TABLE 3.2. CALCULATION OF ELAPSED TIME

-Sub-path- Num	Nodes	Computation	Sub-total
1.1	Interpret cmd	+ 2	
		+ p(conti) * 1	
	Send message	+ p(conti) * 5	
		+ p(conti) * 1	
	Recover info	+ p(conti) * 2	2 + p(conti) * 9
1.2		+ p(new) * 1	
	Parse	+ p(new) * 50	
		+ p(new) * 1	
	Send message	+ p(new) * 5	
		+ p(new) * 1	
	Validate	+ p(new) * 250	
		+ p(new) * 1	p(new) * 309
2.1	Process request:		
	Locate data	+ p(new) * (45 + 36 * #reads)	
		+ p(new) * 1	
	Format screen	+ p(new) * 5	p(new) * (51 + 36 * #reads)
1.2		+ N * 1	N
2.1	Get data:		
3.1	Get record:		
	Check memory	+ N * 2	
		+ N * 0	2N
3.2	Dummy	+ N * 0	
		+ N * p(not) * 1	
	Allocate	+ N * p(not) * 5	
		+ N * p(not) * 1	
	Read	+ N * p(not) * 30	
		+ N * 1	
	Set pointer	+ N * 1	2N + (N * p(not) * 37) = 35.3N when p(not) = .9
2.1		+ N * 1	
	Translate fields	+ N * 5	
		+ N * 1	
	Move data	+ N * 1	8N
1.2		+ 1	
	Save information	+ 2	
		+ 1	
	Write to screen	+ 20	24

TOTAL RESPONSE TIME = 26 + p(conti) * 9
+ p(new) * (360 + 36 * #reads) + 46.3 N

TABLE 3.3. EXAMPLES WITH DATA DEPENDENCY

Scenarios	Specifications p(conti)	p(new)	reads	N	Average Response Time
1. New request 10 qualify	0	1	6	10	1065 ms.
2. Subsequent screens 10 qualify	1	0	0	10	498 ms.
3. Subsequent screens 5 qualify	1	0	0	5	267 ms.
4. New request & 2 subsequent 25 qualify	2/3	1/3	6	25	1830 ms. total 610 ms. average
5. New request* 10 qualify	0	1	6	10	1065 ms.

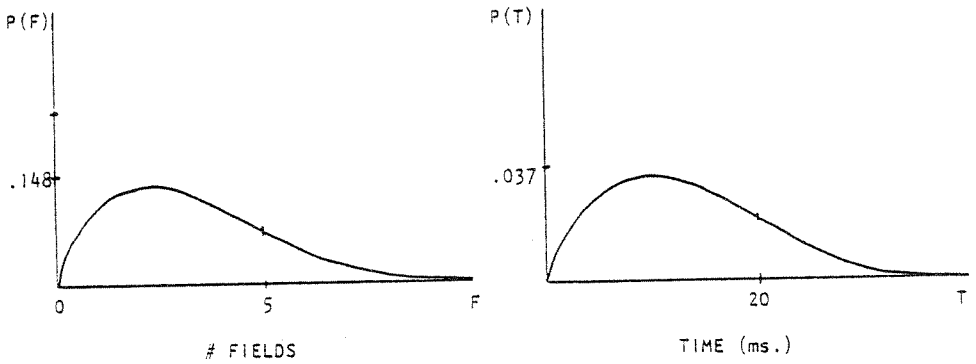
* Data dependent distribution for "Write to screen"

Next, suppose that the number of programs that qualify is 25. If only 10 will fit on a screen, three queries will be needed, a new one and two continuations. This is scenario 4 in Table 3.3. The total response time is actually the sum of that of the first 3 scenarios.

The performance goals are not included in the example. Appropriate values are 2 seconds if less than 10 qualify and 6 seconds for up to 100 qualifying. The performance goal is met in both cases.

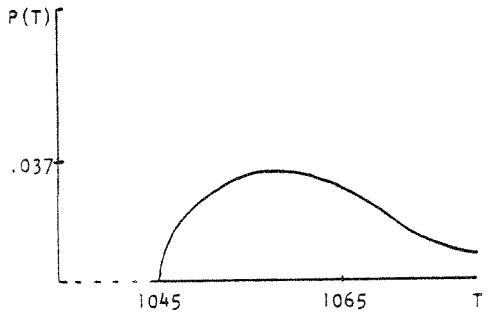
It is often possible to characterize data dependent performance specifications by using random variables. Suppose that the time for "write to screen" in Figure 2.5 is not constant, but depends on the number of fields desired and that this number can be approximated by a hypoexponential distribution with mean 5 and variance 50, as shown in Figure 3.2.

The mean, variance, and probability density function (pdf) are specified for each random variable. The algorithm in Table 2.6 is used to compute the average response time for the graph. The specified mean is substituted for the random variable in this computation. The variance of the response time is computed using the



(a) Distribution of number of fields desired

(b) Processing time for "Write to screen"



(c) Response time for Scenario 5

FIGURE 3.2. DATA DEPENDENT PROBABILITY DENSITY FUNCTIONS

TABLE 3.4. CALCULATION OF THE VARIANCE OF THE RESPONSE TIME

Origin Node Type	Calculation
Basic node	Add the variance to the current variance.
Collapsed node	Determine the variance of the associated sub-graph. Add it to the current variance.
Repetition node	1. Add E(X)**2 * Var N to the current variance where E(X) is the average execution time of one pass through the loop; that is, total time for the loop / E(N), the expected value of N. Var N is the variance of the loop repetition factor. 2. For each node i in the loop, add the product, E(N) * Var i, to the current variance.
Or-node	1. For every combination of 2 paths, i and j, out of the node, add: P(i) * P(j) * (E(i) - E(j)) ** 2 to the current variance. P(i) and P(j) are the probabilities of taking the respective paths. E(i) and E(j) are the expected execution times of the paths beginning at the or-node and ending at either a terminal node or where the paths i and j join. 2. Add to the current variance: P(m) * Var k for each node k on each sub-path m out of the or-node.

algorithm in Table 3.4. It is an enhanced version of the algorithm proposed by Kelly [KEL74], that handles the graphical representation of hierarchical structures, additional types of nodes and arcs, and control mechanisms. The order of evaluation and manipulation of loop repetition factors and execution probabilities are also different.

Kelly also defined a procedure for obtaining the probability density function (pdf) for response time. It is a compound distribution derived by taking the pdf's of the nodes of a graph and combining them according to rules based on several standard graph structures. His procedure is to take the transform functions of each pdf, combine them, then invert the resulting transform to obtain the desired pdf. He concluded that this computation was too complex to be used for non-trivial graphs. Some special cases exist, however, that occur frequently in software systems, and that have computationally tractable solutions for probability density functions.

Consider the previous example where the time for "write to screen" is changed to be a hypoexponential distribution with mean 5 and variance 50 as in Figure 3.2a. The processing time is 4 ms. per field, therefore the distribution for the processing time is also hypoexponential with mean 20 and variance 200, as shown in Figure 3.2b. The other specifications are shown in Table 3.3, scenario 5. The average response time remains the same as in scenario 1, 1065 ms. The variance of the response time is 200 since the variance is zero for all other nodes in the graph. The value of the variance is the same as that for "write to screen", but it is less significant with respect to the mean of the response time.

The response time pdf can also be determined trivially. It is the result of the convolution of a hypoexponential random variable and the constant-valued random variables of the other nodes. This results in a scaled hypoexponential compound pdf as illustrated in Figure 3.2c. That is, the function is the same as that of "write to screen", but it is shifted on the x axis by the sum of the constants

associated with the other nodes.

A study of a number of software systems with data dependent execution characteristics produced the following observations:

1. When the specifications for several components depend on one particular data object, the pdf's for each component are homogeneous. Only the mean and variance differ.
2. Constant, normal, exponential, and hypoexponential pdf's are usually representative.
3. The response time pdf is usually desired in order to get a general idea of the shape of the curve as related to the data objects and to determine the expected response time for a specific quantile. (For example, 90% of the transactions should have a response time less than or equal to 6 seconds.)

These observations suggest the following strategy for obtaining an approximation of the response time pdf. The hypoexponential pdf is actually a sum of two or more exponential random variables. Therefore, when a hypoexponential random variable is convolved with other hypoexponential or exponential random variables, the resulting compound pdf is also hypoexponential. A compound pdf that includes constant branching probabilities or repetition factors is a scaled pdf as before.

Thus, the compound pdf for the response time can easily be determined when constant, exponential, and hypoexponential random variables are summed. When a normal pdf is needed, it can be approximated by a hypoexponential pdf (the sum of a "large" number of exponential random variables) to simplify the computations. Note that when a "large" number of exponential and/or hypoexponential random variables are convolved, the result approaches the normal pdf.

Another approximation can be used to further simplify computation of the compound pdf and to eliminate the rather complex calculation of the variance. The normal and hypoexponential pdf's

Figure 3.3 shows a central server model of a host system with existing work reflected by job type 1. Figure 3.4 shows the execution graph for proposed software. The specifications for the estimated CPU time and number of disk I/O's are shown for each functional component of the graph.

First, the total CPU time and number of disk I/O's are calculated using the standard graph evaluation algorithm in Table 2.6. Then, the branching probabilities out of node BP are calculated from the total number of disk I/O's. Since a type 2 job completes after the last I/O, the branching probability from node BP to queue TERM is the reciprocal of the total I/O's (.1 in this example). Since the branching probabilities must sum to one, the branching probability from node BP to the CPU is calculated from that of node BP to TERM (.9 in this example).

The mean CPU service time is derived by dividing the total CPU time by the total number of disk I/O's. The TERM and DISK service rates are assumed to be the same as for job type 1. Otherwise, the changes are included in the environment specifications. The model parameters for the host with the existing work (type 1) and the new software (type 2) are shown in Table 3.5.

The model solution yields revised response times for the existing work and a response time estimate for the new software. Resource utilizations, throughputs, queue lengths, and other performance indicators are also derived from the model solution.

The analysis of the effect of multiple users of the software is easily handled in this framework. The queueing network model is developed as described. Increasing the degree of multiprogramming for the job of interest then produces revised performance metrics for that number of users. The model can be solved for a range of values for the degree of multiprogramming to determine how many users can be supported before the response time degrades.

The extension to more complex host environments is a straightforward expansion of the queueing network model. The

These findings support the use of a simple approximation technique. It provides good, inexpensive initial feedback on the response time distribution. The extension for analyzing the effects of other work competing for host system resources is described next.

3.2 Competitive Effects

It was assumed in the basic performance prediction methodology that the software system would execute on a dedicated computer system. When other work is processed on the host computer, external competitive effects are introduced. Software performance may be satisfactory on a dedicated system but queueing delays for shared resources such as the CPU and disks can cause significant performance degradations that affect the new software, the existing work, or both.

These external competitive effects are straightforwardly represented in queueing network models of computer systems. The software environment specifications must include a model of the host computer system. Measurements of the existing work on the host are used to derive the parameters for the model. The model is then validated and calibrated until it is an accurate representation of the existing computer system.

Another job type is then added to the model to represent the new software [BAS75]. The resulting model is the elementary model. Recall that in the basic methodology the performance specifications for the software were analyzed and an execution graph was generated. The queueing network model parameters for the new job can easily be derived from this software execution graph [IRA79]. This procedure is best illustrated by an example.

can be approximated by an R-stage Erlangian pdf. It is a sum of R exponential random variables where the means of all the exponential random variables are equal to 1/R times the mean of the Erlangian. When R is one, it is an exponential pdf. R can be selected to yield a curve of the desired shape. From R and the specified mean, the variance of the pdf can easily be computed:

$$\text{Var} = \frac{1}{R\ (\text{Mean})^2}$$

When an R-stage Erlangian is convolved with a constant, the compound pdf is still an R-stage Erlangian that is scaled appropriately. When several R-stage Erlangian random variables are summed the resulting compound pdf is hypoexponential, since the means of the individual R-stage Erlangian random variables are not equal. However, it can be approximated by an R-stage Erlangian with the new R equal to the sum of the R's associated with the individual random variables that were summed. The resulting compound pdf is a close approximation to the pdf obtained from the procedure defined by Kelly. The variance, as computed from the resulting mean and the value of R, will not be exact. However, it is usually acceptable since the specifications on which the analysis is based are rarely precise enough to warrant the extra work required to obtain the exact variance.

These computations for the data dependent mean, variance, and distribution of response time apply to software execution in a limited environment. Competitive effects and other external factors affect the response time characteristics. Lazowska and Sevcik have shown that in the presence of competitive effects, the response time distribution is asymtotically normal [LAZ78]. He also makes the observation that host system bottlenecks can completely dominate the response time distribution thus causing a drastic change to a distribution computed as above.

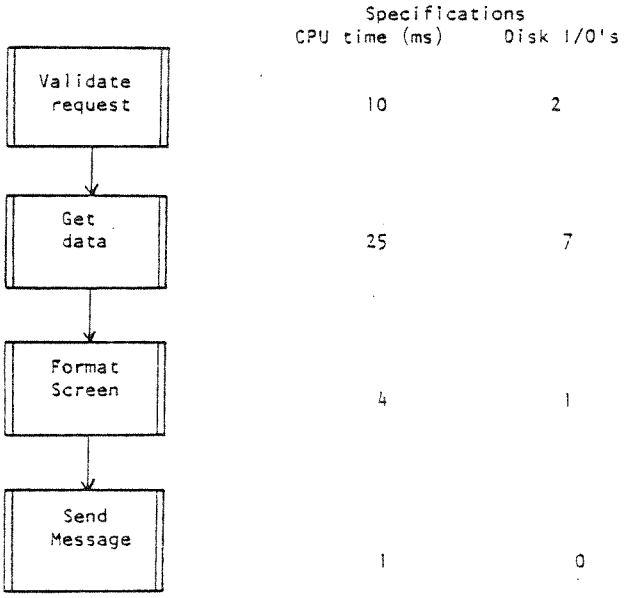


FIGURE 3.4. NEW SOFTWARE EXECUTION GRAPH

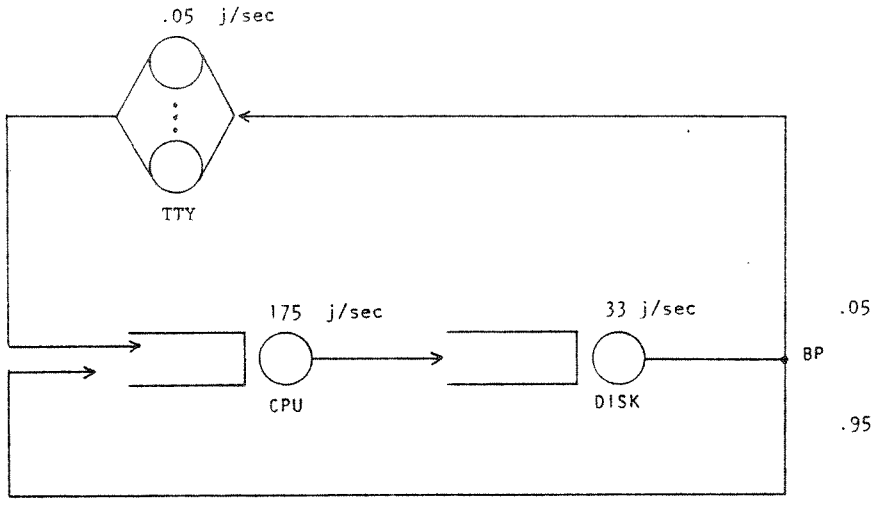


FIGURE 3.3. COMPUTER SYSTEM MODEL

TABLE 3.5. MODEL PARAMETERS WITH NEW JOB

a. Service rates (job/second)

Node	Job Type* 1	2
CPU	175	250
DISK	33	33
TTY	.05	.05

*Type 1 = existing work
Type 2 = new job

b. Branching probabilities for Type 1

From: \ To:	CPU	DISK	TTY	BP
CPU	0	1	0	0
DISK	0	0	0	1
TTY	1	0	0	0
BP	.95	0	.05	0

c. Branching probabilities for Type 2

From: \ To:	CPU	DISK	TTY	BP
CPU	0	1	0	0
DISK	0	0	0	1
TTY	1	0	0	0
BP	.9	0	.1	0

analysis techniques previously developed can be used to obtain the model parameters from more complex software execution graphs.

The additional performance metrics from the elementary model enable the consideration of other performance goals such as bounds on resource utilization and bounds on the degradation of the response time (or throughput) of existing work. They also help to determine bottlenecks: performance limiting devices (resources) are identified by high utilizations. The software design can then be analyzed with

respect to requirements for that resource. Alternate design and implementation strategies that potentially reduce those resource requirements can be developed and evaluated.

For example, suppose the bottleneck device is the CPU. The execution graphs are analyzed and a histogram of the CPU requirement of each component is produced. Those components with the greatest total CPU requirements are studied and alternatives are investigated. For each feasible alternative, the specifications are revised, the graphs are re-evaluated to produce revised model parameters, and the elementary model is re-solved.

Thus, the effect of design alternatives is easily evaluated with quantitative results produced for each. The appropriate design is then selected with consideration of the ease of implementation and the performance benefits. Since the analysis begins prior to implementation, extensive coding modifications to enhance performance are avoided.

Of course, the evaluation may indicate that the performance will be unsatisfactory even after design improvements are incorporated. In this case, a modification to the host configuration is necessary. Various reconfigurations and equipment upgrades are easily evaluated by making appropriate modifications to the queueing network model.

The technique of uniting the graphical representation of software execution and the queueing network model technology supports many additional types of software design evaluation. It is a natural combination whose potential is just beginning to be realized. The queueing network models preserve the criteria of quick, interactive evaluation of designs while adding tremendous modeling power. Extensions to the models to facilitate the analysis of memory contention, a typical and complicated external influence on the software design, are described next.

3.3 Memory Analysis

The effects of the contention for executable non-paged memory on response time are two-fold:

1. Jobs are delayed by a scheduler upon arrival to the system until sufficient memory is available to begin
2. During execution, jobs may be swapped out of memory, when long waits for resources are encountered, to allow other jobs that have sufficient resources to use the memory.

Consider the example in Figure 3.4. A request is entered at a terminal and sent to the host computer system. It then enters a memory wait queue where it remains until sufficient memory is available for the user application program, APP, to be loaded. Once available, APP is loaded and begins execution. APP is a driver that calls each of the components indicated in the graph. There is minimal APP processing between each of the calls. If the elapsed execution time of any of the called components exceeds a predetermined swap threshold, APP is swapped out. It is swapped in again when the component is complete; further user processing is delayed until the swap in is complete. The swap out can execute concurrently with the component processing, but the swap in cannot.

Figure 3.5 illustrates the revised execution graph with nodes representing the swapping activity included. A node is added for each component whose estimated elapsed time exceeds the predefined swap threshhold. These nodes are identified by applying the following algorithm to compute the estimated elapsed time of each component:

1. The elementary model is run to obtain the average queue time for the CPU and DISK.
2. The estimated elapsed time for each component is computed by weighting its CPU and DISK requirements by the expected queue time.

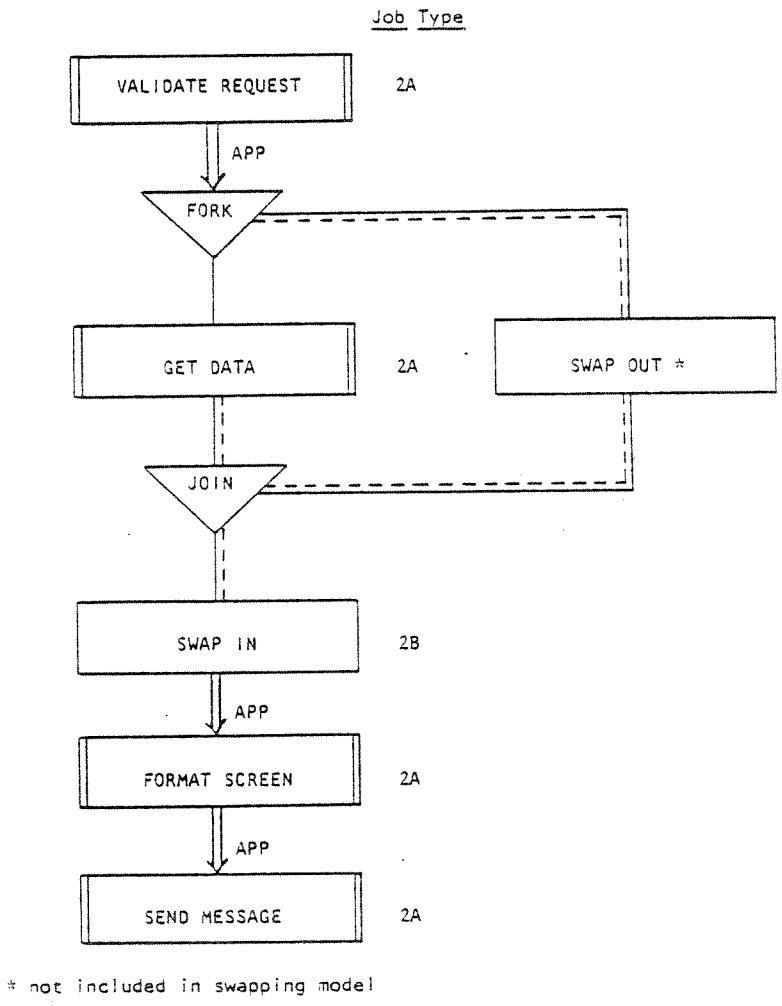


FIGURE 3.5. EXECUTION GRAPH WITH SWAPPING ACTIVITY

Special fork and join nodes have been added to the graph to delineate the concurrent component and swap out processing. The concurrent processing can actually proceed in one of three ways:

1. Processing for one completes before the other begins execution
2. The processing of both proceeds in parallel
3. A combination of these where only part of the processing is parallel.

The major influences of memory contention on response time are analyzed first. It is a best case analysis; extensions for the analysis of the more complicated situations are discussed later. The assumptions for the best case are as follows:

1. There is complete overlap of concurrent processing
2. There is no wait for memory allocation prior to swap in
3. The swap out processing has minimal impact on the host system performance.

The first assumption implies that the swap out processing will complete before the component processing, so it will not be necessary to wait for the swap out to complete before the swap in can be started. The second implies that sufficient memory is always available to swap in the user program. The third implies that the swap out processing can be omitted from the model with little affect on the resulting response time.

None of these assumptions are essential for modeling swapping activity. They are included to simplify this discussion and the model solution. Since it is simple, this swapping model is evaluated first. If the best case results are unsatisfactory, modifications are necessary regardless of the results of the more complicated models. The model variations to preclude these assumptions are discussed later.

The best case swapping model uses dependent job typing to reflect the time required for swap ins [CHA77]. The user application program, APP, is loaded and begins execution as job type 2A. It remains type 2A throughout the component and APP processing. It changes to type 2B when a swap in begins and back to type 2A when the swap is complete. A typical scenario will have several such changes.

The queueing network model is shown in Figure 3.6. The model parameters are computed independently for each type, using the algorithm in Table 2.6. That is, the total CPU time and total number of I/O's for type 2A determine the CPU processing rate for type 2A. The CPU rate for a type 2B, swap in, is specified with the environmental specifications. It is the reciprocal of the average CPU time overhead to initiate the swap in. The service rate for the SWAP device is calculated from the size of the program to be swapped and the environmental specifications for the hardware swapping device (the seek, latency, and transfer time).

The branching probabilities are slightly more complicated than those in the elementary model; these must also include the probability that a job changes type. The job changes type only at the BP node. The number of times a job reaches the BP is the sum of the number of I/O's and swaps, TOTBP. One of these is the job completion, so the branching probability for type 2A from the BP to the TERM is the reciprocal of this total. The branching probability for type 2A from the BP to the CPU and remaining type 2A is the ratio of the number of I/O's minus one to the TOTBP. For type 2A from BP to the CPU and changing to type 2B it is the ratio of the number of swaps to the TOTBP. Type 2B jobs always change to type 2A and branch back to the CPU. These model parameters are shown in Figure 3.6.

The swapping model is solved and the performance metrics are obtained for each job type. The revised response time for the scenario is the sum of the response times for type 2A and 2B. The proportion of the response time due to swapping and its impact on the host system are easily obtained.

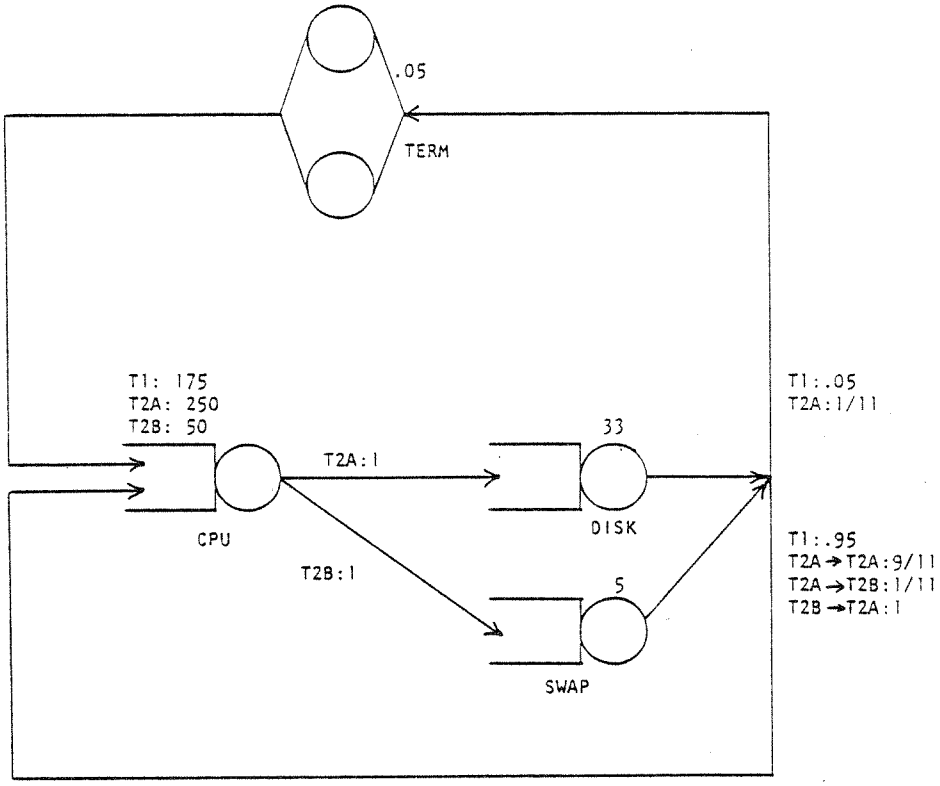


FIGURE 3.6. QUEUEING NETWORK MODEL FOR SWAPPING

The analysis of the impact of the wait for memory allocation is based on the single job-type queueing network model proposed by Brown, et.al. [BRO77]. This model was extended by Keller to handle multiple job types [IRA79]. The additional memory model parameters consist of:

1. A discrete probability distribution of memory requirements for each job type
2. The total memory available for allocation
3. The number of users of each type.

These parameters are used to determine the probability that the various combinations of job mixes will occur. The swapping model is then run with each of the feasible job mixes. The metrics obtained from the swapping model results are then weighted by the probability that the corresponding job mix will occur to obtain the expected performance metrics.

The additional memory model parameters are essentially derived from the execution graphs. The discrete probability distribution,

$P_j(i)$ where $P_j(i)$ is the probability that a type j job will have a memory requirement of i units

is determined from the swapping model results and the execution graphs. The estimated elapsed time algorithm is used to compute the elapsed time of each component in the graph. The ratio of the elapsed time of a component to the total elapsed time of the scenario is the probability that the component is executing; its size is contained in the functional component specifications. The available memory and number of users are contained in the environmental specifications.

Recall that some simplifying assumptions were made earlier. One was the best case assumption that there is no wait for memory allocation prior to each swap in node. This is, perhaps, an

unreasonable assumption for some host systems. A variation of the memory model that precludes this assumption is derived by adjusting the branching probabilities from node BP for type 2B jobs. In the revised model, type 2B jobs change to type 2A jobs and branch to the TERM. These jobs are immediately ready; therefore the TERM service rate is adjusted to reflect the "mean equivalent user think time." It is a weighted average deduced from the number of swap outs and the original think time.

The other simplifying assumptions are concerned with the inherent internal concurrency of swapping activity and actual processing. The modeling of internal concurrent processing is presented next. The techniques described there can be applied to the swapping model to reflect the competitive effects of the concurrent swap out processing.

CHAPTER 4
CONCURRENT PROCESSING

4.0 Overview

The previous chapter is primarily an analysis of the interaction between the software and the environment. Bottlenecks are corrected by adjusting the design, the host environment, or both, so that they are symbionic. Software systems are rarely specifically designed to accommodate the variability of environmental factors.

This chapter contains techniques for the modeling and analysis of aspects of software designs that are more directly related to the resulting performance. Internal concurrency and the synchronization of processing are discussed first. Internal concurrency results when software processing forks and multiple processes execute in parallel. Synchronization occurs when various parallel processes must join (complete) before processing can continue.

The swapping scenario in Figure 3.5 is an example. The processing forks at the point that a swap out begins. The swap out executes in parallel with the component processing. The processing joins again prior to the execution of the swap in. Both parallel processes must complete before the swap in can begin.

Internal concurrency is incorporated into software systems to overlap processing and thus shorten response times. Increasing concurrency also increases the competition for shared resources which increases queue time. It is desirable to model concurrency explicitly to quantify the extent and effectiveness of the

concurrency since various design strategies will exhibit very different performance characteristics.

The second effect of concurrency on software designs that is modeled is the effect of blocking by mutual exclusion. This commonly occurs when highly used portions of data bases such as keys and pointers are held in exclusion to be updated. All other data base processing involving those data elements is then blocked until the update is complete.

Queueing network models are used as the vehicle for the analysis of both of these software design characteristcs. This is necessary for compatibility with the environmental analysis and to preserve the requirement that the design analysis be suitable for interactive evaluation.

The representation of synchronization and blocking in queueing network models is a difficult problem. Analytical solutions for models including blocking behavior are available only for special cases [KON76,BOX79]. Lam has shown that product form solutions can be obtained for networks with population size constraints [LAM77] and Zahorjan has given a general convolution algorithm that includes Lam-type networks [ZAH79b]. None of these results is satisfacotry for modeling general blocking behavior.

Towsley, et.al., formulated a model of concurrent processing for representing the overlap of CPU and I/O processing within a job [TOW78]. He collapses a network into a two queue model then solves the corresponding Markov model to obtain exact analytical results. Brown, et.al., [BRO75] and Zahorjan [ZAH79a] used an iterative solution technique for the modeling of multiple disk per controller systems where holding of the controller for transfer may inhibit the initiation of seeks.

The synchronization and blocking models presented next use the iterative solution approach. The approximation techniques can be used to obtain results quickly. Once again, this approximation is sufficient for design level evaluation when the specification data is

imprecise. The extra effort required for an exact solution is usually not justified. An exact solution technique should, perhaps, be used at later development stages for software systems when more precise performance metrics are crucial.

4.1 Internal Concurrency and Synchronization

The modeling of internal concurrency and synchronizatiion within a software design begins with a specification and resulting graphical representation. The functional component specifications include a description of the linkage between components. A link, FORK, is specified for each concurrent process that can begin execution when a functional component completes. Each FORK begins another concurrent chain. Similarly, a JOIN link is specified from each component that terminates a concurrent chain.

Figure 4.1 depicts a software system with internal concurrency and synchronization inherent in the design: the Harris scenario. A remote terminal user sends a request for data which causes asynchronous (concurrent) processing to begin to satisfy the request. The user is not required to block to wait for a reply; any user processing can be done at this time. When the user desires the reply from the asynchronous process, the request reply is sent. Both of the concurrent chains must complete before the results are returned to the user.

The actual user processing that occurs between the "send request" and the "request reply" is unpredictable. This is another example of data dependency that must be resolved by introducing a conditional performance goal. To simplify this discussion, the assumption is made that no additional processing will occur and the model is evaluated to determine the response time conditioned on this

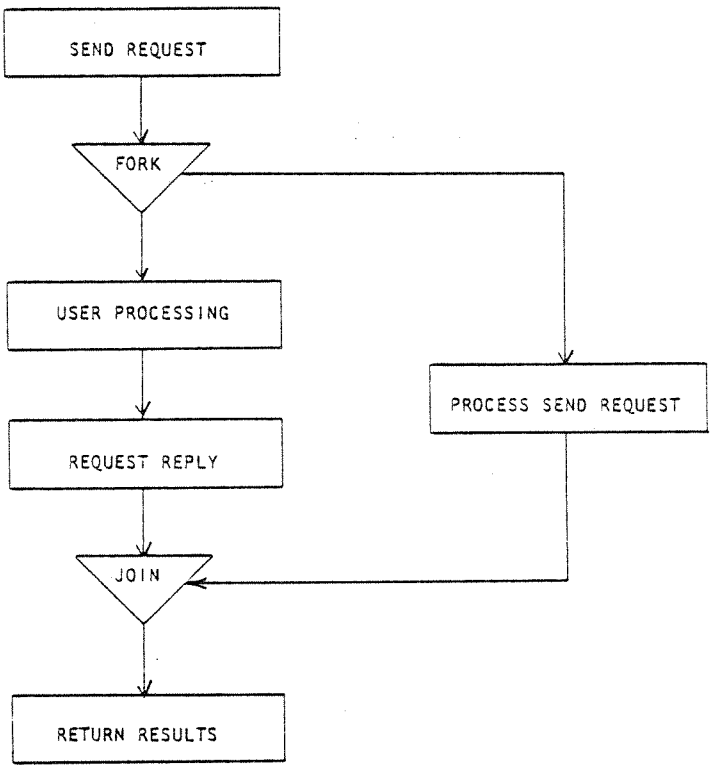


FIGURE 4.1. HARRIS SCENARIO

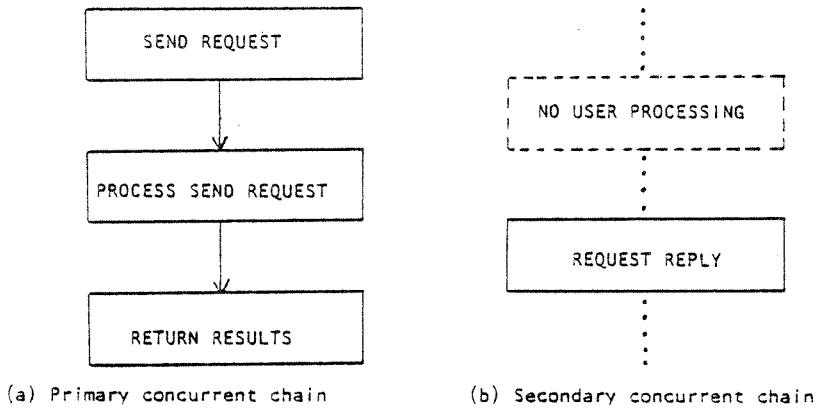


FIGURE 4.2. PARTITIONED HARRIS SCENARIO

event. The evaluation of the model with additional processing included is easily derived from the one presented.

The response time effects are two-fold:

1. Additional competition for shared resources is introduced thus potentially increasing the elapsed time of each concurrent chain.
2. The concurrent chain with the longest elapsed time determines the overall response time.

These effects are quantified by first partitioning the graph into concurrent chains. The primary chain is that with the longest expected elapsed time. Other concurrent chains are secondary. Figure 4.2 illustrates the partitioning of the Harris scenario.

The primary concurrent chain is usually easily identified from the specifications and execution graphs. Generally, one chain clearly dominates the others in terms of resource requirements. If not, it is initially determined by formulating an elementary model with existing competitive work and one additional job type representing one of the concurrent chains. An elementary model is solved for each possible chain and the chain with the longest elapsed time is selected.

Next, queueing network model parameters are derived for each chain independent of the others. Each chain is a distinct job type in the synchronization model shown in Figure 4.3. The primary chain cycles through the network as before. The secondary chains begin at the wait queue, WQ. The length of time they remain there is, on the average, equal to the elapsed execution time of the primary chain from the beginning of execution until the FORK. They enter the SYSTEM and cycle through it (competing with the primary chain for resources) until their actual processing is complete, then return to the WQ. They then remain at the WQ for the remainder of the elapsed time of the primary concurrent chain.

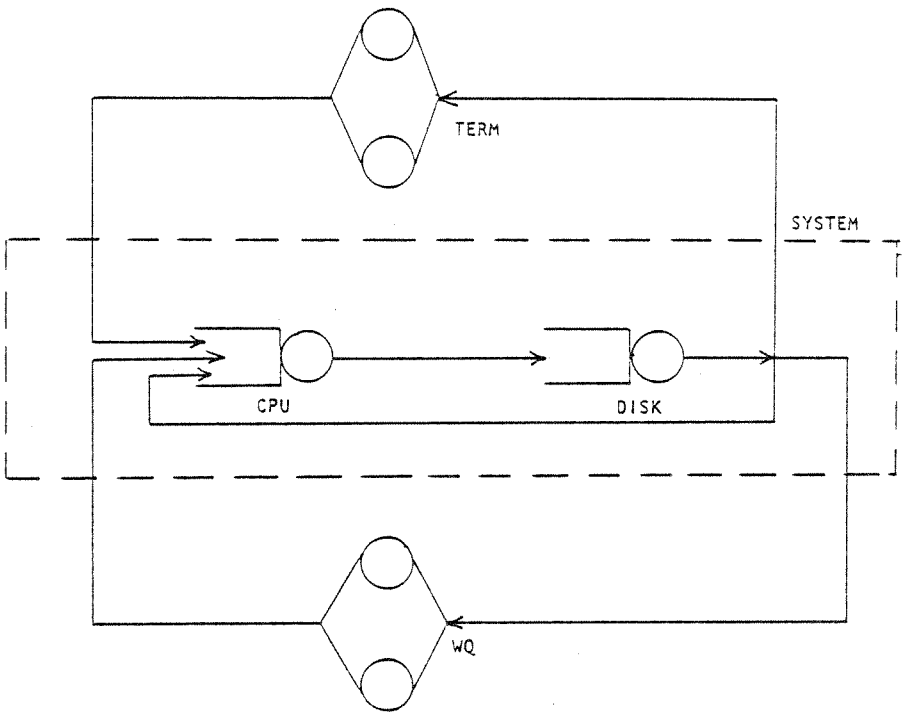


FIGURE 4.3. SYNCHRONIZATION MODEL

The degree of multiprogramming for each of these chains is equal. The WQ is modeled as an infinite server since the time spent there does not directly depend on the number of other jobs of that type at the WQ. (The indirect effect is the effect on the total elapsed time of the primary chain due to the competitive effects of the secondary chains). The WQ service time is first estimated from the results of the elementary model with the primary chain. This will be a lower bound for the elapsed time of the primary chain since the competitive effects are minimal in that model. The WQ time is obtained by computing the estimated elapsed time of each secondary chain using the queue time from the elementary model and the elapsed time algorithm in Chapter 3. Subtracting this figure from the response time of the primary chain gives the estimated total time to be spent at the WQ for each secondary chain.

The resulting synchronization model is then solved. The response time of each chain should be equal (or within some tolerance, say 10%). If so, the synchronization model is an adequate representation of the concurrency and synchronization and the design and configuration analysis proceed as in the elementary model. This will occur when the internal competitive effects are minimal or when the response time is dominated by the other, serial processing in the scenario.

When a particular resource is saturated (utilization >90%) and one concurrent chain requests service from that resource many more times than the other chains, the original estimate of the elapsed time of a chain is not adequate. In this situation, an evaluation of alternate designs or host system configuration upgrades should be pursued rather than a more accurate synchronization model. The competition for the resource is increased by the concurrency, which degrades the response time. It may be that serial processing would result in better overall response time. Nevertheless, a procedure is described next for iteratively adjusting the WQ service time to achieve a more accurate synchronization model.

The primary chain usually requests service from the saturated resource more than the others. This is because, by definition, the primary chain has the longest elapsed time, and queueing for bottleneck devices usually dominates the response time. Note that it may be determined at this point that an incorrect selection of the primary chain was made initially; if so, appropriate modifications are made before proceeding.

Recall that the elapsed time of the primary chain is greater than that of the secondary. A new WQ service time is computed from the previous model results using the elapsed time algorithm. The result is a higher WQ service time which, when used to re-solve the model, reduces the competitive effects and results in a lower response time for the primary chain. The response times for the concurrent chains are more nearly equal in the revised solution. This adjustment is repeated until the resulting response times are within an acceptable bound.

If the WQ service time is inadvertently reduced too much, the resulting model solution will result in a response time for the primary chain that is lower than that of the secondary chain. At that point an upper and lower bound for the actual response time are known. It is likely that the range of response times is small enough that one of the previous solutions is within an acceptable bound of the response times. In fact, the first synchronization model solution is usually adequate.

4.2 Mutual Exclusion and Blocking

There are two primary effects of blocking on the performance of software:

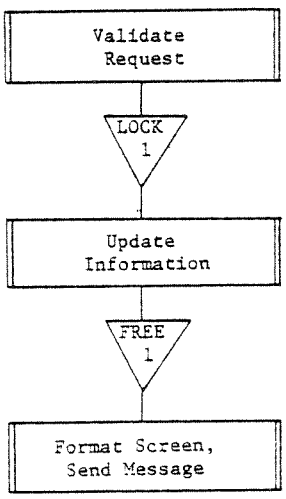
1. The response time is increased by the amount of time that a job must wait before acquiring exclusive access to an object
2. The processing time (and resource utilization) is reduced due to decreased competitive effects while jobs are waiting.

The interaction of the two and the resulting performance metrics are quantified by a queueing network model. It is best explained with an example.

Suppose that the blocking is due to a specific part of a data base index structure. Jobs process normally until exclusive access to this data object is required. At that time, if the object is free (not held by another job), the job obtains a lock and continues processing. If another job has the data object locked, the job desiring the lock must wait until it has been freed. The data object is freed by the job holding the lock when it no longer needs exclusive access.

The software execution graph in Figure 4.4 illustrates this type of processing. The elementary queueing network model is augmented in Figure 4.5 to represent this added complication. A node, WQ, is added to the model to represent a wait queue. When the lock is desired by a job, it proceeds either to the CPU (if no other job holds the lock) or to the wait queue (if the object is already locked).

Dependent job typing is used to reflect locked and unlocked stages. Each stage in the processing is considered a distinct job type as shown in Figure 4.4. The CPU service rate for each job type is once again derived from the graphs in a straightforward manner. The branching probabilities are more complicated. They must reflect the probability that the job changes type as well as the probability



SPECIFICATIONS

Job Type	CPU time (ms)	DISK I/O	Estimated Elapsed Time (ms)
1	10	2	70
2	25	7	265
3	5	1	35
			370

FIGURE 4.4. GRAPH A WITH LOCKING

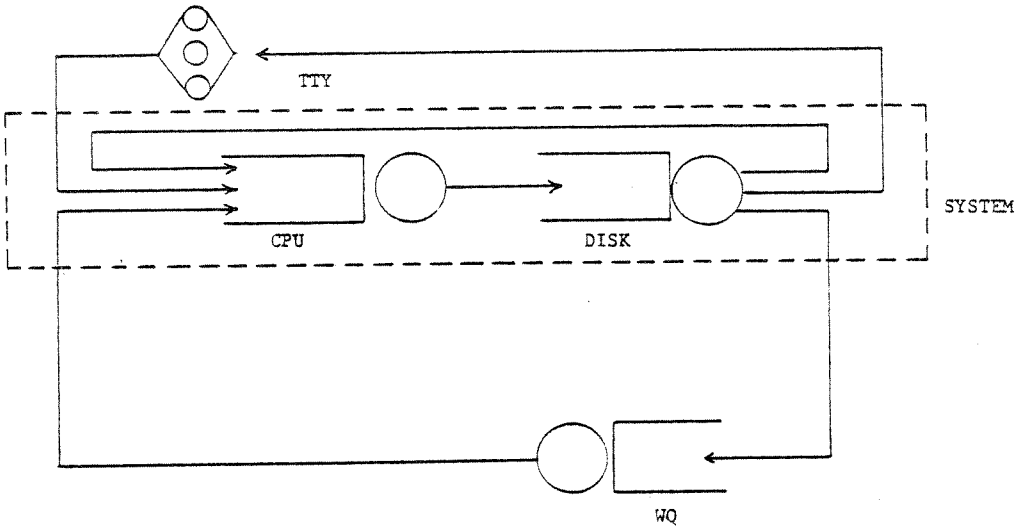


FIGURE 4.5. QUEUEING NETWORK MODEL FOR LOCKING SOFTWARE

that it branches to a particular node.

A job enters the CPU from the TERM node as a type 1 job (with probability 1). A type 1 job branches from the CPU to the DISK as a type 1 job (probability 1) since it does not access the potentially locked data object in this stage of processing. A type 1 job completes processing and changes to a type 2 job after two I/O's. If no other type 2 jobs are in the system, it can branch back to the CPU, otherwise it must branch to the wait queue and stay there until the other type 2 job has changed to type 3.

The mean service time of the wait queue is the mean residual life of a type 2 job. It is derived from the elapsed processing time of a type 2 job, which depends on the number of other type 1,2, and 3 jobs in the system. An iterative solution technique is used to derive the elapsed time. It is described in detail later. It is assumed in this example that the distribution of the elapsed time is exponential so that the mean residual life of a type 2 job is equal to the mean elapsed time. Other distributions that have a computable mean residual life can also be easily handled [DRA67].

The probability that a type 1 job branches back to the CPU as a type 2 job (obtains the lock) is the probability that all other jobs in the SYSTEM are either type 1 or type 3 jobs. The formula for computing this probability is:

$$P_{(1\text{-DISK},2\text{-CPU})} = \frac{ET(1) + ET(3)}{ET(TOT)^{N-1}}$$

where: ET(i) is the elapsed time for job type i

ET(TOT) is the total elapsed time:

ET(1) + ET(2) + ET(3)

N is the total number of jobs

in the SYSTEM (CPU and DISK nodes).

The probability that a type 1 job changes to a type 2 job and branches to the wait queue is:

$$P_{(1\text{-DISK},2\text{-WQ})} = 1 - P_{(1\text{-DISK},2\text{-CPU})}$$

The other model parameters are easily derived from the specifications and are shown in Table 4.2.

Since the above branching probability and the elapsed processing time depend on the state of the SYSTEM and thus are not known a priori, an iterative solution approach is used. First, the probability that a new type 2 job branches to the wait queue is set to 0 and the model is solved, This solution reflects the maximum concurrency in the SYSTEM and thus will give an upper bound for the elapsed processing time (excluding the WQ time). An upper bound is obtained for N, the number of jobs in the SYSTEM.

Next, the probability that a new type 2 job branches to the wait queue is set to 1. The mean service time of the wait queue is the mean elapsed time of a type 2 job running in isolation (the lower bound). The solution to this model gives a lower bound for the elapsed time for each job type (excluding WQ time) since the concurrency in the SYSTEM is minimized. It also gives a lower bound for N.

These two steps yield bounds for the model parameters that depend on the system state. The appropriate selection of parameter values depends on the impact of the blocking on the system. If the arrival rate of blocking jobs is high and/or the duration of the lock is long, the utilization of the wait queue will be high and there will be a type 2 job in the SYSTEM node most of the time. The solution to the model without the wait queue will have the mean queue length of type 2 jobs in the SYSTEM greater than one. The goal of the following model parameter selection is to gradually reduce this queue length until it is 1.

TABLE 4.2. MODEL PARAMETERS FOR LOCKING SOFTWARE

a. Service rates (jobs/second)

Node	Job Type 1	2	3
CPU	200	320	200
DISK	33	33	33
TTY	.05	—	—
WQ	—	3.77	—

b. Branching probabilities

From: Node Type		To: 1	CPU 1	CPU 2	CPU 3	WQ 2	DISK 1	DISK 2	DISK 3
TTY	1		1						
CPU	1						1		
	2							1	
	3								1
WQ	2			1					
	1		1/2	1/2P		1/2(1-P)			
DISK	2			6/7	1/7				
	3	1							

where $P = (105/370)^{N-1}$

and blank entries are all zero

Initially, the probability P (in Table 4.2) is computed using the average of the probabilities derived from the upper bound and lower bounds for elapsed times from the two previous solutions. Similarly, the average of the number of jobs in the system is used. The lower bound is used for the wait queue service time and the model is solved. If the mean queue length of type 2 jobs in the SYSTEM is still high, the service time of the wait queue is increased and the new model is solved, until the appropriate queue length is obtained.

Minor contention due to blocking is indicated by almost equal values for the upper and lower bounds for N and one or fewer type 2 jobs in the SYSTEM. In this case, the lower bounds for ET and N are used to compute the branching probabilities and for the wait queue service times. The model is solved iteratively with the wait queue service time gradually increased, using the results of the previous model solutions, until the mean queue length of type 2 jobs is less than one and the service rate obtained for the wait queue stabilizes.

The preceding model can be used when several execution graphs are to be evaluated. Distinct job types are used for each possible lock that can be held.

The models can be extended to include resource sharing combined with resource locking. This occurs when:

1. Only one job is allowed exclusive access to a particular resource.
2. Any number of jobs may share the resource on a non-exclusive basis.
3. Jobs desiring exclusive access to a resource must wait until no other job has the lock and no other job is sharing the resource.
4. Jobs desiring shared access must wait until no locks are held.

This case can be illustrated by two execution graphs: the locking graph previously described and the sharing graph in Figure 4.6.

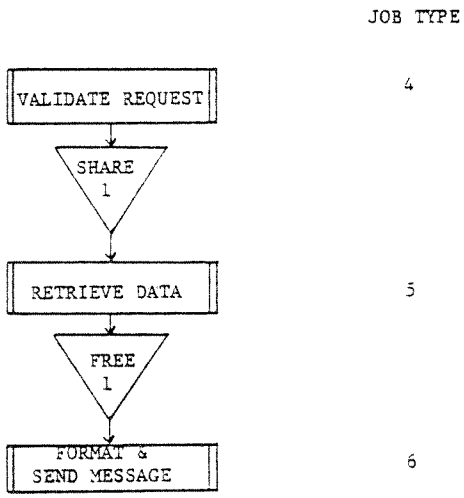


FIGURE 4.6. GRAPH B WITH SHARING

The queueing network model remains the same as in Figure 4.5 with most model parameters the same. The mean service time of the wait queue for type 5 (sharing) jobs will be the mean residual life of type 2 (locking) jobs. For type 2 jobs it is a weighted average of the mean residual life of type 2 and 5 jobs.

The branching probabilities will be slightly different. A type 4 job will change to a type 5 job and branch to the CPU (begin sharing the resource) if all other jobs in the system are not type 2. This is the probability previously calculated for type 1 jobs changing to type 2 and branching to the CPU. If one (or more) of the jobs are type 2 jobs, a type 4 job will change to type 5 and branch to the wait queue. This probability was also previously calculated.

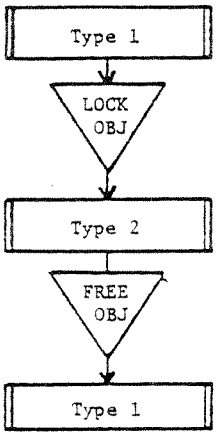
The probability that a type 1 job will change to a type 2 job and branch to the CPU will change in this model. It becomes the probability that there are no type 2 jobs in the system and there are no type 5 jobs in the system. That is,

$$P_{(1\text{-DISK},2\text{-CPU})} = \frac{ET(1) + ET(3)}{ET(TOTA)^{NA-1}} \quad \frac{ET(4) + ET(6)}{ET(TOTB)^{NB}}$$

A similar iterative technique is used to solve this model. Note that external competitive effects can be included in both of these models by adding job types to reflect the competing work.

The preceding model was used to analyze the ensuing software system to verify that it accurately represents blocking. The host computer system is an IBM 3032 running MVS. The workload consists of a transaction processing system (INTERCOMM), time-sharing jobs (SPF and TSO), and batch jobs. The software is a subset of an accounts payable system consisting of four types of transactions that require exclusive use of multiple index records within a file.

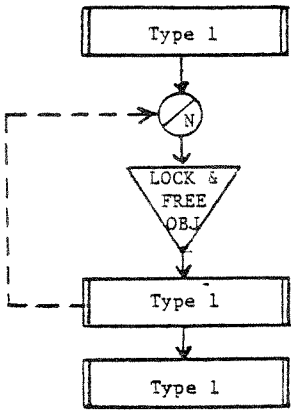
The transactions were analyzed to derive the execution graphs shown in Figure 4.7 One particular index record was isolated as the primary area of contention. It was chosen because of the long duration of the lock in LA and LB and the volume of transactions desiring access to the data object. Other secondary areas of contention are present but should not cause significant degradation in response time. Note that LD and LF require exclusive access to the index record, but once it is obtained it is held for an insignificant amount of time. Therefore, these transactions can be locked out by LA and LB but, since they are unlikely to lock out other jobs, they are always type 1 jobs. In this system index records can be shared (read) at any time regardless of their lock status.



SPECIFICATIONS

	LA: Volume = 477		LB: Volume = 42	
	CPU time	I/O's	CPU time	I/O's
Type 1	.007	1	.006	1
Type 2	.042	13	.045	12
Type 1	.031	2	.032	3

a. Transactions LA & LB



	LD: Volume = 77		LF: Volume = 72	
	CPU time	I/O's	CPU time	I/O's
Type 1	.04	5	.11	8
N	N = 7		N = 3	
Type 1	.024	8	.055	15
Type 1	.05	3	.094	3

b. Transactions LD & LF

FIGURE 4.7. L TRANSACTION EXECUTION GRAPHS

A model similar to that in Figure 4.5 is used. It is shown in Figure 4.8. The transaction processing system runs at the highest priority so the external competitive effects of the other (timesharing and batch) work are minimal and are not considered. The L transactions have the highest priority in the transaction processing system. This means that when several transactions are ready for processing, an L transaction will be selected. But there is no preemption. Therefore, an L transaction will wait on at most one of the other Intercomm transactions: the competitive transactions. This is represented by always having one competitive transaction in the SYSTEM.

L transactions are single-threaded. That is, there can only be one active job for each particular type of transaction in the SYSTEM at a time. Other jobs of the same type must wait at the SSQ. Once a job begins processing it is first loaded (at DLOAD). After the load, the L transactions cycle through the CPU, DISK, and WQ until completion. Competitive transactions use different disks, DISKC. The utilization of the channels is low, so they are not included in the model.

The results of each iteration of the model solution are shown in Table 4.3. The model is first solved with the probability that a job branches to the WQ set to 0. This reflects the maximum concurrency in the SYSTEM and gives an upper bound for the WQ service time. It also gives the minimum total response time for a transaction.

The model is then solved with the probability that a new type 2 job branches to the WQ set to 1. This gives a lower bound for the WQ service time and the maximum total response time.

In this case there is minor contention for exclusive access. The lower bound is chosen for the wait queue service time and the number of jobs in the system. The branching probability is derived

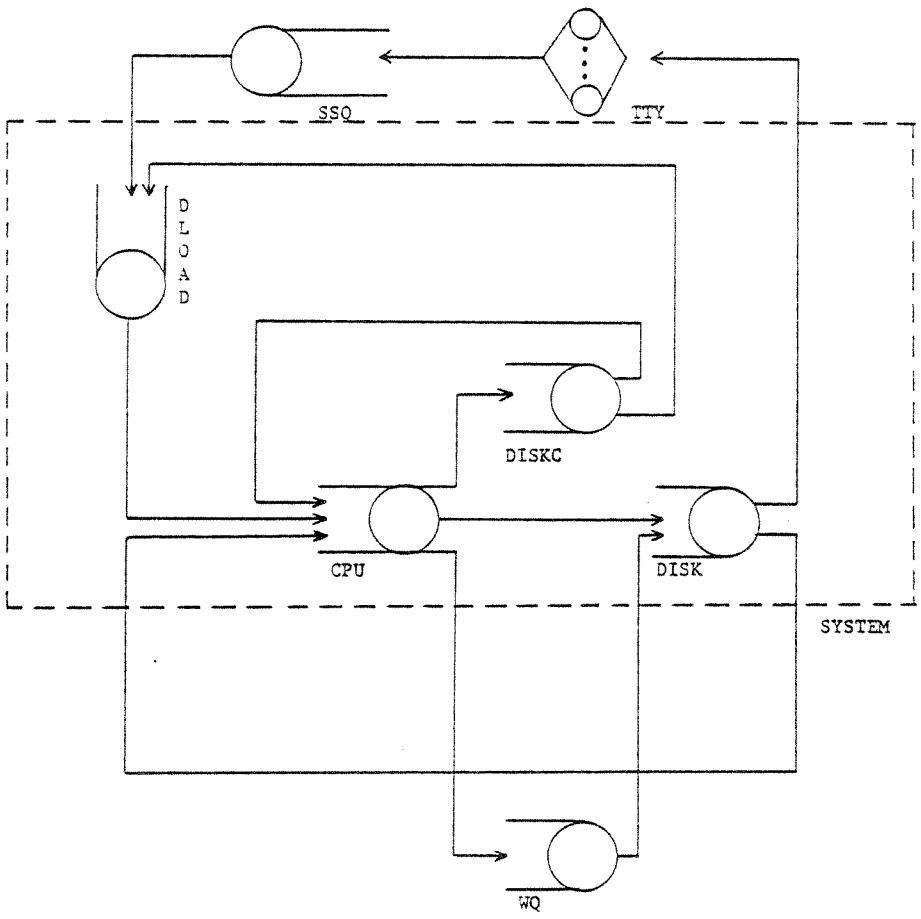


FIGURE 4.8. VALIDATION MODEL

TABLE 4.3. VALIDATION MODEL SOLUTIONS

		Actual System	No Wait	All Wait	Iteration 1	Iteration 2
Model Parameters						
Branching Probability:						
New type 2 jobs CPU to WQ			0	1	.117	.117
WQ Service Time			—	.316	.456	.468
Results						
Response Time:	LA	.7	.65	.97	.69	.69
	LB	1.0	.77	1.11	.85	.85
	LD	2.5	2.43	5.51	3.04	3.04
	LF	3.1	2.61	3.94	2.80	2.35
Weighted elapsed time:						
Type 2			.468	.456	.468	.468

from the number of transactions of each type in the system and from (one minus) the ratio of the elapsed time of type 2 jobs to the elapsed time of the transaction.

The model is then solved using these derived parameters. The elapsed time of type 2 jobs is calculated and found to be equal to the upper bound. The calculated branching probability is equal to the model parameter used and thus has stabilized.

The model is solved again using the revised wait queue service time (the upper bound). The solution yields an elapsed time for type 2 jobs equal to the service time for the WQ. Therefore the model solution has stabilized and no further iterations are necessary. The resulting response times are very close to the measured response times of the system, which indicates that the model is representative of the actual system. Thus, the queueing network model of blocking can be used to obtain suitable values for response time and other performance indicators.

Further research in this area may lead to a refined iterative solution technique to include heuristics to hasten convergence. It may be possible to reformulate the model to use state dependent branching [TOW75] to the wait queue and CPU. However, further research is needed to expand the scope of models of this type that satisfy local balance. In this blocking model a job always enters the SYSTEM when it leaves the wait queue. In Towsley's local balance model, the queue dependent probability of branching to the SYSTEM would also apply to jobs leaving the wait queue.

4.3 Summary

Queueing network models have been described that successfully represent concurrency, synchronization, mutual exclusion, and blocking in software systems. Previously, simulation or hybrid models have been required to analyze the effects of these design decisions on the resulting performance. Approximation techniques are used to obtain performance metrics interactively. These techniques were developed using an example-driven approach. They have produced satisfactory approximations for the representative examples studied. They are applicable to many, typical software designs; however, there may be some unusual types of designs that do not yield to this type of iterative analysis.

The successful use of queueing network models allows quick and easy solutions which are necessary for interactive performance prediction tools. The parameters for the models can easily be derived from the performance specifications. It is possible to automate the procedure to obtain the model parameters from the software execution graphs. The implementation of a comprehensive performance prediction tool that includes the model parameter derivation is discussed next.

CHAPTER 5
IMPLEMENTATION OF THE METHODOLOGY

5.0 Overview

The three previous chapters define the essential data and solution techniques for a performance analyst to predict and evaluate performance characteristics of most general software systems. The requirement stated in the introductory chapter was, however, that the methodology "...be used to quickly and easily predict performance..." and that it be suitable for use by software designers as well as performance analysts. These requirements necessitate an automated tool that encompasses the analysis techniques. This chapter contains functional specifications for such a comprehensive tool. The essential features required for the performance analyses are defined. Additional desirable features are also included.

A Design Evaluation Prototype Tool, ADEPT, was developed to demonstrate the feasibility of building and using an automated performance prediction and evaluation tool. A detailed description of ADEPT is provided which includes:

1. The performance data base design
2. The program specifications
3. The performance reports
4. Sample results

ADEPT was developed to aid in the analysis of the performance of a large data base management system, IPIP, through several design iterations [BOE80]. Examples of its use are shown in this chapter.

Some results of the analysis are in the Appendix.

5.1 Functional Specifications

The tool is to be used primarily for the interactive evaluation of the software and its environment. Therefore, the implementation must be efficient to assure satisfactory response times. It must also be flexible so that new solution techniques can be incorporated as additional performance prediction problems are encountered. It consists of three main components:

1. A data management system
2. A queueing network model solver
3. An interface to the user and the above components.

The data management system must provide easy and efficient data entry, retrieval, and modification of the performance data. It must be easy to represent hierarchical structures and the incremental resolution of processing details.

For example, the first (top-down) design definition includes each major component of the software and an estimate for the performance specifications. The next design definition includes more processing details for the major components. The revised performance specifications may involve data dependency. It must be easy to include processing details as they are available and to change performance specifications of components. Data dependency must be supported by providing the capability to specify (and store) either constant values or variables.

Easy and efficient data modifications are essential. Extensive modifications may be required throughout implementation to reflect design alterations. There is often a rippling effect when a

design modification is made to a component that is called by many other components. The evaluation of conditional performance goals often involves changing the value assigned to many variables. This and the evaluation of alternative design strategies requires retention of data for several versions of the software. Each must be easily identified for later comparison of alternatives. Finally, the data derived from performance measurements must be easily added to the data base.

A data management system interface must have an on-line query language as well as an interface to standard programming languages. The tool can then easily support general performance queries and smooth communication between performance tool components.

The queueing network model solver must provide standard performance metrics for a wide range of general models. It must handle multiple job types, dependent job typing, and hierarchical collapsing. It must support interactive interrogation of model results. The evaluation of both configuration and workload changes must be easy. It needs the memory model extension to calculate the probability of occurrences of various job mixes. The iterative solution capability is necessary and should be as automatic as possible. It should be easily parameterized and incorporate default model parameters where feasible.

The interface component is an important part of the tool since it is visible to the analysts and thus determines the usability and acceptability of the entire tool. The interactions must be smooth and require little effort from the analyst. It is essential that it be friendly and tolerant of errors. Error corrections should be as automatic as possible. For example, specified device service rates should be checked against typical acceptable ranges. When an out-of-range condition is detected, the specifications should be checked for obvious mistakes, and revised specifications suggested to the user.

Flexibility in the input of performance specifications is desirable. The analyst should be able to provide information in a natural format. The interface component should be able to translate it into an appropriate format for storage in the data base. For example, CPU time specifications can be in milliseconds, seconds, or even lines of code.

Reasonable default values for performance specifications are beneficial. Graphical input (and output) of the software structure is highly desirable. The evaluation of design alternatives must be easy. The results of each solution must be identified as to the version of the design and the implementation status of the components. Reports on the actual resource requirements and performance versus the specifications and estimates should be prepared for the analyst automatically.

Bottlenecks should be detected and the appropriate performance reports produced automatically. Automatic design and configuration optimizations should be detected and the performance comparisons produced where feasible.

A tool that incorporates all of these features has a high probability of acceptance and use in data processing installations. A prototype system is presented next that demonstrates the feasibility of developing such a tool.

5.2 Prototype Tool

A modular approach is used in the development of the comprehensive tool. ADEPT is the first version of the tool. It is specifically designed to enable the evaluation of a test case: a large data base management software system, IPIP. Most of the essential functions of the comprehensive tool are provided as well as

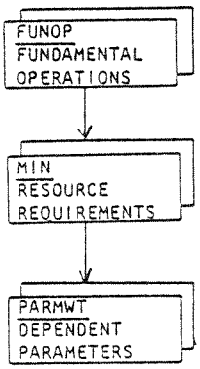
a straight forward means of gradually (modularly) expanding it to incorporate the other desirable functions.

In order to concentrate design and implementation efforts on the newly proposed functions, ADEPT uses commercially available software for the data management and queueing network model solver components. It is argued that if a successful prototype is produced with these components, a fully integrated, special-purpose tool is certainly feasible and will be more suited to the specific task.

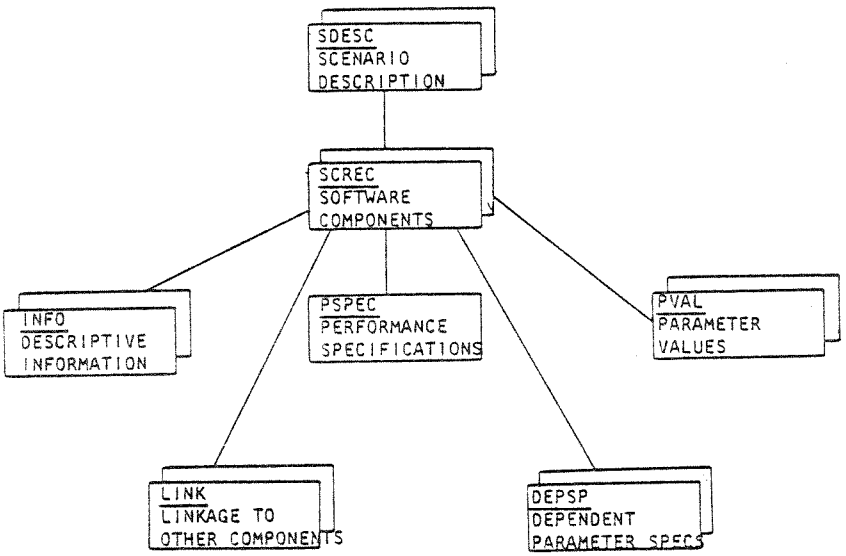
ADEPT uses SYSTEM 2000 for the data management component and the Computer Analysis and Design System, CADS, for the queueing network model solver [MRI73,IRA75]. The functionality of these components is important: they must satisfy enough of the functional requirements previously defined to successfully evaluate software systems. This functionality is demonstrated in the examples presented here and in the Appendix; therefore, a detailed discussion of these software products is not included.

An integral part of ADEPT is the performance data base design. It is illustrated in Figure 5.1. The test case, IPIP, consists of a number of standard data base functions such as FIND, RETRIEVE, ADD, and MODIFY that are called many times in user scenarios. There is a data base of these fundamental operations, FUN. Each one has minimum resource requirements (overhead) that are consumed when it is called. This overhead is contained in the MIN schema. Some of them also have variable resource requirements. The PARMWT schema contains the data dependent parameters and the resource requirements associated with each occurrence of the parameters. There may be multiple parameters for each fundamental operation. Design versions are supported by allowing multiple occurrences of the MIN schema and attaching an identifier to distinguish them. Each MIN occurrence has its own (optional) PARMWT schemas.

The descriptor data base, SDESC, contains information about each user scenario. There is a schema for each software component of the scenario, SCREC. Each component has associated descriptive



(a) Fundamental Operations Data Base (FUN)



(b) Scenario Descriptor Data Base (SDESC)

FIGURE 5.1. ADEPT DATA BASE DESIGN

information, INFO, and linkage with other components, LINK. A performance specification schema, PSPEC, contains the resource requirements of the component. There may be multiple versions of the PSPEC schema.

Software components can represent calls to fundamental operations. The resource requirements of the fundamental operations are then retrieved from the FUN data base. This precludes redundant performance specifications for operations called many times. The parameter value schema, PVAL, is used to assign values to the data dependent parameters of the fundamental operations. There may be multiple versions of the parameter values. This structure controls the "rippling effect": changes in the resource requirements of fundamental operations are isolated and a re-evaluation of software scenarios automatically includes the revised resource requirements.

Since the IPIP system evaluation concerns both the design of the fundamental operations and the user scenarios, the software components of each fundamental operation are defined in the descriptor data base. Design modifications are easily evaluated in this framework. Revised resource requirements are automatically derived by the analysis program and used to update the FUN data base. The dependent specification, DEPSP, is used to define the data dependent parameters for the fundamental operations. With this strategy, the fundamental operations can easily call other fundamental operations. There is no limit to the nesting of fundamental operations. This framework supports the incremental resolution of processing details and also controls the rippling effect of design changes.

The ADEPT data base design is specifically tailored to the IPIP system evaluation, but it is generally applicable to most software systems. Fundamental operations can represent a standard operating system function, such as a data translation routine, for which actual resource requirements can be obtained. They can also be functions developed along with the new software such as table

maintenance routines.

The SYSTEM 2000 Immediate Access Feature is used for general queries against and modifications to the performance data base. There are two interface programs: the first, FUNA, computes the resource requirements of the fundamental operations from the DESCR data base and updates the FUN data base. The second, GENREP, analyzes the software scenarios in SDESC and produces the queueing network model parameters, a workload characterization report and the frequency of the average elapsed time of the components. The specifications for these programs follow.

FUNA Design Specifications
Analysis of Fundamental Operations

Purpose:

To derive the resource requirements for the basic system (IPIP) functions.

General Description

The analysis begins with the independent functions (those that do not call other functions). Dependent functions are analyzed as the dependencies are resolved. The minimum resource requirements are computed and placed in the MIN schema. The dependent parameters and parameter weights are calculated and put in the PARMWT schema.

Input

Functions to be analyzed in order of evaluation.

FUN Data Base, schemas:

1. FUNOP Fundamental operations
2. MIN Resource requirements
3. PARMWT Dependent parameter weights

SDESC Data Base, schemas:

1. SDESC Scenario Description
2. SCREC Software Components
3. PSPEC Performance Specifications
4. PVAL Parameter Values
5. DEPSP Dependent Parameter Specifications

Update

FUN Data Base, schemas:

1. MIN
2. PARMWT