Computing Performance Bounds of Fork-Join Parallel Programs Under a Multiprocessing Environment

John C.S. Lui, *Member*, *IEEE*, Richard R. Muntz, *Fellow*, *IEEE*, and Don Towsley, *Fellow*, *IEEE*

Abstract—We study a multiprocessing computer system which accepts parallel programs that have a fork-join computational paradigm. The multiprocessing computer system under study is modeled as K homogeneous servers, each with an infinite capacity queue. Parallel programs arrive at the multiprocessing system according to a series-parallel phase type interarrival process with mean arrival rate of λ. Upon the program arrival, it forks into K independent tasks and each task is assigned to an unique server. Each task's service time has a k-stage Erlang distribution with mean service time of $1/\mu$. A parallel program is completed upon the completion of its last task. This kind of queuing model has no known closed form solution in the general ($K \geq 2$) case. In this paper, we show that by carefully modifying the arrival and service distributions at some imbedded points in time, we can obtain tight performance bounds. We also provide a computational efficient algorithm for obtaining upper and lower bounds on the expected response time. The methodology is flexible and allows one to trade-off the tightness of the bounds and computational cost.

Index Terms—High performance computing, performance evaluation, performance modeling methodology, analysis of multiprocessing systems.

1 Introduction

With the advent of multiprocessing technology, such as symmetric multiprocessing [29], parallel programming languages [12], [26], and parallel programming environments [3], [9], there is an increasing interest in understanding and modeling the performance of parallel programs. Many parallel programs exhibit the fork-join structure and this type of parallel program paradigm arises in many application areas. For example, in a parallel database machine with a shared-nothing architecture [30], a complex query is often partitioned into many local queries so that each local query will be processed on one processing node. Another example is a computer vision system where an image is fed to a multiprocessor system, each processor does some ray-tracing or rendering computations for a portion of the image. The final image is ready for display when all processors have finished their ray-tracing or rendering operations.

We model this kind of parallel program as follows. Assume that our multiprocessing computing system has K homogeneous servers, each with an infinite capacity queue. A parallel program arrives to the multiprocessing computing environment with mean arrival rate λ. Upon the parallel program's arrival, it splits into K independent tasks t_i, $1 \leq i \leq K$ and task t_i is assigned to the ith server. Each task's service time is k-stage Erlang distributed with a mean service time of $1/\mu$. Each server uses the first-come-first-serve (FCFS) scheduling discipline to service its tasks. When a task is finished and if there are any tasks belonging to the same parallel program still in service, the finished task will wait in the synchronization area. The parallel program is considered complete (and it departs from the computing system) only when all its tasks have been completed.

Performance evaluation of this type of fork-join parallel program is difficult for the following reasons:

1) The arrival processes to the servers are correlated so the model cannot be decomposed,
2) Since each server has infinite queuing capacity, the state space of the system is multidimensional in nature and infinite in each dimension,
3) There is no known closed-form solution and,
4) It is impossible to use conventional numerical methods directly, since the state space of the problem is infinite.

Since this type of work is closely related to fork-join queuing systems, we start with a brief review of the published literature on the analysis of fork-join queuing models. First, exact analysis is possible when the system is significantly simplified, for example, if we assume the job arrival process is Poisson with tasks having exponential service time distribution and the number of servers, K, is equal to two. The exact analysis for this system can be found in [7], [8], [24]. For more than two servers, approximation and heuristic techniques are used to solve the problem. For example, approximate solutions for fork-join synchronization is reported by Thomasian and Tantawi in [31].

- *J.C.S. Lui is with the Department of Computer Science and Engineering, Chinese University of Hong Kong, Shatin, NT, Hong Kong. E-mail: cslui@cse.cuhk.edu.hk.*
- *R.R. Muntz is with the Computer Science Department, University of California at Los Angeles, 3732 Boelter Hall, Los Angeles, CA 90024-1596.*
- *E-mail: muntz@cs.ucla.edu.*
- *D. Towsley is with the Department of Computer Science, University of Massachusetts, Amherst, MA 01003-4610. E-mail: towsley@cs.umass.edu.*

Manuscript received 10 Jan. 1996.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 100089.

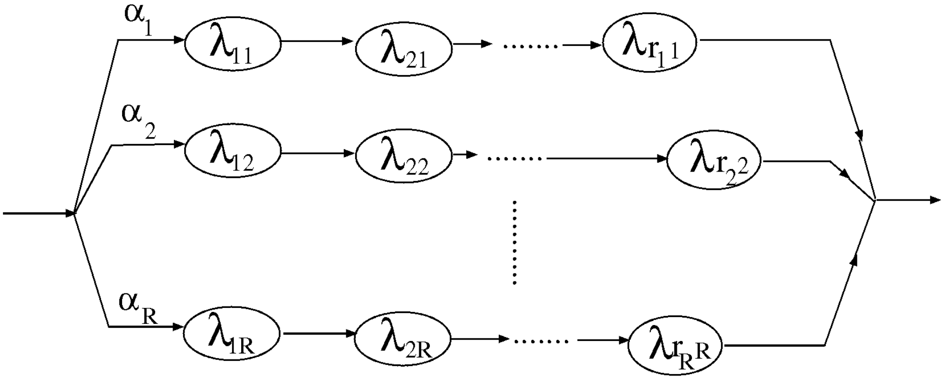

Fig. 1. Series-parallel distribution for the interarrival time of parallel program.

Nelson and Tantawi [24] also proposed a scaling approximation technique for $K \geq 2$ homogeneous exponential servers. Recently, an extension to this approximation approach was developed by Makowski and Varma [22]. Approximation of fork-join queue in terms of virtual waiting time was presented [14], where the solution technique is restricted to the case that the interarrival distributions are exponentially or hyperexponentially distributed and service time distributions are exponential or two-stage Erlangian. Heidelberger and Trivedi [10] developed an approximate solution with bounds for a queuing network with jobs having computation graphs with a fork phase only (i.e., no synchronization of the completion of the tasks). In [11], the model is extended to contain a fork and join node and two approximation techniques are presented, one is based on a decomposition approach and other is based on an iterative solution method. Stability conditions for fork-join queuing networks were investigated by Konstantopoulos and Walrand [16]. Approximate analysis based on nearly-completely decomposability and Gauss-Seidel method for closed fork-join queuing system was reported in [18]. A more general model is considered in [1], [2], where arrival and service processes are general. An upper bound is obtained by assuming K mutually independent $GI/G/1$ parallel queuing systems, while the lower bound assumes $D/G/1$ parallel queuing systems, but the tightness of these bounds is not investigated. In [4], bounds on job response time were derived based on two-server models only. Last, models have been investigated for programs exhibiting parallel fork-join structures that are executed on multiple servers sharing a single queue [17], [25], [32].

The goal of our paper is to provide an efficient algorithm to compute the performance measure, such as the expected response time, of a parallel program that has the fork-join structure. Rather than proposing another approximation technique without presenting the error bounds as in previous literature, our algorithm can guarantee an error bound. If one desires, more computation can be used to obtain a tighter error bound on the derived performance measure.

The approaches we use to solve the problem are:

1) Modify the state space of the original model so that the problem is mathematically tractable,
2) Show that the modified systems lead to a lower and an upper bound on the performance measure,
3) The modifications to the original system should be as few as possible so that the bounds are tight, and
4) The modified systems have a special structure that can be easily solved numerically.

To illustrate the usefulness of our methodology, an example of a parallel program executing under a workstation cluster environment is presented. Through this example, we illustrate that an excellent relative accuracy can be obtained with modest computational cost. Therefore, by providing an efficient computational algorithm and tight error bounds, we distinguished our results from previous work on this problem.

The organization of the paper is as follows. In Section 2, we formally define the parallel program with fork-join computational paradigm that we are analyzing. In Sections 3 and 4, we present two modified models; one gives an upper bound for the expected response time and the other provides a lower bound on the expected response time. We will illustrate how these modified models have a special transition structure which yields an efficient numerical solution. Also, we will prove that these models do provide bounds and we will also present the algorithmic approach to computing these bounds. In Section 5, we present an application with some numerical examples. The example illustrates how to apply the theory and the computational algorithm. Lastly, conclusions are given in Section 6.

2 Model Description

We model a multiprocessing computing system as a queuing system which consists of K identical servers, each having an infinite capacity queue. Let τ be the random variable of the the interarrival time of a parallel program to the multiprocessing system and $A^*(s)$ denotes its Laplace Transform. The class of interarrival distribution we allow is the series-parallel phase type [15] which has the form[1]:

$$A^*(s) = \sum_{j=1}^{R} \alpha_j \prod_{i=1}^{r_j} \frac{\lambda_{ij}}{s + \lambda_{ij}}$$

$$\text{for } 0 < \alpha_j < 1, \ \sum_{j=1}^{R} \alpha_j = 1 \text{ and } \lambda_{ij} > 0 \qquad (1)$$

1. Series-parallel phase type implies that the interarrival time of the next job, with probability of α_j, will have a j-stage general Erlang distribution.

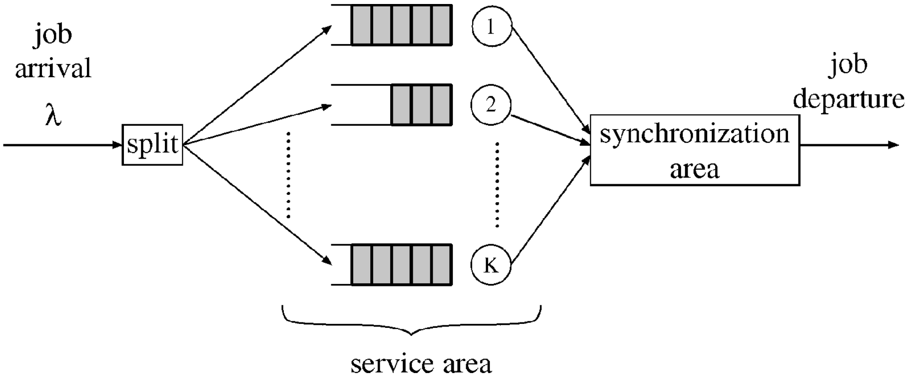

Fig. 2. Parallel program with fork-join model.

such that $-\frac{dA^*(s)}{ds}\Big|_{s=0} = \lambda^{-1}$ and λ is the average arrival rate of parallel programs to the system. This class of series-parallel phase type distribution is very general in that it includes such common distributions like exponential, Erlang, hyperexponential, …, etc. Fig. 1 illustrates this class of series-parallel distribution, where each stage represents an exponentially distributed random variable τ_{ij} with parameter λ_{ij}. Without loss of generality, let there be $R \geq 1$ parallel phases and let phase i have $r_i \geq 1$ series stages.

Upon arrival, the parallel program forks into K tasks, namely t_i, $1 \leq i \leq K$. Task t_i, which is assumed to have a k-stage Erlang distribution with mean service time requirement of $1/\mu$, is assigned to the ith processor. Each processor is modeled as a single server with infinite queuing capacity and the FCFS scheduling discipline. The system model is depicted in Fig. 2. The stability condition for the system is [22]:

$$\lambda < \mu, \tag{2}$$

which is the same as the local stability condition for each server in the system. A job (i.e., parallel program) leaves the system as soon as all its task executions are completed. For a partially completed job, the completed tasks are forced to wait in the synchronization area. A job completes and departs from the system when its last task execution has been completed. We are interested in the expected response time of the parallel program which is denote by T.

We can construct a Markov model, M, for this queuing system with state space description as:

$$N = ([a, b], [N_1, p_1], \ldots, [N_K, p_K])$$

for

$$1 \leq b \leq R,\ 1 \leq a \leq r_b,\ N_i \geq 0,\ 0 \leq p_i \leq k,$$

where $[a, b]$ indicates the state of the interarrival process,[2] N_i is the number of tasks in the waiting queue of the ith server, p_i indicates the servicing stage of the task in the ith server. By convention, $p_i = 0$ when there is no task being serviced at server i. The transition rate matrix G of the Markov process M can be specified as follows. If the event is a transition in the phase of the interarrival distribution,[3] we have:

$$([a, b], [N_1, p_1], \ldots, [N_i, p_i], \ldots, [N_K, p_K]) \rightarrow ([a + 1, b], [N_1, p_1], \ldots, [N_i, p_i], \ldots, [N_K, p_K])\ 1\{a < r_b\}\ \lambda_{ab}$$

$$([a, b], [N_1, p_1], \ldots, [N_i, p_i], \ldots, [N_K, p_K]) \rightarrow ([1, c], \delta([N_1, p_1]), \ldots, \delta([N_i, p_i]), \ldots, \delta([N_K, p_K]))\ 1\{a = r_b\}\ \lambda_{ab}\alpha_c,$$

where:

$$\delta([N_i, p_i]) = \begin{cases} [N_i + 1, p_i] & \text{if } p_i \neq 0 \\ [0, 1] & \text{if } p_i = 0. \end{cases}$$

If the event is a departure from the ith server, we have:

$$([a, b], [N_1, p_1], \ldots, [N_i, p_i], \ldots, [N_K, p_K]) \rightarrow ([a, b], [N_1, p_1], \ldots, [N_i, p_i + 1], \ldots, [N_K, p_K])\ 1\{p_i < k\}\ k\mu$$

$$([a, b], [N_1, p_1], \ldots, [N_i, p_i], \ldots, [N_K, p_K]) \rightarrow ([a, b], [N_1, p_1], \ldots, [N_i - 1, 1], \ldots, [N_K, p_K])\ 1\{p_i = k\}\ 1\{N_i > 0\}\ k\mu$$

$$([a, b], [N_1, p_1], \ldots, [N_i, p_i], \ldots, [N_K, p_K]) \rightarrow ([a, b], [N_1, p_1], \ldots, [0, 0], \ldots, [N_K, p_K])\ 1\{p_i = k\}\ 1\{N_i = 0\}\ k\mu.$$

The unique steady state probability vector for this continuous-time discrete state (CTDS) Markov process M satisfies the following system of linear equations:

$$\vec{\pi} G = \vec{0} \quad \text{and} \quad \vec{\pi}\, \underline{e} = 1, \tag{3}$$

where $\vec{\pi}$ is the K-dimensional steady state probability vector, $\underline{e}$ denotes an appropriately dimensioned column vector of 1s, and G is the transition rate matrix.

Given the probability steady state vector, the expected job response time can be computed by:

2. a and b represent the current parallel and series stage of the arrival job. Note that if the arrival process is Poisson and task service times are exponential, the state space description $(N_1, \ldots, N_K)$ will suffice, where N_i is the number of tasks in the ith server.

3. We assume that the arrival process for stage $[a, b]$ has rate of λ_a and the mean arrival rate of external jobs is λ.

$$T = \frac{\text{Expected number of tasks in the system}}{\text{arrival rate of tasks}} = \frac{\sum_{s \in S} \pi(s) f(s)}{K\lambda}, \quad (4)$$

where S is the state space of the Markov model M, $\pi(s)$ is the steady state probability of state s, and $f(s)$ is the total number of tasks in the system when in state s. To see how to express $f(s)$, we first show how, given a state s, we can find the number of tasks waiting in the synchronization area. Define $\hat{N}_i = N_i + 1\{p_i > 0\}$ so that $\hat{N}_i$ is the number of tasks at the ith server.[4] To derive the number of tasks waiting in the synchronization area, we define $\hat{N}_i = N_i + 1\{p_i > 0\}$. Then, the number of tasks that are waiting in the synchronization area is:

$$\sum_{i=1}^{K} \left(N^* - \hat{N}_i \right) \quad (5)$$

Now we can express $f(s)$ as:

$$f(s) = \sum_{i=1}^{K} \hat{N}_i + \sum_{i=1}^{K} \left(N^* - \hat{N}_i \right) = KN^*. \quad (6)$$

The above model does not possess a known closed form solution, and it is not possible to solve the problem numerically due to its infinite state space. The methodology we choose to solve this model is as follows: We first construct two models that can closely bound the performance (in our case, the performance measure is the expected job response time), then we show that these modified models can be evaluated *efficiently* by numerical methods.

Before we go on to the next section, let us describe briefly the intuition behind the construction of the models that provide the upper and lower bounds on the expected job response time. First, it is reasonable to argue that the distribution of stationary state probabilities for the model M is *skewed*, that is, the probability mass is concentrated in some relatively small subset of the state space, rather than distributed uniformly over the entire state space. The reasoning is that, when a parallel program arrives into the system, it adds load uniformly to all K processors. Therefore, a parallel program arrival will not increase any differences in the queue lengths. On the other hand, since the system is homogeneous, all processors are servicing their workload at the same rate, therefore, the probability of having a large difference in the waiting queue length among all processors should be relatively small. For example, for a four-processors system and parallel programs which are characterized by a Poisson arrival and exponential service times, the steady state probability for state (8, 8, 8, 8) should have a higher state probability than state (29, 1, 1, 1). This insight suggests that we should, in any modified model, represent the exact behavior (e.g., transition rates) for the most *popular* states of the original model M. Note that the number of states in the most popular subset will be a function of the accuracy demanded and the computational cost one is willing to pay. When the modified system leaves these *popular* states, we modify the behavior of the system in such a manner that we can

1) prove the modified model provides a bound (either upper or lower) for the original model, and
2) solve the modified system with an efficient algorithm.

3 UPPER BOUND MODEL M^u

In this section, we describe the behavior of the upper bound model M^u. As in the original model, the upper bound model M^u has K parallel servers. A parallel program arrives to the multiprocessing system according to a series-parallel arrival process with mean rate λ. Upon arrival, a parallel program splits into K tasks, each has a k-stage Erlang distribution with mean $1/\mu$, and task t_i is assigned to the ith processor of the computing system. To model the Erlang distribution of a task, we let each task be composed of k subtasks and each subtask has an exponential service time distribution with mean $(k\mu)^{-1}$. Each server will service subtasks in a FCFS order. Again, a parallel program departs from the system only when its last task execution is complete.

We first give an intuitive argument on why the modifications we will make to the original model provides an upper bound on the expected response time, then we proceed to prove the validity of the bound. The first modification we make to the original model M is to impose a queue length difference threshold, which limits the difference in the number of waiting tasks between any two servers to be less than or equal to d. To satisfy this criterion, we force task departures to observe the following rule: When a task tries to depart from the a server, if its departure would cause the *difference* between the longest and shortest queue of waiting tasks to exceed d, then this task is not allowed to depart but must remain in service and repeat its last stage of service. Note that we only need to impose this condition at a task departure instant because task arrival instants will never increase the difference in the queue lengths. An important point is that, since the parallel program's expected response time is a function of its last departing task, disallowing a task departure due to violation of the d threshold implies we are delaying the departure of a task that would have to wait in the synchronization area anyway. The delay of completion of a task can, intuitively, only delay the completion of the parallel program and, therefore, should increase the expected response time. With this modification, we exploit the *skewness property* we mentioned in the last section and we also reduce the state space of the problem of the original model. Let us now formally define this threshold parameter.

DEFINITION 1. *Let d be a predefined threshold for the upper bound model M^u. We require that, at any instant in time, the difference in the number of waiting tasks in any two servers is less than or equal to d.*

We now proceed to formally prove that this modification provides an upper bound on the expected job response time. It is interesting to point out that the proof is applicable for a general arrival process and it is not restricted to the arrival process specified in (1). The approach

4. This is true because N_i only represents the number of *waiting* tasks at the ith server only.

we use is to show that M^u gives an upper bound on the number of tasks in the system at any given point in time using sample path analysis [19]. In the case that the model exhibits stationary behavior, we apply (4) to obtain the expected job response time.

To establish the upper bound, let us use the notion of *subtask*. Since each task has a service time which is a random variable with a k-stage Erlang distribution, whenever a task arrives to a server, it generates k subtasks to that server. To prove the upper bound, we concentrate on the time instants when certain events occur. There are two kinds of events of interest, namely, job arrivals and subtask departures. In the latter case, it is useful to think of each server as continuously serving customers. Hence, a *service event*[5] at server k occurs as a Poisson process with parameter k/μ. For the original model M, define $\mathcal{N}_i = (\mathcal{N}_{i,1},\cdots, \mathcal{N}_{i,K})$ to be the joint queue lengths (i.e., number of subtasks) immediately after the ith event. Let $\mathcal{N}_0$ denote the initial queue lengths at time 0. We have the following evolution equations. If the $(i + 1)$th event is a job arrival, then,

$$\mathcal{N}_{i+1,l} = \mathcal{N}_{i,l} + k \qquad 1 \le l \le K. \tag{7}$$

If the $(i + 1)$th event is a subtask service event at the jth server, then the joint queue length will be:

$$\mathcal{N}_{i+1,l} = \begin{cases} \mathcal{N}_{i,l}, & l \ne j, \\ (\mathcal{N}_{i,j} - 1)^+ & l = j. \end{cases} \qquad 1 \le l \le K. \tag{8}$$

For the upper bound model M^u, we define a binary valued random variable Y_i that takes on the value 0 if a subtask is not allowed to depart and the value 1 if a subtask is allowed to depart at the ith event, provided that it is a service event. Observe that we have:

$$Y_i = 1 \qquad \text{for model } M$$

$$Y_i \in \{0, 1\} \qquad \text{for model } M^u$$

since the value of Y_i for the model M^u depends on whether the queue length difference threshold d is satisfied or not. Let $\mathcal{N}^u(t)$ be the joint queue lengths (measured in subtasks) in the upper bound model M^u. We have the following evolution equations at the time of arrival and service events: If the $(i + 1)$th event corresponds to a parallel program arrival,

$$\mathcal{N}^u_{i+1,l} = \mathcal{N}^u_{i,l} + k \qquad 1 \le l \le K. \tag{9}$$

If it is a task service event at server j,

$$\mathcal{N}^u_{i+1,l} = \begin{cases} \mathcal{N}^u_{i,l}, & l \ne j, \\ (\mathcal{N}^u_{i,j} - Y_{i+1})^+, & l = j. \end{cases} \tag{10}$$

DEFINITION 2. *Let* **X** *and* **Y** *be vectors of real valued random variables. We say* **Y** *is stochastically larger than* **X** *(written as* $\mathbf{X} \le_{st} \mathbf{Y}$*) iff* $E[h(\mathbf{X})] \le E[h(\mathbf{Y})]$ *for all increasing functions h.*

THEOREM 1. *If* $\mathcal{N}(0) \le_{st} \mathcal{N}^u(0)$*, then* $\mathcal{N}(t) \le_{st} \mathcal{N}^u(t)$.

PROOF. Please refer to the Appendix.

COROLLARY 1. *If* $\mathcal{N}(0) \le_{st} \mathcal{N}^u(0)$*, then* $T \le T^u$*, where* T (T^u) *is the expected response time of parallel program for model* M (M^u).

PROOF. Let $\mathbf{N}(t)$ and $\mathbf{N}^u(t)$ be the joint queue lengths for model M and M^u at time t, respectively, and $\mathcal{N}_i(t)$ $\left(\mathcal{N}^u_i(t)\right)$ be the number of sub-tasks in the ith server for model M (M^u) at time t. We have:

$$\mathbf{N}(t) = (\lceil \mathcal{N}_1(t)/k \rceil, \ldots, \lceil \mathcal{N}_K(t)/k \rceil)$$
$$\mathbf{N}^u(t) = (\lceil \mathcal{N}^u_1(t)/k \rceil, \ldots, \lceil \mathcal{N}^u_K(t)/k \rceil).$$

Based on Theorem 1, we have $\mathcal{N}(t) \le_{st} \mathcal{N}^u(t)$. Since $\lceil \mathcal{N}_i(t)/k \rceil \le \lceil \mathcal{N}^u_i(t)/k \rceil$, we have $\mathbf{N}(t) \le_{st} \mathbf{N}^u(t)$. Now, let $\mathbf{N} = \lim_{t\to\infty} \mathbf{N}(t)$ and $\mathbf{N}^u = \lim_{t\to\infty} \mathbf{N}^u(t)$. Because the function f defined in (6) is an increasing function, then, based on (4), we have $T \le T^u$. □

With the queue length difference threshold d, M^u has a reduced t state space compared with the original model M, but it is still infinite. The next modification is used to impose a *regular, repetitive* transition rate structure to the upper bound model so that an efficient solution technique such as aggregation [5] can be applied. One way to impose a regular and repetitive transition rate structure is to use the concept of nested busy cycles. The next defintion and remark will clarify this concept.

DEFINITION 3. *Let* $\overline{\mathbf{C}}$ *be a K-dimensional vector,* $\overline{\mathbf{C}} = (C, \ldots, C)$*, where C is a predefined trigger threshold for the upper bound model* M^u*. If a parallel program arrives and finds that there are already C waiting tasks in any server, the system will suspend all tasks (except tasks from the current arriving job) and put them into background queues. A new busy cycle begins with the current arriving job.*

REMARKS. A busy cycle starts when an arriving parallel program causes the trigger threshold to be exceeded (or when the arrival finds the system is idle) and the newly arriving job starts the new busy cycle. The busy cycle ends when the foreground queues are all empty. When a busy cycle ends, the last suspended set of tasks (if any) are put into the foreground queues. Note that busy cycles can be *nested*, that is, during a busy cycle, an external parallel program can arrive and trigger another busy cycle. When a busy cycle ends, only the set of tasks suspended at the initiation of that busy period is released for service. Therefore, a foreground task is defined to be any task which arrived during the currently active busy period. The motivation of the trigger threshold $\overline{\mathbf{C}}$ is to create a regular and repetitive transition structure that yields an efficient numerical solution. Intuitively the modified model is an upper bound because, at the end of a nested busy period, we have introduced some extra idle time on some servers; e.g., when some servers are idle but we do not release the suspended tasks until all servers are done with their foreground tasks.

To prove that this modification yields an efficient numerical solution to model M^u, we first describe the state space of the upper bound model. Let us define the following:

DEFINITION 4. *Given parameters K, d,* $\overline{\mathbf{C}}$*, external job interarrival process, and the tasks service time distribution, let* S_{tt}

5. Note that a service event is also a departure event only when there is a subtask in the server.

be an ordered list of states for foreground tasks in the upper bound model M^u such that any additional job arrival will cause the system to switch to a new busy cycle.

EXAMPLE 1. If the arrival process is Poisson, and task service time is exponentially distributed, $K = 3$, $d = 2$, $\overline{\mathbf{C}} = (5, 5, 5)$, then

S_{tt} = {(3, 3, 5), (3, 4, 5), (3, 5, 3), (3, 5, 4), (3, 5, 5), (4, 3, 5), (4, 4, 5), (4, 5, 3), (4, 5, 4), (4, 5, 5), (5, 3, 3), (5, 3, 4), (5, 3, 5), (5, 4, 3), (5, 4, 4), (5, 4, 5), (5, 5, 3), (5, 5, 4), (5, 5, 5)}.

EXAMPLE 2. If the job arrival process and the task service times are two-stage Erlang, $K = 2$, $d = 2$, $\overline{\mathbf{C}} = (3, 3)$, then

S_{tt} = {(1, [1, 1], [3, 1]), (1, [1, 1], [3, 2]), (2, [1, 2], [3, 1]),
(2, [1, 2], [3, 2]), (1, [1, 2], [3, 1]), (1, [1, 2], [3, 2]),
(2, [1, 2], [3, 1]), (2, [1, 2], [3, 2]), (1, [2, 1], [3, 1]),
(1, [2, 1], [3, 2]), (2, [2, 1], [3, 1]), (2, [2, 1], [3, 2]),
(1, [2, 2], [3, 1]), (1, [2, 2], [3, 2]), (2, [2, 2], [3, 1]),
(2, [2, 2], [3, 2]), (1, [3, 1], [1, 1]), (1, [3, 1], [1, 2]),
(2, [3, 1], [1, 1]), (2, [3, 1], [1, 2]), (1, [3, 1], [2, 1]),
(1, [3, 1], [2, 2]), (2, [3, 1], [2, 1]), (2, [3, 1], [2, 2]),
(1, [3, 1], [3, 1]), (1, [3, 1], [3, 2]), (2, [3, 1], [3, 1]),
(2, [3, 1], [3, 2]), (1, [3, 2], [1, 1]), (1, [3, 2], [1, 2]),
(2, [3, 2], [1, 1]), (2, [3, 2], [1, 2]), (1, [3, 2], [2, 1]),
(1, [3, 2], [2, 2]), (2, [3, 2], [2, 1]), (2, [3, 2], [2, 2]),
(1, [3, 2], [3, 1]), (1, [3, 2], [3, 2]), (2, [3, 2], [3, 1]),
(2, [3, 2], [3, 2])}.

Let B (= $|S_{tt}|$) be the cardinality of the set of trigger states S_{tt}. Let $S_{tt}(i)$ be the ith element in S_{tt}, where $0 \le i \le B - 1$. Using Example 1 listed above, we have:

$$B = 19; \qquad S_{tt}(0) = (3, 3, 5), \ldots, S_{tt}(18) = (5, 5, 5).$$

Using Example 2 listed above, we have:

$$B = 40; \qquad S_{tt}(0) = (1, [1, 1], [3, 1]), \ldots, S_{tt}(39) = (2, [3, 2], [3, 2]).$$

DEFINITION 5. *Let $S^u_{i,j}$, $i \ge 0$, $0 \le j \le B^i - 1$, be the set of states for the upper bound model M^u such that*

1) *the system is in the ith (where $i > 0$) nested level of busy cycles,*
2) *the difference in the waiting queue length between foreground tasks among different servers is less than or equal to d, and*
3) *the system entered the ith nested busy cycle from the $(i - 1)$th busy cycle through the trigger state $S_{tt}(j \bmod B) \subset S^u_{i-1,j/B}$ (where "/" is an integer divide operator).*

For the upper bound model M^u, we impose the following rules. Transitions between states in $S^u_{i,j}$ behave like the original model M except:

1) If a task departure from the server would cause the difference between the longest and shortest queue of waiting foreground to exceed d, then this task is not allowed to depart but must remain in service and repeat its last stage of service.
2) If a new job arrives and the system is in one of the trigger states, for example $S_{tt}(l)$, then a new busy cycle begins with this newly arrive job and the system enters state $(1, [0, k], \cdots, [0, k])$ in $S^u_{i+1,jB+l}$.
3) Upon completion of the last foreground job in $S^u_{i,j}$, the previously suspended set of tasks is released for service. That is, the system will make a transition back to one of the trigger states in $S^u_{i-1,j/B}$.

With this definition, we can see that the state space of the upper bound model M^u can be organized as a tree. Each node of the tree represents the set of states $S^u_{i,j}$ of all possible waiting queue lengths for active tasks. Each state in the model has four components, namely

1) a stack of *return states*[6] $(s_1, \ldots, s_i)$, where i is the depth of the node in the tree and,
2) the stage of the interarrival distribution,
3) the waiting queue lengths of active tasks at each servers, and
4) the stage of service for each task in service.

All states in the same node of the tree have the same first component, i.e, the same stack of return states. To illustrate this definition, we use a simple example as depicted in Fig. 3 to illustrate the degenerate case where the arrival process is Poisson and task service times are exponentially distributed, $K = 2$, $d = 2$, $C = 5$, S_{tt} = {(3, 5), (4, 5), (5, 3), (5, 4), (5, 5)}.

We pause now to justify why we want to impose this kind of transition rate structures to M^u. Since the state space for states in $S^u_{i,j}$ is finite and has a special structure, as we will prove in later section, we can easily compute the conditional state probabilities for states in $S^u_{i,j}$ given that the system is operating in $S^u_{i,j}$. Given the conditional state probabilities we can apply exact aggregation [5] to the $S^u_{i,j}$ subsets of states, reducing each subset to a single state. The Markov chain composed of the aggregated states has a tree structure and, therefore, is time reversible and has a closed form solution.

Now we are in a position to prove that the transitional structure we impose on model M^u indeed provides an upper bound on the expected response time of a parallel program. As stated above, upon departure of the last foreground job in $S^u_{i,j}$, there will be a transition back to one of the trigger states in $S^u_{i-1,j/B}$. At this point, we increase the rate of the arrival process (in doing so, we obtain the upper bound) so that we can have an efficient algorithm for solving the upper bound model M^u. Note the we only increase the rate of the arrival process at some imbedded points so as to obtain tight performance bound.

6. In the actual state representation, we do not need to represent the stack of return states since they can be derived easily from the relative position within the tree.

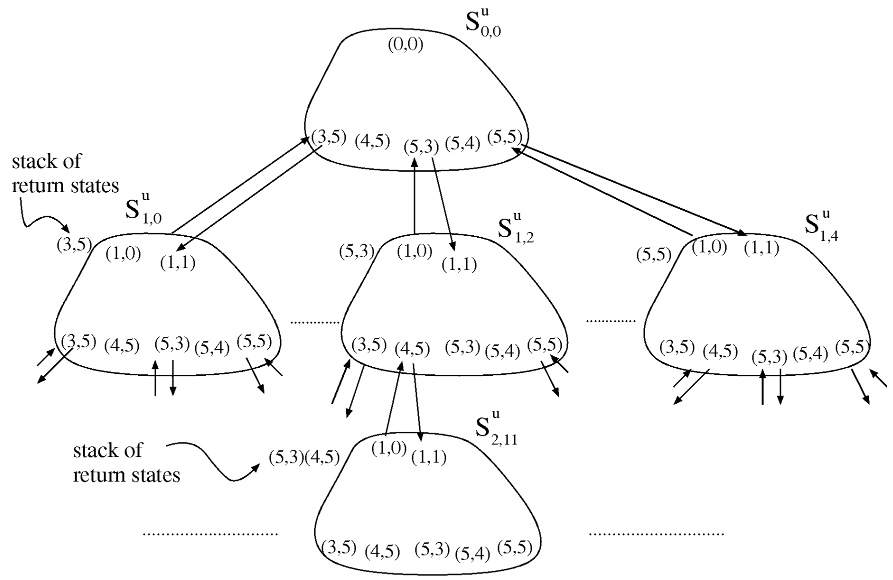

Fig. 3. State space partition for upper bound model M^u.

To formally define this modification, we define random variables $\hat{\tau}_{kj}$ based on the notation used in (1):

$$\hat{\tau}_{kj} = \sum_{l=k}^{r_j} \tau_{lj} \qquad \text{for } 1 \le j \le R \text{ and } 1 \le k \le r_j \tag{11}$$

with the corresponding Laplace transform

$$\hat{A}^*_{kj}(s) = \prod_{l=k}^{r_j} \left(\frac{\lambda_{lj}}{s + \lambda_{lj}} \right).$$

Based on this new notation, we can express the Laplace transform of the parallel program's interarrival time τ as

$$A^*(s) = \sum_{j=1}^{R} \alpha_j \hat{A}^*_{1j}(s).$$

Let us define τ_{max} to be an exponentially distributed random variable with parameter λ_{max} such that:

$$\lambda_{max} = \max_{1 \le j \le R} \lambda_{r_j j}$$

and the corresponding Laplace transform of τ_{max} is

$$A^*_{\lambda_{max}}(s) = \left(\frac{\lambda_{max}}{s + \lambda_{max}} \right).$$

The following theorem relates the original interarrival time of parallel program τ with the new interarrival time random variable τ_{max}.

THEOREM 2. $\tau_{max} \le_{st} \tau$.

PROOF. Please refer to the Appendix.

COROLLARY 2. *Let Z be a random variable that takes on value zero or one. Define*

$$\hat{\tau}_{max} = \begin{cases} \tau & Prob(Z = 0) \\ \tau_{max} & Prob(Z = 1). \end{cases}$$

We have $\hat{\tau}_{max} \le_{st} \tau$.

PROOF. Please refer to the Appendix.

Let $n_F = \{n_F(t), t \ge 0\}$ and let $n_G = \{n_G(t), t \ge 0\}$ denote the stochastic processes for number of tasks in the queuing system with interarrival distribution $F_{\hat{\tau}_{max}}$ and G_τ, respectively. We want to show that:

$$n_G(t) \le_{st} n_F(t) \qquad \text{for } t \ge 0.$$

In preparation for this proof, we need the following lemma about *coupling* between random variables [28]:

LEMMA 1. *If F and G are probability distributions such that $\overline{F}(a) \le \overline{G}(a)$ for all a, then there exist random variables X and Y having distributions F and G, respectively, such that $P\{Y \ge X\} = 1$.*

THEOREM 3. $\{n_G(t), t \ge 0\} \le_{st} \{n_F(t), t \ge 0\}$.

PROOF. To prove the above statement, we constrain the system such that

1) the service times of subtasks are exponential and,
2) there is no a priori knowledge of service time for the scheduling discipline.

Now, only observe the process when there is an arrival. We use the *coupling* argument as follows. Let $X_1, X_2, \ldots$ be independent and distributed according to G_τ. Then, the process generated by X_i, call it n^*_G, has the same distribution as n_G. Now generate independent random variables $Y_1, Y_2, \ldots$ having distribution $F_{\hat{\tau}_{max}}$ such that $Y_i \le X_i$. Then, the process generated by interarrival time Y_i, call it n^*_F, has the same probability distribution as n_F. Since $Y_i \le X_i$ for all i, it follows that:

$$n^*_F(t) \le n^*_G(t) \qquad \text{for all } t$$

$$n_F(t) \le n_G(t) \qquad \text{for all } t$$

$$n_F \le_{st} n_G. \tag{12}$$

□

REMARKS. The significance of Theorem 2, Corollary 2, and Theorem 3 is as follows: At some *imbedded times*, we can modify the interarrival distribution according to Theorem 2 such that we have an upper bound on the number of tasks in the system. Based on Little's result and (4) and (6), we see that it also provides an upper bound on the expected response time. The imbedded times are chosen to be the times at which all the foreground tasks in the model are finished and the system switches to the last suspended set of tasks, which are now to be the foreground tasks. Therefore, the interarrival process of the upper bound model M^u is the same as the original model M, except when a suspended set of tasks are activated. When this occurs, the time till the next arrival is an exponential random variable with parameter λ_{max}. Following this arrival, the original interarrival process is again in effect.

3.1 Computational Procedure for Upper Bound Model M^u

We have proven that the model M^u provides an upper bound on the expected response time of the parallel program, but we need to also have a computationally efficient algorithm for calculating the performance measure. This section presents an efficient algorithm in solving the model M^u. As we have described above, due to the routing of task arrivals, the constraint on tasks departures and the modification of interarrival time distribution, as indicated in Corollary 2, at points where the system switches from $S^u_{i,j}$ to $S^u_{i-1,k}$, the only transition from set $S^u_{i-1,k}$ to set $S^u_{i,j}$ is actually to one state in $S^u_{i,j}$. Furthermore, transitions from $S^u_{i+1,k}$ to $S^u_{i,j}$ can only go to one state in $S^u_{i,j}$. This unique transition structure, as we will show, allows us to efficiently solve the model via *exact* aggregation [5]. After we aggregate each set of states, $S^u_{i,j}$, into a single state denotes as $s^u_{i,j}$, the aggregated model has a tree structure, which implies the aggregated process is time reversible and the steady state probability solution to this aggregated process has a closed form solution [13].

We are now in a position to describe the computational procedure to obtain the expected response time for the upper bound model M^u. Let $Q_{S^u_{i,j},S^u_{l,m}}$ be the transition rate matrix from states in $S^u_{i,j}$ to states in $S^u_{l,m}$. Based on the tree structure of the transitions between states, as depicted in Fig. 3, a brief description of how to compute T^u is as follows:

1) Compute the conditional state probability given that the system is in set $S^u_{i,j}$,
2) Aggregate each $S^u_{i,j}$ into a single state $s^u_{i,j}$,
3) Compute the transition rates between aggregate states,
4) Given the conditional state probabilities of $S^u_{i,j}$ and aggregate state probabilities $s^u_{i,j}$, by applying (4) and (6), we can find the T^u, the expected job response time for the upper bound model M^u.

REMARKS. In computing the conditional probabilities given the system is in $S^u_{i,j}$, observe that the transition from its parents $S^u_{i-1,k}$ is to one state in $S^u_{i,j}$ and the transitions from each of its children $S^u_{i+1,l}$ are through exactly one state in $S^u_{i,j}$, respectively. We need the following lemma from [6], [20], [23] to show that we can compute these conditional probabilities easily.

LEMMA 2. *Assume that we have an irreducible Markov process with state space $S = A \cup B$ and transition rate matrix*:

$$\begin{bmatrix} Q_{A,A} & Q_{A,B} \\ Q_{B,A} & Q_{B,B} \end{bmatrix},$$

where $Q_{i,j}$ is the transition rate submatrix from partition i to partition j. If $Q_{B,A}$ has all zero entries, except for some nonzero entries in the ith column, the conditional steady state probability vector, given that the system is in partition A, is the solution to the following system of linear equations:

$$\vec{\pi}_{|A}\left[Q_{A,A} + Q_{A,B}\underline{e}\underline{e}_i^T\right] = \vec{0}$$
$$\vec{\pi}_{|A}\underline{e} = 1,$$

where $\underline{e}_i^T$ is a row vector with a zero in each component, except for the ith *component, which has the value one.*

The implication of the above lemma is that, if there is one state in A by which A is entered from B, then we can compute the conditional state probability of system in A without knowing the transition structure of $Q_{B,B}$. The following theorem shows that how we can apply Lemma 2 to compute the conditional state probability vector given system is in $S^u_{i.j}$.

THEOREM 4. *Let $\mathcal{M}$ be a Markov process with rate matrix $\hat{Q}_{S^u_{i,j},S^u_{i,j}}$, which is equal to $Q_{S^u_{i,j},S^u_{i,j}}$ except for the following modifications*:

1) *A transition to the parent set of states $S^u_{i-1,k}$ is changed to a transition to the initial state of $S^u_{i,k}$ (which is the state where $S^u_{i,k}$ is entered from $S^u_{i-1,k}$.)*
2) *Each transitions to a child in set $S^u_{i+1,l}$ is changed to a transition to state s, where state s is the entering state for the transition from $S^u_{i+1,l}$ to $S^u_{i,j}$.*

The steady state probability vector for the Markov process with rate matrix $\hat{Q}_{S^u_{i,j},S^u_{i,j}}$ is the conditional state probability vector given the system is in set $S^u_{i,j}$.

PROOF. Please refer to the Appendix.

Due to the way we define our busy cycles, transition rate matrices $\hat{Q}_{S^u_{i,j},S^u_{i,j}}$ are the same for $i \geq 1$. This implies we only need to solve two models, $\hat{Q}_{S^u_{0,0},S^u_{0,0}}$ and $\hat{Q}_{S^u_{1,0},S^u_{1,0}}$, to obtain all the conditional state probabilities. Once we obtain the conditional state probabilities for partition $S^u_{i,j}$, we can aggregate each partition [5] into a single state, which we denote as $s^u_{i,j}$.

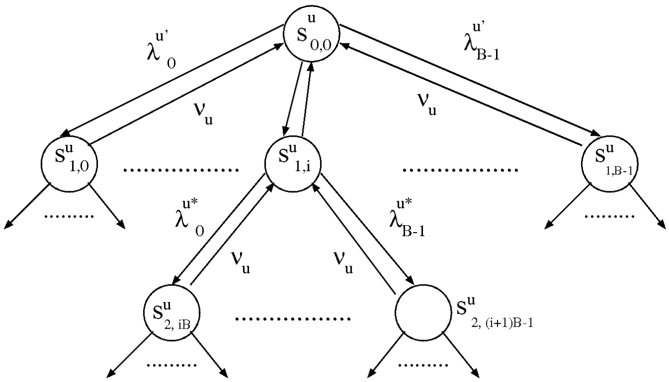

Fig. 4. Aggregated process for upper bound model M^u.

After aggregation, the state transition structure forms a tree, as depicted in Fig. 4, where $\lambda_i^{u'}$, $0 \le i \le B-1$ can be derived from the conditional steady state probabilities of $\hat{Q}_{S^u_{0,0},S^u_{0,0}}$ and λ_i^{u*}, $0 \le i \le B-1$ can be derived from the conditional steady state probabilities of $\hat{Q}_{S^u_{1,0},S^u_{1,0}}$. Let Ψ be the set of states in $S^u_{i,j}$, $i \ge 1$, such that there are transitions from Ψ back to the parent of $S^u_{i,j}$. Then we have:

$$\lambda_i^{u'} = \hat{\pi}'(a)\frac{1}{\tau_{r_j j}}$$
$$\text{for } a \in S_{tt} \text{ where } S_{tt} \subset S^u_{0,0},\ 0 \le i \le B-1,\ 1 \le j \le R \tag{13}$$

$$\lambda_i^{u*} = \hat{\pi}(a)\frac{1}{\tau_{r_j j}}$$
$$\text{for } a \in S_{tt} \text{ where } S_{tt} \subset S^u_{1,0},\ 0 \le i \le B-1,\ 1 \le j \le R \tag{14}$$

$$\nu_u = \sum_{s \in \Psi} \hat{\pi}(s) k\mu, \tag{15}$$

where $\hat{\pi}'(a)$ and $\hat{\pi}(a)$ are the steady state probability for Markov processes with rate matrix $\hat{Q}_{S^u_{0,0},S^u_{0,0}}$ and $\hat{Q}_{S^u_{1,0},S^u_{1,0}}$, respectively.

Let $\pi^*(i, j)$ be the steady state probability for the aggregate state $s^u_{i,j}$. For this aggregated process, we can write the down *local balance equations* [15] as:

$$\pi^*(1,i) = \pi^*(0,0)\frac{\lambda_i^{u'}}{\nu_u} \qquad 0 \le i \le B-1$$

$$\pi^*(k,j) = \pi^*(k-1, j/B)\frac{\lambda^{u*}_{(j \bmod B)}}{\nu_u} \qquad k \ge 1,\ 0 \le j \le B^k - 1.$$

Normalizing all local balance equations, we have:

$$\pi^*(0,0)\left[1 + \frac{1}{\nu_u}\sum_{i=0}^{B-1}\lambda_i^{u'} + \left(\frac{1}{\nu_u}\right)^2 \sum_{i=0}^{B-1}\sum_{j=0}^{B-1}\lambda_i^{u'}\lambda_j^{u*} + \left(\frac{1}{\nu_u}\right)^3 \sum_{i=0}^{B-1}\sum_{j=0}^{B-1}\sum_{k=0}^{B-1}\lambda_i^{u'}\lambda_j^{u*}\lambda_k^{u*} + \cdots\right] = 1.$$

Define $\lambda^{u'} = \sum_{i=0}^{B-1}\lambda_i^{u'}$ and $\lambda^{u*} = \sum_{i=0}^{B-1}\lambda_i^{u*}$, we have:

$$\begin{aligned}\pi^*(i) &= \left[1 + \left(\frac{\lambda^{u'}}{\nu_u}\right) + \left(\frac{\lambda^{u'}}{\nu_u}\right)\frac{\lambda^{u*}}{\nu_u} + \left(\frac{\lambda^{u'}}{\nu_u}\right)\left(\frac{\lambda^{u*}}{\nu_u}\right)^2 + \cdots\right]^{-1} \\ &= \left[1 + \frac{\lambda^{u'}}{\nu_u}\sum_{i=0}^{\infty}\left(\frac{\lambda^{u*}}{\nu_u}\right)^i\right]^{-1} \\ &= \frac{\nu_u - \lambda^{u*}}{\nu_u + \lambda^{u'} - \lambda^{u*}}.\end{aligned} \tag{16}$$

Let $\pi^*(i)$ be the steady state probabilities for all nodes at the ith level of the aggregated process. $\pi^*(i)$ can be expressed as:

$$\pi^*(i) = \sum_{j=0}^{B^i - 1}\pi^*(i,j) = \frac{\lambda^{u'}}{\nu_u}\left(\frac{\lambda^{u*}}{\nu_u}\right)^i\left(\frac{\nu_u - \lambda^{u*}}{\nu_u + \lambda^{u'} - \lambda^{u*}}\right) \qquad i \ge 1. \tag{17}$$

To compute the job response time T^u for the upper bound model, which is based on (4), let us define $r^u_{f_0}$ to be the expected number of tasks given the system is in $S^u_{0,0}$, r^u_f to be the expected number of foreground tasks[7] given the system is in $S^u_{i,j}$ for $i \ge 1$, and $r^u_b(i)$ to be the expected number of background tasks given the system is in $S^u_{i,j}$ for $i \ge 1$. Therefore, given the rate matrix $\hat{Q}_{S^u_{0,0},S^u_{0,0}}$ and $\hat{Q}_{S^u_{i,j},S^u_{i,j}}$, we can compute $r^u_{f_0}$ and r^u_f. The number of background tasks (which includes the completed tasks associated with the jobs that are placed in the background), given that the system is in ith level of busy cycle, is equal to $iK(C + 1)$. Therefore, the total expected number of tasks in the upper bound model M^u is:

$$\begin{aligned}N^u &= \pi^*(0,0)\,r^u_{f_0} + \sum_{i=1}^{\infty}\frac{\lambda^{u'}}{\nu_u}\left(\frac{\lambda^{u*}}{\nu_u}\right)^{i-1}\pi^*(0,0)\left[iK(C+1) + r^u_f\right] \\ &= \pi^*(0,0)\,r^u_{f_0} + \frac{\lambda^{u'}}{\nu_u}\pi^*(0,0)\,r^u_f\sum_{i=0}^{\infty}\left(\frac{\lambda^{u*}}{\nu_u}\right)^i + \\ &\qquad \frac{\lambda^{u'}}{\nu_u}\pi^*(0,0)K(C+1)\sum_{i=0}^{\infty}\left(\frac{\lambda^{u*}}{\nu_u}\right)^i[i+1] \\ &= \pi^*(0,0)\left[r^u_{f_0} + \frac{r^u_f\lambda^{u'}}{\nu_u - \lambda^{u*}} + \frac{\lambda^{u'}K(C+1)\nu_u}{\left(\nu_u - \lambda^{u*}\right)^2}\right] \\ &= \left(\frac{\nu_u - \lambda^{u*}}{\nu_u + \lambda^{u'} - \lambda^{u*}}\right)\left[r^u_{f_0} + \frac{r^u_f\lambda^{u'}}{\nu_u - \lambda^{u*}} + \frac{\lambda^{u'}K(C+1)\nu_u}{\left(\nu_u - \lambda^{u*}\right)^2}\right].\end{aligned} \tag{18}$$

The expected job response time for upper bound model M^u is:

$$T^u = \left[\frac{\nu_u - \lambda^{u*}}{K\lambda\left(\nu_u + \lambda^{u'} - \lambda^{u*}\right)}\right]\left[r^u_{f_0} + \frac{r^u_f\lambda^{u'}}{\nu_u - \lambda^{u*}} + \frac{\lambda^{u'}K(C+1)\nu_u}{\left(\nu_u - \lambda^{u*}\right)^2}\right]. \tag{19}$$

7. Note that r^u_f is the same for $S^u_{i,j}$ for $i \ge 1$.

4 Lower Bound Model

The lower bound model M^l is modeled as having K parallel servers. A parallel program arrives to the system according to a series-parallel phase type distributions. Let τ denote the random variable equal to the interarrival time of parallel programs to the computing system and let $A^*(s)$ denote the Laplace transform of its distribution. The series-parallel phase type distribution we consider in the lower bound model M^l is that each stage represents an exponentially distributed random variable τ_{ij} with parameter λ_{ij}. Without loss of generality, the class of interarrival distributions has the following form:

$$A^*(s) = \sum_{j=1}^{R} \alpha_j \prod_{i=1}^{r_j} \frac{\lambda_{ij}}{s+\lambda_{ij}} \quad \text{for } 0 < \alpha_j < 1,\ \sum_{j=1}^{R} \alpha_j = 1,\ \lambda_{ij} > 0 \tag{20}$$

$$r_i \le r_1 \qquad \text{for } 2 \le i \le R \tag{21}$$

$$\lambda_{i1} \le \lambda_{ij} \qquad \text{for } 2 \le j \le R \text{ and } 1 \le i \le r_j \tag{22}$$

such that $-\frac{dA^*(s)}{ds}\Big|_{s=0} = \lambda^{-1}$, where λ is the average arrival rate of the parallel program to the system. Note that because of (21) and (22), the class of distributions allowed is more restrictive than the interarrival distribution of the upper bound model M^u. But even with these constraints, this class of interarrival distributions can accommodate a large class of distributions such as exponential, Erlang, hyperexponential, …, etc.

Upon arrival, a parallel program is split into K tasks. Task t_i, $1 \le i \le K$, has a k-stage Erlang distribution with mean $1/\mu$, and is assigned to the ith server of the system. Each server will service its assigned tasks in a FCFS order. Again, a parallel program departs from the system when its last task execution is complete. Similar to the upper bound model M^u, we have a threshold d, which places a constraint on the maximum difference in the queue lengths. There is also a threshold trigger $\overline{C}$, such that, if a parallel program arrives and finds that there are already C foreground tasks waiting at any server, a new busy cycle begins with this newly arrived job. The busy cycle ends when all servers complete all tasks which arrive in the current busy cycle. Once again, the definition of a busy cycle implies nested busy cycles, i.e., during a busy cycle, a parallel program arrival can trigger the start of a new busy cycle. When a busy cycle ends, only the set of tasks suspended at the initiation of that busy cycle is released for service.

The lower bound model operates as follows:

- If a task departure would violate the threshold d, it remains in the server for another phase of service (e.g., for k-stage Erlang, the task remains in the last stage of service) and we force a foreground task departure from one of the longest queues.
- When the system has some background tasks waiting and there are K or less foreground tasks in the system, any task departure will force all other foreground tasks to depart from the system.
- When the system has some background tasks waiting and there are more than K foreground tasks, if a foreground task departure would create an idle server (e.g., there is no more foreground tasks for that particular server, while there are still some foreground tasks in other servers), the task remains in the server for another phase of service and we force a task departure from one of the longest foreground task queues instead.

The intuitive argument that the modified model yields a lower bound expected response time is as follows: Since a parallel program's response time is a function of the last task's departure, the constraint on threshold d can force a parallel program to complete *earlier* and thereby obtain a lower bound response time. The second and third departure constraints described above not only force parallel programs to finish earlier, thereby getting lower bound response time, but combined with the trigger threshold $\overline{C}$, this will create a tree structured transition diagram for M^l. With these modifications to the original model, we can use an approach similar to that developed in the last section to efficiently compute the job response time of M^l.

We proceed with the proof that the model M^l provides a lower bound on the expected response time for the parallel program. We will use the concept of *majorization* [21] in the proof, so we first state some basic facts about this concept.

Definition 6. ***X*** *is said to majorize* ***Y*** (*written* $\boldsymbol{X} \prec \boldsymbol{Y}$) *iff*

$$\sum_{l=1}^{k} \hat{X}_l \le \sum_{l=1}^{k} \hat{Y}_l, \quad k = 1, \ldots, K-1$$
$$\sum_{l=1}^{K} \hat{X}_l = \sum_{l=1}^{K} \hat{Y}_l, \tag{23}$$

where $\hat{X}_l$ ($\hat{Y}_l$) *is the l-largest component of* ***X*** (***Y***). *If we replace the equality in* (23) *by* $\sum_{l=1}^{K} \hat{X}_l \le \sum_{l=1}^{K} \hat{Y}_l$, *we obtain a weaker ordering. In this case, we say that* ***Y*** weakly majorizes ***X*** (*written* $\boldsymbol{X} \prec_w \boldsymbol{Y}$).

The following lemma states some properties regarding operations that can be performed on ***X*** and ***Y*** such that weak majorization is preserved.

Lemma 3. *Let* $\boldsymbol{X}, \boldsymbol{Y} \in IN^K$ *such that* $\boldsymbol{X} \prec_w \boldsymbol{Y}$, *then*

1) $\left(\hat{X}_1, \ldots, \hat{X}_k, \ldots, \hat{X}_l + 1, \ldots, \hat{X}_K\right) \prec_w \left(\hat{Y}_1, \ldots, \hat{Y}_k + 1, \ldots, \hat{Y}_l, \ldots, \hat{Y}_K\right)$, *for* $1 \le k \le l \le K$.

2) $\left(\hat{X}_1, \ldots, \left(\hat{X}_k - 1\right)^+, \ldots, \hat{X}_l, \ldots, \hat{X}_K\right) \prec_w \left(\hat{Y}_1, \ldots, \hat{Y}_k, \ldots, \left(\hat{Y}_l - 1\right)^+, \ldots, \hat{Y}_K\right)$, *for* $1 \le k \le l \le K$.

Proof. The proof follows in a straightforward manner from the definition of "$\prec_w$". The reader is referred to [21] for a detailed proof. □

In the following definition, we introduce the concept of a *Schur-convex* function, which will be useful in applying stochastic comparison based on majorization.

Definition 7. *A function* $\phi : IN^K \to IR$ *is said to be Schur-convex iff*

$$\phi(\boldsymbol{X}) \le \phi(\boldsymbol{Y}) \qquad \textit{whenever } \boldsymbol{X} \prec \boldsymbol{Y}, \forall \boldsymbol{X}, \boldsymbol{Y} \in IN^K.$$

DEFINITION 8. *If $\boldsymbol{X}, \boldsymbol{Y} \in IN^K$ are random variables, then we say $\boldsymbol{X}$ is smaller than $\boldsymbol{Y}$ in the sense of Schur-convex order (written $\boldsymbol{X} \leq_{scx} \boldsymbol{Y}$) iff*

$$\phi(\boldsymbol{X}) \leq_{st} \phi(\boldsymbol{Y}), \qquad \text{Schur–convex } \phi.$$

If the class of functions is restricted to be increasing Schur-convex, then we say that $\boldsymbol{X}$ is smaller than $\boldsymbol{Y}$ in the sense of increasing Schur-convex order ($\boldsymbol{X} \leq_{iscx} \boldsymbol{Y}$).

We have a property of these ordering as expressed in the following lemma [21].

LEMMA 4. *Let $\boldsymbol{X}, \boldsymbol{Y} \in IN^K$ be vector valued random variable's such that*:

$$\boldsymbol{X} \leq_{scx} \boldsymbol{Y} \qquad (\boldsymbol{X} \leq_{iscx} \boldsymbol{Y}).$$

There exist two random variable's $\tilde{\boldsymbol{X}}$ and $\tilde{\boldsymbol{Y}}$ such that:

$$\tilde{\boldsymbol{X}} =_{st} \boldsymbol{X}, \tilde{\boldsymbol{Y}} =_{st} \boldsymbol{Y} \text{ and } \tilde{\boldsymbol{X}} \prec \tilde{\boldsymbol{Y}}\left(\tilde{\boldsymbol{X}} \prec_w \tilde{\boldsymbol{Y}}\right).$$

With this lemma, we can couple the behavior of models M and M^l so that we can concentrate on comparing deterministic vectors. Letting $\boldsymbol{N}(t)$ ($\boldsymbol{N}^l(t)$) be the joint queue length of tasks for the model M (M^l) at time t. We have the following theorem:

THEOREM 5. *If $\boldsymbol{N}^l(0) \leq_{iscx} \boldsymbol{N}(0)$, then $\boldsymbol{N}^l(t) \leq_{iscx} \boldsymbol{N}(t)$ for $t \geq 0$.*

PROOF. Please refer to the Appendix.

COROLLARY 3. *If $\boldsymbol{N}^l(0) \leq_{iscx} \boldsymbol{N}(0)$, then $\boldsymbol{N}^l(t) \leq_{st} \boldsymbol{N}(t)$, for $t \geq 0$.*

PROOF. This follows from the preceding theorem and the fact that the function f defined in (6) is a Shur-convex function. □

To illustrate that there is an efficient numerical solution to the model M^l, it is important that we first describe the state space of the lower bound model. Let us define the following.

DEFINITION 9. *Let $S^l_{i,j}$, $i \geq 0$, $0 \leq j \leq B^i - 1$, be the set of states in the lower bound model M^l where $S^l_{i,j}$ contains all states that are*

1) *in the* ith (*where* $i > 0$) *nested level of busy cycles,*
2) *the difference in the waiting queue lengths of foreground tasks among different servers is less than or equal to d, and*
3) *the* ith *nested busy cycle was entered from the* $(i-1)$th *nested busy cycle through the trigger state* $S_{tt}(j \bmod B) \in S^l_{i-1,j/B}$.

For the lower bound model M^l, we impose the following constraints. Transitions between states in $S^l_{i,j}$ behave like the original model M except :

1) For task departures that would violate the threshold d, the task has to remain in the server for another phase of service and the system forces a foreground task departure from one of the longest queues.
2) If a foreground task departure would create an idle server (e.g., there are no more foreground tasks for that particular server), the task has to remain in the server for another phase of service and the system forces a task departure from one of the longest foreground task queues.
3) If the system has some background tasks waiting and there are K or less foreground tasks in the system, any task departure will force all foreground tasks to depart from the server.
4) If a new job arrives and the system is in one of the trigger states, for example $S_{tt}(l)$, then a new busy cycle begins with this newly arrived job and the system enters state (1, [0, k], L, [0, k]) in $S^l_{i-1,jB+l}$.
5) Upon completion of the last foreground job in $S^l_{i,j}$, the previously suspended set of tasks is released for service. That is, the system will make a transition back to one of the trigger states in $S^l_{i-1,j/B}$.

With these constraints, the state space of the lower bound model can be organized as a tree with each node of the tree containing the states in $S^l_{i,j}$. Fig. 5 illustrates the lower bound model M^l for a system where the parallel program has a Poisson arrival process, the task has an exponential service time distribution and with the following system parameters: $K = 2$, $d = 2$, $C = 5$. Also note that, although the definition of $S^l_{i,j}$ in M^l and of $S^u_{i,j}$ in M^u are similar, the internal transition structures are different.

Now we are in the position to prove that the transition structure we impose on the model M^l indeed provides a lower bound on the expected response time of the parallel program. For the first three constraints, Theorem 5 and Corollary 3 indicate that the lower bound on expected response time can be guaranteed. For the last two constraints, the lower bound on the expected response time can be guaranteed by the following theorem and corollaries. As stated above, upon departure of the last foreground job in $S^l_{i,j}$, there will be a transition back to one of the trigger states in $S^l_{i-1,j/B}$. At this point, similar to what was done in the proof for the upper bound model M^u, we change the arrival process so that we can have an efficient algorithm for solving the lower bound model. First, we define random variables $\hat{\tau}_{kj}$ as:

$$\hat{\tau}_{kj} = \sum_{l=k}^{r_j} \tau_{lj} \qquad \text{for } 1 \leq j \leq R \text{ and } 1 \leq k \leq r_j \tag{24}$$

and the Laplace transform of $\hat{\tau}_{kj}$ is $\hat{A}^*_{kj}(s) = \prod_{l=k}^{r_j}\left(\frac{\lambda_{lj}}{s+\lambda_{lj}}\right)$. Based on this notation, we can express the Laplace transform of the parallel program's interarrival time t as

$$A^*(s) = \sum_{j=1}^{R} \alpha_j \hat{A}^*_{1j}(s).$$

First, we have the following observation:

THEOREM 6. $\tau \leq_{st} \hat{\tau}_{11}$.

PROOF. Please refer to the Appendix.

COROLLARY 4. *Let Z be a random variable that takes on value of* 0 *or* 1. *Define*

$$\hat{\tau}^* = \begin{cases} \tau & \text{Prob}(Z = 1) \\ \hat{\tau}_{11} & \text{Prob}(Z = 0) \end{cases}$$

We have $t \leq_{st} \hat{\tau}^*$.

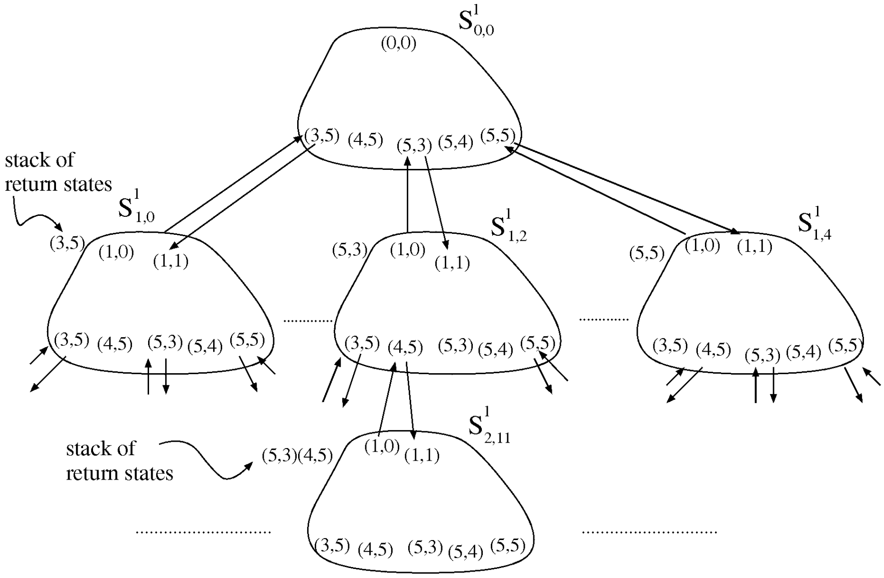

Fig. 5. State space partition for lower bound model M^l.

PROOF. Please refer to the Appendix. □

Let $n_i = \{n_i(t), t \geq 0\}$ for $i = 1, 2$ denote the process defined by the number of tasks at a server when the interarrival distribution is G_τ and $F_{\hat{\tau}^*}$ respectively. We want to show that

$$\{n_2(t), t \geq 0\} \leq_{st} \{n_1(t), t \geq 0\}.$$

COROLLARY 5. $\{n_2(t), t \geq 0\} \leq_{st} \{n_1(t), t \geq 0\}$.

PROOF. The proof is similar to Theorem 3 and is omitted in the interest of space. □

REMARKS. The significance of Theorem 6, Corollary 4, and Corollary 5 is that, at some imbedded time, we can modify the interarrival distribution according to Theorem 6 and obtain a lower bound on the number of tasks in the system. These imbedded times are defined to be the times when all the foreground tasks execution are completed and the system is switching to the last suspended set of tasks (which then becomes the foreground tasks). When this occurs, the next parallel program arrival occurs with an interarrival time $\hat{\tau}_{11}$. After this arrival, the original interarrival process is again in effect.

4.1 Computational Procedure for Lower Bound Model M^l

We now present an efficient algorithm in computing the expected job response time T^l for the lower bound model M^l. Note that the general approach is similar to the approach we took in obtaining the T^u in M^u, that is:

1) Compute the conditional state probability given that the system is in set $S^l_{i,j}$.
2) Aggregate each $S^l_{i,j}$ into a single state $s^l_{i,j}$.
3) Compute the transition rates between aggregate states.
4) Given the conditional state probabilities of $S^l_{i,j}$ and aggregate state probabilities $s^l_{i,j}$, applying (4) and (6), we can find the T^l, the expected job response time for lower bound model M^l.

First, we show how to obtain the conditional state probability given that the system is in set $S^l_{i,j}$.

THEOREM 7. *Consider a Markov process with rate matrix $\hat{Q}_{S^l_{i,j},S^l_{i,j}}$, which is equal to $Q_{S^l_{i,j},S^l_{i,j}}$ except for the following modification*:

1) *A transition to the parent set of states, $S^l_{i-1,k}$, is changed to a transition to the initial state, which is the state where $S^l_{i-1,k}$ is entered.*
2) *Each transition to a child in set $S^l_{i+1,l}$ is changed to the return state from the child in set $S^l_{i+1,l}$.*

The steady state probability vector for the Markov process with rate matrix $\hat{Q}_{S^l_{i,j},S^l_{i,j}}$ is the conditional state probability vector given the system is in set $S^l_{i,j}$.

PROOF. The proof is similar to the proof in Theorem 4. □

With the above theorem, we can aggregate each set of states $S^l_{i,j}$ as a single state $s^l_{i,j}$. After aggregation, the state space is depicted in Fig. 6. The transitions between aggregates states can be computed as follow:

$$\lambda_i^{l'} = \hat{\pi}'(a)\lambda \qquad \text{for } a \in S_{tt}, \text{ where } S_{tt} \subset S^l_{0,0}, 0 \leq i \leq B-1 \quad (25)$$

$$\lambda_i^{l*} = \hat{\pi}(a)\lambda \qquad \text{for } a \in S_{tt}, \text{ where } S_{tt} \subset S^l_{1,0}, 0 \leq i \leq B-1 \quad (26)$$

$$v_l = \sum_{s \in \Psi} \hat{\pi}(s)\mu, \quad (27)$$

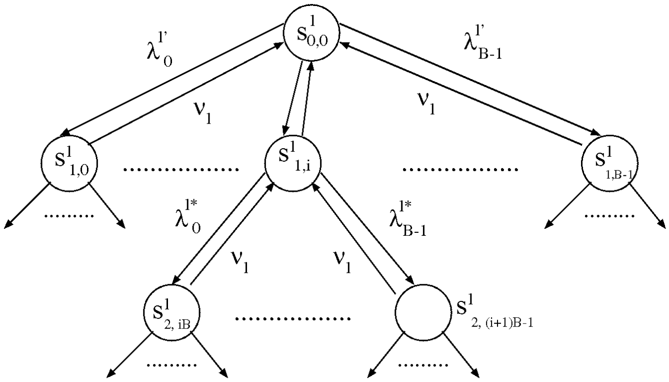

Fig. 6. Aggregated process for lower bound model M^l.

where Ψ is the set of states in $S^l_{i,j}$, $i \geq 1$, such that there are transitions from Ψ back to parents of $S^l_{i,j}$. $\hat{\pi}'(a)$ and $\hat{\pi}(a)$ are the steady state probability for Markov processes with rate matrix $\hat{Q}_{S^l_{0,0},S^l_{0,0}}$ and $\hat{Q}_{S^l_{1,0},S^l_{1,0}}$, respectively.

Let $\pi^*(0, 0)$ be the steady state solution for the aggregated state $s^l_{0,0}$. It is equal to:

$$\pi^*(0,0) = \frac{\nu_l - \lambda^{l*}}{\nu_l + \lambda^{l'} - \lambda^{l*}}. \tag{28}$$

To compute the expected job response time T^l for the lower bound model M^l, let r^l_{f0} be the expected number of tasks given the system is in $S^l_{0,0}$, r^l_f be the expected number of foreground tasks given the system is in $S^l_{i,j}$ for $i \geq 1$ and, $r^l_b(i)$ be the expected number of background tasks given the system is in $S^l_{i,j}$ for $i \geq 1$. Therefore, given the rate matrix $\hat{Q}_{S^l_{0,0},S^l_{0,0}}$ and $\hat{Q}_{S^l_{i,j},S^l_{i,j}}$, we can compute r^l_{f0} and r^l_f. The number of background tasks given the system is in ith level of busy cycle is equal to iKC. Therefore, the total number of tasks in the lower bound model M^l is:

$$N^l = \left(\frac{\nu_l - \lambda^{l*}}{\nu_l + \lambda^{l'} - \lambda^{l*}}\right)\left[r^l_{f0} + \frac{r^l_f \lambda^{l'}}{\nu_l - \lambda^{l*}} + \frac{\lambda^{l'} KC\nu_l}{\left(\nu_l - \lambda^{l*}\right)^2}\right] \tag{28}$$

and the expected job response time T^l for the lower bound model M^l is:

$$T^l = \left[\frac{\nu_l - \lambda^{l*}}{K\lambda\left(\nu_l + \lambda^{l'} - \lambda^{l*}\right)}\right]\left[r^l_{f0} + \frac{r^l_f \lambda^{l'}}{\nu_l - \lambda^{l*}} + \frac{\lambda^{l'} KC\nu_l}{\left(\nu_l - \lambda^{l*}\right)^2}\right]. \tag{29}$$

5 Applications and Numerical Examples

In this section, we present several examples to illustrate the applicability of the bounding algorithm.

Let us consider a computer vision system implemented in a homogeneous workstation cluster under the *Parallel Virtual Machines* (*PVM*) environment [27]. An image will arrive to the computer vision control manager, which will then assign a portion of the image to each workstation for computation. Each workstation will in turn perform some ray-tracing calculation for its part of the image and the total image is ready for display when all workstations have finished their ray-tracing operations. This type of application can be mapped to the fork-join paradigm and the environment is illustrated in Fig. 7.

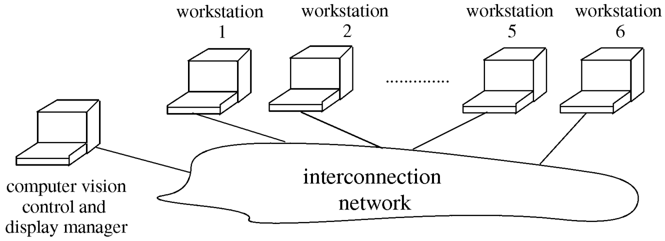

Fig. 7. Computer vision program in workstation cluster environment.

Assume that there are six workstations ready for ray-tracing operations The job arrival distribution is Poisson and the service time distribution of each task is exponential with rate equal to 1.0. We vary the utilization of the system from 0.1 to 0.9 by changing the arrival rate. Table 1 gives the upper and lower bound of the expected job response time. Percentage error is defined to be $\frac{R_u - R_l}{R_u + R_l} \times 100\%$.

The second scenario we illustrate is the same system with six servers but the job arrival process is a Erlang-2 distribution. The service time distribution of tasks are exponential with rate 1.0. Again, we vary the input rate so the utilization of the system can varies from 0.1 to 0.9. Table 2 illustrates the upper and lower bound for the expected job response time.

The third system we illustrate is similar to the second system except the arrival process is Poisson with rate one, but the service time distribution is Erlang-2 distribution. Table 3 illustrates the upper and lower bound for the expected job response time.

For the last example, we want to illustrate the trade-off between computational cost and accuracy of the bounds. Let us consider the system where the job arrival process is Poisson and the task service time is Erlang-2 distribution. By fixing the system utilization at 0.7 and increasing the number of states generated, we see the improvement of the bounds on the mean response time. The results are illustrated in Table 4.

6 Conclusion

The fork-join structure is one of the basic modeling structures for parallel processing. Due to the correlation of arrival processes to all servers and the infinite queuing capacities at each server, no closed-form solution exists for the general case. The problem cannot be solved by direct numerical computation due to its infinite state space. In this paper, we have presented an elegant methodology for computing performance bounds on the expected response time of parallel programs. We take advantage of the skewness property of the original model and then modify the original model such that an efficient computational algorithm is available to solve the modified model. The algorithm also provides the flexibility to tradeoff computational resources and tighter bounds.

TABLE 1
POISSON ARRIVAL AND EXPONENTIAL SERVICE TIME

System Utilization	T^u	T^l	Spread of Bounds	C	d	States Generated	Percentage Error
0.1	2.64	2.67	0.03	8	5	1,218	0.56 %
0.2	2.91	2.96	0.05	8	5	1,218	0.85 %
0.3	3.26	3.32	0.06	8	5	1,218	0.91 %
0.4	3.73	3.81	0.08	8	6	1,848	1.06 %
0.5	4.35	4.45	0.10	8	6	1,848	1.14 %
0.6	5.31	5.44	0.13	9	7	3,300	1.21 %
0.7	7.00	7.20	0.20	9	7	3,300	1.41 %
0.8	10.21	10.52	0.31	10	7	4,092	1.49 %
0.9	19.80	20.33	0.63	11	8	6,864	1.57 %

TABLE 2
TWO-STAGE ERLANG ARRIVAL AND EXPONENTIAL SERVICE TIME

System Utilization	T^u	T^l	Spread of Bounds	C	d	States Generated	Percentage Error
0.1	2.50	2.52	0.02	6	4	924	0.40 %
0.2	2.65	2.68	0.03	6	4	924	0.56 %
0.3	2.92	2.97	0.05	6	4	924	0.85 %
0.4	3.21	3.30	0.09	6	5	1,428	1.38 %
0.5	3.74	3.85	0.11	6	5	1,428	1.45 %
0.6	4.58	4.72	0.14	6	5	1,428	1.50 %
0.7	5.70	5.89	0.19	7	5	1,932	1.64 %
0.8	8.40	8.70	0.30	7	5	1,932	1.75 %
0.9	16.22	16.87	0.65	8	6	3,696	1.96 %

TABLE 3
POISSON ARRIVAL AND TWO-STAGE ERLANG SERVICE TIME

System Utilization	T^u	T^l	Spread of Bounds	C	d	States Generated	Percentage Error
0.1	2.14	2.16	0.02	6	4	5440	0.46 %
0.2	2.31	2.34	0.03	6	4	5440	0.65 %
0.3	2.53	2.57	0.04	6	4	5440	0.78 %
0.4	2.82	2.87	0.05	6	4	5440	0.88 %
0.5	3.29	3.37	0.08	6	5	8,568	1.20 %
0.6	3.90	4.00	0.10	6	5	8,568	1.52 %
0.7	4.90	5.07	0.17	7	5	11,592	1.72 %
0.8	7.01	7.27	0.26	7	6	16,632	1.82 %
0.9	14.70	15.29	0.59	8	7	30,096	1.97 %

TABLE 4
COMPUTATIONAL COST VS. ACCURACY

C	d	States Generated	Response Time Upper Bound(T^u)	Response Time Lower Bound(T^l)	Spread of Bounds	Percentage Errors
7	5	11,592	4.80	5.25	0.44	4.38 %
7	6	16,632	4.90	5.07	0.17	1.72 %
8	7	30,096	4.97	5.05	0.08	0.80 %

There is ongoing work to investigate the possibility of a priori determining d and C_i to obtain specified error bounds and feedback capability of the parallel program as well as extending to non-FCFS service discipline.

APPENDIX

THEOREM 1. *If* $\mathcal{N}(0) \le_{st} \mathcal{N}^u(0)$, *then* $\mathcal{N}(t) \le_{st} \mathcal{N}^u(t)$.

PROOF. Couple the initial joint queue length of subtasks such that $\mathcal{N}(0) \le \mathcal{N}^u(0)$. Condition on the initial joint queue length, arrival event instants, and service event instants. The proof is by induction on all event times to establish the deterministic relationship:

$$\mathcal{N}_i \le \mathcal{N}_i^u, \quad i \ge 0,$$

where $\le$ is taken to mean componentwise.

Basis step. For $i = 0$, $\mathcal{N}(0) \le \mathcal{N}^u(0)$.

Inductive step. For the induction step, assume that $\mathcal{N}_i \le \mathcal{N}_i^u$ holds for $i = k$. For $i = k + 1$, if the ith event is a parallel program arrival, the inequality obviously holds since both models never reject any task arrival.

If the ith event is a service event, since Y_i is less than or equal to one for model M^u, the relationship holds also. Therefore, we have $\mathcal{N}(t) \le \mathcal{N}^u(t)$. By removing the conditioning on initial joint queue length of subtasks, arrival event times, and service event times, we have:

$$\mathcal{N}(t) \le_{st} \mathcal{N}^u(t), \quad \text{for} \quad t \ge 0. \qquad \square$$

THEOREM 2. $\tau_{max} \le_{st} \tau$.

PROOF. First, we show $\tau_{max} \le_{st} \hat{\tau}_{kj}$ for $1 \le j \le R$ and $1 \le k \le r_j$. Since

$$\overline{F}_{\tau_{max}}(x) = e^{-\lambda_{max}x} \qquad \text{for } x \ge 0 \text{ and}$$

$$\overline{F}_{\tau_{r_j},j}(x) = e^{-\lambda_{r_j},j^x} \qquad \text{for } x \ge 0,$$

we have $\overline{F}_{\tau_{max}}(x) \le \overline{F}_{\tau_{r_j},j}(x)$ for all $x \ge 0$, therefore, $\tau_{max} \le_{st} \tau_{r_j,j}$. Since

$$\hat{\tau}_{k,j} = \tau_{k,j} + \cdots + \tau_{r_j,j} \qquad \text{for } 1 \le j \le R \text{ and } 1 \le k \le r_j$$

and τ_{ij} are nonnegative random variables, it is easy to see that $\tau_{max} \le_{st} \hat{\tau}_{k,j}$ for $1 \le j \le R$ and $1 \le k \le r_j$. The interarrival time random variable t, can be expressed as $\tau = \sum_{j=1}^{R} \alpha_j \hat{\tau}_{1j}$, because $\le_{st}$ relationship is preserved under linear combination [28]. Therefore:

$$\tau_{max} = \sum_{j=1}^{R} \alpha_j \tau_{max} \quad \le_{st} \quad \sum_{j=1}^{R} \alpha_j \hat{\tau}_{1j} = \tau. \qquad \square$$

COROLLARY 2. *Let Z be a random variable that takes on value zero or one. Define*

$$\hat{\tau}_{max} = \begin{cases} \tau & Prob(Z = 0) \\ \tau_{max} & Prob(Z = 1) \end{cases}$$

then we have $\hat{\tau}_{max} \le_{st} \tau$.

PROOF. Since

$$\text{Prob}(Z = 1)\tau + \text{Prob}(Z = 0)\tau_{max} \le_{st} \text{Prob}(Z = 1)\tau + \text{Prob}(Z = 0)\tau,$$

therefore,

$$\hat{\tau}_{max} \le_{st} \tau. \qquad \square$$

THEOREM 4. *Let $\mathcal{M}$ be a Markov process with rate matrix $\hat{Q}_{S^u_{i,j},S^u_{i,j}}$, which is equal to $Q_{S^u_{i,j},S^u_{i,j}}$ except for the following modifications*:

1) *A transition to the parent set of states $S^u_{i-1,k}$ is changed to a transition to the initial state of $S^u_{i,k}$ (which is the state where $S^u_{i,k}$ is entered from $S^u_{i-1,k}$.)*
2) *Each transitions to a child in set $S^u_{i+1,l}$ is changed to a transition to state s, where state s is the transition from $S^u_{i+1,l}$ to $S^u_{i,j}$.*

The steady state probability vector for the Markov process with rate matrix $\hat{Q}_{S^u_{i,j},S^u_{i,j}}$ is the conditional state probability vector given the system is in set $S^u_{i,j}$.

PROOF. Without loss of generality, we show how to compute the conditional state probabilities given that the system is in $S^u_{1,0}$. Let us partition the rate matrix of the model M^u as follows:

$$\begin{bmatrix} Q_{S^u_{1,0},S^u_{1,0}} & Q_{S^u_{1,0},T_1} & \cdots & Q_{S^u_{1,0},T_B}, & Q_{S^u_{1,0},S'_T} \\ Q_{T_1,S^u_{1,0}} & Q_{T_1,T_1} & 0 & 0 & 0 \\ \vdots & 0 & \ddots & 0 & 0 \\ Q_{T_B,S^u_{1,0}} & 0 & \vdots & Q_{T_B,T_B} & 0 \\ Q_{S'_T,S^u_{1,0}} & 0 & 0 & 0 & Q_{S'_T,S'_T} \end{bmatrix},$$

where T_i, $1 \le i \le B$ is the ith subtree under $S^u_{1,0}$ and T' is :

$$T' = S^u - S^u_{1,0} - \bigcup_{i=1}^{B} T_i,$$

and S^u is the state space of the upper bound model M^u. Note that transition matrices $Q_{T',S^u_{1,0}}$ and $Q_{T_i,S^u_{1,0}}$, for $i \ge 1$ are transition rate matrices with one nonzero column vector; this implies that there is only one way to enter $S^u_{1,0}$ through its parent and only one way via its children. By Lemma 2, the modification (1) provides the conditional state probabilities vector given the system is in $S^u_{1,1}$ and all its children. $\square$

THEOREM 5. *If $\mathbf{N}^l(0) \le_{iscx} \mathbf{N}(0)$, then $\mathbf{N}^l(t) \le_{iscx} \mathbf{N}(t)$ for $\forall t \ge 0$.*

PROOF. Using Lemma 4, we first couple the initial queue length of these two models such that $\mathbf{N}^l(0) \prec_w \mathbf{N}(0)$. Now, condition on the arrival times so that we can examine the two systems with exactly the same arrival sample path. For each server, we associate a service event process, which is a Poisson process, with parameter $k\mu$, where k is the number of stages for the task service Erlang distribution. Whenever a service event occurs associated with a server, a departure occurs if there is one or more tasks in the queue at the time of the event. Note that the coupling of the service event times at the different servers is only possible if the service times at the servers are all mutually independent sequences of i.i.d exponential random variables with the same parameter.[8]

Let $\{t_i\}_{n=0}^{\infty}$ be the sequence of times at which arrivals or service events occur ($t_0 \equiv 0$). We will establish the relation $\mathbf{N}^l(t) \prec_w \mathbf{N}(t)$ by induction on the event times.

Basis step. This follows from the coupling of the initial queue lengths.

Inductive step. Assume that $\mathbf{N}^l(t) \prec_w \mathbf{N}(t)$ for $t < t_i$. We will establish that $\mathbf{N}^l(t_i) \prec_w \mathbf{N}(t_i)$. Now there are several cases:

8. This is the origin of the restriction of service time to Erlangian service variables.

Arrival Event. If the event is a parallel program arrival, since it adds k subtasks to each server, the $\boldsymbol{N}^l(t_i) \prec_w \boldsymbol{N}(t_i)$ relationship is preserved (this is easily seen as a generalization of Lemma 3).

Service Event. There are several cases:

- *Normal task departure*: If a service event occurs at the ith server and both systems, M and M^l, allow this departure (i.e., the ith server is busy in both systems), $\prec_w$ is clearly preserved by Lemma 3, Property 1.
- *Constraint on threshold d*: If the lower bound model M^l disallows the task departure from the ith server and forces a task departure from the longest queue due to threshold d constraint, then Property 2 of Lemma 3 can be applied to see that the $\prec_w$ relationship is preserved.
- Constraint on task departures when there are background tasks: In the lower bound model M^l, we have the constraint that if there are background tasks, we disallow any task departure which will create an idle server by replacing the original departure with either a departure of a foreground task from the longest queue or a flush of all foreground tasks. In either case, we see that by Property 2 of Lemma 3, the $\prec_w$ relationship is preserved.

This completes the induction step and, thus, we have $\boldsymbol{N}^l(t) \prec_w \boldsymbol{N}(t)$, $t \geq 0$. By the definition of weak majorization ($\prec_w$), this implies that $f(\boldsymbol{N}^l(t)) \leq f(\boldsymbol{N}(t))$ for any increasing Shur-convex function $f(t)$. Removing the conditioning on the arrival times and service times, we have

$$\boldsymbol{N}^l(t) \leq_{iscx} \boldsymbol{N}(t) \qquad \forall t > 0. \qquad \square$$

THEOREM 6. $\tau \leq_{st} \hat{\tau}_{11}$.

PROOF. Based on (22), we have $\lambda_{ij} \geq \lambda_{i1}$ for $2 \leq j \leq R$ and $1 \leq i \leq r_j$, therefore, we have $\lambda_{ij} \leq_{st} \lambda_{i1}$ for $2 \leq j \leq R$ and $1 \leq i \leq r_j$. We then have the following observations:

$$\sum_{k=i}^{r_j} \tau_{kj} \leq_{st} \sum_{k=i}^{r_j} \tau_{k1} \qquad \text{for } 2 \leq j \leq R \text{ and } 1 \leq i \leq r_j$$

$$\hat{\tau}_{1j} \leq_{st} \sum_{k=1}^{r_j} \tau_{k1} \leq_{st} \hat{\tau}_{11} \qquad \text{for } 2 \leq j \leq R$$

$$\tau = \sum_{j=1}^{R} \alpha_j \hat{\tau}_{1j} \leq_{st} \sum_{j=1}^{R} \alpha_j \hat{\tau}_{11} = \hat{\tau}_{11}. \qquad \square$$

COROLLARY 4. *Let Z be a random variable that takes on value of zero or one. Define*:

$$\hat{\tau}^* = \begin{cases} \tau & \text{Prob}(Z=1) \\ \hat{\tau}_{11} & \text{Prob}(Z=0). \end{cases}$$

We have $\tau \leq_{st} \hat{\tau}^*$.

PROOF. Since

$$\text{Prob}(Z=1)\tau + \text{Prob}(Z=0)\tau \leq_{st} \text{Prob}(Z=1)\tau + \text{Prob}(Z=0)\hat{\tau}_{11},$$

therefore, $\tau \leq_{st} \hat{\tau}^*$.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous referees for their helpful and insightful comments. John C.S. Lui was supported in part by the Croucher Foundation and the UGC Research Grant. Richard R. Muntz was supported in part by the U.S. National Science Foundation under Grant IRI-9527178. Don Towsley was supported by the U.S. National Science Foundation under Grant NCR-9116183.

REFERENCES

[1] F. Baccelli, A.M. Makowski, and A. Shwartz. "The Fork-Join Queue and Related Systems with Synchronization Constraints: Stochastic Ordering and Computable Bounds," *Advanced Applied Probability*, vol. 21, pp. 629-660, 1989.

[2] F. Baccelli, W.A. Massey, and D. Towsley, "Acyclic Fork-Join Queueing Networks," *J. ACM*, vol. 36, no. 3, pp. 615-642, July 1989.

[3] A. Beguelin, J.J. Dongarra, A. Geist, R. Manchek, and V. Sunderam, "PVM and HeNCE: Tools for Heterogeneous Network Computing," *Advances in Parallel Computing: Environments and Tools for Parallel Scientific Computing*, pp. 139-153, 1993.

[4] S. Chen and D. Towsley, "Design and Modeling Policies for Two Server Fork/Join Queueing Systems," COINS Technical Report 91-39, Univ. of Massachusetts, 1991.

[5] P.J. Courtois, *Decomposability—Queueing and Computer System Applications*. New York: Academic Press, 1977.

[6] P.J. Courtois and P. Semal, "Computable Bounds for Conditional Steady-State Probabilities in Large Markov Chains and Queueing Models," *IEEE J. Systems and Circuits*, vol. 4, no. 6, Sept., 1986.

[7] L. Flatto and S. Hahn, "Two Parallel Queues Created by Arrivals with Two Demands I," *SIAM J. Applied Math.*, vol. 44, pp. 1,041-1,053, Oct. 1984.

[8] L. Flatto, "Two Parallel Queues Created by Arrivals with Two Demands II," *SIAM J. Applied Math.*, vol. 45, pp. 861-878, Oct. 1985.

[9] G.A. Geist and V.S. Sunderam, "The PVM System: Supercomputing Level Concurrent Computations on Heterogeneous Network of Workstations," *Sixth Distributed Memory Computing Conf. Proc.*, pp. 258-261, Apr. 1991.

[10] P. Heidelberger and K.S. Trivedi. "Queueing Network Models for Parallel Processing with Asynchronous Tasks." *IEEE Trans. Computers*, vol. 31, no. 11, pp.1,099-1,109, Nov. 1982.

[11] P. Heidelberger and K.S. Trivedi, "Analytic Queueing Models for Programs with Internal Concurrency," *IEEE Trans. Computers*, vol. 32, no. 11, pp. 73-82, Nov. 1983.

[12] C.A.R. Hoare, *Communicating Sequential Processes*. London: Prentice-Hall Int'l, 1985.

[13] F.P. Kelly, *Reversibility and Stochastic Networks*. New York: Wiley, 1979.

[14] C. Kim and A.K. Agrawala. "Analysis of the Fork-Join Queue." *IEEE Trans. Computers*, vol. 38, no. 2, pp. 250-255, Feb. 1989.

[15] L. Kleinrock, *Queueing Systems Vol. I: Theory*. Wiley Int'l, 1975.

[16] P. Konstantopoulos and J. Walrand, "Stationary and Stability of Fork-Join Networks," *J. Applied Probability*, vol. 26, pp. 604-614, 1989.

[17] C.P. Kruskal and A. Weiss, "Allocating Independent Substasks on Parallel Processors," *IEEE Trans. Software Eng.*, vol. 11, pp. 1,001-1,016, Oct. 1985

[18] Y.C. Liu and H.G. Perros. "A Decomposition Procedure for the Analysis of a Closed Fork/Join Queueing System," *IEEE Trans. Computers*, vol. 40, no. 3, pp. 365-770, Mar. 1991.

[19] J.C.S. Lui, R.R. Muntz, and D. Towsley, "Bounding the Response Time of a Minimum Expected Delay Routing System: An Algorithmic Approach," *IEEE Trans. Computers*, vol. 44, no. 5, pp. 1,371-1,382, May 1995.

[20] J.C.S. Lui and R.R. Muntz, "Computing Bounds on Steady State Availability of Repairable Computer Systems," *J. ACM*, vol. 41, no. 4, pp. 676-707, July 1994.

[21] A.W. Marshall and I. Olkin, *Inequalities: Theory of Majorization and Applications*. New York: Academic Press, 1979.

[22] A. Makowski and S. Varma, "Interpolation Approximations for Symmetric Fork-Join Queues," *Proc. Performance '93*. Also in *Performance Evaluation 20*, pp. 145-165, 1994.
[23] R.R. Muntz, E. De Souza, E Silva, and A. Goyal. "Bounding Availability of Repairable Computer Systems," *Proc. 1989 ACM SIGMETRICS and PERFORMANCE '89*. Also in special issue of *IEEE Trans. Computers*, vol. 38, no. 12, pp. 19-30, Dec. 1989.
[24] R. Nelson and A.N. Tantawi, "Approximate Analysis of Fork/Join Synchronization in Parallel Queues," *IEEE Trans. Computers*, vol. 37, no. 6, June, 1988.
[25] R. Nelson, D. Towsley, and A.N. Tantawi, "Performance Analysis of Parallel Processing Systems," *IEEE Trans. Software Eng.*, vol. 14, no. 4, pp. 532-540, Apr. 1988.
[26] I.C. Pyle, *The Ada Programming Language*. London: Prentice-Hall Int'l, 1981.
[27] A. Giest, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam, "PVM 3.0 User's Guide and Reference Manual," Oak Ridge National Lab., 1993.
[28] S.M. Ross, *Introduction to Probability Models*. Academic Press, 1970.
[29] C. Schimmel, *Unix Systems for Modern Architectures: Symmetric Multiprocessing and Caching for Kernel Programmers*. Addison-Wesley, 1994.
[30] M.R. Stonebraker, "The Case for Shared-Nothing," *Proc. 1986 Data Engineering Conf.* IEEE, 1986.
[31] A. Thomasian and A.N. Tantawi, "Approximate Solutions for M/G/1 Fork/Join Synchronization," *Proc. 1994 Winter Simulation Conf.*, pp. 361-368, Orlando, Fla., Dec. 1994.
[32] D. Towsley, J.A. Rommel, and J.A. Stankovic, "The Performance of Processor Sharing Scheduling Fork-Join in Multiprocessors". *High-Performance Computer Systems*, E. Gelenbe, ed., pp. 146-156. Amsterdam, North-Holland: 1988.

John C.S. Lui received his PhD in computer science from the University of California, Los Angeles, in 1991. He then joined the IBM Almaden team in San Jose and participated in a research and development of a parallel I/O architecture project. In the summer of 1993, he joined the Department of Computer Science and Engineering at the Chinese University of Hong Kong. His research interests are in parallel and distributed system design, distributed multimedia systems, parallel I/O architectures, communication network, mobile computing, and performance evaluation theory. Dr. Lui was a guest coeditor for a special issue of the *International Journal of Intelligent Systems*, a program committee member of ACM SIGMETRICS '97, '98, the IEEE International Conference on Protocols for Multimedia Systems and Multimedia Networking 1997, and High Performance Computing Asia '98. He is a member of the IEEE and the ACM.

Richard R. Muntz received the BEE from Pratt Institute in 1963, the MEE from New York University in 1966, and the PhD. in electrical engineering from Princeton University in 1969. Currently, he is a professor in the Computer Science Department, School of Engineering and Applied Science, University of California, Los Angeles. His current research interests are multimedia storage servers, distributed and parallel database systems, scientific database systems, data mining, and computer performance evaluation.

Dr. Muntz is a member of the board of directors for SIGMETRICS. He was an associate editor for the *Journal of the ACM* from 1975 to 1980 and the editor-in-chief of *ACM Computing Surveys* from 1992 to 1995. He is a member of Sigma Xi, Tau Beta Pi, a fellow of the ACM, and a fellow of the IEEE.

Don Towsley received the BA degree in physics and the PhD degree in computer science from University of Texas 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. He is currently a professor of computer science at the University of Massachusetts. He has held visiting positions at the IBM T.J. Watson Research Center, Yorktown Heights, New York (1982-1983), Laboratoire MASI, Paris, France (1989-1990), and INRIA, Sophia-Antipolis, France (1996). His research interests include networks, multimedia systems, and performance evaluation.

Dr. Towsley currently serves on the editorial board of *Performance Evaluation* and has previously served on numerous editorial boards, including those of the *IEEE Transactions on Communications* and *IEEE/ACM Transactions on Networking*. He was a program cochair of the joint ACM SIGMETRICS and PERFORMANCE '92 conference. He is a member of the ACM, ORSA and the IFIP Working Groups 6.3 and 7.3. He has received two best paper awards from the ACM SIGMETRICS annual conferences and has been elected a fellow of both the ACM and IEEE.