Performance Analysis of Parallel Processing Systems

RANDOLPH NELSON, DON TOWSLEY, MEMBER, IEEE, AND ASSER N. TANTAWI, MEMBER, IEEE

Abstract—**A bulk arrival $M^X/M/c$ queueing system is used to model a centralized parallel processing system with job splitting. In such a system, jobs wait in a central queue, which is accessible by all the processors, and are split into independent tasks that can be executed on separate processors. The job response time consists of three components: queueing delay, service time, and synchronization delay. An expression for the mean job response time is obtained for this centralized parallel processing system. Centralized and distributed parallel processing systems (with and without job splitting) are considered and their performances compared. Furthermore, the effects of parallelism and overheads due to job splitting are investigated.**

Index Terms—**Bulk arrival queues, fork/join queues, job splitting, modeling and analysis, parallel processing, synchronization delay.**

I. Introduction

PERFORMANCE modeling of parallel processing systems must take into account the job structure and the system structure. The *job structure* describes the precedence relationships among the various components, called tasks, of the job and is usually given by a task graph model. A simple job structure is a fork/join one where a job consists of a set of independent tasks. For such a structure, a job is considered completed only when all of its tasks are completed. The *system structure* is modeled by a queueing system consisting of a system of queues and a set of servers. Generally speaking, we may have either a central queue which is accessible by all the processors or a distributed queue where each processor serves its own queue. A queue may hold jobs or tasks depending on whether jobs are split into tasks before entering the queue or not. We index the tasks composing a job by their order of departure and define the time between the departure of the first and last task to be the *synchronization delay* of that job. The job response time, which includes this synchronization delay, is hard to obtain even for simple system structures [1], [3]–[5]. In the nonsplitting case, the analysis is much easier since all tasks from the same job are executed sequentially on the same processor.

Spies and Geilenkeuser [6] study a related system. They consider multiple processors serving jobs that arrive according to a Poisson process. Each job consists initially of a single task. Upon completion, each task generates zero or more additional tasks that require further processing. Hence a job is represented by a tree precedence graph where control flows from the root to the leaves. The statistics of the number of tasks in the system are derived under the assumption that task service times are independent and identically distributed exponential random variables. However, no results are reported regarding the job response time characteristics.

In this paper, we model a parallel processing system with a central queue and job splitting as a bulk arrival $M^X/M/c$ queueing system where customers and bulks correspond to tasks and jobs, respectively. Such a system has been studied in [2], [7] and an expression for the mean response time of a random customer is obtained. However, since we are interested in the time that a job spends in the system, including the synchronization delay, we must evaluate the bulk response time rather than simply the customer response time.

In Section II we describe the queueing system under consideration. The job response time is the sum of the job waiting time and the job service time. In Section III we analyze the bulk queueing system and obtain an expression for the mean job waiting time. The mean job service time is given by a set of recurrence equations. In Section IV we consider queueing models of a wide range of parallel processing systems covering systems with centralized and distributed queues as well as with and without job splitting. In these models we assume that jobs consist of fork/join task graphs in which each task has the same distribution of service which is assumed to be exponential. These are simplifying assumptions and are made so the resultant queueing systems are mathematically tractable. In real systems jobs could consist of an arbitrary task graph and could have very different task service distributions. We believe, however, that fundamental performance characteristics are captured by our models and that the relationships found by comparing their relative response time, in Section V, continue to hold under less restrictive assumptions. Other characteristics that we study in Section V include the effects of parallelism on response time and the overhead delays due to job splitting.

II. Model Description

We consider a system of c identical servers that serve a single queue. Jobs enter the system according to a Poisson process with parameter λ. Each job consists of one or more tasks that can be processed independently of each other. Specifically, let X be a random variable (r.v.) that

Manuscript received March 31, 1987; revised September 30, 1987. The work of D. Towsley was supported in part by the National Science Foundation under Contract ECS8406402.

R. Nelson and A. N. Tantawi are with the IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598.

D. Towsley is with the Department of Computer and Information Science, University of Massachusetts, Amherst, MA 01003.

IEEE Log Number 8819727.

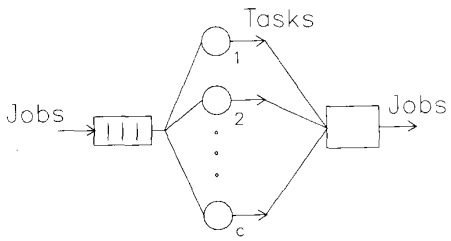

Fig. 1. Queueing model.

denotes the number of tasks within a job. We assume that X has probability distribution $\alpha_i = P[X = i]$, $i = 1, 2, \cdots$ and probability generating function (p.g.f.) $X(z) = \Sigma_{i=1}^{\infty} \alpha_i z^i$. The service time required by a task is assumed to be an exponential r.v. with parameter μ and is independent of the service requirements of all other tasks.

We are interested in the steady state behavior of this system. In particular, we focus on the response time of a random job, i.e., the interval of time measured from the arrival of a job until the service completion of the *last task* associated with that job. One can envision the system as consisting of a queue for tasks awaiting service, c servers, and a waiting area for tasks that have completed service but are awaiting the completion of the last task associated with a job (Fig. 1). All of the tasks within a job depart this last waiting area when all have completed their service. Denote this response time as T. Last, we assume that jobs are scheduled into service in a first-come first-serve manner. Tasks within a single job are scheduled in random order.

III. Analysis

As described, this system is modeled as a continuous time, discrete state Markov process. We obtain the statistics for the response time of a job by decomposing the problem into two parts each of which can be handled in a simple manner. Specifically, the response time T of a random job can be expressed as the sum of two terms: $T = W + S$. The first term W is the *job waiting time* and corresponds to the time that the job waits in the queue before the first of its tasks is scheduled. If the job arrives to a system where one or more of the servers are idle then $W = 0$, otherwise $W > 0$. The second term S is the *job service time* and corresponds to the time required to process all of the tasks associated with the job once the first task is scheduled. The mean response time is given by

$$E[T] = E[W] + E[S]. \tag{1}$$

The statistics for W can be obtained by studying the bulk arrival $M^X/M/c$ queue that underlies the parallel processing system of interest to us. Specifically, W in our system corresponds to the time that a bulk of customers in the $M^X/M/c$ must wait before the first customer begins service. Section III-A focuses on this random variable. The statistics of the job service time are obtained in Section III-B. Unfortunately, we are unable to obtain closed form expressions for these statistics, but must numerically solve a set of recurrence relations.

A. Mean Job Waiting Time

In this section, we analyze the behavior of the $M^X/M/c$ queue. This provides us with the statistics for W, the time that a job waits in the queue of the parallel processing system. The queue length distribution for the bulk arrival $M^X/M/c$ queue will also be used to obtain the service time statistics in the parallel processing system. We shall use the following terminology. The bulk arrival system processes *tasks*. Tasks that arrive simultaneously form a *group of tasks*. The reader should note that a job in the parallel processing system corresponds to a group of tasks in the bulk arrival system. In addition to the notation introduced in the previous Section, we define N to be a random variable denoting the steady state number of tasks in the system. N has distribution $\pi_i = P[N = i]$, $i = 0, 1, \cdots$, and p.g.f. $\Pi(z) = \Sigma_{i=0}^{\infty} \pi_i z^i$. The steady state queue length distribution, π_i, $i = 0, 1, \cdots$ satisfies the following difference equations,

$$\lambda \pi_0 = \mu \pi_1, \tag{2}$$

$$(\lambda + i\mu)\,\pi_i = \lambda \sum_{k=0}^{i-1} \pi_k \alpha_{i-k} + (i+1)\,\mu \pi_{i+1}, \quad 0 < i < c, \tag{3}$$

$$(\lambda + c\mu)\,\pi_i = \lambda \sum_{k=0}^{i-1} \pi_k \alpha_{i-k} + c\mu \pi_{i+1}, \quad i \geq c. \tag{4}$$

By multiplying both sides of each of the above equations by z^i and summing, we obtain

$$(\lambda + c\mu)\,\Pi(z) - \sum_{i=0}^{c-1} \mu(c-i)\,\pi_i z^i = \lambda \sum_{i=1}^{\infty} z^i \sum_{k=0}^{i-1} \pi_k \alpha_{i-k} + \sum_{i=1}^{c-1} i\mu \pi_i z^{i-1} + \sum_{i=c}^{\infty} c\mu \pi_i z^{i-1}.$$

By defining $A(z) = \Sigma_{i=0}^{c-1} (c-i)\,\pi_i z^i$ and substituting it into the above equation, we finally obtain

$$\Pi(z) = A(z)\,\mu(z-1) \big/ \big[\lambda z(1 - X(z)) + c\mu(z-1)\big]. \tag{5}$$

By taking the limit of the above equation as z goes to 1, we get $\Pi(1) = A(1)\,\mu/[c\mu - \lambda E[X]]$, which, as $\Pi(1) = 1$, yields $A(1) = (c\mu - \lambda E[X])/\mu$. The above expression along with (2) and (3) can be used to obtain an expression for $\Pi(z)$. The moment generating properties of the p.g.f. allow us to obtain the expected number of tasks in the system, i.e., $E[N] = d\Pi(z)/dz \,|_{z=1}$. Let Q be an r.v. that denotes the number of tasks in the queue awaiting service; then we have

$$E[Q] = E[N] - c + \sum_{i=0}^{c-1} (c-i)\,\pi_i = E[N] - c + A(1).$$

The mean job waiting time $E[W]$ in the original parallel processing system corresponds to the average time that a group of tasks waits in the $M^X/M/c$ queue before the first job enters a server. $E[W]$ is therefore given by

$$E[W] = (E[Q] + P[N \geq c])/(c\mu). \quad (6)$$

This completes our discussion of the bulk arrival $M^X/M/c$ queue.

B. Mean Job Service Time

In this section we determine the value of the mean job service time $E[S]$ for a random job. The analysis is simplified if we condition the job service time on the number of processors that can be initially scheduled (i.e., at the time that the first task is scheduled) and on the total number of tasks that must be completed (including those in service).

Let $s_{i,n}$, $(0 < i \leq \min\{c, n\}, 1 \leq n < \infty)$, be the mean job service time given that there are n tasks to be executed and that i tasks are initially scheduled to servers. The quantity $s_{i,n}$ satisfies the following equations,

$$s_{1,1} = 1/\mu,$$

$$s_{n,n} = s_{n-1,n-1} + 1/(n\mu) = H_n/\mu, \quad 1 \leq n \leq c,$$

$$s_{c,n} = 1/(c\mu) + s_{c,n-1} = (n-c)/(c\mu) + H_c/\mu, \quad n > c,$$

$$s_{i,n} = 1/(c\mu) + is_{i,n-1}/c + (c-i)s_{i+1,n}/c, \quad 1 \leq i < \min\{c, n\},$$

where $H_n = \sum_{k=1}^{n} 1/k$ is the harmonic series.

The number of servers available when the job is first scheduled is related to the number of idle servers at job arrival time. If the job arrives when all servers are busy, $N \geq c$, then exactly one server will be available when the first task is scheduled. If there are one or more idle servers at the time that the job arrives, $N < c$, then the job is scheduled immediately and will initially schedule min $\{X, c - N\}$ servers at that time. Consequently, removal of the conditioning on the number of servers initially available yields

$$E[S \mid X = n] = \sum_{i=0}^{c-1} \pi_i s_{\min\{n, c-i\}, n} + \left(1 - \sum_{i=0}^{c-1} \pi_i\right) s_{1,n}, \quad 1 \leq n.$$

Removal of the conditioning on the number of tasks in a job yields

$$E[S] = \sum_{i=0}^{c-1} \pi_i \sum_{n=1}^{\infty} s_{\min\{n, c-i\}, n} \alpha_n + \left(1 - \sum_{i=0}^{c-1} \pi_i\right) \sum_{n=1}^{\infty} s_{1,n}\alpha_n. \quad (7)$$

The mean job response time $E[T]$ is obtained by substituting (6) and (7) into (1).

As an example, we consider a special case where the number of tasks in a job is a geometric r.v., i.e., $P[X = n] = p^{n-1}(1-p)$, $n > 0$. We shall focus on deriving the expected service time, $E[S]$. The following analysis is made possible by the memoryless property of the geometric distribution. Let s_i, $0 < i \leq c$, be the mean service time of a job given that i tasks are initially scheduled to i processors and that it contains *at least one* task in the queue. This conditional expectation satisfies the following equations,

$$s_i = 1/(c\mu) + (1-p)\left(\frac{i}{c}H_i + \frac{c-i}{c}H_{i+1}\right)\Big/\mu + p\left(\frac{i}{c}s_i + \frac{c-i}{c}s_{i+1}\right), \quad 0 < i < c,$$

$$s_c = 1/(c\mu) + (1-p)H_c/\mu + ps_c = 1/((1-p)c\mu) + H_c/\mu, \quad i = c.$$

By substitution into (7) we obtain

$$E[S] = \sum_{i=0}^{c-1} \pi_i \left(\sum_{n=1}^{c-i} p^{n-1}(1-p)H_n/\mu + p^{c-i+1}s_{c-i}\right) + \left(1 - \sum_{i=0}^{c-1} \pi_i\right)[(1-p)/\mu + ps_1]. \quad (8)$$

IV. Parallel Processing Models

In the remainder of this paper we consider the following question. How should one architect a system that processes an arrival stream of jobs composed of independent tasks on a system consisting of c identical processors. Since the queueing system analyzed in the previous section is just one way to architect a parallel processing system, we find it to be of more interest in this section to expand the class of architectures that we consider.

In Fig. 2 we present four models of parallel processing systems: distributed/splitting (D/S), distributed/no splitting (D/NS), centralized/splitting (C/S), and centralized/no splitting (C/NS). In each of these systems there are c processors, jobs are assumed to consist of set of tasks that are independent and have exponentially distributed service requirements, and arrivals of jobs are assumed to come from a Poisson point source with intensity λ. The systems differ in the way jobs queue for the processors and in the way jobs are scheduled on the processors. The queueing of jobs for processors is *distributed* if each processor has its own queue, and is *centralized* if there is a common queue for all the processors. The scheduling of jobs on the processors is *no splitting* if the entire set of tasks composing that job are scheduled to run sequentially on the same processor once the job is scheduled. On the other hand, the scheduling is *splitting* if the tasks of a job are scheduled so that they can be run independently and potentially in parallel on different processors. In the splitting case a job is completed only when all of its tasks have

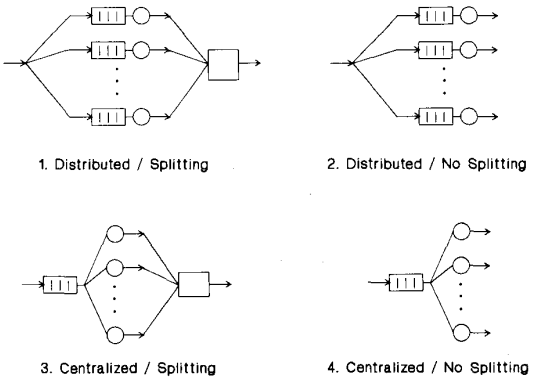

Fig. 2. Parallel processing models.

finished execution. This is known as a *join* operation. We distinguish this case in the figures by showing a "waiting" box after the processors in the ./S systems.

In our study we compare the mean response time of jobs in each of the systems for differing values of the number of processors, number of tasks per job, server utilization, and certain overheads associated with splitting up a job. The $M^X/M/c$ system studied in the previous section corresponds to the C/S system. In this system, as processors become free they serve the first task in the queue. D/. systems are studied in [5]. We use the approximate analysis of the D/S system and the exact analysis of the D/NS system that are given in that paper. For systems with 32 processors or less, the relative error in the approximation for the D/S system was found to be less than 5 percent. The approximation for the D/S system given in [5] requires that the number of tasks per job equals the number of processors, each task is scheduled on a different processor. In the D/NS system, jobs are assigned to processors with equal probabilities. The approximation we use for the mean job response time for the C/NS system is found in [8]. Although an extensive error analysis for this system over all parameter ranges has not been carried out, the largest relative error for the $M/E_2/10$ system reported in [8] is about 0.1 percent.

V. Results

In this section we present the results of our investigations into the performance of the four parallel processing systems described in Section IV. We ran four different experiments to compare and contrast the performance of the different systems. The performance metric we use in these comparisons is the mean job response time and, for comparative purposes, often plot the ratio of mean job response times for the different systems. We plot the curves in this section as a function of the utilization of each server in the system given by $\rho = \lambda E[X]/c\mu$. We compare systems for a common ρ since this corresponds to viewing each system under an equal load. Experiment 1 is a comparison of the mean job response times for the four systems to determine their relative performance for different utilizations. In experiment 2 we quantify the effects of executing the tasks of a job in a parallel, rather than sequential, manner. In experiment 3 we consider a system which has two types of jobs: edit jobs and batch jobs. Edit job are assumed to consist of a single task whereas batch jobs may consist of many tasks. With the idea that the response time of edit jobs might be severely increased by the existence of batch jobs on the same system, we consider the effects of partitioning the processors of the system so that edit and batch jobs are run on different sets of processors. Finally, in experiment 4, we investigate the effects of including overhead resulting from the splitting of jobs into tasks.

A. A Comparison of Parallel Processing Systems

In this section we compare the mean job response time for the four parallel processing systems described in Section IV to determine their relative performance over different utilizations. In Fig. 3 we plot the mean job response time for the four different systems for a system consisting of $c = 8$ processors and where jobs are assumed to consist of exactly 8 tasks.

The figure contains several interesting comparisons. For any server utilization ρ, we note that the ./S systems yield lower mean job response times than the ./NS systems. This implies that, in the absence of overhead associated with splitting jobs up into tasks, it is better to run the tasks of a job in parallel and incur a synchronization delay than it is to avoid this delay and run tasks sequentially on the same processor. This same observation, for the D/. systems, was made in [5]. The intuition behind this observation lies in the fact that in the splitting case work is spread in smaller increments over all the processors. Intuitively, it is more likely in the no splitting case for a set of processors to be idle when others are busy processing larger, unsplit jobs, than it is in the splitting case. In comparing distributed and centralized systems as shown in Fig. 3 we see that the C/S system has a better performance than that of the D/S system. Once again the same observation explains this effect, namely one is more likely to find idle processors even when there is work available for processing in the distributed system than in the centralized system.

From Fig. 3 we conclude that, for all utilizations, the C/S system has the least mean job response time, the D/NS system the greatest, and that the D/S system has a lower response time than the D/NS system. These qualitative relationships are not actually surprising and could have been anticipated from basic queueing considerations. What could not have been so easily predicted, however, is the relationship between the D/S and C/NS systems. The relationship between the mean job response time for these systems, as shown in Fig. 3, depends on the utilization of the system. Specifically, we see that for $\rho < 0.7$ the D/S system has a lower mean job response time than the C/NS system and that the roles are reversed for higher utilizations. Evidently the parallelism found in splitting up jobs into tasks in the D/S system reduces the mean job response time for low to moderate utilizations

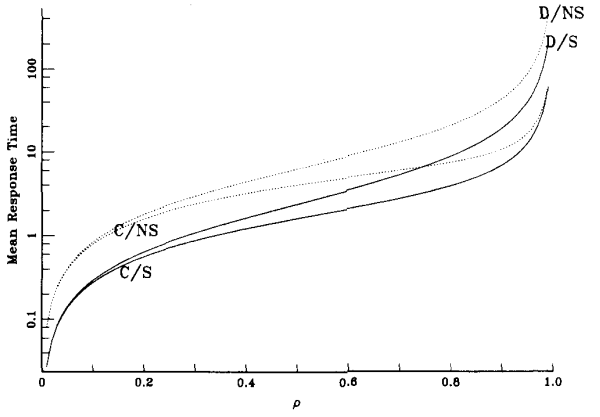

Fig. 3. Comparison of four systems, $C = 8$, $\lambda = 1$.

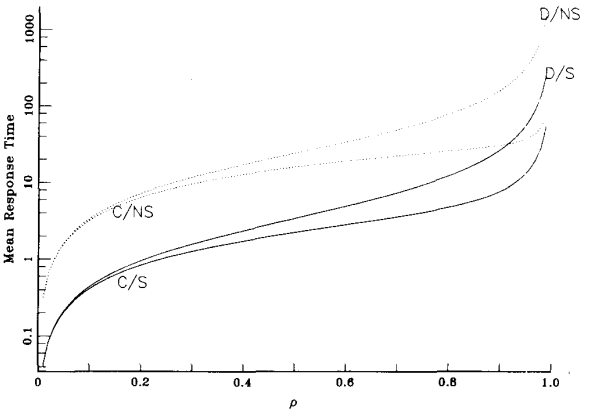

Fig. 4. Comparison of four systems, $C = 32$, $\lambda = 1$.

so that its mean job response time is lower than in the C/NS system. For high utilizations, however, the synchronization delay incurred in the D/S system becomes an increasing factor in the overall mean job response time and, as shown in the figure, grows sufficiently so that the C/NS system, which suffers no such delay, yields a lower mean job response time.

In Fig. 4 we plot the same set of curves for the case of $c = 32$ and jobs consisting of 32 tasks. Observe the similarity between this figure and Fig. 3. The point at which the C/NS system has a lower mean job response time than the D/S system increases to $\rho = 0.9$. This can be explained by the fact that with 32 processors there is an increased opportunity for parallelism in processing tasks in the D/S system than with 8 processors.

The relative performances of the D/S and C/NS systems to that of the system with the minimal mean job response time, the C/S system, are compared and plotted in Fig. 5. We have eliminated the D/NS system from consideration because of its poor performance. There are two sets of curves in Fig. 5. In the first set, we investigate the effects of splitting jobs into tasks for a centralized queueing structure. The top two curves in the figure are the ratios of the mean job response times for the C/NS system to the C/S system for varying utilizations with $c = 8$ and $c = 32$. As before, we assume that a job consists of the same number of tasks as processors. For centralized systems, these curves show that for low utilizations it is very important to split jobs into tasks. This importance decreases with increasing utilization and, at very heavy traffic, both systems perform similarly. These relationships can best be explained by considering each system for low utilizations. Assume in such a case that an arriving job finds no jobs in the queue and thus the mean job response time is simply the service time for the job. In the splitting case, the work of the job is distributed over all the processors whereas in the no splitting case all of the work is performed by one processor. Clearly the splitting case will be faster than the no splitting case and as the system approaches saturation, and queueing delay becomes a larger component of the total job response time,

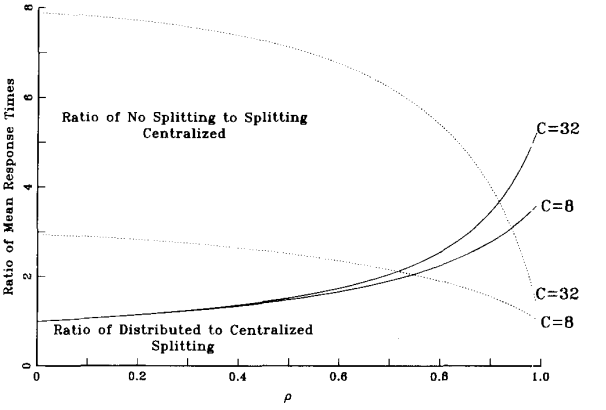

Fig. 5. Ratio of mean response times, $\lambda = 1$.

this effect becomes less important. These effects will be analyzed more fully in experiment 2.

In the second set of curves in Fig. 5 we evaluate the effect of the queueing structure on the mean job response time. In this case we plot the ratio of the mean job response times for the D/S system to that of the C/S system. In this case it is evident from the figure that keeping jobs in a central queue becomes more important as the utilization of the system increases. This follows from the fact that, because of the differences in the queueing structures for these systems, the synchronization time for tasks in the C/S system is lower than that for the D/S system.

B. The Effect of Parallelism

Having established that the C/S system yields the least mean job response time over all utilizations, we now use it as a touchstone for further experiments. In the second set of experiments we explore the effect of parallelism on the mean job response time by comparing the performance of the C/NS system to that of the C/S system. In Fig. 6 we plot the ratio of mean job response times for the C/NS system to that of the C/S system for various utilizations as a function of c. We assume that each job consists of 8 tasks. As noted in the last part of experiment

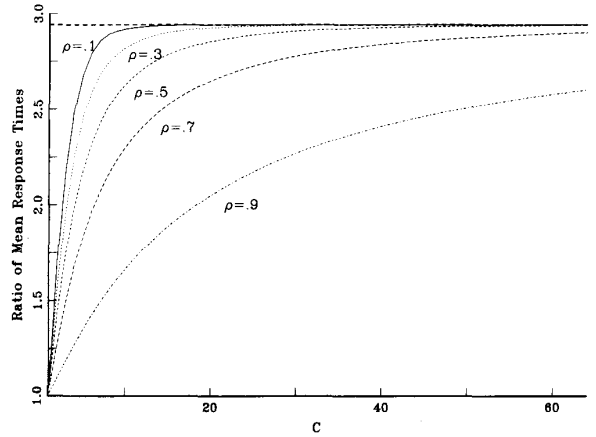

Fig. 6. Ratio of mean response times, no splitting to splitting, centralized.

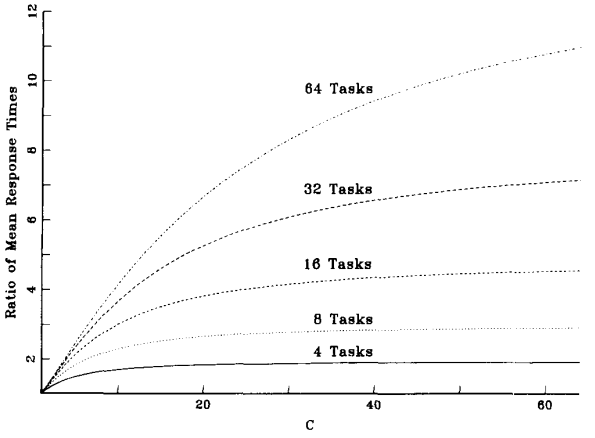

Fig. 7. Ratio of mean response times, no splitting to splitting, centralized.

1, when c is fixed, the performance improvement obtained from splitting jobs into tasks decreases with increasing utilization. For a fixed utilization, however, this performance improvement increases with c. This arises from the fact that as c increases, the chance for processing tasks of the same job in parallel increases. As seen in Fig. 6, for all utilizations all curves increase asymptotically to the solid line on the top of the figure. To derive the value of this asymptote consider the ratio of the mean job response times for low utilizations. This will effectively be the ratio of the two service times, or $8/(\Sigma_{j=1}^{8} 1/j) = 2.943$. As c increases, for a fixed processor utilization, the probability that a job arrives and has all of its tasks scheduled simultaneously increases to one. Thus, as the number of processors grows to infinity and for any fixed processor utilization, the ratio of the mean job response times reaches the above value.

In Fig. 7 we investigate the effects of parallelism as we vary the number of tasks per job. The utilization is fixed at $\rho = 0.7$. As just noted, as c approaches infinity the ratio of the mean job response times reaches the limit $X/(\Sigma_{j=1}^{X} 1/j)$, where X is the number of tasks in a job. For a fixed c, it is interesting to note that the ratio of mean job response times increases with increasing X. Thus breaking up jobs into a large number of smaller pieces is beneficial to a parallel processing system. Naturally this conclusion is only valid if one ignores any overhead incurred with splitting jobs up into pieces. We investigate the phenomena of including overhead associated with this process in experiment 4.

C. Processor Partitioning

In this set of experiments we assume that there are two classes of jobs each with different processing requirements. *Edit* jobs are assumed to consist of simple operations that have no inherent parallelism and thus consist of only one task. *Batch* jobs, on the other hand, are assumed to be inherently parallel and can be broken up into tasks. All tasks from either class are assumed to have the same service requirements. It is not at all clear that two such divergently differing job classes should be executed on the same system since edit jobs could potentially suffer a poor performance because of the presence of the more computationally bound batch jobs. Furthermore, the existence of the edit jobs on the system could decrease the parallelism available to the batch jobs and thus increase their response time. In order to investigate the interaction of these two classes of jobs, we compare the performance of a combined system to that of a partitioned system. We refer to the C/S system, where both classes share the processors, as a combined system. A partitioned system is a system where processors are partitioned into two sets, one set for each class. We assume that batch jobs consist of 16 tasks, that there are $c = 16$ processors, and that in the partitioned system K, $1 \leq K \leq c$, processors are allocated to edit jobs and $c - K$ processors are allocated to batch jobs. Our performance metric is the ratio of mean response times for each class of jobs of the partitioned system to that of the combined system. For either class, a ratio greater than (less than) 1 indicates that in the partitioned system that class of jobs has a greater (lesser) mean response time than that obtained in the combined system. We should note here that our comparison assumes that jobs in either system are processed FCFS. We expect a real system to use some scheduling discipline that, in general, gives priority to the edit class of jobs.

In Fig. 8 we assume that edit jobs compose 50 percent of the job stream and plot the ratio of mean job response times in the partitioned system to that of the combined system as a function of utilization, for different values of K. These curves reveal a very interesting phenomena that can arise in a parallel processing system. Specifically, one sees that when one processor is allocated to the edit jobs, $K = 1$, the mean response time for *both* edit and batch jobs increases. In other words, both classes are negatively influenced by partitioning the system in this manner. Improvement to edits jobs, at the cost of increasing the mean response time of batch jobs, results only when the number of processors allocated to edit jobs increases to 2, as shown in the figure. Clearly, for $K > 2$ the performance

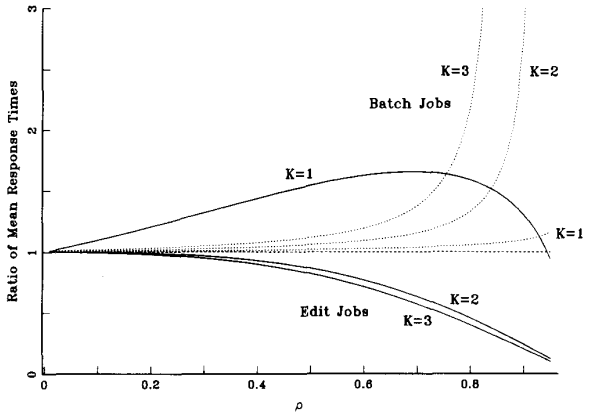

Fig. 8. Comparing combined system to partitioned system, 50 percent edit jobs.

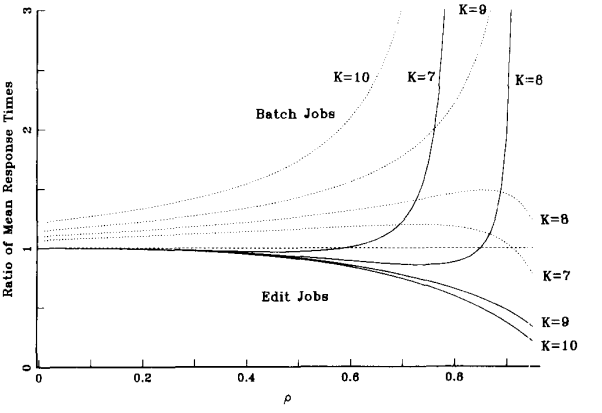

Fig. 9. Comparing combined system to partitioned system, 95 percent edit jobs.

of the edit jobs improves as seen in the figure. As the relative arrival rate of edit jobs increases, as shown in Fig. 9 where the proportion of edit jobs is 95 percent, we see that more processors must be allocated to edit jobs before their mean response time is decreased. As shown in the figure, nine processors are required to decrease the mean job response time over all utilizations. Once again there are regions over which the mean response times of both job classes increase in the partitioned system. For parallel processing systems, these considerations suggest that it is desirable to have a controllable boundary for processor partitioning. We should observe here that since there are regions over which both classes of jobs have their mean response time increased in the partitioned system, one might hope to find regions over which both classes had their mean response time *decreased*. Looking for such a region is a vain pursuit, however, as one might expect upon reflection.

D. Effect of Overhead

So far, we have ignored any costs associated with splitting jobs up into separately executable tasks. Such costs may arise from data dependencies between tasks of the same job. To model these costs we associate with each task an increase in service requirement that depends upon the number of tasks per job. Specifically we will assume that the average service requirement for a task of a job consisting of X tasks, is given by $1/X + H$, where H is the overhead cost. Each job thus requires a total of one unit of processing time in the absence of overhead. The tradeoff we investigate in this set of experiments concerns breaking up a job into a number of tasks. As noted in discussing Fig. 7, if $H = 0$, for a given c and processor utilization, the mean job response time decreases with X. This is no longer true if there is overhead associated with splitting jobs into tasks since increasing the number of tasks of a job also results in higher service times.

In Fig. 10 we plot the mean job response time for the C/S system for different arrival rates λ as a function of X. We assume that $H = 0.1$, which may correspond to an overhead of 10 percent of the total service required by a

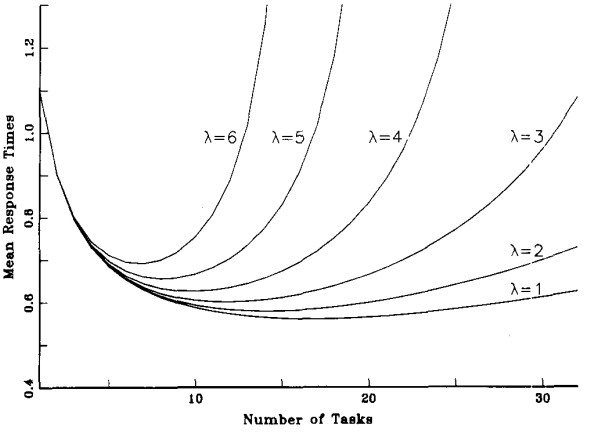

Fig. 10. Centralized/splitting, fixed overhead, $H = 0.1$.

job per task. The convex nature of the curves arises from the tradeoff just mentioned, namely that increasing the number of tasks at first initially decreases the mean job response time until a certain point after which the overhead costs predominate and the mean job response time increases. The optimal number of tasks, as shown by the lowest point on these curves, decreases as the arrival rate increases. Note that, as the system becomes more heavily loaded, queueing effects, which are increased because of the existence of overhead, become more predominant. This also explains why the curves become steeper with increasing λ.

In Fig. 11 we show, on the same scale as in Fig. 10, the effects of increasing the overhead to $H = 0.2$. Each curve increases due to the effect of the increased load on the system and similarly the optimal values of X also decrease from that shown in Fig. 10. When the overhead becomes a substantial portion of the service requirement of a task, for example if $H = 1$, than it is no longer advisable, for any arrival rate, to split jobs up into tasks. For this case, parallelism is too expensive and for a real system could correspond to jobs which have strong data dependencies.

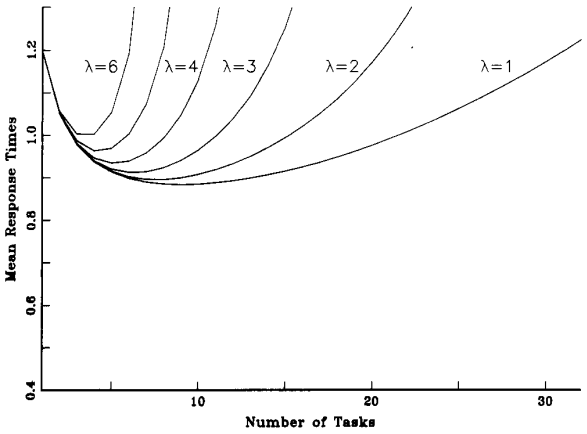

Fig. 11. Centralized/splitting, fixed overhead, $H = 0.2$.

VI. Conclusion

We have analyzed the bulk arrival $M^X/M/c$ queueing system and obtained an expression for the mean job response time in centralized parallel processing systems with job splitting into independent tasks.

To compare this system with other parallel processing systems, the following four systems are considered: distributed/splitting, distributed/no splitting, centralized/splitting, and centralized/no splitting. For all values of utilization ρ and similar queueing structures, our results show that the splitting systems yield lower mean job response time than the no splitting systems. This follows from the fact that, in the splitting case, work is distributed over all the processors. For any ρ, the lowest (highest) mean job response time is achieved by the C/S system (the D/NS system). The relative performance of the D/S system and the C/NS system depends on the value of ρ. For small ρ, the parallelism achieved by splitting jobs into parallel tasks in the D/S system reduces its mean job response time as compared to the C/NS system, where tasks of the same job are executed sequentially. However, for high ρ, the C/NS system has lower mean job response time than the D/S system. This is due to the long synchronization delay incurred in the D/S system at high utilizations.

The effect of parallelism on the performance of parallel processing systems is studied by comparing the performance of the C/NS system to that of the C/S system. The performance improvement obtained by splitting jobs into tasks is found to decrease with increasing utilization. For a fixed number of processors and fixed ρ, we find that by increasing the number of tasks per job, i.e., higher parallelism, the mean job response time of the C/NS system relative to that of the C/S system increases. By considering an overhead delay associated with splitting jobs into independent tasks, we observe that the mean job response time is a convex function of the number of tasks, and thus, for a given arrival rate, there exists a unique optimum number of tasks per job.

We also consider problems associated with partitioning the processors into two sets, each dedicated to one of two classes of jobs: edit jobs and batch jobs. We analyzed an example where the mean job response time for both classes of jobs *increases* if one processor is strictly allocated to edit jobs. Improvement to edit jobs, at a cost of increasing the mean job response time of batch jobs, results only when the number of processors allocated to edit jobs is increased to two. This, and other results, suggest that it is desirable for parallel processing systems to have a controllable boundary for processor partitioning.

Lastly we comment on the effect of some of our assumptions on the relative performance of the four parallel processing models studied in the paper. Although we do not have an analysis for the case in which task service times are generally distributed, we believe that the synchronization time, and thus the response time, increases with the variance of service time. Intuitively this follows from the fact that for a large service variation there is a greater probability of a job waiting a long time for a large task to finish execution. Since tasks in real systems are bounded and, in some cases, tend to have less variation than that of an exponential distribution, we believe the analysis for the splitting cases found in the paper is pessimistic for some systems. This suggests that, for these systems, the relative performance of splitting to nonsplitting would be better than that given in this paper. Acting against this, however, is the fact that there are some overheads associated with splitting jobs into, perhaps dependent, tasks that have not been modeled. As these overheads grow, it is clear that the nonsplitting models perform relatively better.

References

[1] F. Baccelli and A. M. Makowski, "Simple computable bounds for the fork-join queue," presented at the Johns Hopkins Conf. Inform. Sci., Johns Hopkins Univ., 1985.

[2] M. L. Chaudhry and J. G. C. Templeton, *A First Course in Bulk Queues*. New York: Wiley, 1983.

[3] L. Flatto, "Two parallel queues created by arrivals with two demands II," *SIAM J. Appl. Math.*, vol. 45, pp. 861-878, Oct. 1985.

[4] L. Flatto and S. Hahn, "Two parallel queues created by arrivals and with two demands I," *SIAM J. Appl. Math.*, vol. 44, pp. 1041-1053, Oct. 1984.

[5] R. Nelson and A. N. Tantawi, "Approximate analysis of fork/join synchronization in parallel queues," IBM Res. Rep. RC11481, Oct. 1985; also *IEEE Trans. Comput.*, to be published.

[6] P. P. Spies and W. J. Geilenkeuser, "Queueing systems serving task families," in *Performance'83*, A. K. Agrawala and S. K. Tripathi, Eds., 1983, pp. 187-201.

[7] D. D. Yao, "Some results for the queues $M^X/M/c$ and $GI^X/G/c$," *Oper. Res. Lett.*, vol. 4, pp. 79-83, July 1985.

[8] ——, "Refining the diffusion approximation for the $M/G/m$ queue," *Oper. Res.*, vol. 33, pp. 1266-1277, Nov. 1985.

Randolph Nelson received the B.S. degree in physics from Rutgers University, New Brunswick, NJ, the M.S. degree in mathematics from Arizona State University, Tempe, and the Ph.D. degree in computer science from the University of California, Los Angeles. At UCLA his main research interests were in computer communication systems with a special emphasis on multihop packet radio networks.

Since 1982 he has been employed as a Research Staff Member at the IBM Thomas J. Watson Research Center, Yorktown Heights, NY. His current research interests are concerned with performance evaluation of computer systems.

Don Towsley (M'78) received the B.A. degree in physics and the Ph.D. degree in computer sciences from the University of Texas at Austin in 1971 and 1975, respectively.

From 1976 to 1985 he was a member of the faculty of the Department of Electrical and Computer Engineering at the University of Massachusetts, Amherst, where he achieved the rank of Associate Professor. He is currently an Associate Professor of Computer and Information Science at the University of Massachusetts. During 1982–1983, he was a Visiting Scientist at the IBM Thomas J. Watson Research Center, Yorktown Heights, NY. His research interests are in computer networks, distributed computer systems, and performance evaluation.

Dr. Towsley is currently an Associate Editor of *Networks* and IEEE TRANSACTIONS ON COMMUNICATIONS. He is also a member of the Association for Computing Machinery and the Operations Research Society of America.

Asser N. Tantawi (M'87) received the B.S. and M.S. degrees in computer science from Alexandria University, Alexandria, Egypt, and the Ph.D. degree in computer science from Rutgers University, New Brunswick, NJ.

He joined the IBM Thomas J. Watson Research Center, Yorktown Heights, NY, in 1982, as a Research Staff Member, where he is currently associated with the Systems Analysis Group of the Department of Computer Sciences. His fields of interest include performance modeling, queueing theory, load balancing, parallel processing and reliability modeling.

Dr. Tantawi is a member of the Association for Computing Machinery and the Operations Research Society of America. He has served as an ACM National Lecturer since 1984.