Computer Science Department Technical Report
University of California
Los Angeles, CA 90024-1596

DYNAMIC SHARING OF PROCESSORS IN PARALLEL PROCESSING SYSTEMS: A PERFORMANCE EVALUATION

Jau-Hsiung Huang
Leonard Kleinrock

October 1988
CSD-880082

Dynamic Sharing of Processors in Parallel Processing Systems:

A Performance Evaluation

Jau-Hsiung Huang and Leonard Kleinrock

Computer Science Department
University of California, Los Angeles

Abstract - We model a job in a parallel processing system as a concatenation of stages which must be processed in a given sequence, each stage of which requires a certain number of processors for an interval of time which is exponentially distributed. A job is allowed to receive service as long as there are available processors in the system following a preemptive priority service discipline. A job's priority increases each time it begins a new stage. A finite number of processors are assigned preemptively according to a job's priority; jobs with equal priority are resolved according to a first-come-first-serve rule. If a job with higher priority requires more processors than it currently has assigned when it finishes one stage of work and advances to the next, then it can obtain processors by preempting jobs with a lower priority. When all the processors of the lowest priority job are taken away (totally preempted), this job will be pushed back to the head of the queue and will wait for the next available processor. (This use of processors by more than one job at a time is an important extension of the model described in our earlier paper [12]).

An exact analysis for the mean response time for the general case of this model has not yet been obtained. However, we are able to solve certain two-stage cases exactly. A Scale-up rule is then introduced. This rule is used when we are given the mean response time for a parallel processing system with a fixed number of processors and we wish to find the mean response time when the number of processors is m times larger. The Scale-up rule provides a very good approximation. By using the Scale-up rule and the two-stage cases, we are able to provide an approximation for finding the mean response time for any general application.

Index Terms - parallel processing, scale-up rule, processor sharing, processor utilization.

I. INTRODUCTION

We model a parallel processing system with P processors as a queueing system with a single queue. A job is modeled as a concatenation of stages which must be processed in a given sequence and the number of processors required in each stage may be different. Because of the characteristic of jobs and the preemptive priority scheme used, the number of processors possessed by a job during its execution time varies. This characteristic makes the exact analysis for the general case very difficult because we must keep track of how many jobs there are in each stage. Later in this paper we show that an $M/E_r/m$ queueing system is a special case of this model and $M/E_r/m$ has no known closed form solution as yet. The main goal of this paper is to find the mean response time of the system.

In this paper, we are able to find the exact solution for jobs with two stages when the number of processors available in the system equals the maximum number of processors required by the job. We then propose a Scale-up Rule. This rule is used to approximate the mean response time for a queueing system with $m \cdot P$ processors when the result with P processors is given. For example, we can use the Scale-up rule and the result for M/G/1 queueing

This research is sponsored by the Defense Advanced Research Projects Agency, Department of Defense (Contract number MDA903-87-C-0663).

systems to find the approximate mean response time for an M/G/m system (which has no known exact solution). Combining our result for the two-stage model and the Scale-up Rule, we give an approximation for the general case (i.e., more than two stages). >From these studies, we propose a rule of thumb for designing parallel algorithms.

The rest of this paper is organized as follows. Section II provides the model description and assumptions. In Section III we find the exact mean response time of a job with a serial stage, which requires no processing capacity at all, and a parallel stage, which requires all the processing capacity, during its execution. In Section IV we analyze, exactly, the mean response time of a two-stage model in which one stage requires P_1 processors and another stage requires $P = mP_1$ processors. In Section V we introduce the Scale-up Rule with some examples. In Section VI we use the Scale-up Rule and the exact result obtained in the two-stage model to approximate the mean response time for a general model given any number of processors. A final conclusion is given in Section VII.

II. MODEL DESCRIPTION AND ASSUMPTIONS

In this paper, jobs arrive to the system according to a Poisson process with rate λ. A job is assumed to be composed of several stages which must be processed one after the other; in each stage up to a given maximum number of processors can be used. The amount of service required by stage i is exponentially distributed with a mean, say W_i seconds; if stage i uses P_i processors, then the time to serve this stage will also be exponentially distributed with a mean equal to $\frac{W_i}{P_i}$ seconds. The numbers of processors needed for different stages need not be the same. Hence, a job can be described by using two vectors. The first vector is the *processor vector* which specifies the number of processors required by each stage. The second vector is the *time vector* which specifies the *average* service time required for each stage. We denote these two vectors as:

$$\vec{P} = [P_1, P_2, P_3, \ldots\ldots, P_n]$$

$$\vec{t} = [t_1, t_2, t_3, \ldots\ldots, t_n]$$

where $\vec{P}$ and $\vec{t}$ stand for the processor vector and the time vector respectively and n is defined as the number of stages in a job. We assume the system has P processors available.

Whenever a job in a stage requires more processors than it is allowed, this job simply uses all the processors available to it with an appropriately elongated stage service time such that the average work done for that stage is constant. That is, the stage time will still be exponentially distributed, but with a larger mean. For example, if a job in a stage needs 10 processors for a mean of 1 second while there are only 5 processors available to it, it uses these 5 processors for an average of 2 seconds (in a processor-sharing fashion [10]). This is similar to the elongation of service described in [12].

The service discipline applied in this paper is a version of priority queueing with preemption. Specifically, if the highest priority job requires less than P processors, these processors which are not needed by this job are assigned to the next highest priority job, etc. However, the priority is not based upon the different classes of jobs entering the system; instead, the priority is based upon the stage that the job is currently working on. The purpose of this priority assignment is to give a higher priority to a job which has fewer stages of work to finish (because this job is perhaps closer to completion). Hence, when a job finishes one stage and advances to the next stage, the priority ranking of this job will be advanced. If by advancing to the next stage this job requires more processors than it currently possesses, this job will preempt processors from other jobs in the system beginning with the lowest priority group until either it has gained enough processors or there are no more lower priority jobs in service. Ties are resolved by a FCFS rule. Those jobs with all processors preempted will be pushed back to the head of the queue. The following example gives an illustration of how the system works.

Example: In this example we assume each job has two stages, the first stage requires 2 processors and the second stage requires 5 processors (i.e., $\vec{P} = [2,5]$). The time vector need not be specified for the purpose of this example. The system has a total of 5 processors. and initially the system is empty.

Event 1: Job A arrives.
Job A uses two processors to work on the first stage at a rate of 2 seconds per second. The other three processors remain unused.

Event 2: Job B arrives while Job A is still in stage 1.
Job B uses two processors from the unused processors to work on the first stage at a rate of 2 seconds per second. Only one processor remains unused.

Event 3: Job C arrives while Jobs A and B are still in stage 1.
Job C uses the only unused processor to work on the first stage at a rate of 1 second per second. There are no more processors unused.

Event 4: Job D arrives while Jobs A, B, and C are still in stage 1.
Job D joins the queue waiting for any processor to become available.

Event 5: Job A finishes stage one and begins stage two.
Job A now has higher priority than jobs B and C and job A needs three more processors to work on the second stage; therefore, jobs B and C are both preempted and are pushed back to the head of the queue such that job A can use all the processors. After gaining all the processors, job A works at a rate of 5 seconds per second in stage 2.

Event 6: Job A finishes stage two and leaves the system.
All the five processors are released. Jobs B and C reenter service and both use two processors to work on the first stage at a rate of 2 seconds per second. Job D also enters service and uses the remaining processor to work on the first stage at a rate of 1 second per second.

The reason we claim the exact analysis is very difficult can be illustrated by using one special case. In this special case, the system has m processors and the number of processors required by all stages equals "1"; the mean service time for all stages is the same. Obviously this very simple special case corresponds to an $M/E_n/m$ queue which is a queueing system without a known closed form solution.

In fact, a generalization of this model leads to a new class of queueing problems (much harder than those addressed in this paper) as follows [11]. Consider that each arriving job has a random function, $r(x)$, which describes the rate at which that job can be processed, given that it has so far received x seconds of service (the units for $r(x)$ are seconds of work per second). If $r(x)$ exceeds C, the maximum processing rate of the system, then the service time is elongated as shown, for example, in Figure 1 of [12]. The capacity C is allocated according to a FCFS preemptive rule or, possibly, according to an arbitrary priority rule (as, for example, pure processor sharing as discussed in [1]). In this paper, we have studied some special cases of this new class of queueing systems.

III. A SERIAL - PARALLEL MODEL

In this section, we assume the execution of a job requires two stages, namely a *serial* stage and a *parallel* stage. In the serial stage, the job requires a negligible amount of total processors while in the parallel stage the job requires all P processors from the system. One practical example of this model is for systems with a large number of processors from which the serial stage requires only one processor while the parallel stage requires all processors. Therefore, if the parallel stage has a higher priority, then there can be at most one job working in the parallel stage. Once there is a job working in the parallel stage, all other jobs will be stopped because there are no available processors left. On the other hand, if the serial stage has a higher priority, then all the jobs working in the serial stage can work concurrently and they will have no effect on jobs working in the parallel stage because the capacity taken by the jobs in the serial stage is negligible. We first study the case when the serial stage precedes the parallel stage (Figure 1(a)); then we study the case when the parallel stage precedes the serial stage (Figure 1(b)). Finally we study the case when the parallel stage is preceded and followed by a serial stage (Figure 1(c)). We derive both the mean response time and the z-transform of the distribution of the number of jobs in the system.

FIGURES 1a, 1b, and 1c GO HERE.

A. The Serial Stage Precedes the Parallel Stage

In this case, stage one is the serial stage and stage two is the parallel stage and the higher priority is given to stage two (the parallel stage) as mentioned earlier. If there are no jobs in stage two, then all jobs in stage one are allowed to work concurrently since there are always enough processors for each job. However, if one of the jobs finishes stage one and advances to stage two, this job will occupy all the processors and preempt all other jobs still in stage one.

One application of this model is for a distributed database which uses a locking mechanism to preserve data integrity whenever more than one transaction tries to write into the database. We assume there are a large number of workstations sharing this database. We model a transaction as one query (read) followed by one update (write). If concurrent read is allowed in the database, then we allow all queries to be processed in the system concurrently. Furthermore, when a transaction is in the query phase, it occupies only the processor which issues the transaction. This read phase corresponds to the serial stage in our model. However, once a transaction finishes the read phase and issues a write update, the entire database will be locked such that all other transactions will be stopped. This update phase corresponds to our parallel stage. If we further assume the time required for the read phase and the write phase are both exponentially distributed, then our model matches this distributed database model perfectly.

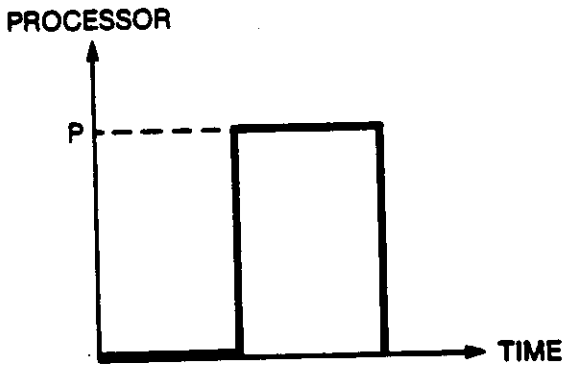

FIGURE 1a. Serial Phase Precedes Parallel Phase

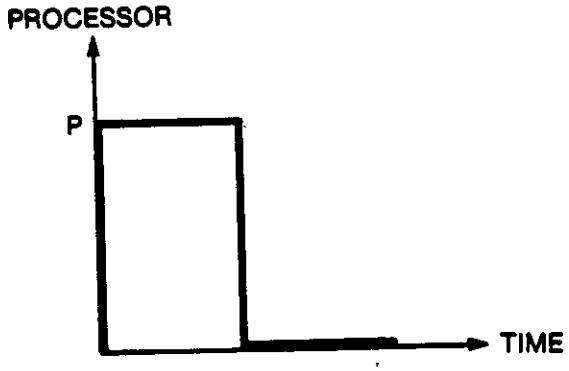

FIGURE 1b. Parallel Phase Precedes Serial Phase

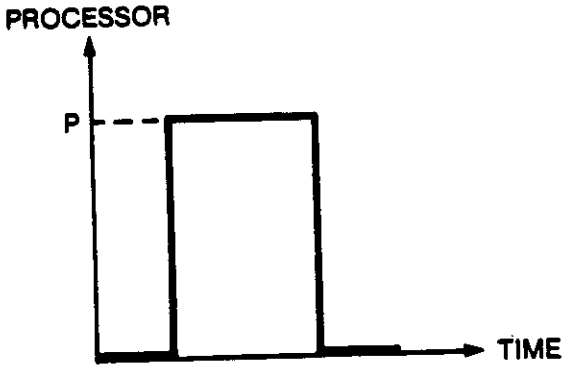

FIGURE 1c. Serial Phase Precedes and Follows Parallel Phase

Let us define the service rate of the serial stage to be μ_1 and the service rate of the parallel stage to be μ_2. The Markov chain for this system is shown in Figure 2 where the state (k, j) represents the situation where there are k jobs in stage one (the serial stage) and j jobs in stage two (the parallel stage).

FIGURE 2 GOES HERE.

THEOREM 1: Given the serial - parallel model as shown in Figure 1(a), where μ_1 is the service rate of the serial stage and μ_2 is the service rate of the parallel stage, the z-transform of the number of jobs in the system is:

$$P(z) = \frac{\mu_2 - \lambda}{\mu_2 - \lambda z} e^{\frac{\lambda}{\mu_1}\left[z - 1 + \ln\frac{\mu_2 - \lambda}{|\lambda z - \mu_2|}\right]}$$

and the mean response time of the system, T_{sp} (where "sp" stands for "serial-parallel"), is

$$T_{sp} = \frac{1/\mu_1 + 1/\mu_2}{1 - \lambda/\mu_2} \tag{1}$$

[Proof] See Appendix A.

Interestingly, note that $\frac{\lambda}{\mu_2}$ in (1) is the load of the system since the serial stage contributes no load at all. Also note that $1/\mu_1 + 1/\mu_2$ is the service time of a job. Hence, if we define

$$\bar{x} = \frac{1}{\mu_1} + \frac{1}{\mu_2} \tag{2}$$

and

$$\rho = \frac{\lambda}{\mu_2}$$

then (1) becomes

$$T_{sp} = \frac{\bar{x}}{1 - \rho} \tag{3}$$

which is the expression of the mean response time of an M/M/1 queue.

B. The Parallel Stage Precedes the Serial Stage

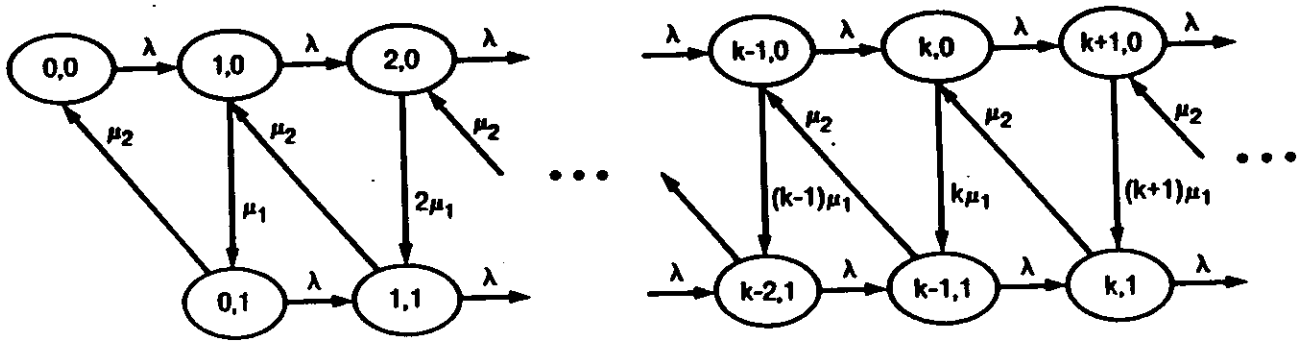

FIGURE 2
The Markov Chain for Serial Stage - Parallel Stage

Here we change the sequence of the stages such that stage one is the parallel stage and stage two is the serial stage. We assume the service rate for the serial stage is μ_1 and the service rate for the parallel stage is μ_2, as defined in the previous model.

The analysis of this model is simple since the serial stage (with the higher priority) will never affect the parallel stage because the serial stage requires no capacity at all. This model can be treated as an M/M/1 queueing system with service rate μ_2 with the modification that each job will suffer an extra delay of mean $\frac{1}{\mu_1}$ before leaving the system. Let us define the mean response time of this model to be T_{ps} (where "ps" stands for "parallel-serial").

THEOREM 2:

$$T_{ps} = \frac{1/\mu_2}{1-\lambda/\mu_2} + \frac{1}{\mu_1} \tag{4}$$

[Proof] This result follows directly from the result for M/M/1.

Q.E.D.

Let us compare T_{sp} and T_{ps} from (1) and (4); it can be shown that T_{sp} is always greater than T_{ps} for $\rho = \frac{\lambda}{\mu_2} < 1$.

Indeed,

$$T_{sp} - T_{ps} = \frac{1}{\mu_1} \cdot \frac{\lambda/\mu_2}{1-\lambda/\mu_2} = \frac{1}{\mu_1} \cdot \frac{\rho}{1-\rho} > 0$$

T_{sp} can be much greater than T_{ps} when ρ approaches one as shown in Figure 3. If we regard the parallel stage as the blocking stage (since it blocks all other jobs in the system), then this result suggests that a job with an early blocking stage performs better (i.e., has a lower mean response time) than a job with a late blocking stage assuming other conditions remain the same.

FIGURE 3 GOES HERE.

C. A Combined Model

In this model, we assume the job begins with a serial stage which is followed by a parallel stage and which is again followed by a serial stage as shown in Figure 1(c). We define "f" to be the fraction of the mean service time the job spends in the serial stage during the entire mean service time, i.e.

$$f \triangleq \frac{1/\mu_1}{1/\mu_1 + 1/\mu_2}$$

Without loss of generality, we normalize the mean service time to be unity, i.e.

$$\frac{1}{\mu_1} + \frac{1}{\mu_2} = 1$$

Thus

$$\frac{1}{\mu_1} = f \quad \text{and} \quad \frac{1}{\mu_2} = 1 - f$$

We define "g" to be the fraction of the serial stage which precedes the parallel stage (i.e., a percent g of the serial stage precedes the parallel stage and the remaining fraction (1-g) of the serial stage follows the parallel stage). We denote the mean response time of this system to be T_{fg}.

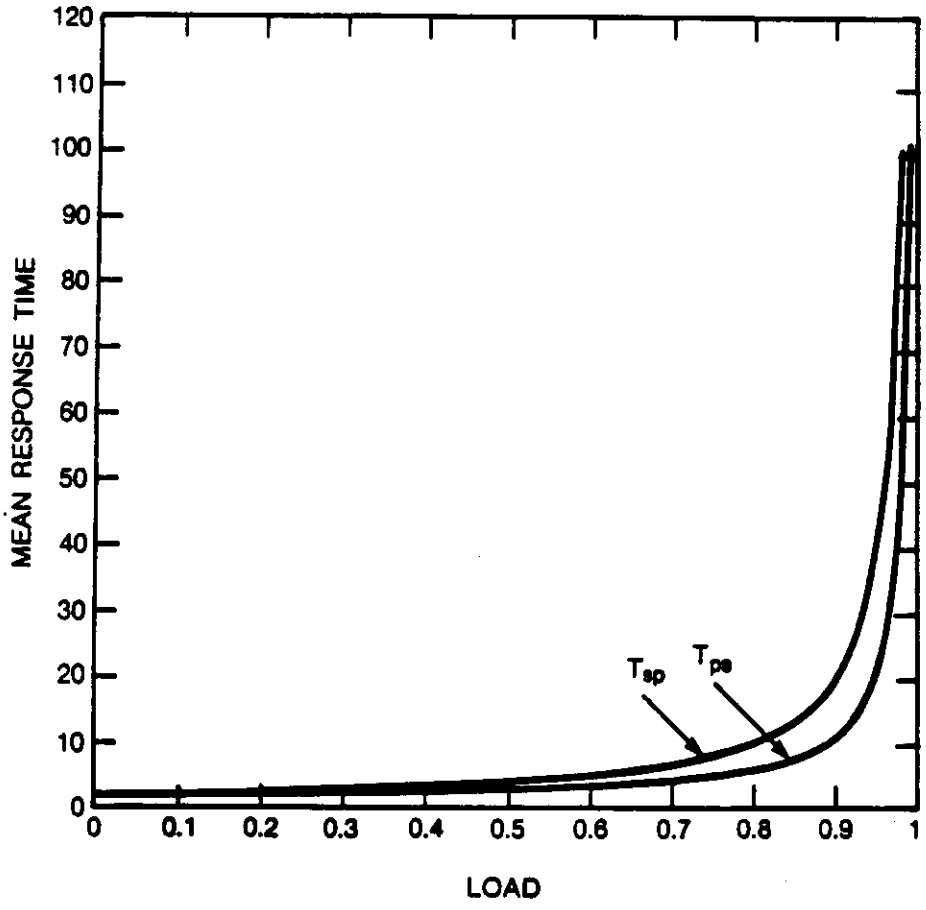

FIGURE 3. Average System Time versus Load ($\mu_1 = \mu_2 = 1$)

THEOREM 3: For the serial - parallel model shown in Figure 1(c), the mean response time is

$$T_{fg} = \frac{(1-f)+gf}{1-\lambda(1-f)} + (1-g)f \tag{5}$$

under the condition that

$$\lambda(1-f) < 1$$

[Proof] By using the results of (1) and (4) we obtain

$$T_{fg} = \frac{\frac{1}{\mu_2} + g\cdot\frac{1}{\mu_1}}{1-\frac{\lambda}{\mu_2}} + (1-g)\cdot\frac{1}{\mu_1} = \frac{(1-f)+gf}{1-\lambda(1-f)} + (1-g)f$$

The utilization of the processors is

$$\rho = \frac{\lambda}{\mu_2} = \lambda(1-f)$$

Hence, the condition for the system not to saturate is

$$\rho < 1 \qquad or \qquad \lambda(1-f) < 1$$

Q.E.D.

Figure 4 shows the mean response time versus load over different values of g and f. From these results, we have the following theorem.

Corollary 1: If $g_1 < g_2$, we have

$$T_{g_1 f} < T_{g_2 f}$$

[Proof] This can easily be proved from (5).

FIGURE 4 GOES HERE.

IV. A TWO-STAGE MODEL

In this section, we consider a job with only two stages. In one stage, the job uses P_1 processors to process its work concurrently. In another stage, the job uses mP_1 processors to process the work concurrently, where m is a real number greater than 1. We define the stage which uses P_1 processors as the *low-concurrency* stage and the other stage which uses mP_1 processors concurrently as the *high-concurrency* stage. We assume there are $P = mP_1$ processors available in the system, i.e., the number of processors in the system equals the number of processors required by the high-concurrency stage. In this section, two cases are analyzed: one is when the low-concurrency stage precedes the high-concurrency stage and the other is when the high-concurrency stage precedes the low-concurrency stage. For cases when the number of processors in the system is greater than mP_1, a good approximation will be given after the introduction of the Scale-up rule, which will be described in Section V.

A. The Low-Concurrency Stage Precedes the High-concurrency Stage

In this section, stage one is the low-concurrency stage while stage two is the high-concurrency stage. We assume the service rate for stage one is μ_1 and the service rate for stage two is μ_2. This job can be modeled as shown in Figure 5. This model is similar to the model described in Section III except that in this model there can be at most m jobs working concurrently in the first stage. For such a model, we can regard P_1 processors as one server and regard the whole system as having m servers. Hence, the Markov chain of this model can be shown as Figure 6.

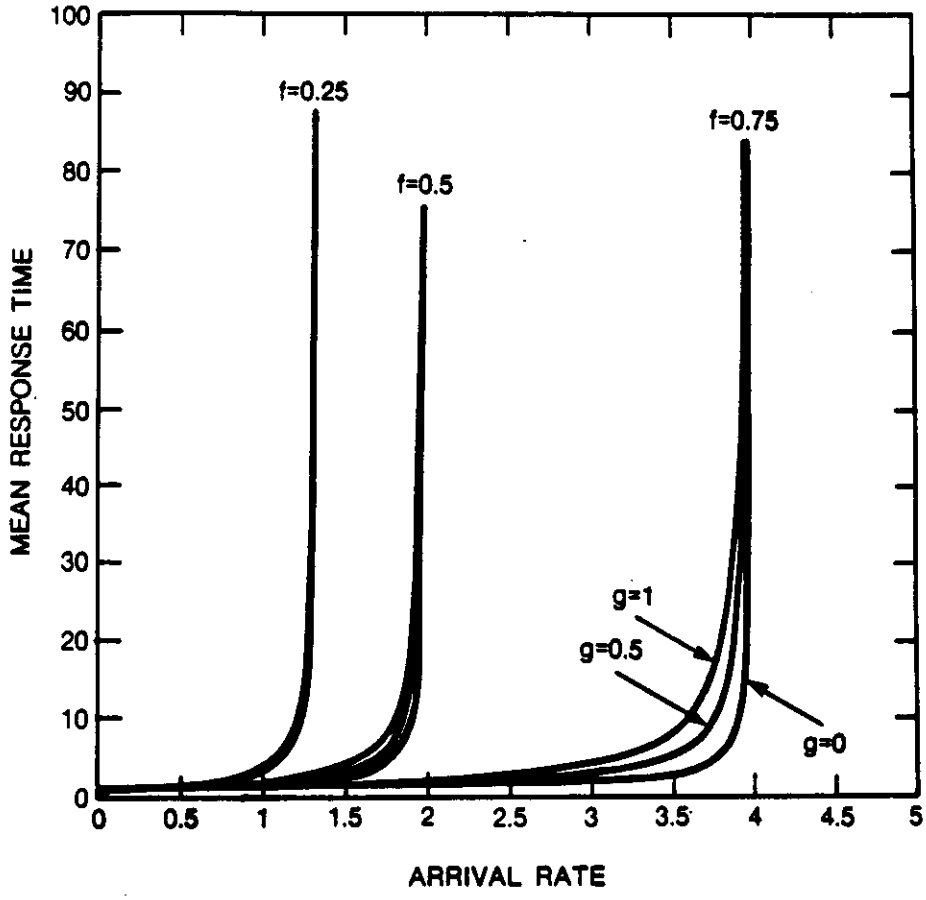

FIGURE 4. Mean Response Time versus Arrival Rate for Various g and f (For Each Set of Curves, g=1,0.5, and 0 Respectively From Left to Right)

We define $p(k,j)$ to be the probability that there are k jobs in stage 1 and j jobs in stage 2 in the system.

FIGURE 5 GOES HERE.

FIGURE 6 GOES HERE.

THEOREM 4: For the two-stage model as shown in Figure 5, the z-transform of the number of jobs in the system is

$$P(z) = \frac{\mu_1\mu_2}{-\lambda^2 z + m\mu_1\mu_2 - m\lambda\mu_1 z - \lambda\mu_2 z + \lambda^2 z^2} \sum_{k=0}^{m-1} (m-k)p(k,0)z^k$$

and the mean response time is

$$T = \frac{\mu_1\mu_2}{\lambda(m\mu_1\mu_2 - m\lambda\mu_1 - \lambda\mu_2)} \sum_{k=1}^{m-1} k(m-k)p(k,0) - \frac{\lambda - m\mu_1 - \mu_2}{m\mu_1\mu_2 - m\lambda\mu_1 - \lambda\mu_2} \tag{6}$$

where $p(k,0)$ for $0 \le k \le m-1$ can be obtained from the following:

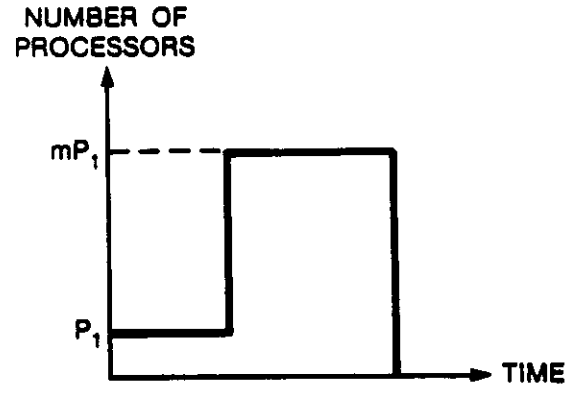

FIGURE 5. A Job's Profile

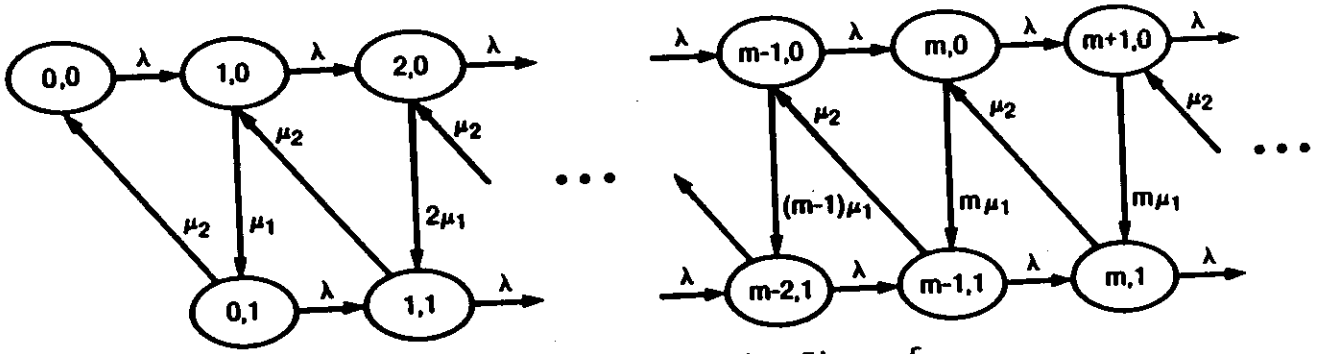

FIGURE 6. The Markov Chain for Figure 5

$$\sum_{k=0}^{m}(m-k)p(k,0)=\frac{m\mu_1\mu_2-m\mu_1\lambda-\mu_2\lambda}{\mu_1\mu_2} \tag{7}$$

$$p(1,0)=\frac{\lambda(\lambda+\mu_2)}{\mu_1\mu_2}p(0,0) \tag{8}$$

$$p(2,0)=\frac{\lambda^2[(\lambda+\mu_2)^2+\lambda\mu_1]}{2\mu_1^2\mu_2^2}p(0,0) \tag{9}$$

$$p(k,0)=\frac{(\lambda+\mu_2)[\lambda+(k-1)\mu_1]}{k\mu_1\mu_2}p(k-1,0)-\frac{\lambda[\lambda+2\mu_1(k-1)]}{k\mu_1\mu_2}p(k-2,0)-\frac{\lambda^2}{k\mu_1\mu_2}p(k-3,0) \quad k\geq 3 \tag{10}$$

The condition for the system not to saturate is

$$\lambda<\frac{m\mu_1\mu_2}{m\mu_1+\mu_2} \tag{11}$$

[**Proof**] See Appendix B.

>From (6) we observe that the constraint on λ to have a finite delay is the same as that given in (11). The processor utilization (u) can be found as

$$u=\frac{\lambda(\mu_2+m\mu_1)}{m\mu_1\mu_2}$$

Hence, the constraint on λ is equivalent to

$$u<1$$

An example is now given for m=2 which will be used later for comparison. >From (7) we have

$$2p(0,0)+p(1,0)=\frac{2\mu_1\mu_2-2\mu_1\lambda-\mu_2\lambda}{\mu_1\mu_2} \tag{12}$$

>From (8) we have

$$\mu_1\mu_2p(1,0)=\lambda(\lambda+\mu_2)p(0,0) \tag{13}$$

Solving (12) and (13) we have

$$p(0,0)=\frac{2\mu_1\mu_2-2\mu_1\lambda-\mu_2\lambda}{\lambda^2+\mu_2\lambda+2\mu_1\mu_2} \tag{14}$$

$$p(1,0)=\frac{(\lambda^2+\mu_2\lambda)(2\mu_1\mu_2-2\mu_1\lambda-\mu_2\lambda)}{\mu_1\mu_2(\lambda^2+\mu_2\lambda+2\mu_1\mu_2)} \tag{15}$$

Substituting (14) and (15) into (6) we have

$$T=\frac{4\mu_1\mu_2^2+4\mu_1^2\mu_2-\lambda^3}{(\lambda^2+\mu_2\lambda+2\mu_1\mu_2)(2\mu_1\mu_2-2\mu_1\lambda-\mu_2\lambda)} \tag{16}$$

B. The High-concurrency Stage Precedes the Low-concurrency Stage

Here we study the case when the high-concurrency stage precedes the low-concurrency stage. We assume the service rate for stage one is μ_1 and the service rate for stage two is μ_2. This model is shown in Figure 7.

FIGURE 7 GOES HERE.

Unfortunately, we are not able to solve the general case for arbitrary values of m. A study using the Markov chain approach is given in [8] without obtaining the exact solution.

However, the result for $m = 2$ can be derived as follows. The Markov chain for $m = 2$ is shown in Figure 8.

FIGURE 8 GOES HERE.

From this Markov chain, we have

$$(\lambda + \mu_1)p(k,0) = \lambda p(k-1,0) + \mu_2 p(k,1) \qquad k \geq 1 \tag{17}$$

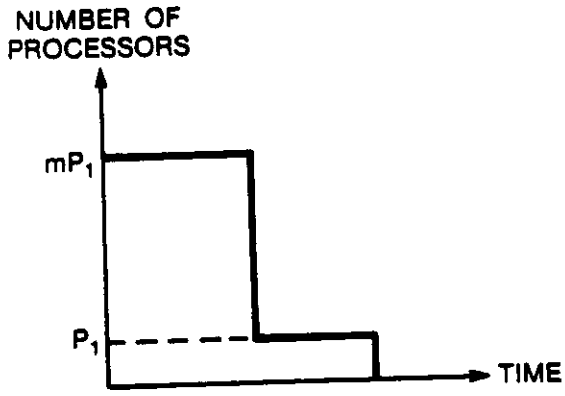

FIGURE 7. A Job's Profile

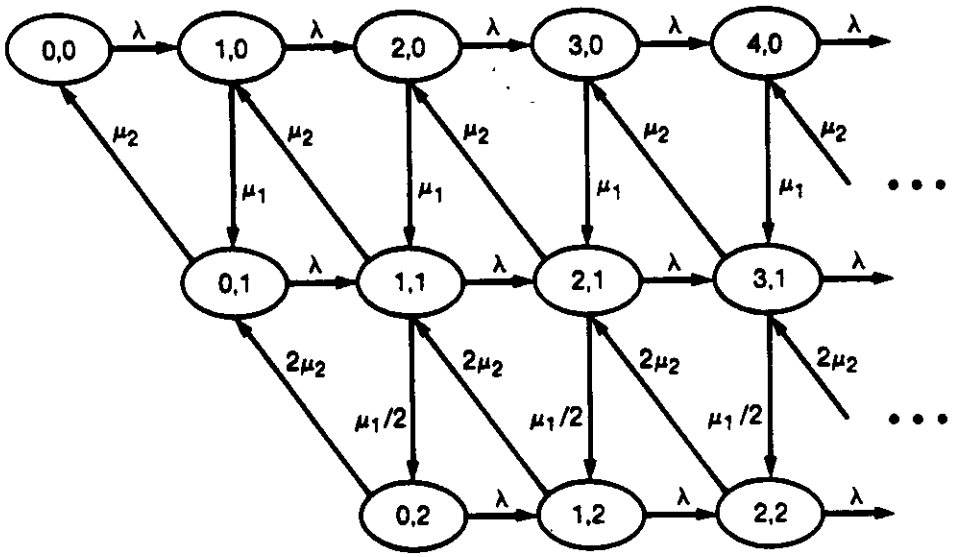

FIGURE 8. Markov Chain for Figure 7

$$\left(\lambda+\mu_2+\frac{1}{2}\mu_1\right)p(k,1)=\lambda p(k-1,1)+\mu_1 p(k+1,0)+2\mu_2 p(k,2) \qquad k\geq 1 \tag{18}$$

$$(\lambda+2\mu_2)p(k,2)=\lambda p(k-1,2)+\frac{1}{2}\mu_1 p(k+1,1) \qquad k\geq 1 \tag{19}$$

The boundary conditions are

$$\lambda p(0,0)=\mu_2 p(0,1)$$

$$(\lambda+\mu_2)p(0,1)=\mu_1 p(1,0)+2\mu_2 p(0,2)$$

$$(\lambda+2\mu_2)p(0,2)=\frac{1}{2}\mu_1 p(1,1)$$

Defining $P_i(z)\triangleq\sum_{k=0}^{\infty}p(k,i)z^k$ and $p(n)\triangleq$ Prob[n jobs in the system], we can show that

$$p(n)=\sum_{j=0}^{\min(2,n)}p(n-j,j)$$

and

$$P(z)=\sum_{n=0}^{\infty}p(n)z^n=\sum_{n=0}^{2}\sum_{j=0}^{n}p(n-j,j)z^n+\sum_{n=3}^{\infty}\sum_{j=0}^{2}p(n-j,j)z^n=\sum_{j=0}^{2}z^j P_j(z) \tag{20}$$

From $P(z)$ we can derive, $\bar{N}$, the mean number of jobs in the system, as

$$\bar{N}=\frac{d}{dz}P(z)\Big|_{z=1}=P'(z)\Big|_{z=1}=\sum_{j=0}^{2}\left[jz^{j-1}P_j(z)+z^j\frac{d}{dz}P_j(z)\right]_{z=1}=\sum_{j=0}^{2}\left[jP_j(z)+\frac{d}{dz}P_j(z)\right]_{z=1} \tag{21}$$

From these conditions it is shown in [8] that

$$\bar{N}=P_1(1)+2P_2(1)+P_0'(1)+P_1'(1)+P_2'(1)$$

$$=\frac{\lambda}{\lambda+2\mu_2}\cdot\frac{(3\mu_2^2+\mu_1\mu_2+2\mu_1^2)\lambda^2-\mu_2(\mu_1^2-\mu_1\mu_2-4\mu_2^2)\lambda-4\mu_2^2(\mu_1^2+3\mu_1\mu_2+2\mu_2^2)}{\mu_2(\mu_1+2\mu_2)(\mu_1\lambda+2\mu_2\lambda-2\mu_1\mu_2)}$$

Using Little's result [13] we finally have

$$T=\frac{\bar{N}}{\lambda}=\frac{(3\mu_2^2+\mu_1\mu_2+2\mu_1^2)\lambda^2-\mu_2(\mu_1^2-\mu_1\mu_2-4\mu_2^2)\lambda-4\mu_2^2(\mu_1^2+3\mu_1\mu_2+2\mu_2^2)}{\mu_2(\lambda+2\mu_2)(\mu_1+2\mu_2)(\mu_1\lambda+2\mu_2\lambda-2\mu_1\mu_2)} \tag{22}$$

C. A Comparison

Let us compare the results given in (16) and (22) for $\mu_1=\mu_2=1$. These two examples have every condition the same except that the sequence of the stages is different. The result from (16) shows that

$$T_1=\frac{1+\lambda}{\lambda^2+\lambda+2}+\frac{3-\lambda}{2-3\lambda}$$

where T_1 is defined as the mean response time when the low-concurrency stage precedes the high-concurrency stage. The result from (22) shows that

$$T_2 = \frac{6\lambda^2 + 4\lambda - 24}{9\lambda^2 + 12\lambda - 12}$$

where T_2 is defined as the mean response time when the high-concurrency stage precedes the low-concurrency stage. The difference, $T_1 - T_2$, is

$$T_1 - T_2 = \frac{\lambda(4 - \lambda - \lambda^2)}{3(\lambda + 2)(\lambda^2 + \lambda + 2)}$$

It can be shown easily that for $\lambda < \frac{2}{3}$ (or, $u < 1$), $T_1 - T_2$ is always positive. If we regard the high-concurrency stage as the blocking stage (since it blocks other jobs in the system), then $T_1 - T_2 > 0$ is consistent with the result we obtained in Section IIIB. Hence, the rule of thumb in designing parallel algorithms is to put the blocking stage as early as possible (if allowed). T_1 and T_2 versus load are shown in Figure 9. Note that T_1 and T_2 are very close to each other; this is because there is no pole in the denominator in the expression of $T_1 - T_2$.

Figure 9: Comparison Between T_1 and T_2

V. THE SCALE-UP RULE

The results derived in Sections III and IV are for cases when the number of processors in the system equals the maximum number of processors required by the job. This assumption simplified the Markov chain dramatically, hence making the analysis feasible. However, this analysis is infeasible when the number of available processors is greater than the maximum number of processors required by the job. In order to solve this problem, we propose the Scale-up Rule which gives a very good approximation result without adding any analytical complexity. An application of the Scale-up rule to the two-stage model when the number of processors in the system is greater than the

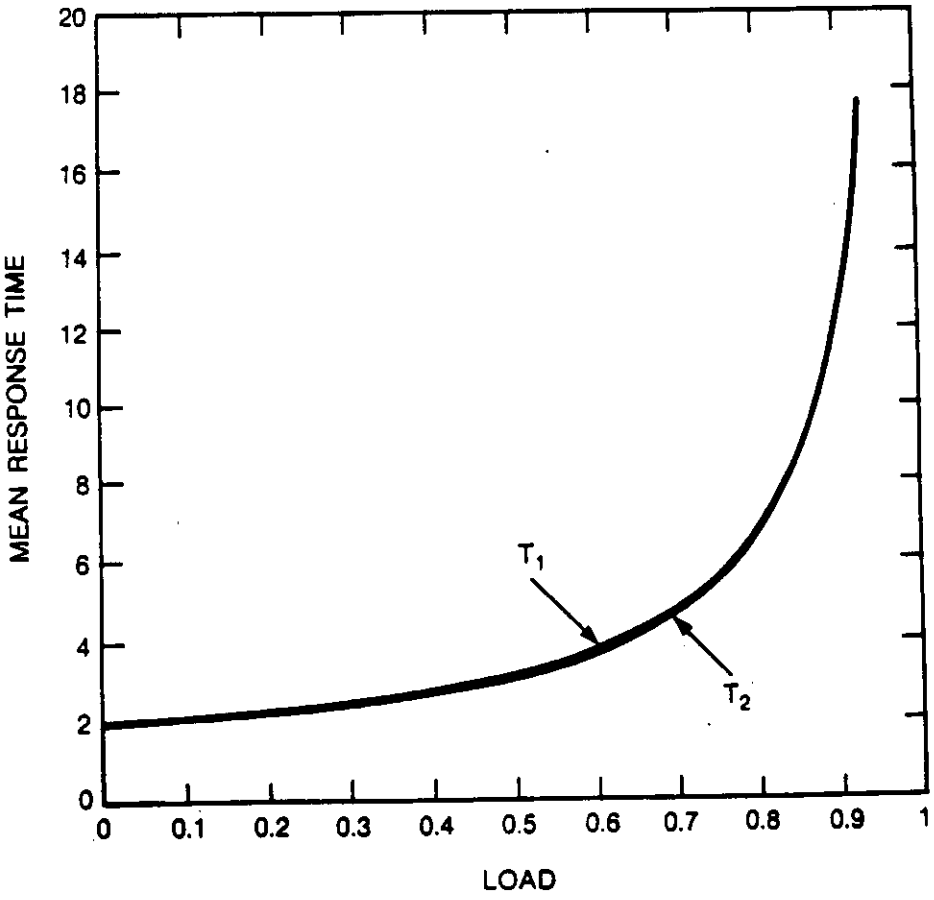

FIGURE 9. Comparison Between T_1 and T_2

maximum number of processors required by the job is given.

A. Previous Work: Classical Queueing Model

Although there has been considerable effort paid to the analysis of the M/G/m queue, almost no exact result has been achieved for the mean waiting time. Two exceptions are provided by M/M/m and M/D/m systems although in both cases the solutions are not in closed form ([5], [16]). Besides these two exceptions, only bounds, approximations, and numerical results have been found for other cases. In [6] some numerical results are given for the $M/E_k/m$ queue. In [3] and [9], some bounds are provided. In [2], [7], and [17] some approximation results for such systems are derived.

In this section, we focus our attention on the results in [14] and [15] which provide an approximation for the mean waiting time in an M/G/m system. In [2] and [4] this model was extended to achieve a better result. Let us define

$W_{M/G/m} \triangleq$ mean waiting time for an M/G/m system

In [14] and [15] the following approximation for $W_{M/G/m}$ was suggested:

$$W_{M/G/m} \approx \frac{W_{M/G/1}}{W_{M/M/1}} \cdot W_{M/M/m} \tag{23}$$

This expression proved to be a very good approximation as some simulation results in [8] demonstrated.

B. A Queueing Model with a Varying Number of Required Processors

Although the number of processors required by a job varies during its execution time, we were surprised (and pleased) to discover that the rule which applies to the classical queueing system as stated in (23) also applies in our model. In other words, if we have the mean response time for a system with P processors, we can find a good approximation result when the number of processors is greater than P by using the following Scale-up rule.

The Scale-up Rule: Given a job specification and the number of processors in the system, P (which equals the maximum number of processors required by the job), we define the average waiting time of the system to be $W_{M/PT/1}(\rho)$, where ρ is the system load and PT stands for "Processor-Time vector". Define $W_{M/PT/m}(\rho)$ to be the average waiting time if the number of processors in the system is mP. Similarly, define $W_{M/M/1}(\rho)$ to be the average waiting time of an M/M/1 queueing system with the same mean service time as the job. The Scale-up Rule says that

$$W_{M/PT/m}(\rho) \approx \frac{W_{M/PT/1}(\rho)}{W_{M/M/1}(\rho)} \cdot W_{M/M/m}(\rho)$$

The Scale-up rule can be useful in two situations. First, if we can analytically obtain $W_{M/PT/1}$, as we did in Sections III and IV, then we can use the Scale-up rule to approximate $W_{M/PT/m}$. Second, if we cannot obtain the analytical result, then we have to run simulations. Since the time required to run simulations for the $M/PT/m$ system is longer than the time required for the $M/PT/1$ system, we can run simulations to find the mean waiting time for the $M/PT/1$ system and then apply the Scale-up rule to find the mean waiting time for the $M/PT/m$ system for any m. A large amount of time can be saved using this rule.

C. The Case When the Available Processors Exceed the Maximum Number of Processors Required in the Two-Stage Model

In this section we study the case where the available processors exceed the maximum number of processors required in the two-stage model. The exact solution is not feasible; hence, an approximation model will be given. The approximation result is a combination of the use of the exact result from Section IV and the use of the Scale-up rule.

Suppose the number of available processors is m times of the number of processors required by the high-concurrency stage. We first find the results of the system with the same job specification assuming the number of processors in the system equals the number of processors required by the high-concurrency stage. The result for this model is provided in Section IV. With these results, we apply the Scale-up rule to get the approximate mean response time. Some examples will help to show how good these approximation results are. Figures 10 to 13 give a comparison between simulation results and approximation results using the Scale-up rule. Two cases are given for two different jobs: one is with $\vec{P} = [\,1, 2\,]$ and $\vec{T} = [\,1, 1\,]$, and the other is with $\vec{P} = [\,2, 1\,]$ and $\vec{T} = [\,1, 1\,]$. Figures 10 and 11 show the result when the available number of processors is twice as many as the maximum number of processors required. Figures 12 and 13 shows the result when the available number of processors is five times as many as the maximum number of processors required.

FIGURES 10, 11, 12, 13 GO HERE.

VI. AN APPROXIMATION FOR THE GENERAL CASE

As mentioned earlier, to evaluate the performance of a general case exactly is extremely difficult. In this section, using the exact solution of the two-stage model and the Scale-up rule, we give an approximation method for a general job specification and any number of processors in the system. The simulation of this approximation method shows reasonably good results.

Given the job specification, we divide the service time for the job into two stages such that the mean service time for each stage is the same. In each stage, we average the work load over its service time to find the average number of processors needed. By doing this, we achieve a two-stage model as analyzed in Section IV. We use this two-stage model as the basis for the approximate model.

If the first stage requires fewer processors than the second stage, we use the result obtained in Section IVA. An example is shown in Figure 14. In Figure 14(a), the processor vector for this 4-stage job is [3,1,3,9] and the corresponding time vector is $\left[\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}\right]$. By dividing the time vector into two equal intervals, the first two stages of Figure 14(a) will be merged into the first stage of the approximate model as shown in Figure 14(b). Similarly, the last two stages of Figure 14(a) will be merged into the second stage of the approximation model as shown in Figure 14(b). In order to equate the work in Figure 14(b) to that in Figure 14(a), the processor vector for Figure 14(b) is [2,6] and the time vector is [1,1]. Figures 15 and 16 shows the approximation result for this example given 12 and 45 processors in the system, respectively. From these figures, we see that the approximation results are very close to the simulation results.

If the first stage requires more processors than the second stage, there is one more step to be applied in the approximation method. An example is given in Figure 17. In Figure 17(a), the processor vector is [3,9,3,1] and the time vector is $\left[\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}\right]$. Applying the same rule as we did to Figure 14, we convert Figure 17(a) into 17(b) such that the processor vector and the time vector of Figure 17(b) are [6,2] and [1,1] respectively. Since we have analyzed the system only when the number of processors required in the first stage is exactly twice that of the second stage, we must modify the two-stage approximate model by using a three-stage approximate model. The first stage of the three-stage model (as shown in Figure 17(c)) is the same as the first stage of the two-stage model (as shown in Figure 17(b)). The second stage of the three-stage model is modified such that it requires exactly half

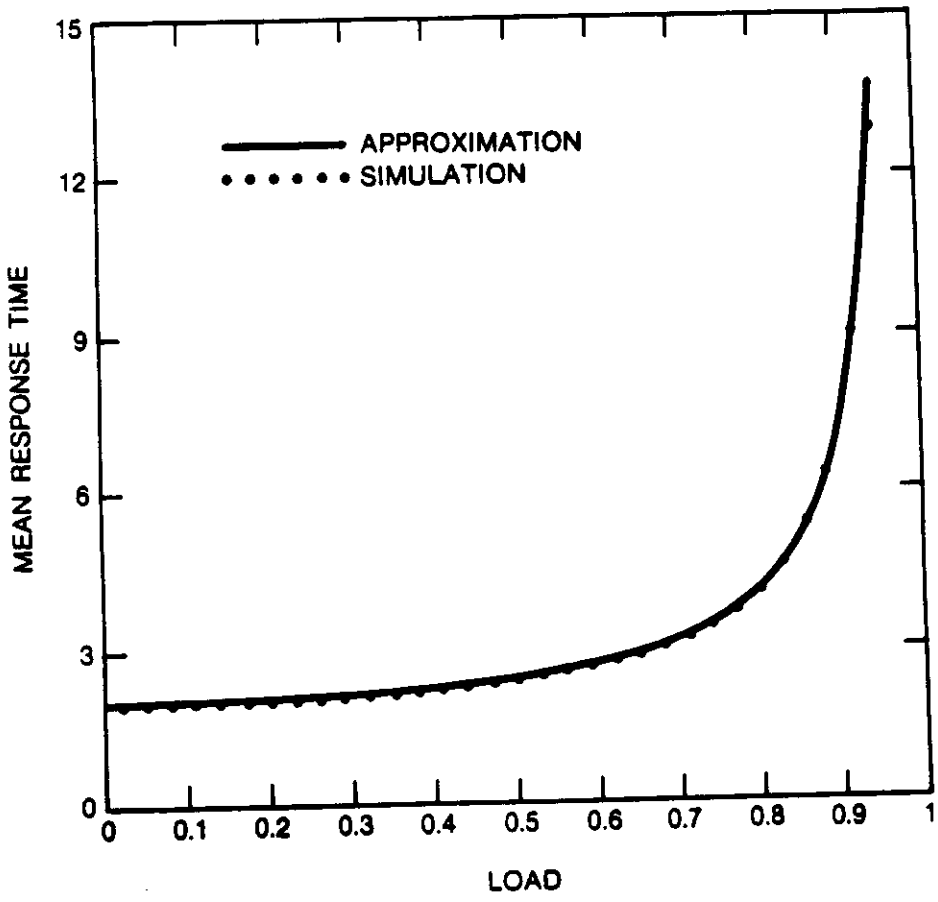

FIGURE 10. An Approximation Result for $\vec{P} = [1,2]$ and P=4

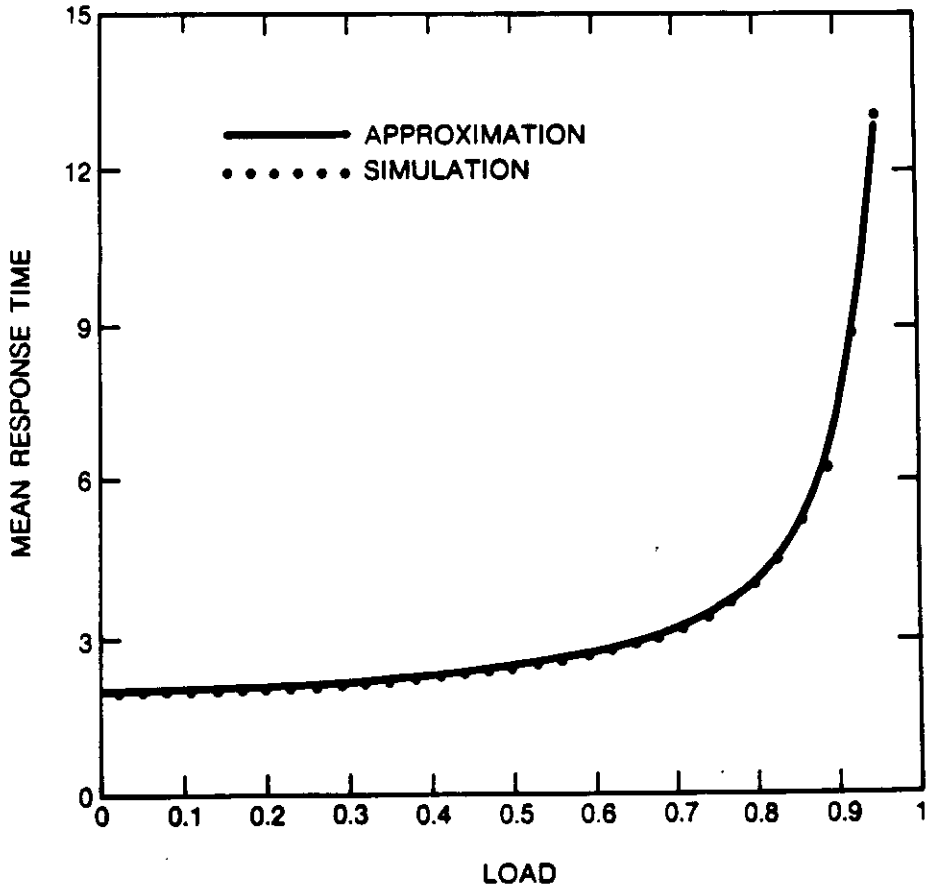

FIGURE 11. An Approximation Result for $\vec{P}$ = [2.1] and $_{P=4}$

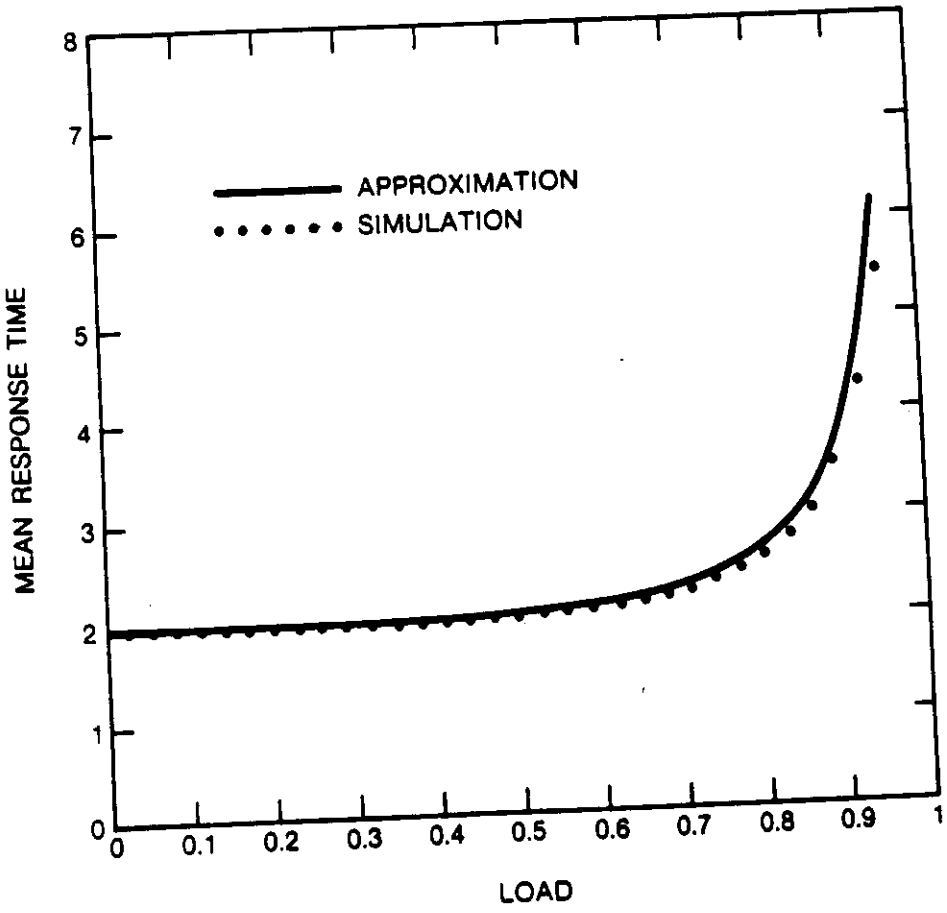

FIGURE 12. An Approximation Result for $\vec{P} = [1,2]$ and P=10

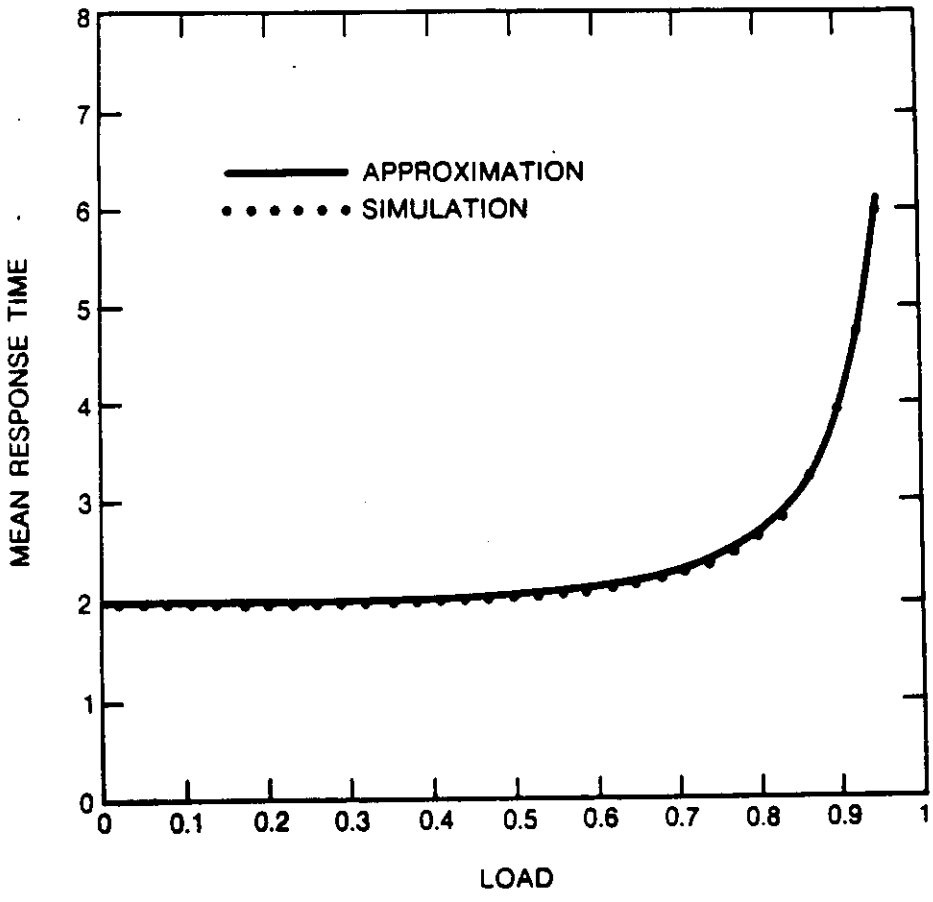

FIGURE 13. An Approximation Result for $\vec{P}$ = [2,1] and P=10

FIGURES 14a, 14b, 15, and 16 GO HERE.

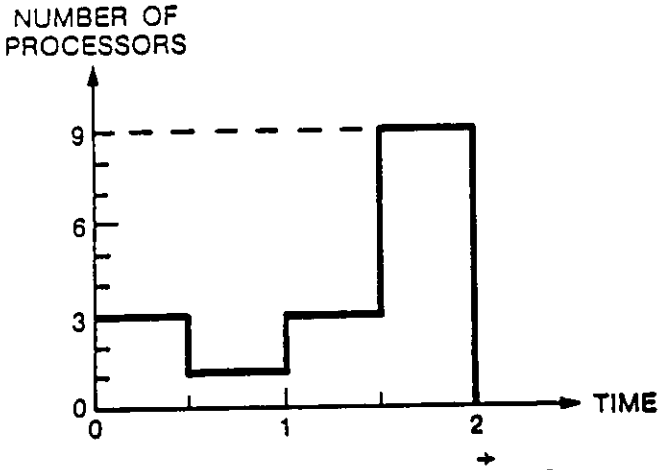

FIGURE 14a. An Example with $\vec{P} = [3,1,3,9]$ and $\vec{t} = [\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}]$

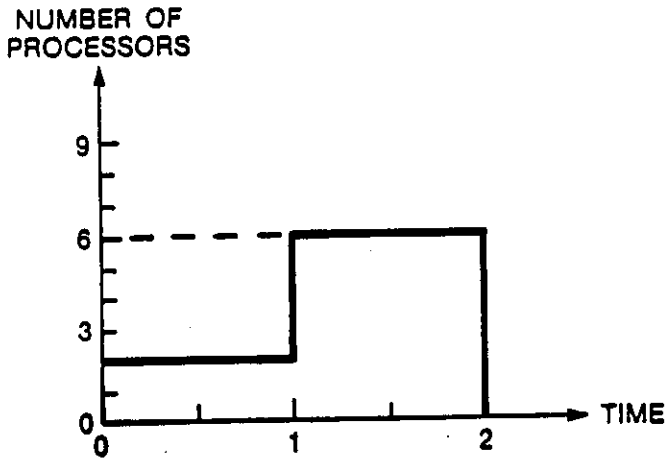

FIGURE 14b. An Approximation Model for Figure 14a

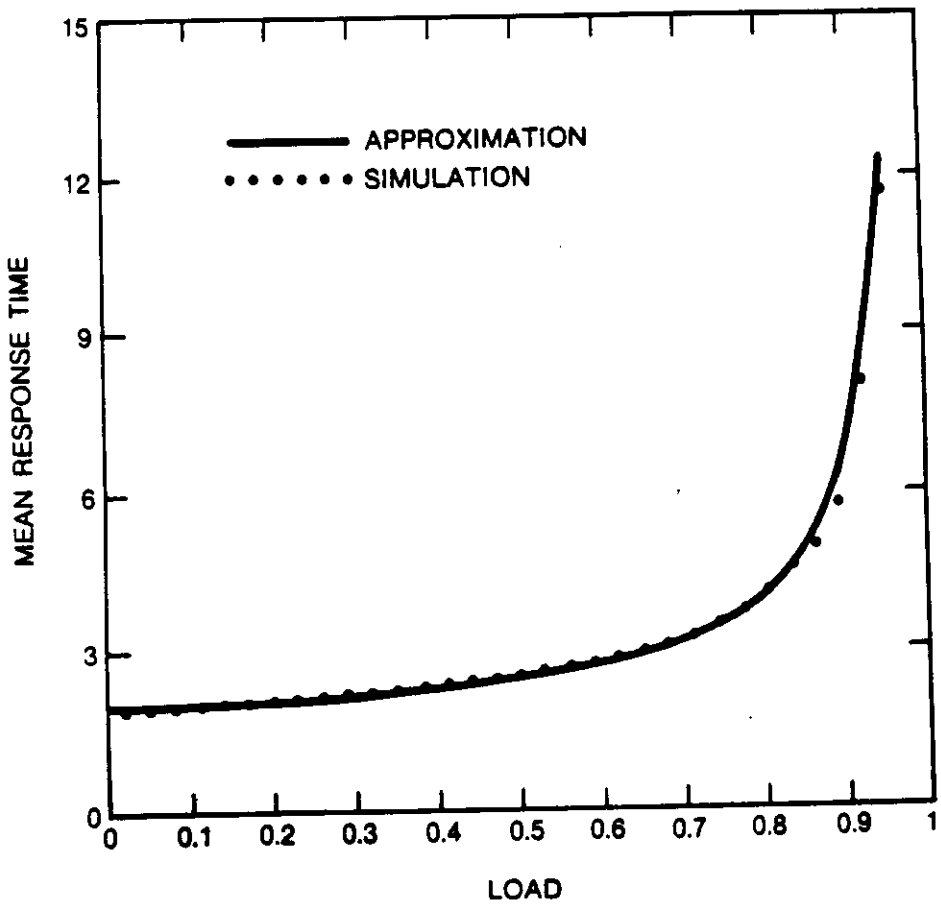

FIGURE 15. An Approximation Result for Figure 3.14a and P=12

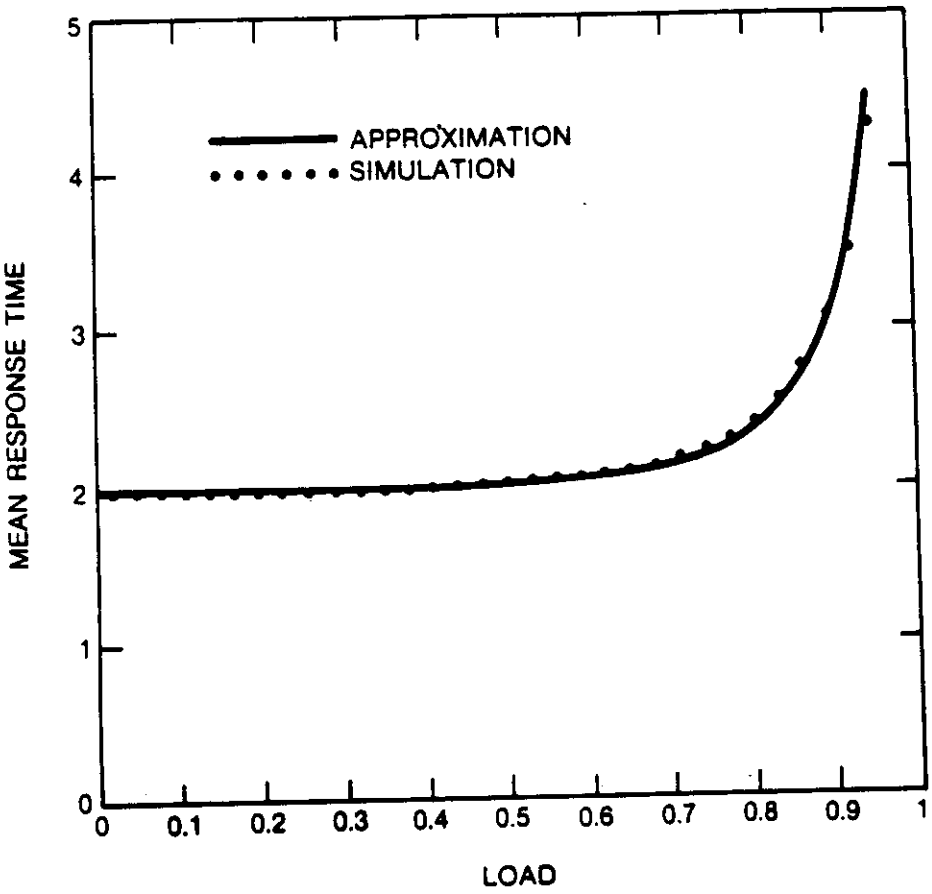

FIGURE 16. An Approximation Result for Figure 3.14a and P=45

FIGURES 17a, 17b, and 17c GO HERE.

of the processors required in the first stage and the total work required in this stage is the same as that of the second stage from the two-stage model. The third stage in the 3-stage model is used to adjust the service time such that it is the same as the 2-stage approximation model. After these approximations, the processor vector and the time vector for Figure 17(c) are [6,3,0] and [1, $\frac{2}{3}$, $\frac{1}{3}$] respectively. Although the model in Figure 17(c) is different from the model described in Section IVB, it can be solved by using the result from Section IVB plus an extra delay for stage three in Figure 17(c). Note that the third stage has the highest priority and requires no processors from the system at all; the existence of stage three has no impact on stages one and two. Hence, we can solve this problem by first neglecting the third stage in Figure 17(c) and apply the results obtained from Section IVB; from this result, we add

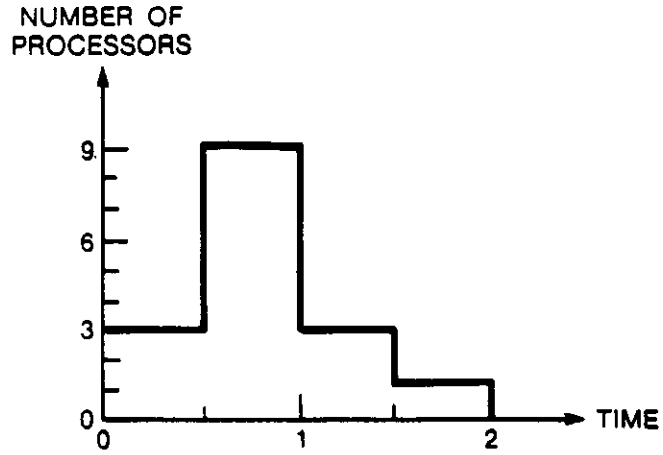

FIGURE 17a. An Example with $\vec{P} = [3,9,3,1]$ and $\vec{t} = [\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}]$

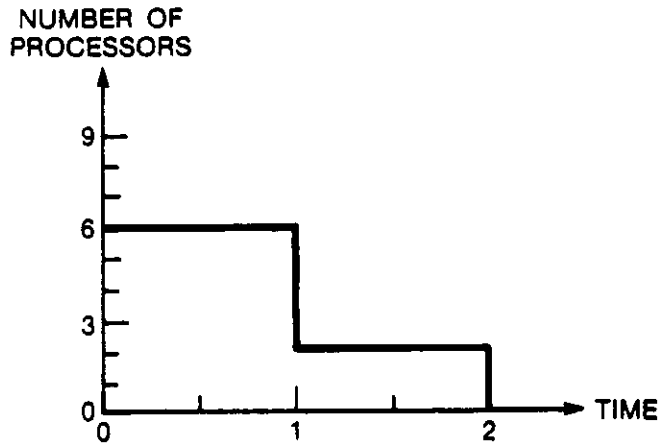

FIGURE 17b. The First Step Toward Approximation Model for Figure 17a

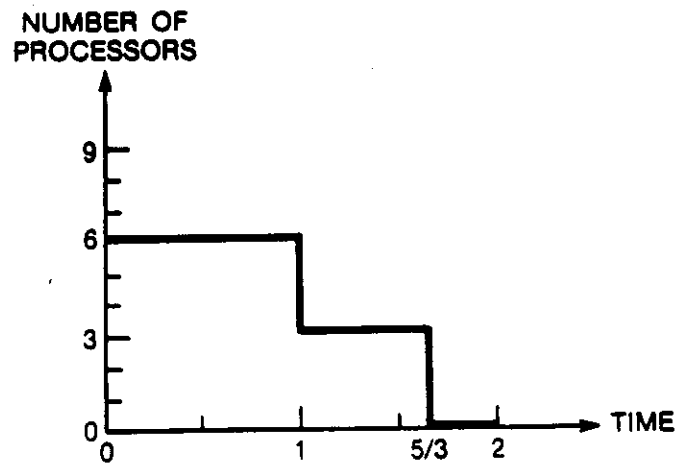

FIGURE 17c. The Second Step Toward Approximation Model for Figure 17a

the mean service time of stage three to it to get the overall mean response time. Figures 18 and 19 show the approximation results for this example given there are 12 and 45 processors in the system, respectively.

FIGURES 18 and 19 GO HERE.

VII. CONCLUSION

In this paper, we were able to analyze some interesting models exactly to study the behavior of jobs dynamically sharing P processors in a multi-processor system. By introducing the Scale-up rule, which is one of the major contributions of this paper, we were able to give an approximation method for finding the mean response time for a general application.

By looking at the comparisons in Sections IIIB and IVC, we have the following rule of thumb in designing parallel algorithms: If there is blocking in the algorithm, late blocking will generate a longer mean response time than early blocking if the service discipline is the same as given in this paper. Hence, if we have the choice, we would like to have early blocking rather than late blocking in the algorithm.

References

[1] Belghith, A., "Response Time and Parallelism in Parallel Processing Systems with Certain Synchronization Constraint," Ph.D. Dissertation, Computer Science Department, UCLA, 1986.

[2] Boxma, O.J., Cohen, J.W., and Huffels, N.,"Approximations of the mean waiting time in an M/G/c queueing system," *Oper. Res.* 27, 1979. pp. 1115-1127.

[3] Brumelle, S.L., "Some Inequalities for Parallel Server Queues," Operations Research, 19, 1971, pp 402-413.

[4] Cosmetatos, G.P., "Some Approximate Equilibrium Results for the Multi-server Queue (M/G/r)," Opnl. Res. Quart. 27, 1976, pp.615-620.

[5] Crommelin, C.D., "Delay Probability Formulae," P.O. Elec. Engrs. J. 26, 1934, pp. 266-274

[6] Hillier F.S. and Lo F.D., "Tables for Multiple-Server Queueing Systems Involving Erlang Distribution," Technical Report No. 31, Dept. of Operations Research, Stanford University, 1971.

[7] van Hoorn, M.H. and Tijms, H.C., "Approximations for the Waiting Time Distribution of the M/G/c Queue," Performance Evaluation 2, 1982, pp. 22-28.

[8] Huang, J.,"On the Behavior of Algorithms in a Multiprocessing Environment," Ph.D. dissertation, Computer Science Department, UCLA. 1988.

[9] Kingman J.F.C., "Inequalities in the Theory of Queues," Journal of the Royal statistical Society, Series B, 32, 1970, pp 102-110.

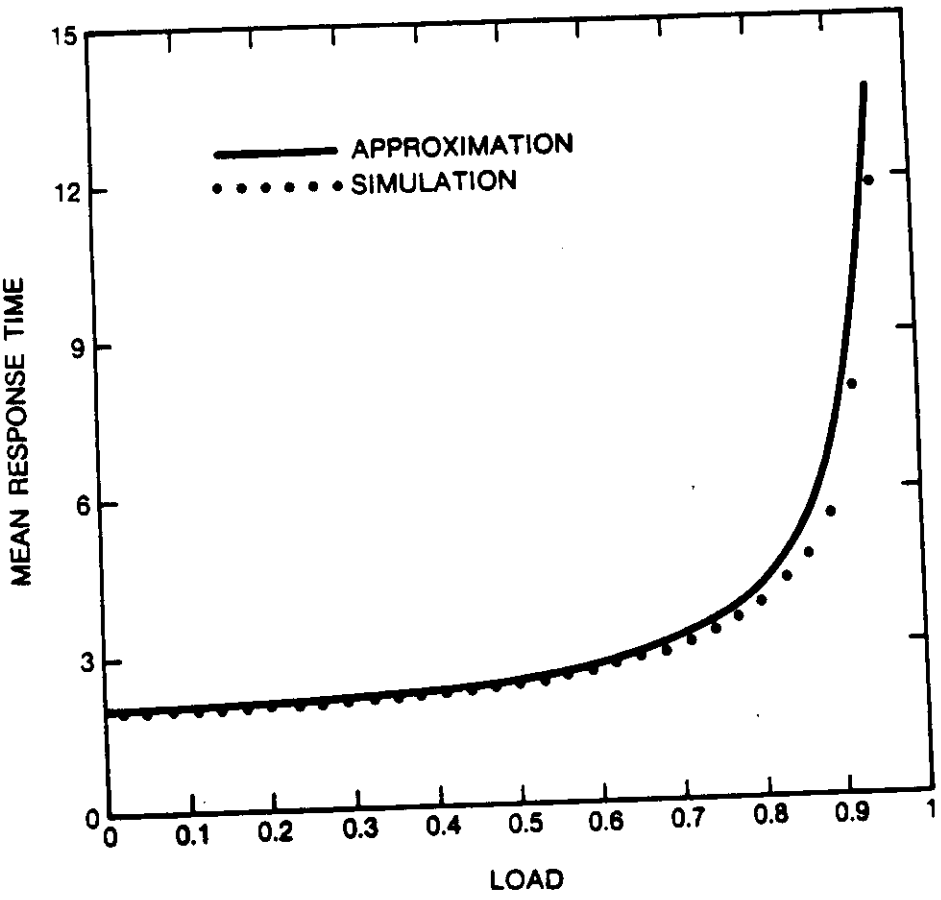

FIGURE 18. An Approximation Result for Figure 3.17a and P=12

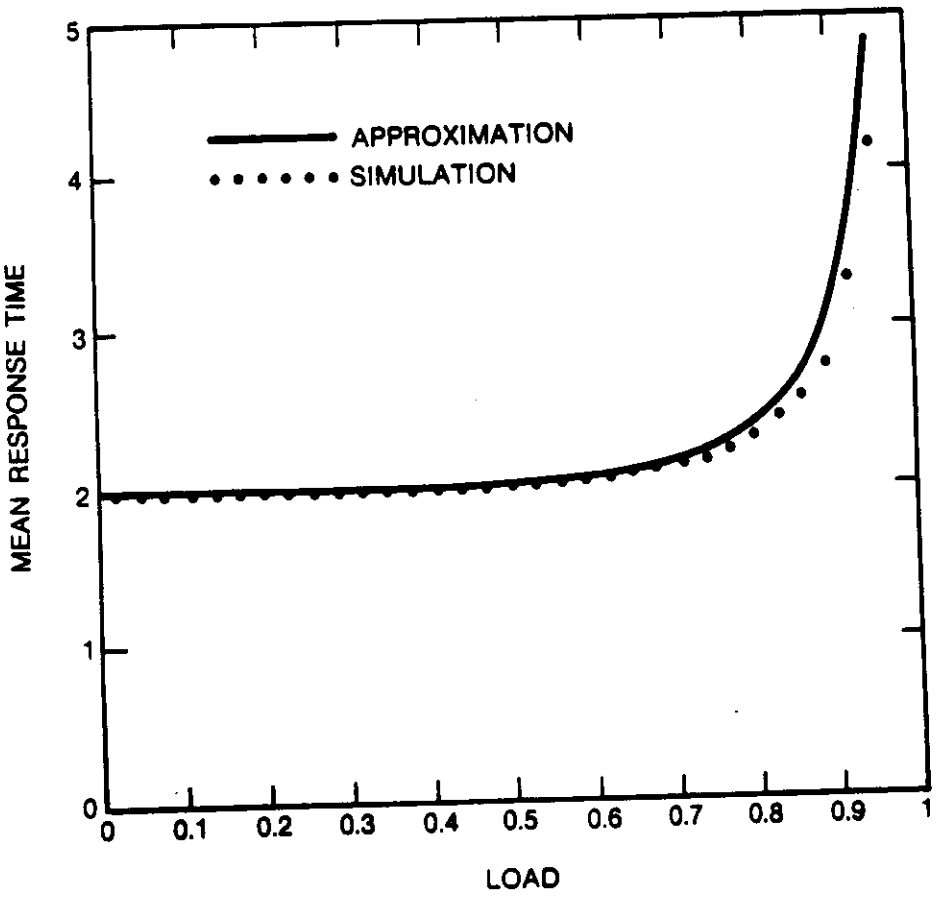

FIGURE 19. An Approximation Result for Figure 3.17a and P=45

[10] Kleinrock, L., *Queueing Systems, Vol. 2: Computer Applications*, Wiley- Interscience, New York, 1976.

[11] Kleinrock, L., "Performance Models for Distributed Systems," Teletraffic Analysis and Computer Performance Evaluation, Proceedings of the International Seminar held at the centre for Mathematics and Computer Science (CWI), June 2-6, 1986, Amsterdam, the Netherlands. pp. 1-15.

[12] Kleinrock, L. and Huang, J., "On Parallel Processing Systems: Amdahl's Law Generalized and Some Results on Optimal Design," submitted to IEEE Trans. on Computers, 1988.

[13] Little, J.D.C., "A Proof of the Queueing Formula $L = \lambda W$," *Operations Research*, 9, 1961. pp. 383-387.

[14] Maaloe, E., "Approximation Formulae for Estimation of Waiting-time in Multiple-Channel Queueing System," *mgmt. Sci.* 19, 1973. pp. 703-710.

[15] Nozaki, S.A., and Ross, S.M.,"Approximations in Multi-server Poisson Queues," Report University of California, Berkeley, 1975.

[16] Syski, R., *Introduction Congestion Theory in Telephone Systems (second edition)*, North Holland, 1984.

[17] Yakahashi, Y. "An Approximation Formula for the Mean Waiting Time of an M/G/m Queue," J. Oper. Res. Soc. Japan 20, 1977, pp. 150-163.

APPENDIX

A. Proof of Theorem 1:

From the Markov chain given in Figure 2, we have the following equilibrium balance equations:

$$(\lambda + k\mu_1)p(k,0) = \lambda p(k-1,0) + \mu_2 p(k,1) \qquad k \geq 1 \tag{A1}$$

$$(\lambda + \mu_2)p(k-1,1) = k\mu_1 p(k,0) + \lambda p(k-2,1) \qquad k \geq 2 \tag{A2}$$

The boundary conditions are

$$\lambda p(0,0) = \mu_2 p(0,1) \tag{A3}$$

$$(\lambda + \mu_2)p(0,1) = \mu_1 p(1,0) \tag{A4}$$

Define the following two z-transforms:

$$P_0(z) \triangleq \sum_{k=0}^{\infty} p(k,0)z^k$$

$$P_1(z) \triangleq \sum_{k=0}^{\infty} p(k,1)z^k$$

Define $p(k)$ to be the probability that there are a total of k jobs in the system. Each job is either in stage one or in stage two. Define the z-transform of $p(k)$ to be

$$P(z) \triangleq \sum_{k=0}^{\infty} p(k)z^k$$

From the definition we have

$$p(k) = p(k,0) + p(k-1,1) \qquad k \geq 1 \tag{A5}$$

$$p(0) = p(0,0) \tag{A6}$$

From (A5) and (A6) we have

$$P(z) = P_0(z) + zP_1(z) \tag{A7}$$

From (A1) and (A3) we have

$$\lambda P_0(z) + \mu_1 z \frac{d}{dz} P_0(z) = \lambda z P_0(z) + \mu_2 P_1(z) \tag{A8}$$

From (A2) and (A4) we have

$$(\lambda + \mu_2)P_1(z) = \mu_1 \frac{d}{dz} P_0(z) + \lambda z P_1(z) \tag{A9}$$

From (A8) and (A9) we have the following differential equation for $P_0(z)$.

$$(\lambda z - \mu_2)\mu_1 \frac{d}{dz} P_0(z) + \lambda(\lambda + \mu_2 - \lambda z)P_0(z) = 0$$

Solving this linear differential equation we obtain the following explicit expression of $P_0(z)$:

$$P_0(z) = c \cdot e^{\frac{\lambda}{\mu_1}(z - \ln|\lambda z - \mu_2|)} \tag{A10}$$

where c is a constant yet to be determined. From (A8) and (A9) we have

$$P_1(z) = \frac{\lambda}{\mu_2 - \lambda z} P_0(z) \tag{A11}$$

Combining (A7), (A10), (A11), and $P(1) = 1$ we find c to be

$$c = \frac{\mu_2 - \lambda}{\mu_2} e^{\frac{\lambda}{\mu_1}\left[1 - \ln(\mu_2 - \lambda)\right]}$$

From the above results we have the z-transform of the number of jobs in the system as:

$$P(z) = \frac{\mu_2 - \lambda}{\mu_2 - \lambda z} e^{\frac{\lambda}{\mu_1}\left[z - 1 + \ln\frac{\mu_2 - \lambda}{|\lambda z - \mu_2|}\right]}$$

We define $\bar{N}$ to be the average number of jobs in the system; hence,

$$\bar{N} = \frac{dP(z)}{dz}\Big|_{z=1} = \frac{\lambda(\mu_1 + \mu_2)}{\mu_1(\mu_2 - \lambda)}$$

Let us define the mean response time of this model to be T_{sp} where "sp" stands for serial-parallel. Using Little's result [13], we finally arrive at

$$T_{sp} = \frac{\bar{N}}{\lambda} = \frac{1/\mu_1 + 1/\mu_2}{1 - \lambda/\mu_2} \tag{A12}$$

B. Proof of Theorem 4

From the Markov chain given in Figure 6, we have

$$(\lambda + k\mu_1)p(k, 0) = \lambda p(k - 1, 0) + \mu_2 p(k, 1); \qquad 1 \le k \le n \tag{A13}$$

$$(\lambda + m\mu_1)p(k,0) = \lambda p(k-1,0) + \mu_2 p(k,1); \qquad m < k \tag{A14}$$

$$(\lambda + \mu_2)p(k-1,1) = \lambda p(k-2,1) + k\mu_1 p(k,0); \qquad 2 \le k \le m \tag{A15}$$

$$(\lambda + \mu_2)p(k-1,1) = \lambda p(k-2,1) + m\mu_1 p(k,0); \qquad m < k \tag{A16}$$

The boundary conditions are

$$\lambda p(0,0) = \mu_2 p(0,1) \tag{A17}$$

$$(\lambda + \mu_2)p(0,1) = \mu_1 p(1,0) \tag{A18}$$

We define the following two z-transforms:

$$P_0(z) \triangleq \sum_{k=0}^{\infty} p(k,0) z^k$$

$$P_1(z) \triangleq \sum_{k=0}^{\infty} p(k,1) z^k$$

We also define

$$p(k) \triangleq Prob[\ k\ jobs\ in\ the\ system\ ]$$

$$P(z) \triangleq \sum_{k=0}^{\infty} p(k) z^k$$

From the above definitions we have

$$P(z) = P_0(z) + zP_1(z)$$

From (A13) and (A14) we have

$$\sum_{k=1}^{m} (\lambda + k\mu_1)p(k,0) z^k + \sum_{k=m+1}^{\infty} (\lambda + m\mu_1)p(k,0) z^k = \sum_{k=1}^{\infty} \lambda p(k-1,0) z^k + \sum_{k=1}^{\infty} \mu_2 p(k,1) z^k$$

After some algebra we have

$$(\lambda + m\mu_1 - \lambda z)P_0(z) = \mu_2 P_1(z) + \mu_1 \sum_{k=0}^{m-1} (m-k)p(k,0)z^k \tag{A19}$$

Similarly, from (A15) and (A16) we have

$$\sum_{k=2}^{\infty} (\lambda + \mu_2)p(k-1,1)z^k = \sum_{k=2}^{\infty} \lambda p(k-2,1)z^k + \sum_{k=2}^{m} k\mu_1 p(k,0))z^k + \sum_{k=m+1}^{\infty} \mu_1 p(k,0)z^k$$

After some algebra we have

$$\left[(\lambda + \mu_2)z - \lambda z^2\right]P_1(z) = m\mu_1 P_0(z) - m\mu_1 - \mu_1 P_0(z) - \mu_1 \sum_{k=0}^{m-1} (m-k)p(k,0)z^k \tag{A20}$$

From (A19) and (A20) we have

$$P_1(z) = \frac{\lambda}{\mu_2 - \lambda z} P_0(z) \tag{A21}$$

Using (A19), (A20), and (A21) we have

$$P_0(z) = \frac{(\mu_2 - \lambda z)\mu_1}{-\lambda^2 z + m\mu_1\mu_2 - m\lambda\mu_1 z - \lambda\mu_2 z + \lambda^2 z^2} \sum_{k=0}^{m-1} (m-k)p(k,0)z^k$$

$$P_1(z) = \frac{\lambda\mu_1}{-\lambda^2 z + m\mu_1\mu_2 - m\lambda\mu_1 z - \lambda\mu_2 z + \lambda^2 z^2} \sum_{k=0}^{m-1} (m-k)p(k,0)z^k$$

Thus

$$P(z) = \frac{\mu_1\mu_2}{-\lambda^2 z + m\mu_1\mu_2 - m\lambda\mu_1 z - \lambda\mu_2 z + \lambda^2 z^2} \sum_{k=0}^{m-1} (m-k)p(k,0)z^k \tag{A22}$$

The remaining problem is to find all $p(k,0)$ for $0 \le k \le m-1$. Using (A22) and $P(1) = P_0(1) + P_1(1) = 1$, we have

$$\sum_{k=0}^{m} (m-k)p(k,0) = \frac{m\mu_1\mu_2 - m\mu_1\lambda - \mu_2\lambda}{\mu_1\mu_2} \tag{A23}$$

Rearranging (A13) and (A15) we have

$$p(k,1) = \frac{\lambda + k\mu_1}{\mu_2} p(k,0) - \frac{\lambda}{\mu_2} p(k-1,0); \qquad 1 \le k \le m \tag{A24}$$

$$p(k,0) = \frac{\lambda + \mu_2}{k\mu_1} p(k-1,1) - \frac{\lambda}{k\mu_1} p(k-2,1); \qquad 2 \le k \le m \tag{A25}$$

From (A17) and (A18) we have

$$p(0,1) = \frac{\lambda}{\mu_2} p(0,0) \tag{A26}$$

$$p(1,0) = \frac{\lambda(\lambda + \mu_2)}{\mu_1 \mu_2} p(0,0) \tag{A27}$$

From (A25), (A26), and (A27) we have

$$p(2,0) = \frac{\lambda^2[(\lambda + \mu_2)^2 + \lambda\mu_1]}{2\mu_1^2\mu_2^2} p(0,0) \tag{A28}$$

Applying (A24) to (A25) we have for $3 \le k \le m$,

$$p(k,0) = \frac{(\lambda + \mu_2)[\lambda + (k-1)\mu_1]}{k\mu_1\mu_2} p(k-1,0) - \frac{\lambda[\lambda + 2\mu_1(k-1)]}{k\mu_1\mu_2} p(k-2,0) - \frac{\lambda^2}{k\mu_1\mu_2} p(k-3,0) \tag{A29}$$

From (A27), (A28), (A29), and (A23) we are able to find all $p(k,0)$ for $0 \le k \le m-1$. Applying these results we obtain $P(z)$ and consequently $\bar{N}$ and T.

Differentiating (A22) we have

$$\bar{N} = \frac{d}{dz} P(z)\Big|_{z=1} = \frac{\mu_1\mu_2}{m\mu_1\mu_2 - m\lambda\mu_1 - \lambda\mu_2} \sum_{k=1}^{m-1} k(m-k) p(k,0) - \frac{\lambda^2 - m\lambda\mu_1 - \lambda\mu_2}{m\mu_1\mu_2 - m\lambda\mu_1 - \lambda\mu_2}$$

Hence,

$$T = \frac{\bar{N}}{\lambda} = \frac{\mu_1\mu_2}{\lambda(m\mu_1\mu_2 - m\lambda\mu_1 - \lambda\mu_2)} \sum_{k=1}^{m-1} k(m-k) p(k,0) - \frac{\lambda - m\mu_1 - \mu_2}{m\mu_1\mu_2 - m\lambda\mu_1 - \lambda\mu_2}$$