Performance Evaluation of Dynamic Sharing of Processors in Two-Stage Parallel Processing Systems

Jau-Hsiung Huang, *Member, IEEE*, and Leonard Kleinrock, *Fellow, IEEE*

***Abstract*—In this paper, we first study the performance of job scheduling in a large parallel processing system where a job is modeled as a concatenation of two stages which must be processed in sequence. We denote Pi as the number of processors required by stage i and denote P as the total number of processors in the system. The service time requirement of stage i, given the required Pi processors, is exponentially distributed with mean $1/\mu_i$. Hence, a job can be fully described by a quadruple (P_1, P_2, μ_1, μ_2). Three service disciplines which can fully utilize all processors in the system are studied in this paper.**

We first consider a *large* parallel computing system where $\text{Max}(P_1, P_2) \geq P >> 1$ and $\text{Max}(P_1, P_2) >> \text{Min}(P_1, P_2)$. For such systems, exact expressions for the mean system delay are obtained for various job models and disciplines. Our results show that the priority should be given to jobs working on the stage which requires fewer processors. We then relax the *large* parallel system (i.e., $P >> 1$) condition to obtain the mean system time for two job models when the priority is given to the second stage. Moreover, a *Scale-up Rule* is introduced to obtain the approximated delay performance when the system provides more processors than the maximum number of processors required by both stages (i.e., $P > \text{Max}(P_1, P_2)$). Lastly, an approximation model is given for jobs with more than two stages.

***Index Terms*—High-concurrency stage, low-concurrency stage, parallel processing system, processor sharing, Scale-up rule.**

I. Introduction

In this paper, we first consider a parallel computing system in which jobs are composed of two stages which must be processed in sequence. In each stage, up to a given maximum number of processors can be used concurrently and the number of processors required by different stages need not be the same. For instance, we can view a job as a program and a stage in a job as a procedure in the program where each procedure can be executed using a certain amount of processors concurrently. The number of processors required by a procedure depends on how many processors it can use in its algorithm. We denote P_i as the number of processors required by stage i and we denote P to be the total number of processors in the system. The service time of stage i given the required P_i processors is exponentially distributed with mean $1/\mu_i$. Hence, a job can be fully described by a quadruple (P_1, P_2, μ_1, μ_2). A general concept of this kind of job model was first proposed in [11].

Manuscript received July 19, 1990; revised June 3, 1991 and September 14, 1991. This work was supported by the Defense Advanced Research Projects Agency, Department of Defense, under Contracts MDA 903-87-C-0064 and MDA 903-87-0663.

J.-H. Huang is with the Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan.

L. Kleinrock is with the Computer Science Department, University of California, Los Angeles, Los Angeles, CA 90024.

IEEE Log Number 9205852.

For easier notation, we denote the *low-concurrency stage* as the stage where the number of processors required equals $\text{Min}(P_1, P_2)$. Similarly, we denote the *high-concurrency stage* as the stage where the number of processors required equals $\text{Max}(P_1, P_2)$. Hence, the job models considered in this paper may be either 1) the low-concurrency stage precedes the high-concurrency stage or 2) the high-concurrency stage precedes the low-concurrency stage. These two job models are denoted as the *LH (Low-High) job model* and *HL (High-Low) job model*, respectively.

Other than considering the characteristics of jobs, we propose three service disciplines for such systems. The basic principle of the service disciplines studied in this paper is to fully utilize all processors such that we do not allow the system to have idle processors while there are jobs waiting in the queue. To achieve this, the system has to share processors among jobs according to a service discipline. Two of the disciplines studied in this paper use priority schemes which assign the priority to a job according to which stage the job is working on. The third discipline follows a straight First Come First Serve rule. Hence the disciplines studied are 1) priority given to jobs working on stage 1 with preemption, 2) priority given to jobs working on stage 2 with preemption, and 3) FCFS without preemption. More details about job models and service disciplines are given in the following section.

For the 2-stage job model, we first find the mean system time of LH and HL job models under different kinds of disciplines. From the obtained mean system time analysis, we then find the best service discipline to minimize the mean system time for both LH and HL job models. These results shed light on designing the operating system for a parallel computing system.

The paper is organized as follows. In Section II, we give a detailed description to the job models and various service disciplines. In Section III, we assume the number of processors required by the low-concurrency stage is far fewer than that of the high-concurrency stage (i.e., $\text{Max}(P_1, P_2) >> \text{Min}(P_1, P_2)$) and we further assume the number of processors in the system is no more than the number of processors required by the high-concurrency stage (i.e., $P \leq \text{Max}(P_1, P_2)$). The performance of various combinations of job models and service disciplines are given and a comparison to find the best service discipline is also provided. In Section IV, we drop the $\text{Max}(P_1, P_2) >> \text{Min}(P_1, P_2)$ condition and find the delay performance using the discipline which gives the priority to jobs working on stage 2 with preemption. In Section V, we propose a *Scale-up Rule* which gives a good approximation

result of the mean system time when the number of processors in the system is more than the number of processors required by the high-concurrency stage (i.e., $P > \text{Max}(P_1, P_2)$). Section VI contains an approximation model for jobs with more than two stages. The concluding remarks are given in Section VII.

II. Model Description and Assumptions

In our model, jobs arrive to the system according to a Poisson process with rate λ. Whenever a job in a stage requires more processors than the system currently can provide, this job simply uses all the processors available to it with an appropriately elongated stage service time such that the work done for that stage remains unchanged. That is, the service time for that stage will still be exponentially distributed but with a larger mean. For example, if a job in a stage can use 10 processors for one second of work and if there are only 5 processors available, it will use these 5 processors and require two seconds of work to complete its work. This elongated stage service time with 5 processors ensures the conservation of the work required in that stage. As mentioned earlier, this is to fully utilize all the processors in the system. An example of an LH job model is shown in Fig. 1(a) and an HL job model is shown in Fig. 1(b) where $\text{Max}(P_1, P_2)/\text{Min}(P_1, P_2)$ equals m.

The first two service disciplines considered in this paper are a version of priority queueing with preemption. If the total number of processors occupied by higher priority jobs in the system is less than P, those processors which are not needed by these higher priority jobs are assigned to lower priority jobs. The assignment of priorities to a job is based upon which stage the job is currently working on. Hence, when a job finishes the first stage and advances to the second stage, the priority ranking of this job will be changed.

For instance, if the priority is assigned to jobs working on stage 2, then a job advances from stage 1 to stage 2 will gain a higher priority. Moreover, in cases where $P_2 > P_1$ and jobs working in stage 2 have higher priorities, a job advances from stage 1 to stage 2 will achieve a higher priority and will need more processors to work on stage 2. In this case, this job will preempt processors from those jobs working on stage 1 until either it has gained enough processors for stage 2 or there are no more jobs working on stage 1. Those jobs with all processors preempted will be pushed back to the head of the queue waiting for available processors. The third discipline considered in this paper does not allow processor preemption and follows a First Come First Serve rule.

Furthermore, for all three disciplines, if there are more than one job wanting to share processors in the same stage, we will first satisfy the processor requirement of the first job before allocating processors to the second job and so on. This process continues until all processors are allocated to jobs or until all jobs are satisfied. If there are jobs which do not receive processors, they stay in the queue of that stage. Hence, the service discipline for each stage is a FCFS scheme for all disciplines. The details of the service disciplines follows.

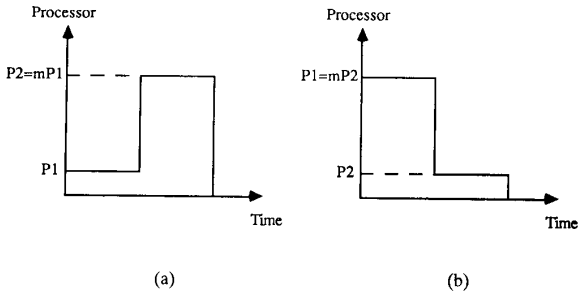

Fig. 1. LH and HL job models. (a) LH job model. (b) HL job model.

A. Priority Given to Jobs Working on Stage One with Preemption

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 decreases when it finishes stage 1 and enters stage 2. Under such a preemptive priority scheme, when a job enters the system while there are no available processors in the system, it can preempt processors from jobs working on stage 2, if there is any, to start servicing its stage 1.

B. Priority Given to Jobs Working on Stage Two with Preemption

In this discipline, a job's priority increases when it finishes stage 1 and enters stage 2. Hence, when a job enters stage 2 and requires more processors than it currently possesses from stage 1, it can preempt processors from jobs working on stage 1, if there is any, to start servicing its stage 2.

C. FCFS Discipline without Preemption

The discipline introduced here is a FCFS discipline where preemption is not allowed. A job is allowed to receive service as long as there are available processors in the system. Further, the processors occupied by a job cannot be preempted by any other jobs. However, if there are processors released by a job, those jobs already in service which need more processors have a higher priority than jobs in the queue to occupy the released processors. A job may release processors by either advancing from the high-concurrency stage to the low-concurrency stage or by finishing stage 2 and leaving the system.

III. Performance Evaluation of Different Job Models Under Various Disciplines

In this section, we consider a large parallel computing system where $\text{Max}(P_1, P_2) \geq P >> 1$ and $\text{Max}(P_1, P_2) >> \text{Min}(P_1, P_2)$. Before further discussion, we will first examine the cases when $P < \text{Max}(P_1, P_2)$. When $P < \text{Max}(P_1, P_2)$, the high-concurrency stage can actually use only P processors, hence the quadruple job description should be modified. For instance, if stage 2 is the high-concurrency stage (i.e., $P_2 > P > P_1$), then the quadruple (P_1, P_2, μ_1, μ_2) should be modified as $(P_1, P, \mu_1, P/P_2\mu_2)$. That is, stage 2 can use all P processors for an elongated service time. Hence, we will only consider the case when $P = \text{Max}(P_1, P_2)$ without loss of generality.

To give an example for the job model discussed in this section, the low-concurrency stage can be regarded as a *serial* stage which requires only one processor and the high-concurrency stage as a *parallel* stage which can use all processors in the system. For such a system, job models shown in Fig. 1(a) and (b) can be approximated as shown in Fig. 2(a) and (b), respectively since the number of processors required by the low-concurrency stage is negligible compared to that required by the high-concurrency stage. For the rest of this section, we will use these approximated job models for analysis. It will be shown in Section IV-A and Fig. 6 and when $\mathrm{Max}(P_1, P_2) \geq 20\mathrm{Min}(P_1, P_2)$, the delay performance of the job models shown in Fig. 1 is very close to the job models shown in Fig. 2, which will be analyzed in this section.

For such a system, the low-concurrency stage requires a negligible amount of total processors while the high-concurrency stage requires *all* P processors in the system. Therefore, if the high-concurrency stage has a higher priority, then there can be at most one job working on the high-concurrency stage since it will occupy all processors in the system. Under such a circumstance, all other jobs in the system will be forced to wait in the queue since there are no available processors left.

On the other hand, if the low-concurrency stage has a higher priority, then all jobs working on the low-concurrency stage can work concurrently and they will have no effect on jobs working on the high-concurrency stage because the processors taken by jobs in the low-concurrency stage are negligible. That is, in this case, there can be as many jobs working on the low-concurrency stage and one job working on the high-concurrency stage at the same time.

In this section, we will give results of the mean system time of LH and HL job models under various disciplines. Clearly, there will be six models from a combination of two job models and three scheduling disciplines. These six models are classified into three groups in according to the service disciplines. Note that the formula of infinite server queues are approximations to the actual system.

A. Priority Given to Jobs Working on Stage One with Preemption

In this section, we give the priority to jobs working on stage 1. For the two job models, we denote T_{LH} and T_{HL} as the mean system time for the LH job model and the HL job model, respectively. These notation will be used throughout this section.

Theorem 1: For systems with priority given to jobs working on stage 1 with preemption, the Laplace Transform, $Y^*_{HL}(s)$, of the system time of the HL job model is shown at the bottom of this page where ρ_1 equals λ/μ_1 and $G^*(s)$ is the root of the following quadratic equation:

$$G^*(s)^2 - \frac{s + \lambda + \mu_1}{\lambda} G^*(s) + \frac{\mu_1}{\lambda} = 0.$$

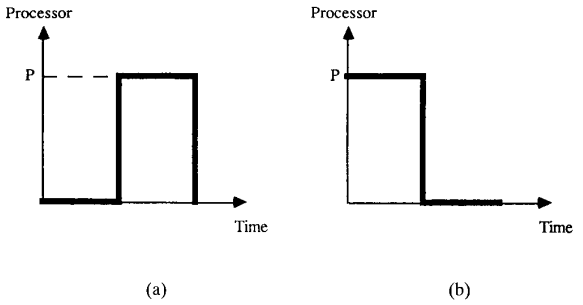

Fig. 2. The approximated LH and HL job models when $P = \mathrm{Max}(P_1, P_2) >> \mathrm{Min}(P_1, P_2)$. (a) LH job model. (b) HL job model.

Proof: See Appendix A.

Theorem 2: For systems with priority given to jobs working on stage 1 with preemption,

$$T_{HL} = \frac{1/\mu_1 + 1/\mu_2}{1 - \rho_1} + \frac{\lambda}{\mu_1^2(1 - \rho_1)^2} \tag{2}$$

$$T_{LH} = \frac{1}{\mu_1} + \frac{1/\mu_2}{1 - \lambda/\mu_2}. \tag{3}$$

Proof: Equation (2) can easily be derived from (1). To find T_{LH}, since stage 1 has higher priority, stage 1 can be regarded as an $\mathrm{M/M/\infty}$ system. Further, since jobs in stage 1 occupy no processors at all, stage 2 can be regarded as an $\mathrm{M/M/1}$ queue. Hence, T_{LH} can be shown as in (3). Q.E.D.

Note that $\lambda/\mu_1 (= \rho_1)$ is the system load for the HL job model since only stage 1 will contribute workload to the system. Similarly, $\lambda/\mu_2 (= \rho_2)$ is the system load for the LH job model. These can be seen from (2) and (3).

B. Priority Given to Jobs Working on Stage Two with Preemption

For systems which give priorities to jobs working on stage 2 with preemption, we first derive the z-transform of the number of jobs in the system as given in the following equation.

$$P(z) = \frac{\mu_2 - \lambda}{\mu_2 - \lambda z} \cdot \exp\left(\lambda/\mu_1 \cdot \left[z - 1 + \ln \frac{\mu_2 - \lambda}{|\lambda z - \mu_2|}\right]\right) \tag{4}$$

where $P(z)$ is defined as the z-transform of the number of jobs in the system. The proof of (4) is provided in Appendix B. From (4), we obtain the following theorem.

Theorem 3: For systems with priority given to stage 2 with preemption,

$$T_{LH} = \frac{1/\mu_1 + 1/\mu_2}{1 - \lambda/\mu_2} \tag{5}$$

$$T_{HL} = \frac{1/\mu_1}{1 - \lambda/\mu_1} + \frac{1}{\mu_2}. \tag{6}$$

$$Y^*_{HL}(s) = \frac{\mu_1^2 \mu_2 (1 - \rho_1)^2}{G^*(s)[s + \mu_1(1 - \rho_1)][S + \lambda - \lambda G^*(s) + \mu_2][s - \lambda G^*(s) + \mu_1]} \tag{1}$$

Proof: From (4), we can easily derive the mean number of customers in the system for the LH job model and we can hence prove (5) using Little's result [12]. Again, to find T_{HL} is easy since stage 1 performs like an M/M/1 queue and stage 2 is like an M/M/∞ queue. Q.E.D.

C. First Come First Serve Discipline

The performance of First Come First Serve discipline for the HL job model is the same as when the priority is given to stage 2 with preemption and the reason follows. Since jobs in stage 1 possess more processors than they will need in stage 2, hence whenever a job finishes stage 1 and enters stage 2, it does not need to ask for more processors than it already possesses. Actually it will release processors to other jobs. Hence, all jobs in stage 2 will always have enough processors. Therefore, the FCFS discipline for the HL job model is the same as the case when the priority is given to stage 2.

However, for the LH job model, the exact result is not obtained. Nonetheless, we can easily show that the performance is worse than the case when priority is given to stage 1 with preemption by the following reason. For FCFS discipline, whenever there is a job in stage 2, new arriving jobs cannot start receiving service for the low-concurrency stage since all processors are occupied by old jobs in stage 2. Hence, these new jobs would have to wait in the queue before receiving service for the low-concurrency stage. However, if the discipline used is to give priority to stage 1, then these new arriving jobs can immediately start receiving service for the low-concurrency stage and will not have to wait in the queue. Moreover, jobs in stage 1 do not really occupy processors; hence, it does not interfere with jobs working on stage 2. Therefore, the mean system time for the case when priority is given to stage 1 with preemption for the LH job model is better than that using the FCFS nonpreemptive service discipline. Applying the similar argument, we can show that the LH job model performs better than the case when priority is given to stage 2 with preemption. From these conclusions, we arrive at the following results for the FCFS system.

$$T_{HL} = \frac{1/\mu_1}{1-\lambda/\mu_1} + \frac{1}{\mu_2} \tag{7}$$

$$\frac{1/\mu_1 + 1/\mu_2}{1-\lambda/\mu_2} > T_{LH} > \frac{1}{\mu_1} + \frac{1/\mu_2}{1-\lambda/\mu_2}. \tag{8}$$

D. Performance Comparison Among Various Disciplines

Comparing (3), (5), and (8), we find that for LH job models, the system which gives priority to stage 1 with preemption has the smallest mean system time. On the other hand, for HL job models, we find that the system which gives priority to stage 2 with preemption and FCFS without preemption achieve the smallest average system time by comparing (2), (6), and (7). From the results derived above, we arrive at the following conclusions:

1) FCFS nonpreemptive discipline is not the optimal discipline.
2) The best service discipline depends on job models.
3) For the disciplines considered in this paper, the priority should be given to the low-concurrency stage which requires fewer processors.

Conclusion 3) above may not seem obvious. As a known fact, the best service discipline to minimize the mean system time is to give higher priority to jobs with less remaining service time if that is known. In our 2-stage job model, jobs working on stage 2 clearly are closer to completion than jobs working on stage 1; hence, the priority should be given to jobs working on stage 2 for both job models. However, from conclusion 3) stated above, we know that the priority should be given to stage 1 in the LH job model. This is opposite to our intuitive reasoning.

The reasons for this can be explained as follows. The problem of giving the priority to the high-concurrency stage is that by so doing, whenever there is a job in the high-concurrency stage, all jobs in the low-concurrency stage will be forced to wait since all processors are occupied by the job in the high-concurrency stage. Once all jobs in the high-concurrency stage finish services, all the processors will be released to jobs waiting to receive service for the low-concurrency stage. Since the number of processors required by all jobs working in the low-concurrency stage is small compared to the number of processors in the system, therefore, most of the processors will be idle. It is this waiting time in the low-concurrency stage and the inefficiency of utilizing processors which causes the poor performance.

On the other hand, if the priority is given to the low-concurrency stage, then all jobs in the low-concurrency stage would not have to wait in any case. Further, jobs in the high-concurrency stage can also receive service since the number of processors occupied by jobs in the low-concurrency stage is negligible. Hence, by giving the priority to the low-concurrency stage obtains a better delay performance for all job models. An example is shown in Fig. 3 for $(P_1, P_2, \mu_1, \mu_2) = (1, 20, 1, 1)$ and $P = 20$. Fig. 3 shows that the system with priority given to jobs working on stage 1 has the best performance while the performance of the system with FCFS discipline is very close to it. In Fig. 3, the mean system time for the FCFS discipline is obtained by simulations.

This conclusion can be further extended to a 3-stage job model where a job is composed of a low-concurrency stage followed by a high-concurrency stage followed by another low-concurrency stage as shown in Fig. 4. From the results obtained in Theorems 2 and 3 and by defining ρ_2 to be λ/μ_2, we can easily obtain the mean system delay for the following cases.

Case 1: If the highest priority is given to jobs working on stage 3, the next priority to jobs working on stage 2, and the lowest priority to jobs working on stage 1; we have

$$T_{LHL} = \frac{1/\mu_1 + 1/\mu_2}{1-\rho_2} + 1/\mu_3.$$

Case 2: If the highest priority is given to jobs working on stage 3, the next priority to jobs working on stage 1, and the lowest priority to jobs working on stage 2; we have

$$T_{LHL} = 1/\mu_1 + \frac{1/\mu_2}{1-\rho_2} + 1/\mu_3.$$

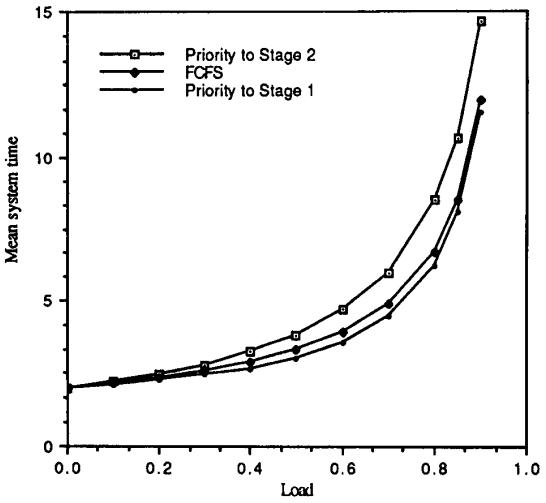

Fig. 3. The comparison for three service disciplines for $(P_1, P_2, \mu_1, \mu_2) = (1, 20, 1, 1)$ and $P = 20$.

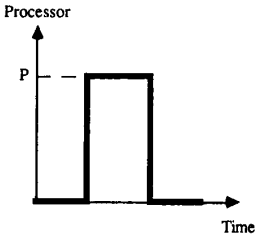

Fig. 4. A 3-stage job model with one high-concurrency stage and two low-concurrency stages.

Case 3: If the highest priority is given to jobs working on stage 1, the next priority to jobs working on stage 2, and the lowest priority to jobs working on stage 3; we have

$$T_{LHL} = 1/\mu_1 + \frac{1/\mu_2 + 1/\mu_3}{1-\rho_2} + \frac{\lambda}{\mu_2^2(1-\rho_2)^2}$$

Note that in Case 1, we give the priority to jobs closer to completion while in Case 2 the priority is given to jobs working on stages which require fewer processors. From these results, we can clearly see that Case 2 has the smallest mean system time. Hence, our conclusion in this section also works for 3-stage job models with one high-concurrency stage.

IV. Delay Analysis by Relaxing $\mathrm{Max}(P_1, P_2) >> \mathrm{Min}(P_1, P_2)$ Condition

In this section, we relax the $\mathrm{Max}(P_1, P_2) >> \mathrm{Min}(P_1, P_2)$ condition as required in the previous section. As before, we define m to be $\mathrm{Max}(P_1, P_2)/\mathrm{Min}(P_1, P_2)$ and denote P as the total number of processors and assume $P = \mathrm{Max}(P_1, P_2)$, i.e., the number of processors in the system equals the number of processors required by the high-concurrency stage. For cases when $P > \mathrm{Max}(P_1, P_2)$, a good approximation will be given after the introduction of the Scale-up rule, which will be described in the following section. In this section, we will only consider the discipline which gives the priority to stage 2 with preemption.

A. The LH Job Model

In this subsection, we consider the situation when the low-concurrency stage precedes the high-concurrency stage. This job can be modeled as shown in Fig. 1(a). The difference between this model and the one given in the previous section is that in this model there can be at most $\lceil m \rceil$ jobs working concurrently on the low-concurrency stage if there is no job working on the high-concurrency stage. As before, if there is a job working on the high-concurrency stage, all jobs in the low-concurrency stage will be forced to wait in the queue.

To draw a Markov chain for such a model, since $P_2 = mP_1$, we can normalize the number of processors required by stages by using $P_1 = 1$ and $P_2 = P = m$ without affecting the result. For example, a job model with $P_1 = 2$, $P_2 = 6$, and $P = 6$ should have the same performance with the job model with $P_1 = 1$, $P_2 = 3$, and $P = 3$. 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. Since the priority is given to stage 2, hence whenever $j > 0$, then all k jobs in stage 1 have to wait in the queue. Furthermore, all jobs in stage 1 will be paused whenever $j = 1$; hence no more job can advance from stage 1 into stage 2. Hence, the value of j can only be 0 or 1. If $j = 0$ and $k \leq m$, then all jobs in stage 1 are in service. If $j = 0$ and $k > m$, then there are m jobs working on stage 1 and the other $k - m$ jobs are waiting in the queue. Hence, the Markov chain of this model can be shown in Fig. 5.

Note that the overall system load can be expressed as $\lambda(m\mu_1 + \mu_2)/m\mu_1\mu_2$, hence the condition for a stable system is $\lambda < m\mu_1\mu_2/(m\mu_1 + \mu_2)$. We obtain the z-transform of the number of jobs in the system in the following equations.

For the LH job model as shown in Fig. 1(a), the z-transform of the number of jobs in the system is

$$P(z) = \frac{\mu_1\mu_2}{-\lambda^2 z + m\mu_1\mu_2 - m\lambda\mu_1 z - \lambda\mu_2 z + \lambda^2 z^2} \sum_{k=0}^{m-1} (m-k)p(k,0)z^k \tag{9}$$

where $p(k, 0)$ for $0 \leq k \leq m-1$ can be obtained from (10) to (13).

$$\sum_{k=0}^{m} \cdot (m-k)p(k,0) = \frac{m\mu_1\mu_2 - m\lambda\mu_1 - \lambda\mu_2}{\mu_1\mu_2} \tag{10}$$

$$p(1,0) = \frac{\lambda(\lambda+\mu_2)}{\mu_1\mu_2}\, p(0,0) \tag{11}$$

$$p(2,0) = \frac{\lambda^2[(\lambda+\mu_2)^2 + \lambda\mu_1]}{2\mu_1^2\mu_2^2}\, p(0,0). \tag{12}$$

For $k \geq 3$

$$p(k,0) = \frac{(\lambda+\mu_2)[\lambda+(k-1)\mu_1]}{k\mu_1\mu_2}\, p(k-1,0) - \frac{\lambda[\lambda+2(k-1)\mu_1]}{k\mu_1\mu_2}\, p(k-2,0) - \frac{\lambda^2}{k\mu_1\mu_2}\, p(k-3,0). \tag{13}$$

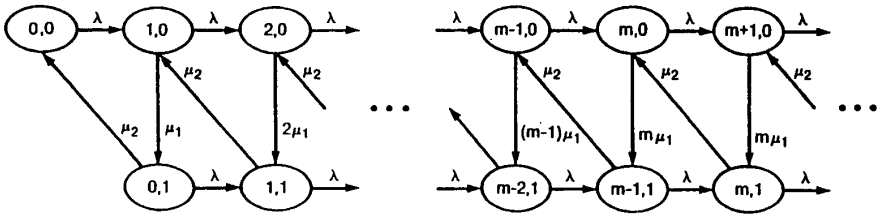

Fig. 5. Markov chain for LH job model.

The proofs of (9) to (13) are included in the Appendix C. From the above results, we can easily find the mean number of jobs in the system. Using the mean number of jobs in the system and Little's result [12], we can find the mean system time as follows.

$$T_{LH} = \frac{\mu_1\mu_2}{\lambda(m\mu_1\mu_2 - m\lambda\mu_1 - \lambda\mu_2)} \sum_{k=1}^{m-1} \cdot k(m-k)p(k,0) - \frac{\lambda - m\mu_1 - \mu_2}{m\mu_1\mu_2 - m\lambda\mu_1 - \lambda\mu_2} \tag{14}$$

where $p(k,0)$ for $0 \le k \le m-1$ can be obtained from (10) to (13).

Here we will compare the results shown in (5) and (14). For both cases, the job model considered are the LH job model. As mentioned in Section III, the results obtained from (5) should be a good approximation to that from (14) when $m >> 1$. Fig. 6 gives the delay performance of an example with $(P_1, P_2, \mu_1, \mu_2) = (1, 20, 1, 1)$ using (14) and another curve derived from (5) using $\mu_1 = \mu_2 = 1$. Fig. 6 shows that these two curves are very close to each other. Intuitively, we know that the results derived from (5) and (14) will be even closer for a larger m. This confirms our assumption in Section III.

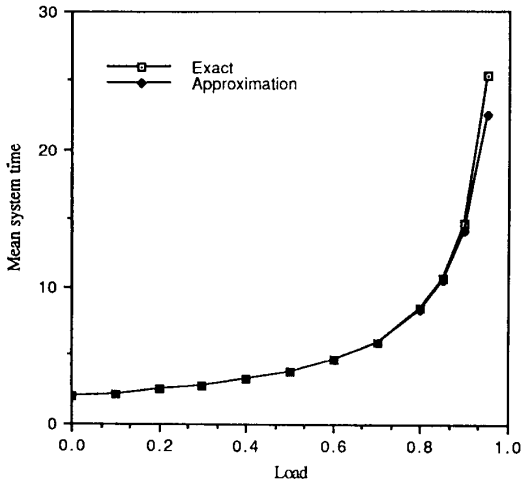

Fig. 6. Comparing (5) and (14) for $m = 20$.

B. The HL Job Model

We study the case when the high-concurrency stage precedes the low-concurrency stage in this subsection. As defined above, 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. Since the priority is given to stage 2, hence whenever there are j jobs working in stage 2 (i.e., $j = m = P$), then all k jobs in stage 1 are forced to wait in the queue. Therefore, there can be at most m jobs in stage 2. Unfortunately, we are not able to solve the general case for an arbitrary value of m. However, the result for $m = 2$ can be derived and are given at the bottom of this page. The Markov chain for $m = 2$ is shown in Fig. 7.

To prove (15) is easy but tedious. We omit the details which can be found in [7].

V. Scale-up Rule

The results derived in previous sections 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 (i.e., $P > \mathrm{Max}(P_1, P_2)$). 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 2-stage job model when $P > \mathrm{Max}(P_1, P_2)$ is provided in this section. For the rest of the paper, all simulation results are represented by 90% confidence interval using t-distribution and the approach used here is the "replicate/deletion" approach as indicated in [20, p. 551].

There has been considerable effort paid to the analysis of the mean waiting time of an M/G/n queue. Some of them provided bounds [2], [8] while some of them provided approximations [1], [4], [5], [6], [15], [16]. In this paper, we focus our attention of the results obtained in [13] and [14]

$$T_{HL} = \frac{(3\mu_2^2 + \mu_1\mu_2 + 2\mu_1^2)\lambda^2 - \mu_2(\mu_1^2 - \mu_1\mu_2 - 4\mu_2^2)\lambda - 4\mu_2^2(\mu_1^2 + 3\mu_1\mu_2 + 2\mu_2^2)}{\mu_2(\lambda + 2\mu_2)(\mu_1 + 2\mu_2)(\lambda\mu_1 + 2\lambda\mu_2 - 2\mu_1\mu_2)} \tag{15}$$

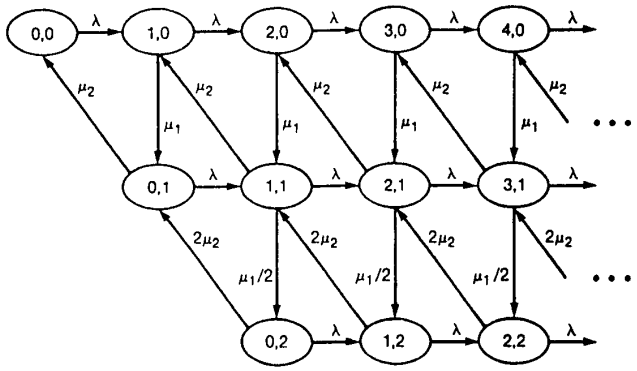

Fig. 7. Markov chain for HL job model.

which provided an approximation for such a system. In [1] and [3] this model was extended to achieve a better result. We define $W_{M/G/n}$ as the mean waiting time for an $M/G/n$ system. In [13] and [14] the following approximation for $W_{M/G/n}$ was suggested as shown in (16). This expression proved to be a very good approximation as some simulation results in [7] demonstrated.

$$W_{M/G/n} \simeq \frac{W_{M/G/1}}{W_{M/M/1}} \cdot W_{M/M/n}. \tag{16}$$

A queueing model with a varying number of required processors has been studied in some previous works [17]–[19]. However, in these works, none of them tried to make a full use of all processors. That is, only our model will not allow cases where there are available processors idle in the system while there are also jobs waiting in the queue.

In this subsection, we will extend (16) to the 2-stage job model when $P > \text{Max}(P_1, P_2)$. Although the number of processors required by a job varies during its execution on different stages, we were surprised, and pleased, to discover that the rule which applies to the classical queueing system as stated in (16) also applies in our model. In other words, if we have to find the mean waiting time for a parallel system with $P > \text{Max}(P_1, P_2)$, we will first find the mean waiting time for the system with P' processors where $P' = \text{Max}(P_1, P_2)$ using the method derived in the previous sections. From the result obtained for $P' = \text{Max}(P_1, P_2)$, we apply the following Scale-up rule to obtain the result for system with P processors $(P > P')$.

Scale-up Rule: Given a system with P processors and the job model (P_1, P_2, μ_1, μ_2), we want to find the mean waiting time if $P > \text{Max}(P_1, P_2)$. We first obtain the mean waiting time of the system assuming the number of processors in the system, denoted as P', equals $\text{Max}(P_1, P_2)$. This result can be obtained from the previous section. We denote the mean waiting time for such a system with P' processors as $W_{M/JM/1}(\rho)$, where ρ is the system load and JM stands for "Job Model." For the original system with P processors and denoting P/P' as n $(n > 1)$, we define $W_{M/JM/n}(\rho)$ to be the mean waiting time of the system with $P = nP'$ processors. Similarly, we define $W_{M/M/n}(\rho)$ to be the mean waiting time of an ordinary $M/M/n$ queueing system with mean service time equals $1/\mu_1 + 1/\mu_2$. The Scale-up Rule says that

$$W_{M/JM/n}(\rho) \simeq \frac{W_{M/JM/1}(\rho)}{W_{M/M/1}(\rho)} \cdot W_{M/M/n}(\rho). \tag{17}$$

Since the parameters in the right hand side of (17) are all known values, hence $W_{M/JM/n}(\rho)$ in the left hand side can be calculated. The approximation result is a combination of the use of the exact result from Section IV and the use of the Scale-up rule. From the obtained mean waiting time, we can easily obtain the mean system time by adding into the mean service time which equals $1/\mu_1 + 1/\mu_2$. Some examples are given to show how good these approximation results are. Figs. 8 to 11 give a comparison between simulation results and approximation results using the Scale-up rule where the priority is given to stage 2. The approximation results are obtained by first exactly finding the mean system time for $P = 2$ using the technique given in Section IV, then the scale-up rule is used to obtain the results for $P > 2$. Two cases are given for two different job models. One is described as $(P_1, P_2, \mu_1, \mu_2) = (1, 2, 1, 1)$ and the other is described as $(P_1, P_2, \mu_1, \mu_2) = (2, 1, 1, 1)$. Figs. 8 and 9 show the results when $P = 4$ and Figs. 10 and 11 show the result when $P = 10$. From these figures we show that the Scale-up Rule is indeed a very good approximation method.

VI. An Approximation for the General Cases

As we mentioned earlier, to exactly evaluate the performance of a general case with N $(N > 2)$ stages is extremely difficult. In this section, using the exact solution of the 2-stage model and the scale-up rule, we give an approximation method for any general processor-time task graph and any number of processors in the system. In this model, we give higher priority to jobs closer to completion. That is, jobs in the last stage (the Nth stage) have the highest priority while jobs in the first stage have the lowest priority. Again, preemption is assumed. The simulation of this approximation method shows reasonably good results.

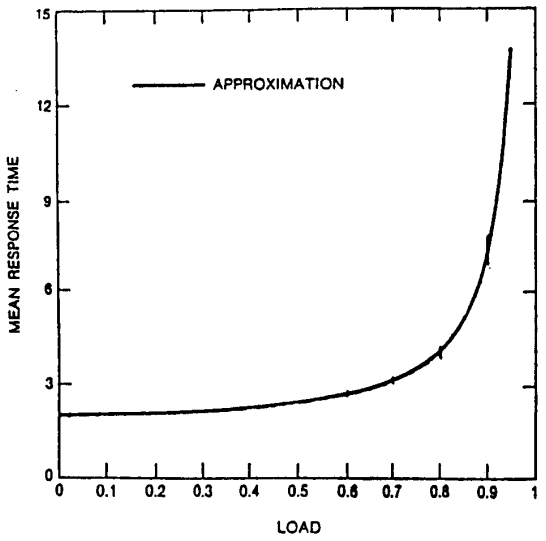

Fig. 8. An approximation result for job model $(1, 2, 1, 1)$ and $P = 4$.

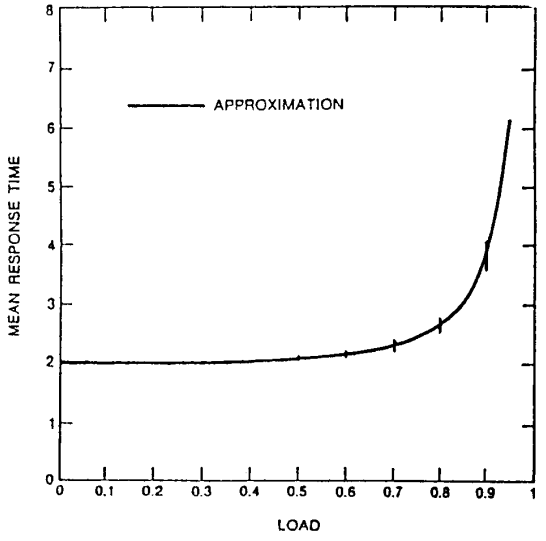

Fig. 10. An approximation result for job model $(1, 2, 1, 1)$ and $P = 10$.

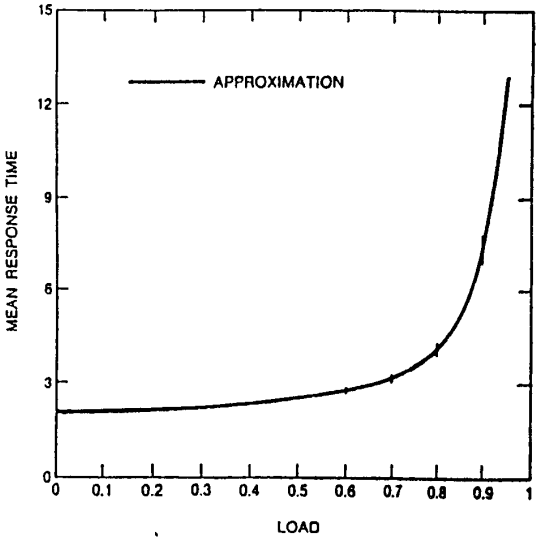

Fig. 9. An approximation result for job model $(2, 1, 1, 1)$ and $P = 4$.

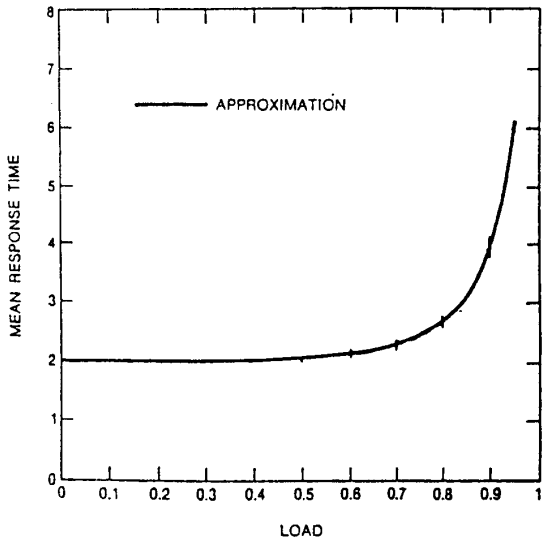

Fig. 11. An approximation result for job model $(2, 1, 1, 1)$ and $P = 10$.

Given a processor-time task graph, we divide the processor-time task graph into two pieces such that the mean service time for each piece is the same. In each piece, we average the work load over its service time to find the *average* number of processors needed. By so doing, we achieve a 2-stage model as analyzed in Section IV. We use this modified 2-stage model as the basis for the approximation model.

If the first stage requires fewer processors than the second stage in the modified model, we use the result obtained in Section IV-A. An example is shown in Fig. 12. In Fig. 12(a), the processor vector for this 4-stage job is $[3, 1, 3, 9]$ and the corresponding time vector is $[1/2, 1/2, 1/2, 1/2]$. By dividing the time vector into two equal pieces, the first two stages of Fig. 12(a) will be merged into the first stage of the modified model as shown in Fig. 12(b). Similarly, the last two stages of Fig. 12(a) will be merged into the second stage of the modified model as shown in Fig. 12(b). In order to make the workload in Fig. 12(b) to be the same as in Fig. 12(a), the processor vector of Fig. 12(b) is $[2, 6]$ and the time vector is $[1, 1]$. Figs. 13 and 14 show the approximation results 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 in the modified model, there is one more step to be done in the approximation method. An example is given in Fig. 15. In Fig. 15(a), the processor vector is $[3, 9, 3, 1]$ and the time vector is $[1/2, 1/2, 1/2, 1/2]$. Applying the same rule as we did in Fig. 12, we convert Fig. 15(a) into Fig. 15(b) such that the processor vector and the time vector of Fig. 15(b)

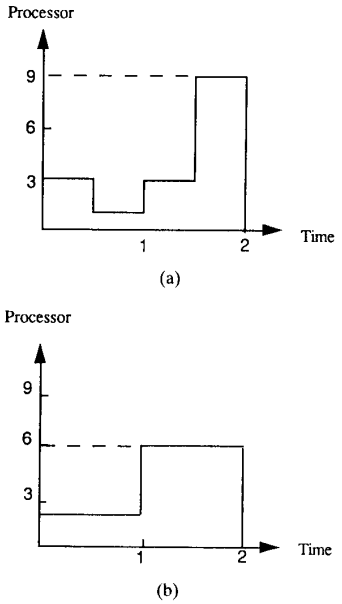

Fig. 12. (a) An example with $[P_1, P_2, P_3, P_4, \mu_1, \mu_2, \mu_3, \mu_4] = [3, 1, 3, 9, 1/2, 1/2, 1/2, 1/2]$. (b) An approximation model for (a).

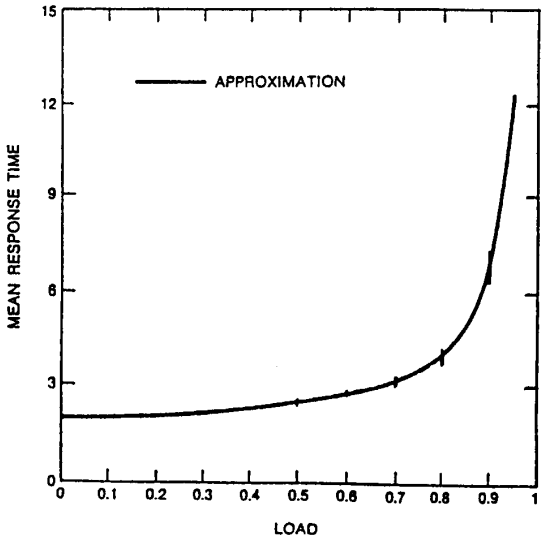

Fig. 13. An approximation result for Fig. 12(a) and $P = 12$.

are $[6, 2]$ and $[1, 1]$, respectively. Since we can analyze the system only when the number of processors required in the first stage is exactly twice of that in the second stage, we have to modify the modified 2-stage model by using a modified 3-stage model. The first stage of the modified 3-stage model [as shown in Fig. 15(c)] is the same as the first stage of the modified 2-stage model [as shown in Fig. 15(b)]. The second stage of the modified 3-stage model is modified such that it requires exactly half of the processors required in the first stage (of the modified 3-stage model) and the total workload required in the stage is the same as that of the second stage

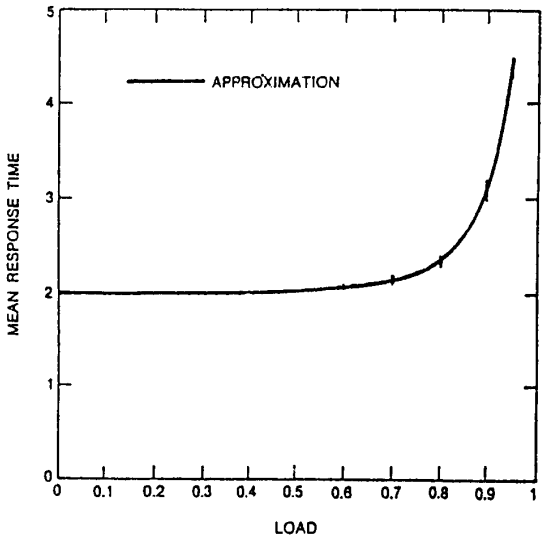

Fig. 14. An approximation result for Fig. 12(a) and $P = 45$.

from the 2-stage model. The third stage in the 3-stage model is used to adjust the no-queueing service time such that it is the same for the modified 2-stage model and the modified 3-stage model.

The processor vector and the time vector for Fig. 15(c) are $[6, 3, 0]$ and $[1, 2/3, 1/3]$, respectively. Although the model in Fig. 15(c) is different from the model described in Section IV-B, it can be solved by using the result from Section IV-B. Notice 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 Fig. 15(c) and apply the results obtained from Section IV-B; from this result, we add the mean service time of stage three to it to get the overall mean response time. Figs. 16 and 17 show the approximation results for this example given there are 12 and 45 processors in the system, respectively.

VII. Conclusions

In this paper we are able to find the average system delay of various job models and disciplines when $\text{Max}(P_1, P_2) \geq P >> 1$ and $\text{Max}(P_1, P_2) >> \text{Min}(P_1, P_2)$ in a large parallel processing system. Further, we achieve the following conclusion that the priority should be given to jobs working on the low-concurrency stage to achieve a better delay performance. This priority assignment scheme remains true for 3-stage job models with a high-concurrency stage preceded and followed by low-concurrency stages. By dropping the $\text{Max}(P_1, P_2) >> \text{Min}(P_1, P_2)$ condition, we also obtain the mean system delays for cases when the priority is given to jobs working on stage 2. A Scale-up Rule is further proposed which gives very good approximation results for systems when the number of processors in the system is greater than $\text{Max}(P_1, P_2)$. Finally, an approximation model for the general cases with N $(N > 2)$ stages is included which shows reasonably good results.

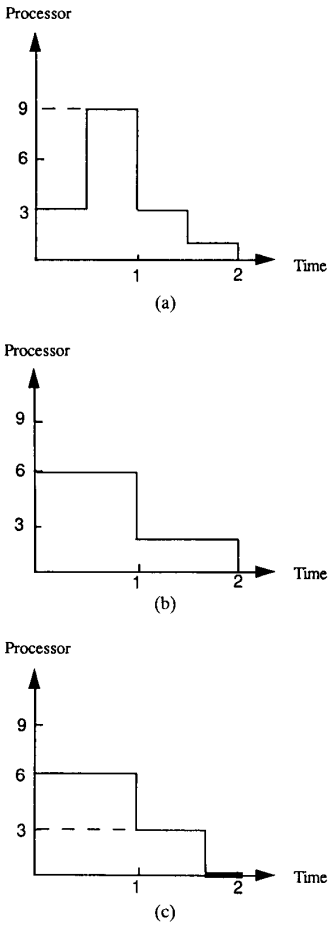

Fig. 15. (a) An example with $[P_1, P_2, P_3, P_4, \mu_1, \mu_2, \mu_3, \mu_4] = [3, 9, 3, 1, 1/2, 1/2, 1/2, 1/2]$. (b) The first step toward approximation model for (a). (c) The second step toward approximation model for Fig. 15(a).

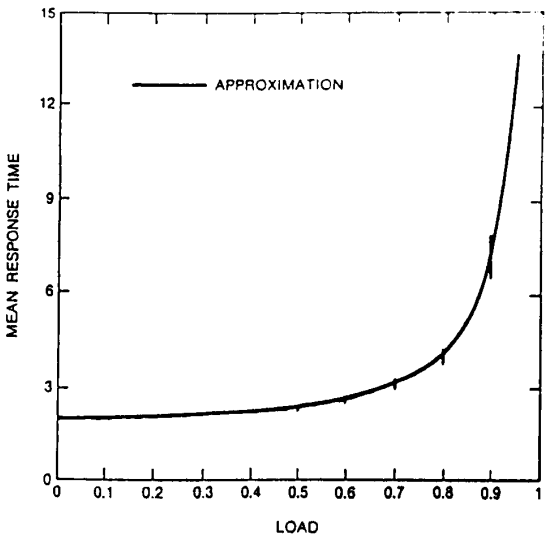

Fig. 16. An approximation result for Fig. 15(a) and $P = 12$.

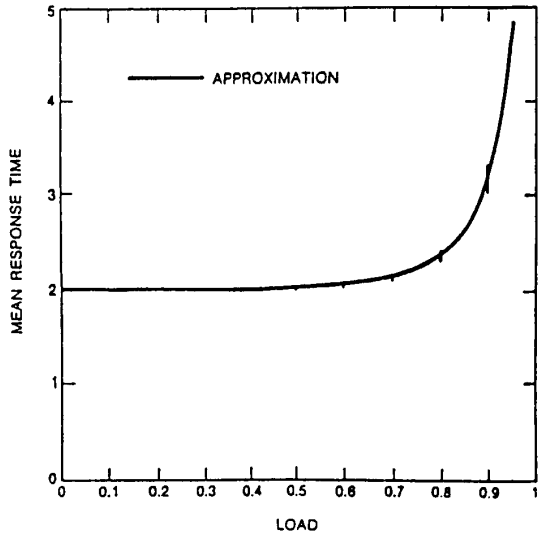

Fig. 17. An approximation result for Fig. 15(a) and $P = 45$.

APPENDIX A

A. Proof of Theorem 1

Proof: For the HL job model, we can regard stage 1 as an M/M/1 queue. Hence, we have the Laplace transform of the system time for stage 1, denoted as $Y_1^*(s)$, given in [9, p. 195]

$$Y_1^*(s) = \frac{\mu_1(1-\rho_1)}{s+\mu_1(1-\rho_1)}. \tag{A1}$$

For stage 2, the system time can be found by using the delay cycle analysis given in [10, p. 110]. When stage 1 is idle, jobs in stage 2 have an exponential service time distribution with parameter μ_2. However, when stage 1 becomes busy, jobs in stage 2 will all be paused until stage 1 becomes idle again and resume the work. Clearly, the system time for stage 2 can be modeled as a delay cycle. The distribution of the initial delay cycle can be found in the following.

It is shown in [9] in page 176 that in an M/G/1 system, the probability that a departure finds k jobs in the system equals the probability that there are k jobs in the system in equilibrium. Hence,

$$Pr[\text{a departure from stage 1 finds } k \text{ jobs still in stage 1}] = (1-\rho_1)\rho_1^k. \tag{A2}$$

A departure from stage 1 to begin receiving service in stage 2 finds k jobs still in stage 1 has an initial delay cycle whose Laplace transformed distribution, denoted as $G_0^{(k)}(s)$, can be represented as

$$G_0^{(k)}(s) = \left(\frac{\mu_1}{s+\mu_1}\right)^k \cdot \frac{\mu_2}{s+\mu_2}. \tag{A3}$$

From the delay cycle analysis formula [10], we have the Laplace transform of the system time for stage 2, denoted as

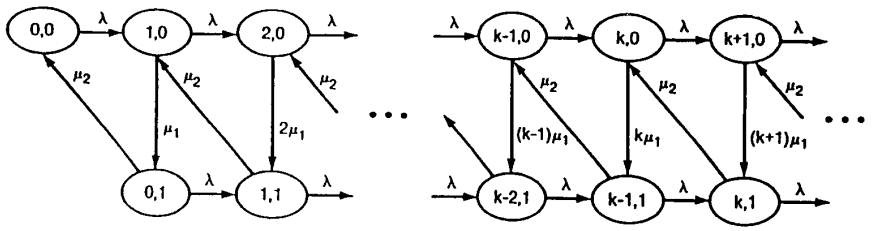

Fig. 18. Markov chain for LH job model.

$Y_2^*(s)$, given as

$$\begin{aligned} Y_2^*(s) &= \sum_{k=0}^{\infty} \cdot (1-\rho_1)\rho_1^k \cdot G_0^{(k)}(s+\lambda-\lambda G^*(s)) \\ &= \sum_{k=0}^{\infty} \cdot (1-\rho_1)\rho_1^k \cdot \left[\frac{\mu_1}{s+\lambda-\lambda G^*(s)+\mu_1}\right]^k \\ &\quad \cdot \frac{\mu_2}{s+\lambda-\lambda G^*(s)+\mu_2} \\ &= \mu_2(1-\rho_1) \cdot \frac{1}{s+\lambda-\lambda G^*(s)+\mu_2} \\ &\quad \cdot \frac{s+\lambda-\lambda G^*(s)+\mu_1}{s-\lambda G^*(s)+\mu_1} \\ &= \frac{\mu_1\mu_2(1-\rho_1)}{G^*(s)[s+\lambda-\lambda G^*(s)+\mu_2][s-\lambda G^*(s)+\mu_1]} \end{aligned}$$

where ρ_1 equals λ/μ_1 and $G^*(s)$ is the Laplace transform of the busy period distribution of stage 1.

From (A1), (A2), and (A3), the Laplace Transform, $Y_{\mathrm{HL}}^*(s)$, of the system time of the HL job model equals $Y_1^*(s) \cdot Y_2^*(s)$ as shown in (1). Q.E.D.

B. Proof of (4)

Proof: By defining state (n_1, n_2) as n_1 jobs in stage 1 and n_2 jobs in stage 2, we have the Markov chain given in Fig. 18 and the following equilibrium balance equations:

$$(\lambda + k\mu_1)p(k,0) = \lambda p(k-1,0) + \mu_2 p(k,1) \qquad k \geq 1 \tag{B1}$$

$$(\lambda + \mu_2)p(k-1,1) = k\mu_1 p(k,0) + \lambda p(k-2,1) \qquad k \geq 2 \tag{B2}$$

$$\lambda p(0,0) = \mu_2 p(0,1) \tag{B3}$$

$$(\lambda + \mu_2)p(0,1) = \mu_1 p(1,0). \tag{B4}$$

We define $p(k)$ to be the probability that there are totally k jobs in the system and define the following notation:

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

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

Since $p(k) = p(k,0) + p(k-1,1)$, it can easily be shown that $P(z) = P_0(z) + zP_1(z)$. From (B1) and (B3) we have

$$\lambda P_0(z) + \mu_1 z \frac{d}{dz} P_0(z) = \lambda z P_0(z) + \mu_2 P_1(z). \tag{B5}$$

From (B2) and (B4) we have

$$(\lambda + \mu_2)P_1(z) = \mu_1 \frac{d}{dz} P_0(z) + \lambda z P_1(z). \tag{B6}$$

From (B5) and (B6) 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 $P_0(z)$:

$$P_0(z) = c \cdot \exp((\lambda/\mu_1)(z - \ln|\lambda z - \mu_2|)) \tag{B7}$$

where c is a constant yet to be determined. From (B5) and (B6) we have

$$P_1(z) = \frac{\lambda}{\mu_2 - \lambda z} P_0(z). \tag{B8}$$

Combining (B7), (B8), and $P(1) = 1$ we find c to be

$$c = \frac{\mu_2 - \lambda}{\mu_2} \exp((-\lambda/\mu_1)(1 - \ln(\mu_2 - \lambda))).$$

From the above results we have the z-transform of the number of jobs in the system as shown in (4). Q.E.D.

C. Proof of (9) to (13)

Proof: From the Markov chain given in Fig. 6, we have

$$(\lambda + k\mu_1)p(k,0) = \lambda p(k-1,0) + \mu_2 p(k,1) \qquad 1 \leq k \leq m \tag{C1}$$

$$(\lambda + m\mu_1)p(k,0) = \lambda p(k-1,0) + \mu_2 p(k,1) \qquad k > m \tag{C2}$$

$$(\lambda + \mu_2)p(k-1,1) = \lambda p(k-2,1) + k\mu_1 p(k,0) \qquad 2 \leq k \leq m \tag{C3}$$

$$(\lambda + \mu_2)p(k-1,1) = \lambda p(k-2,1) + m\mu_1 p(k,0) \qquad k > m \tag{C4}$$

$$\lambda p(0,0) = \mu_2 p(0,1) \tag{C5}$$

$$(\lambda + \mu_2)p(0,1) = \mu_1 p(1,0). \tag{C6}$$

We define $p(k) = \Pr[k \text{ jobs in the system}] = p(k,0) + p(k-1,1)$. We further define

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

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

Hence, we have $P(z) = P_0(z) + zP_1(z)$. From (C1) and (C2) we have

$$(\lambda + m\mu_1 - \lambda z) P_0(z) = \mu_2 P_1(z) + \mu_1 \sum_{k=0}^{m-1} \cdot (m-k) p(k,0) z^k. \quad \text{(C7)}$$

Similarly, from (C3) and (C4) we have

$$[(\lambda + \mu_2) z - \lambda z^2] P_1(z) = m\mu_1 P_0(z) - \mu_1 \sum_{k=0}^{m-1} \cdot (m-k) p(k,0) z^k. \quad \text{(C8)}$$

From (C7) and (C8) we have

$$P_1(z) = \frac{\lambda}{\mu_2 - \lambda z} P_0(z). \quad \text{(C9)}$$

From (C7), (C8), and (C9) we have $P(z)$ as shown in (9). The remaining job is to find all $p(k,0)$ for $0 \le k \le m-1$. Using (9) and $P(1) = P_0(1) + P_1(1) = 1$, we can obtain (10). Equations (11), (12), and (13) can easily be obtained from arranging (C1), (C3), (C5), and (C6). From (10) through (13) we are able to find all $p(k,0)$ for $0 \le k \le m-1$. Q.E.D.

REFERENCES

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

[2] S. L. Brumelle, "Some inequalities for parallel server queues," *Oper. Res.*, vol. 19, pp. 402–413, 1971.

[3] G. P. Cosmetatos, "Some approximation equilibrium results for the multi-server queue (M/G/r)," *Opnl. Res. Quart.*, vol. 27, pp. 615–620, 1976.

[4] C. D. Crommelin, "Delay probability formulae," *P.O. Elec. Eng. J.*, vol. 26, pp. 266–274, 1934.

[5] F. S. Hillier and F. D. Lo, "Tables for multiple-server queueing systems involving Erlang distribution," Tech. Rep. 31, Dep. of Oper. Res., Stanford Univ., 1971.

[6] M. H. van Hoorn and H. C. Tijms, "Approximations for the waiting time distribution of the M/G/c queue," *Perform. Eval*, vol. 2, pp. 22–28, 1982.

[7] J. Huang, "On the behavior of algorithms in a multiprocessing environment," Ph.D. dissertation, Comput. Sci. Dep. U.C.L.A., 1988.

[8] J. F. C. Kingman, "Inequalities in the theory of queues," *J. Royal Statistical Society, Series B*, vol. 32, pp. 102–110, 1970.

[9] L. Kleinrock, *Queueing Systems, Vol. 1: Theory.* New York: Wiley-Interscience, 1975.

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

[11] ——, "Performance models for distributed systems," in *Teletraffic Analysis and Computer Performance Evaluation*, Proc. International Seminar held at the centre for Mathematics and Computer Science (CWI), Amsterdam, The Netherlands, June 2–6, 1986, pp. 1–15.

[12] J. D. C. Little, "A proof of the queueing formula $L = \lambda W$," *Oper. Res.*, vol. 9, pp. 383–387, 1961.

[13] E. Maaloe, "Approximation formulae for estimation of waiting-time in multiple-channel queueing system," *Mgmt. Sci.*, vol. 19, pp. 703–710, 1973.

[14] S. A. Nozaki and S. M. Ross, "Approximations in finite-capacity multi-server queues with poisson arrivals," *J. Appl. Probability*, vol. 15, no. 4, pp. 826–834, Dec. 1978.

[15] R. Syski, *Introduction Congestion Theory in Telephone Systems*, second ed. Amsterdam, The Netherlands: North Holland, 1984.

[16] Y. Yakahashi, "An approximation formula for the mean waiting time of an M/G/m queue," *J. Oper. Res. Soc. Japan*, vol. 20, pp. 150–163, 1977.

[17] Y. De Serres and L. G. Mason, "A multiserver queue with narrow- and wide-band customers and wide-band restricted access," *IEEE Trans. Commun.*, vol. 36, no. 6, pp. 675–684, June 1988.

[18] B. Kraimeche and M. Schwartz, "Bandwidth allocation strategies in wide-band integrated networks," *IEEE J. Select. Areas Commun.*, vol. SAC-4, pp. 869–878, Sept. 1986.

[19] L. Green, "A queueing system in which customers require a random number of servers," *Oper. Res.*, vol. 28, no. 6, pp. 1335–1346, Nov.–Dec. 1980.

[20] A. M. Law and W. D. Kelton, *Simulation Modeling & Analysis*, second ed. New York: McGraw-Hill, 1991.

Jau-Hsiung Huang (A'89) received the B.S. degree in electrical engineering from National Taiwan University in 1981, and the M.S. and Ph.D. degrees in computer science from the University of California, Los Angeles, in 1985 and 1988, respectively.

He is Associate Professor of Department of Computer Science and Information Engineering at National Taiwan University, Taipei, Taiwan. He joined the faculty at National Taiwan University in 1988. His research interests include design and performance evaluation of high speed networks, multimedia systems, and parallel and distributed systems.

Leonard Kleinrock (S'55–M'64–SM'71–F'73) received the B.S. degree in electrical engineering from the City College of New York in 1957 (evening session) and the M.S.E.E. and Ph.D.E.E. degrees from the Massachusetts Institute of Technology in 1959 and 1963, respectively.

He is Professor and Chairman of Computer Science at the University of California, Los Angeles. While at M.I.T., he worked at the Research Laboratory for Electronics, as well as with the computer research group of Lincoln Laboratory in advanced technology. He joined the faculty at U.C.L.A. in 1963. His research interests focus on performance evaluation of high speed networks and parallel and distributed systems. He has had over 170 papers published and is the author of five books–*Communication Nets: Stochastic Message Flow and Delay*, 1964; *Queueing Systems, Volume I: Theory*, 1975; *Queueing Systems, Volume II: Computer Applications*, 1976; *Solutions Manual for Queueing Systems, Volume I*, 1982, and most recently, *Solutions Manual for Queueing Systems, Volume II*, 1986. He is a well-known lecturer in the computer industry. He is the principal investigator for the DARPA Parallel Systems Laboratory contract at U.C.L.A.

Dr. Kleinrock is a member of the National Academy of Engineering, is a Guggenheim Fellow, and a member of the Computer Science and Technology Board of the National Research Council. He has received numerous best paper and teaching awards, including the ICC 1978 Prize Winning Paper Award, the 1976 Lanchester Prize for outstanding work in Operations Research, and the Communications Society 1975 Leonard G. Abraham Prize Paper Award. In 1982, as well as having been selected to receive the C.C.N.Y. Townsend Harris Medal, he was co-winner of the L. M. Ericsson Prize, presented by His Majesty King Carl Gustaf of Sweden, for his outstanding contribution in packet switching technology. In July of 1986, he received the 12th Marconi International Fellowship Award, presented by His Royal Highness Prince Albert, brother of King Baudoin of Belgium, for his pioneering work in the field of computer networks. In the same year, he received the U.C.L.A. Outstanding Teacher Award. In 1990, he received the ACM SIGCOMM award recognizing his seminal role in developing methods for analyzing packet network technology. He was a co-founder of Linkabit Corporation. He is also the founder and CEO of Technology Transfer Institute, a computer/communications seminar, conference and consulting organization located in Santa Monica, CA.