesting allocation of the required memory. If the RST function were issued, 1RJ would still punt if some other RST "up" function should be serviced first leaving less free memory than required to hold the job. 1RJ punts cause the job scheduler to execute more often than might otherwise be necessary, resulting in higher CPU overhead for CPM.

5.2.1 Experiment Description

Experiment B attempts to reduce the number of punts by modifying the SPG for 1RJ. Figure 5.3 depicts how the first portion of the modified 1RJ SPG would be structured.

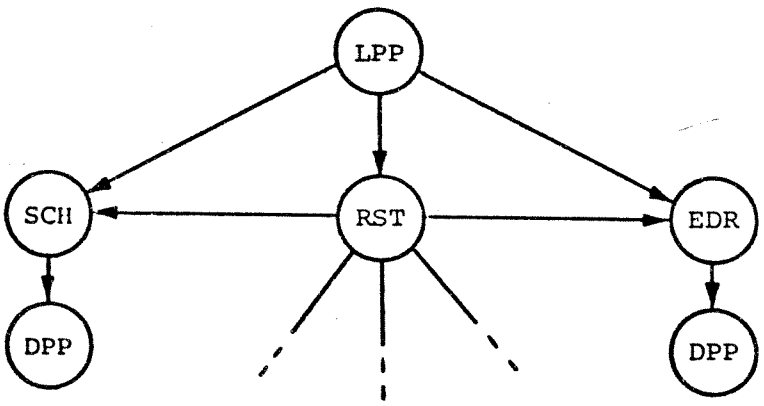

Figure 5.3

1RJ Modified

The subroutines attached to the LPP and RST nodes (section 4.2.2.6) are modified so that the first traversal of 1RJ never results in the SCH function's being issued. Instead, if at either of the two points insufficient free memory exists, 1RJ enters a delayed request for itself. An arbitrary delay of 50 milleseconds was used. During the second traversal, insufficient free space would result in a punt as before.

5.2.2 Comparison of Results

Table 5.3 compares various measures of the modified and unmodified model. Two of the goals of this experiment were realized. The number of punts were reduced and the job scheduler ran less often.

MEASURE	SIM 1	EXP. B-1	SIM 2	EXP. B-2
Elapsed Time (seconds)	1541.406	1584.183	1238.074	1233.603
Utilization				
CPU				
Control Points	.3959	.3852	.6788	.6812
CPM	.2620	.2522	.2506	.2512
PPU	.9955	.9941	.8772	.8773
Central Memory	.90	.91	.91	.92
Disk Channel Requests per second	14.56	14.62	16.77	16.71
Number of Executions				
Job Scheduler	4629	4508	3660	3633
1SJ	3589	3550	2487	2491
1RJ	4294	4384	2999	3045
Number of Punts	792	414	516	278

Experiment B - Reduce Punts

Table 5.3

Comparison of Performance Measures