Performance Modeling of Concurrent Systems under Resource Constraints

M. Ghodsi and K. Kant

Computer Science Department
The Pennsylvania State University, University Park, PA 16802, USA

Abstract

In this paper, we propose an approximate, iterative algorithm to compute the performance parameters of a resource constrainted computer system running jobs with internal concurrency. The jobs are represented by a task graph with arbitrary topology. The computer system is intended to be a multiprocessor or a parallel machine, and is represented by a closed queueing network. There are only limited number of passive resources in the system, called tokens, such as memory blocks, channels, or permission to use processors, which restricts the number of tasks that can be processed simultaneously. A task must acquire one token before being admitted for processing. While holding on to a token, a task can access active resources in the system, e.g., CPU, disks, busses, etc. The model, therefore, allows simultaneous possession of active and passive resources. Experimental results show that the algorithm converges very fast and is reasonably accurate.

1 Introduction

In many areas of concurrent system research, there is a need for tools to predict the performance of computations with internal concurrency when they are run on a multiprocessor system or a parallel machine. Unfortunately, such problems are not amenable to the classical product-form queueing network based modeling. In most cases, it is still possible to construct a Markovian model, but the direct solution to such a model quickly becomes intracable even for medium size problems. Therefore, there is real need for approximate solutions that are both efficient and accurate.

The problem that we have considered can be stated as follows: We have a closed queueing network model of a computer system and a parallel computation, henceforth called a *job*, represented by a task graph. The queueing network which is intended to represent a multiprocessor system, consists of a *central subsystem* (CS) which includes a number of service stations to perform active processing (e.g., computation, I/O, etc.) There are also a limited number of passive resources in the system, such as memory blocks, channels, permission to use processors, etc. that are needed by tasks in order to receive service. For simplicity, we shall assume that the passive resource are all of one type and call them *tokens*. Each task must obtain the required units of tokens (assumed one for simplicity) before entering the central subsystem, and release them upon exit. During the period that a task is in the CS, while holding on to a token, it can access other system resources represented by the service stations. The problem is to estimate the throughput and response time of jobs and their constituent tasks.

The main contribution of this paper is to develop an efficient algorithm for solving the problem in a general setting. There have been several works on similar problems, but to the best of our knowledge, this paper is the first attempt in providing performance prediction for concurrent systems with simultaneous possession of resources. In addition, our algorithm allows the following properties for the model which have not been considered simultaneously before: (a) the task graph representation of the job has a general structure, (b) the number of available passive resources is arbitrary, (c) the central subsystem is an arbitrary product form network, and (d) the job population is arbitrary (i.e., not necessarily one or infinite). Moreover, because the solution technique is based on product form queueing networks and well known techniques for modeling resource constraints, several extensions

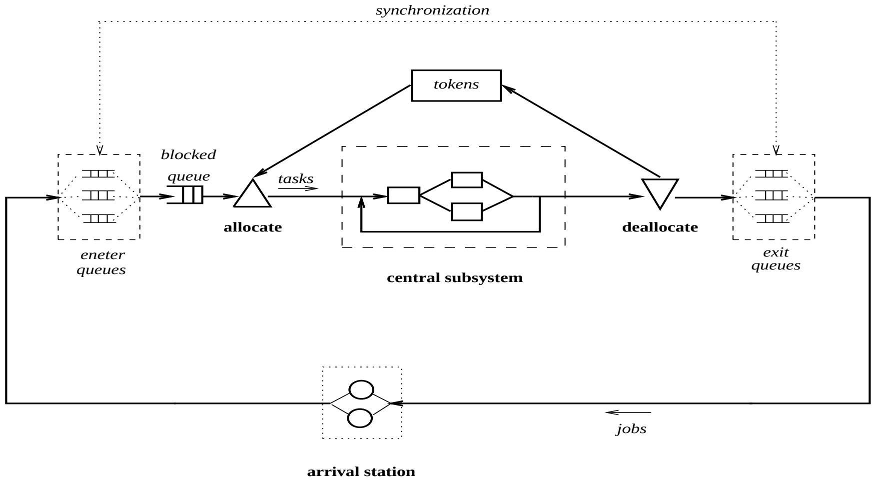

Figure 1: The queueing system.

(e.g., contention for multiple resources, sharing of processors, etc.) can be made. Finally, the techniques appears to converge fast and give good results.

Several researchers have considered the modeling of parallel systems without considering simultaneous possession of resources. Some of the more theoretical results can be found in [2, 6, 7], where very simple task graphs and queueing models are considered. Other approaches admit more general task graphs and more complicated closed queueing models, but are heuristic in nature [3, 5, 9]. For the systems with simultaneous access to resources, but without internal concurrency in jobs, several approximate algorithm have been proposed [4, 8].

In the next sections, we describe the model, solution technique, and the experimental results. The details are omitted and may be found in [1].

2 Model Description

It is assumed that there are N jobs in the system which are all instances of a single fork-join task graph with K tasks $T_1, \ldots, T_K$. A portion of the queueing network is designated as the central subsystem (CS) and includes all active service stations. There is a delay station outside the CS with the service time of E, called the *external arrival station*, that controls the interarrival time of the jobs. The model also includes an allocate and a de-allocate station and M tokens that represent the passive resources in the system.

Figure 1 shows the overall structure of the queueing system. When job j leaves the arrival station, all its tasks are created. Of these, the first task is put at the end of the *blocked queue*, and all others into the jth *enter queue*. Tasks are loaded into the CS from the blocked queue according to a FCFS policy and executed, as already stated. When a running task belonging to the jth job leaves the CS, it is put into the jth *exit queue*. A task from an enter queue is removed and put into the blocked queue immediately after its predecessor tasks, satisfying its precedence constraints, are present at the jth exit queue. After the terminal task belonging to the jth job arrives at its exit queue, all the tasks in the jth exit queue are removed from the system, and a new job is created which goes to the arrival station.

Based on its location in the system, we define the status of a task as following. A task is *blocked* if it is in the blocked queue waiting to be loaded. It is *loaded* if it is inside the CS. It is *active* if it is either blocked or loaded. Otherwise, the task is called *inactive*. The delays that a task T_t encounters are: B_t, its blocking delay which is the average period that T_t spends in the blocked queue; R_t^{cs}, the CS response time of T_t; and ID_t its average inactive period. Then the system throughput is $\lambda = 1/(R_t + ID_t)$ where $R_t = B_t + R_t^{cs}$ is the response time of T_t. The *loading probability* of task T_t, ρ_t, is defined as the probability that T_t is loaded given that it is active.

The input parameters of the system are the TG topology, delay at the external arrival station (E), job population (N), the number of available tokens (M), central subsystem structure, and the service demand of each task at each CS station (D_{tk}). The performance parameters of interest are throughput, response time of each task and that of the complete job, and *token utilization*, defined as the average fraction of tokens being used.

3 Solution Technique

The algorithm is iterative in nature and is based on decomposition and flow-equivalent-aggregation. We identify two different high-level models in the system each of which takes into account different interdependencies among the tasks. First is a high level task graph model (HLTG) which has the same topology as the TG, but the processing time of a task in the HLTG model is its response time which includes its blocking delay. These processing times are computed and updated in each iteration of the algorithm. There are N HLTG models, one for each job, that are repeatedly initiated after E units of time. The goal of defining the HLTG's is mainly to capture the effects of TG topology on system performance. We also consider K high level queueing network models (HLQN), one for each task in the TG. The HLQN model for task T_t is a closed network with population of N which consists of two stations: a delay station representing the total inactive period of T_t (ID_t), and a flow equivalent aggregate station replacing the entire CS and the allocate/deallocate stations. The service rates for this load-dependent station at different queue lengths are computed by solving the HLTG models. Because of the precedence constraints between the tasks, different HLQN models are not independent. However, in our approximation, the main purpose of the HLQN models is to estimate the blocking delays caused by resource contention; the effect of the precedence constraints is taken care of in the solution of the HLTG models. Therefore, we ignore the interdependencies in the HLQN models, thereby enabling us to solve them using the standard techniques for solving closed single chain product form networks.

Without going into details, the basic formulae for computing the approximated service rates of the load-dependent station in the HLQN model for task T_t are given below. In the following, all N tasks T_t ($t = 1..K$) that are present in the system are said to be of *type t*. First some notations.

$\mu_t(n)$	Mean departure rate of type t tasks from the CS when a total of n tasks from this type are active.
$\alpha_t(j\|n)$	Probability that a total of j tasks of type t are loaded when n tasks of the same type are active.
$\beta_t(j\|n)$	Probability that total of j tasks of types other than t are loaded into the CS given that total of n tasks of type t are active.
$\vec{L}_t(j,n)$	The average number of each task that are loaded in the CS when n tasks of type t are active, j of which are loaded.
$\phi_t(\vec{L}_t(j,n))$	Mean departure rate of type t tasks from the CS when the population vector of the tasks inside CS is $\vec{L}_t(j,n)$.

Then,

$$\mu_t(n) \approx \sum_{j=1}^{n} \alpha_t(j|n)\, \phi_t(\vec{L}_t(j,n)) \tag{1}$$

where, by definition, $\alpha_t(j|n)$ is related to $\beta_t(.|.)$ as

$$\alpha_t(j|n) = \begin{cases} \beta_t(M-j|n) & \text{if } j < n \text{ and } j \le M \\ \sum_{k=0}^{M-j} \beta_t(k|n) & \text{if } j = n \le M \\ 0. & \text{if } j > M \end{cases} \tag{2}$$

This approximation is based on the ideas presented in [4]. The main difficulty regarding our algorithm is the computation of above parameters considering the effect of the interdependencies among the tasks due to the precedence constraints. This is done with the help of following simplifying assumptions: (1) task response times are assumed exponentially distributed, and (2) the loading behavior of two tasks is assumed independent. i.e., the probability that two tasks are loaded given that they are both active is assumed to be the product of the individual loading probabilities. In reality, none of these assumptions hold. Without them, however, the solution becomes intractable. The reasonableness of these assumptions is borne out by the good accuracy of the final results.

Informally, the algorithm starts with an estimate of the blocking delay and the CS response time for every task. Using these quantities, we solve the HLTG models to compute $\beta_t(j|n)$ and $\vec{L}_t(j,n)$ for all tasks and for all possible values of j and n. Equations 2 and 1 are then used to compute $\mu_t(.)$. In equation 1, $\phi_t(\vec{L}_t(j,n))$ is computed by solving the CS in isolation with the population vector of $\vec{L}_t(j,n)$. This gives the service rates of the aggregateed stations in the HLQN models. We then solve the HLQN models using the independence assumptions and obtain new values for the task blocking delays and response times. The iterations are continued until convergence is achieved.

For solving the HLTG models, we first construct a Markov chain (MC) for a single HLTG with states representing the tasks that are active and those that are not. This MC is correct with the exponential assumption we made. The solution to this MC provides values for parameters needed when the number of jobs is one. Then, with the help of an efficient approximation technique, we compute the values of the parameters to be used in the algorithm. The details of theses computations are not included in this paper.

4 Experimental Results

The algorithm has been tested and compared with simulation using seven basic medium size task graphs with different structures and degrees of parallelism. This includes completely serial, completely parallel, and general task graphs. Four queueing network structures were chosen as the central subsystem, one of which is just a delay station. The task demands were chosen for each CS and assigned randomly to the tasks so that the number of chains inside the CS remains less than five. This was done solely to reduce the complexity of solving the CS in isolation. To test the algorithm even for extreme cases, the number of tokens for each TG was chosen from as low as 2 up to a number which makes the blocking delays negligible. In all these cases, the delay at the external arrival station was set to 1.0 second. The job population was chosen from 1 to 8. The combination of all these, created a total of 244 different cases.

The convergence criterion was chosen as the relative error of 0.001 for the task throughputs. The average and the maximum number of iterations over all 244 cases were 6.4 and 18 respectively, while 82% of these cases needed less than 10 iterations to converge.

There were some extreme cases in the tests in which a large number of tasks compete for a small number of tokens. These cases produced the maximum errors. Yet, the algorithm produced reasonably accurate results, especially for the system throughput and token utilization. In all these cases, the average absolute relative errors for the system throughput and token utilization were 5.9% and 7.9% respectively. The maximum of these errors were 26.1% and 20.6%. The results for the task response times were not as good. The average relative error for the task response time was 9.5%, with the maximum of 67.0% in one case. The average tolerance error, defined as the absolute error divided by the cycle time, is a better measure of error for the response times. The average and maximum relative tolerance error of the task response times were 1.5% and 11.1% respectively.

In general, the algorithm performs well specially when the token utilization is less then 90%. This happens at medium job population and when the task graph does not have a large degree of parallelism. In the cases where the CS is just a delay station, the algorithm is very accurate. For over 50 cases that we experimented, the maximum and average relative errors for the system throughput were 15.0% and 4.3% respectively. These values for token utilization were 14.5% and 4.3% and for task response times were 45.7% and 4.6% respectively.

5 Conclusions

In this paper, we have proposed a new technique based on an approximate, iterative algorithm for computing the performance parameters of a resource constrained queueing network model, when it is driven by a number of statistically identical jobs with internal concurrency represented by an arbitrary task graph. The queueing network model is intended to represent a parallel computer with a limited number of passive resources, e.g., memory blocks or permission to use processors, etc. The model allows simultaneous possession of resources. The algorithm is based on the decomposition and flow-equivalent-aggregation techniques and takes into account the precedence relations of the tasks as well as the resource constraints. The experimental results show that the algorithm converges quite fast and is reasonably accurate. The algorithm can be extended to accept multiple tokens requests for tasks, sharing of or contention for multiple passive resources, etc. Further work on these topics is underway.

References

[1] M. Ghodsi and K. Kant, *Simultaneous modeling of resource and precedence constraints*, Tech. Report CS-88-36, Computer Science Dept., Penn State University, Univ Park, PA 16801, Oct. 1988.

[2] P. Heidelberger and K. Trivedi, *Queueing network models for programs with internal concurrency*, IEEE Transactions on Computer, (Jan. 1983), pp. 73–82.

[3] A. Kapelnikov, R. Muntz, and M. Ercegovac, *A modeling methodology for the analysis of concurrent systems and computations*, PhD thesis, Computer Science Dept., UCLA, Los Angeles, CA 90024-1600, July 1987. (tech report# CSD-870038).

[4] E. D. Lazawoska and J. Zoharjan, *Multiple class memory constrained queueing network*, in ACM SIGMETRICS conference, 1982, pp. 130–140.

[5] V. W. K. Mak, *Performance prediction of concurrent systems*, PhD thesis, Computer System Lab., Stanford University, Stanford, CA 94305-4055, Dec. 1987. (tech report# CSL-TR-87-344).

[6] R. Nelson and A. N. Tantawi, *Approximate analysis of fork/join synchronization in parallel queues*, IEEE Transactions on Computer, (June 1988), pp. 739–743.

[7] R. Nelson, D. Towsley, and A. N. Tantawi, *Performance analysis of parallel processing systems*, IEEE Transactions on Software Engineering, (April 1988), pp. 532–540.

[8] E. S. Silva and R. R. Muntz, *Approximate solutions for a class of non-product form queueing network models*, Performance Evaluation 7, (1987), pp. 221–242.

[9] A. Thomasian and P. F. Bay, *Analytical queueing network models for parallel processing of task systems*, IEEE Transactions on Computer, (Dec. 1986), pp. 1045–1054.