Approximate Analysis of Priority Scheduling Systems Using Stochastic Reward Nets

Varsha Mainkar
Department of Computer Science
Duke University
Durham, NC 27708

Kishor S. Trivedi*
Department of Electrical Engineering
Duke University
Durham, NC 27708-0291

Abstract

We present a performance analysis of a heterogeneous multiprocessor system where tasks may arrive from Poisson sources as well as by spawning and probabilistic branching of other tasks. Non-preemptive priority scheduling is used between different tasks. We use Stochastic Reward Nets as our system model, and solve it analytically by generating the underlying continuous-time Markov chain. We use an approximation technique based on fixed-point iteration to avoid the problem of a large underlying Markov chain. The iteration scheme works reasonably well, and the existence of a fixed point for our iterative scheme is guaranteed under certain conditions.

1 Introduction

Resource sharing systems require scheduling disciplines to share the resources in a fair manner. When the processor is shared, disciplines such as FIFO, round-robin or priority may be employed to schedule the tasks, depending on the particular performance requirements of that system. In cases where it is important that the response time for certain jobs should be predictably short, priority scheduling of tasks is used [5, 6, 7]. In this paper we present a performance analysis of a heterogeneous multiprocessor system which uses priority discipline to schedule its tasks. The tasks may arrive to ths system from a Poisson source or due to spawning or conditional branching of other tasks in the system.

Priority queues with tasks that could feedback into the system through spawning and conditional branching were analyzed in [4, 12]. Similar multi-tasking systems have also been modeled in [11, 13] though not in the domain of priority scheduled systems. Recently, Nishida [8] analyzed a heterogeneous multiprocessor system with priority scheduled jobs which arrive from a Poisson source. A continous-time Markov chain (CTMC) is employed for the analysis, and a lumping scheme is used to tackle the problem of large state space of the Markov chain.

Our focus in this paper is to consider a more general task arrival behavior. Specifically, we allow task arrivals to the system to depend on the number of executions of another task; we also allow tasks that can arrive to the system externally, as well as through feedback; furthermore, tasks can be brought to the system by spawning/probabilistic branching of a number of other tasks.

This task structure models many computing systems of today. In fault-tolerant computer systems, diagnostic tasks are often spawned by other tasks on occurrence of certain events or errors. On the other hand, a routine which uses a hardware component may initiate diagnostic routines on that component after every n uses of the component. Our focus in this paper is to model such systems.

We choose Stochastic Reward Nets (SRN) as our analytical model for this system. SRNs are essentially a specification technique for automatic generation of Markov reward models [1]. SRNs have built-in constructs, such as transition priorities, which render them very useful in modeling priority scheduling systems. Rewards, which can represent performance indices, can be specified at the net level.

Since SRN models are solved by generation of the underlying continuous-time Markov chain, we eventually confront problems of a large state space as the model size increases. We resolve this problem by using state lumping along with a fixed-point iteration technique [3] to avoid generation and solution of a large state space and balance equations. The existence of a solution for our fixed-point iterative scheme is guaranteed under certain conditions.

The rest of the paper is organized as follows: in Section 2 we provide an overview of SRNs, in Section 3 we describe our system specification. Section 4 describes the SRN model corresponding to such a system. In Section 5 we present a performance evaluation of an example system using SRNs. Section 6 introduces our iterative scheme. In Section 7 we illustrate the scheme with our example, compare the approximation results with the exact results, and prove the existence of a fixed point for the example model. We conclude with some remarks on future work in Section 8.

*This research was supported in part by the National Science Foundation under Grant CCR-9108114.

2 Overview of SRNs

A Petri net [10] is a directed bipartite graph with two types of nodes called **places** (represented by circles) and **transitions** (represented by rectangles or bars). Directed arcs (represented by arrows) connect places to transitions, and vice versa. If an arc exists from a place (transition) to a transition (place), then the place is called an input (output) place of that transition. Places may contain **tokens** (represented by dots). The state of a Petri net is defined by a vector of the number of tokens in each place, called a **marking** of the Petri net. Typically, places represent conditions or resources, and transitions represent events or choices. The token flow along the directed arcs reflect the changes in the system due to various events.

A **multiplicity** is a non-negative integer that may be associated with an input or output arc. A transition is said to be **enabled** if each of its input places contains at least as many tokens as that input arc's multiplicity. An enabled transition can **fire**. When it fires, as many tokens as the corresponding input arc's multiplicity are removed from each input place, and as many tokens as the corresponding output arc's multiplicity are deposited in each output place.

Structural extensions to Petri nets include **inhibitor** arcs (denoted by an arc with a circle instead of an arrow head), which connect places to transitions. A transition can be enabled only if the number of tokens in its inhibitor place is less than the multiplicity of the inhibitor arc. A set of transitions is said to be **conflicting** when the firing of one disables the rest. Transitions may be assigned priorities that can be used to resolve conflicts between transitions.

Timed Petri nets associate a time delay with transitions. The **Stochastic Reward Net** [1] is a type of Petri net that allows transitions to have an exponentially distributed time delay (**timed** transitions, represented by rectangles) or a zero time delay (**immediate** transitions, represented by bars). The **firing rate** of the timed transitions may be marking-dependent. A marking of a SRN is said to be **vanishing** if at least one immediate transition is enabled in it and is said to be **tangible** otherwise. Conflicts among immediate transitions in a vanishing marking may be resolved by assigning probabilities to conflicting sets of immediate transitions. This probability may also be marking-dependent. Multiplicities of arcs may be marking-dependent. Such arcs are termed **variable cardinality** arcs. Further, enabling functions or **guards** may be associated with transitions. Guards are marking-dependent predicates which must be satisfied for transitions to be considered enabled.

The reward structure is obtained by associating **reward rates** with markings of the SRN. We associate a reward rate r_i with every tangible marking of the SRN. This SRN can be mapped to a CTMC. If this CTMC is irreducible, we can solve it for its steady state probability vector π, then the expected reward rate at steady-state can be computed as $\sum_i r_i \pi_i$.

With the use of appropriate reward rates, the expected reward at steady state can yield various useful measures of a model. In our SRN models, reward rates will be various performance indices; thus expected reward rate at steady-state will give us the average values of the performance measures of the system.

3 System description

We study a heterogeneous multiprocessor system with non-preemptive priority scheduling of tasks. We classify the tasks by the way they arrive to the system.

- **Poisson Tasks**: these arrive according to a Poisson process, so that the interarrival time of these tasks is exponentially distributed.
- **Sporadic Tasks**: these arrive to the system by spawning or conditional branching of other Poisson and/or sporadic tasks.
- **Poisson and Sporadic**: These arrive from a Poisson source or are created by other tasks.

Each task has a buffer where its instances wait to execute on a processor. For instances of the same task, the service discipline is FIFO. Different tasks are served according to different priorities. After a task acquires a processor, it executes on the processor for an exponentially distributed amount of time. A task may create some instances of other sporadic tasks. Each wait buffer has a limit on how many tasks can be waiting in it to be executed. When this limit is reached, any new arriving task of the corresponding type is lost.

The task system consists of a set of tasks, $\mathcal{T} = \{T_1, T_2, \ldots, T_N\}$. Each task T_i is characterized by:

- λ_i : the parameter of the Poisson process according to which the tasks arrive to the computing system. If the task is strictly sporadic, $\lambda_i = 0$.
- A parent set : $\mathcal{P}_i = \{(T_j, m_{j,i}, n_{j,i}, q_{j,i}) \mid T_j \in \mathcal{T}, T_j \text{ spawns task } T_i\}$. In the 4-tuple $(T_j, m_{j,i}, n_{j,i}, q_{j,i})$, $q_{j,i}$ is the probability that $n_{j,i}$ instances of task T_i are spawned by task T_j after the $m_{j,i}$th execution of T_j. This set is empty for strictly Poisson tasks.
- μ_i : the parameter of the exponentially distributed service demand. (Service demand could be number of instructions to be processed, or some other measure of the amount of service required by a task.)
- M_i : the maximum number of instances of task T_i that can be waiting to be executed.
- p_i, $1 \leq p_i \leq N$: task priority; we use the convention that $p_i > p_j$ implies that T_i has higher priority than T_j.

A task system in which each task is a strictly sporadic task, is a degenerate system. Hence we require that at least one of the tasks in the task system be Poisson.

The processing system is specified by the following:

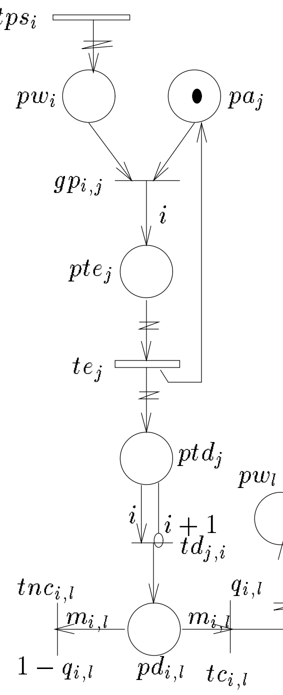


Figure 1: The Stochastic Reward Net Model

- Number of processors: P, the jth processor is denoted by P_j.
- The **capacity** of each processor: $C_i, i = 1, \ldots, P$. The capacity of a processor is its rate of providing service, for example it could be number of instructions that it can process in one second.
- Discipline for allocation of idle processors to tasks: we assume this also to be by order of preassigned priority. Let pp_i denote the priority of the processor P_i, where $1 \leq pp_i \leq P$.

We would like to compute performance measures such as the throughputs, utilizations, mean queue lengths and mean response times of this system.

4 The Stochastic reward net model

The system described in the above section can be translated into a stochastic reward net as shown in Figure 1. In the net a subscript i corresponds to a task i and subscript j corresponds to a processor P_j. Corresponding to each task T_i there is a place pw_i which represents the wait buffer. Suppose that T_i is a Poisson task. A timed transition tps_i, with firing rate λ_i, deposits tokens in place pw_i, representing the arrivals of the task according to the Poisson process. The arc from tps_i to pw_i has the marking dependent multiplicity of $\min\{1, M_i - \#(pw_i)\}$. This represents the fact that tasks arriving to a full wait buffer are lost. (In the figure, variable cardinality arcs are denoted by a "Z" across the arcs.) A token in place pa_j says that processor P_j is available. Transition $gp_{i,j}$ represents the acquisition of processor P_j by task T_i. The priority scheduling discipline is reflected in the net by assigning the following priority to transition $gp_{i,j}$: $\max\{P, N\} \times p_i + pp_j$. On firing, transition $gp_{i,j}$ puts i tokens into place pte_j, indicating that task T_i is executing on processor P_j. The firing rate of te_j must be the execution rate of task T_i on processor P_j and is given by $\mu_i C_j$, when $\#(pte_j) = i$. On firing, transition te_j removes all tokens from place pte_j and deposits a token back in pa_j. It may also deposit the same number of tokens that it removes from pte_j, into ptd_j, if the task T_i that just finished is a parent of any other tasks. If it is not a parent of any other task, te_j puts zero tokens in place ptd_j. Note that the number of tokens in place pte_j is used to identify the task that is executing on processor P_j.

If T_i does not spawn any task, the description above is sufficient to model the execution of task T_i. Suppose T_i spawns a task T_l, i.e., $(T_i, m_{i,l}, n_{i,l}, q_{i,l}) \in \mathcal{P}_l$. Then transition $td_{j,i}$ recognizes the completion of task T_i by the arrival of i tokens in place ptd_j. This is accomplished by having an input arc from ptd_j with multiplicity i, and an inhibitor arc from ptd_j with multiplicity $i+1$. Thus $td_{j,i}$ fires when there are exactly i tokens in place ptd_j. For each child task T_l there is a place $pd_{i,l}$ with an output arc from $td_{j,i}$ to $pd_{i,l}$. This place counts the number of completions of task T_i required to spawn an instance of task T_l. Place $pd_{i,l}$ together with transitions $tc_{i,l}$ and $tnc_{i,l}$ counts the number of completions of task T_i. This is done by setting the multiplicity of the arc from $pd_{i,l}$ to $tc_{i,l}$ (and $tnc_{i,l}$) equal to $m_{i,l}$. The firing probability of transition $tc_{i,l}$ $\{tnc_{i,l}\}$ is $q_{i,l}$ $\{1 - q_{i,l}\}$, which represents the probabilistic spawning of T_l. Transition $tc_{i,l}$ deposits tokens in place pw_l; the marking-dependent multiplicity of the arc from $tc_{i,l}$ to pw_l is defined as $\min\{n_{i,l}, M_l - \#(pw_l)\}$. Transition $tnc_{i,l}$ sinks the tokens out of place $pd_{i,l}$ signifying the event that task T_l was not scheduled. The priorities of immediate transitions $td_{j,i}, tc_{i,l}$ and $tnc_{i,l}$ are set to be higher than the priorities of all the transitions $gp_{i,j}$.

We illustrate the above model with an example, in the next section.

5 An example

Consider a two processor heterogeneous system, that maintains information which is regularly read and updated. In this system, we would like to provide the `read` tasks with as up-to-date information as possible. One way to achieve this effect is to give preference to the `update` tasks, so that the `read` tasks are executed *after* the latest update has been performed. Thus `update` tasks are assigned higher priority than the `read` tasks. To avoid excessive scheduling overhead, the system adopts non-preemptive priority.

Let T_1 denote the `update` task, and T_4 denote the `read` task. During its execution, the update task may come across an erroneous condition in the system. If this error is not critical, the update task spawns an

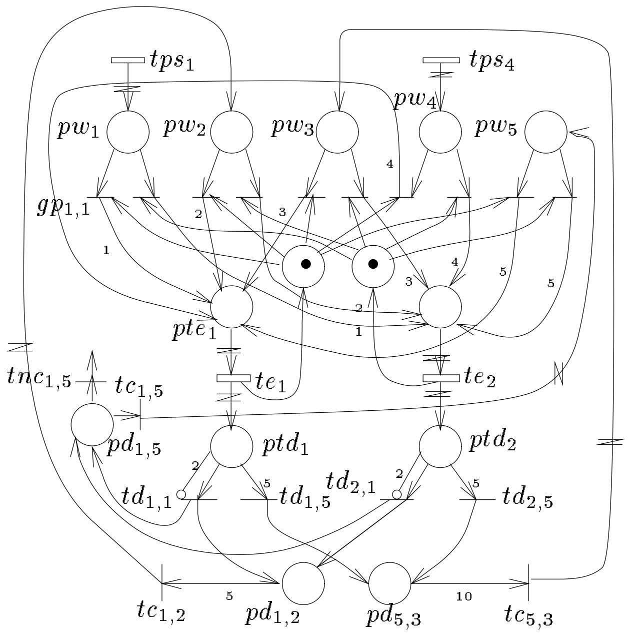
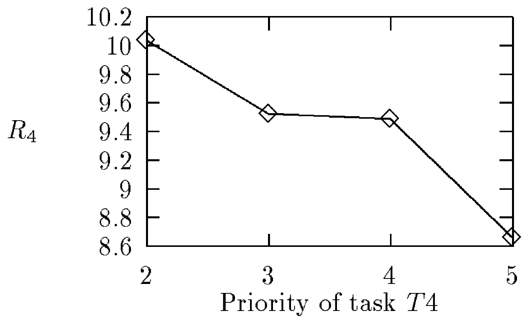


Figure 2: SRN corresponding to the example



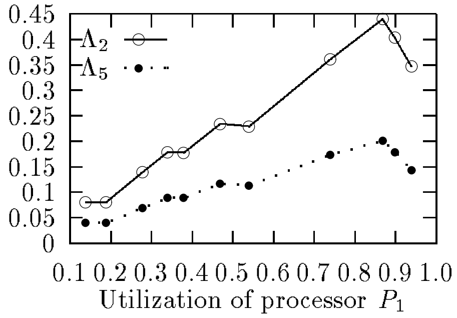


Figure 3: (a)Response Time of T_4 *vs* Priority if T_4 (b) Throughput of tasks *vs* processor utilization

error-handler task[1] denoted by T_5. As a measure of preventive diagnostics, the update task also schedules a diagnostic task (denoted by T_2) after every five executions. Task T_2 runs diagnostic checks on the secondary storage on which the updates are performed. After every ten executions of the error-handler task (T_5), it schedules another diagnostic task, T_3. This task runs further in-depth diagnostics on the entire system. We assume that the priority of these tasks is in the order $p_1 > \ldots > p_5$. We also assume that priority of processor P_1 is greater than that of P_2. We further assume that tasks T_1 and T_4 arrive to the system according to independent Poisson processes with parameters λ_1 and λ_4 respectively. Finally, let q denote the conditional probability that a non-critical error occurs in the system given that T_1 is running.

As described in Section 4, this system can be represented by an SRN (Figure 2). This SRN can be mapped to an irreducible continuous time Markov chain (CTMC) [1]. We solve this CTMC for its steady-state behavior, and compute the required performance measures, using the SRN specification and solution tool SPNP [2].

5.1 Performance measures

Performance measures of the system are derived using appropriate rewards. Define $I_{i,j}(m)$, a function of marking m, as 1 if $\#(pte_j) = i$ and 0 if $\#(pte_j) \neq i$ in marking m. If we assign a reward rate $I_{i,j}(m)$ to marking m, the expected reward rate at steady state is the utilization of P_j by T_i. Average throughput of task T_i is calculated by assigning a reward $\mu_i C_1 I_{i,1}(m) + \mu_i C_2 I_{i,2}(m)$ to a marking m of the SRN. The mean queue length, of task T_i is calculated by assigning a reward of $\#(pw_i) + I_{i,1} + I_{i,2}$ to each marking.

Let Λ_i be the throughput of task T_i, and let L_i be its average queue length. Then by Little's law [14], mean response time is given by: $R_i = L_i/\Lambda_i$.

Figure 3(a) shows how average response time of task T_4 varies as its priority varies from 2 to 5 (low to high). Figure 3(b) shows the throughputs of tasks T_2 and T_5 as processor utilization varies.

6 The iterative decomposition scheme

The small example model described in Section 5 gave rise to 17574 states in the underlying CTMC, when $M_i = 1, \forall i = 1, \ldots, 5$. The state space increases exponentially in terms of the model parameters such as the number of task types.

[1]If the error is critical, the processor may take drastic actions such as shutting the system down. Since in this paper we are concerned only with performance of a system while it is operational, we do not consider this possibility.

We use a well-known lumping technique [8] to reduce the size of the state space generated by the SRN model. This lumping technique is based on the following observation: from the viewpoint of task T_i, there are only two sets of tasks: tasks that have a priority higher than itself, and tasks that have a priority lower than itself. In other words, it does not matter *which* of the higher priority tasks are waiting to be executed on the processor; if *any* of them is, task T_i will not acquire the processor. Similarly, all lower priority tasks matter only when they are already executing on the processor(s), and task T_i has to wait for them to finish. Thus in our lumping technique, we create a series of sub-models, each corresponding to one task in the system, where we have three tasks: the original task T_i, an "aggregate" task T'_{i-1} which represents all the tasks that have a priority higher than T_i and a task T'_{k+1}, which represents all the tasks with priority lower than task T_i.

This lumping can be further generalized, so that we do not necessarily create one sub-model for each task, but, rather, for a group of tasks. Suppose the tasks in a system are $T_1, \ldots, T_N$, where $p_1 > p_2 > \ldots > p_N$. (Task T_1 has highest priority and T_N has lowest priority.) Thus we create a series of sub-models in which we represent tasks $T_i, T_{i+1}, \ldots, T_k$ individually and lump all the tasks $T_1 \ldots, T_{i-1}$ into one higher priority task, and all the tasks $T_{k+1}, \ldots, T_N$ into one lower priority task. This generalization is very beneficial, because for some models the acute lumping (one sub-model per task) is not necessary, and the generalized lumping delivers more accurate results.

In the kind of systems considered in this paper, lumping affects the interdependence between task arrivals. We have to especially account for tasks that are created by parent tasks. These parent tasks may no longer be represented individually in the sub-model, hence we need to approximate the creation of sporadic tasks.

In particular, Suppose that a task system $\mathcal{T} = \{T_1, \ldots, T_N\}$ is ordered so that $p_1 > p_2 > \ldots > p_N$. Further, suppose one of the lumped submodels represents tasks $T_i, T_{i+1}, \ldots, T_k$ individually. Let T'_{i-1} denote the aggregate task which represents tasks $T_1, \ldots, T_{i-1}$ and T'_{k+1} represent the aggregate task which represents tasks $T_{k+1}, \ldots, T_N$. Then we define a modified task system for the submodel as follows:

1. The set of tasks is $\{T'_{i-1}, T_i, T_{i+1}, \ldots, T_k, T'_{k+1}\}$.

2. For every sporadic $T_l \in \{T_1, \ldots, T_N\}$ and every parent task T_p, of T_l, where $T_p \notin \{T_i, \ldots, T_k\}$,

   (a) If $(T_p, m, n, q) \in \mathcal{P}_l$, approximate the sporadic creation of T_l by a batch Poisson process with parameter $\lambda_{p,l} = \Lambda_p \times q/m$.

   (b) Remove the tuple corresponding to T_p from the parent set $\mathcal{P}_l$ of T_l.

3. Define the service demand of the aggregate tasks T'_{i-1} and T'_{k+1} to be exponentially distributed with rate μ'_{i-1} amd μ'_{k+1} respectively where,

$$\mu'_{i-1} = \frac{\sum_{j=1}^{i-1} s_j \mu_j}{\sum_{j=1}^{i-1} s_j}, \quad \mu'_{k+1} = \frac{\sum_{j=k+1}^{N} s_j \mu_j}{\sum_{j=k+1}^{N} s_j}$$

   and s_j is the total arrival rate of task T_j defined as

$$s_j = \lambda_j + \sum_{(T_p,m,n,q) \in \mathcal{P}_j} \frac{\Lambda_p \ q \ n}{m}$$

4. Define the priority of task T'_{i-1} equal to p_{i-1} and priority of task T'_{k+1} equal to p_{k+1}.

5. Define the maximum size of the wait buffer for task T'_{i-1} to be $M'_{i-1} = M_1 + M_2 + \ldots + M_{i-1}$ and that of task T'_{k+1} to be $M'_{k+1} = M_{k+1} + \ldots + M_N$.

6. Set the parent set of the task T'_{i-1} equal to $\bigcup_{j=1}^{i-1} \mathcal{P}_j$, and that of T'_{k+1} equal to $\bigcup_{j=k+1}^{N} \mathcal{P}_j$.

We use the description for the general case in Section 4 to construct an SRN for this task system, with the following modifications:

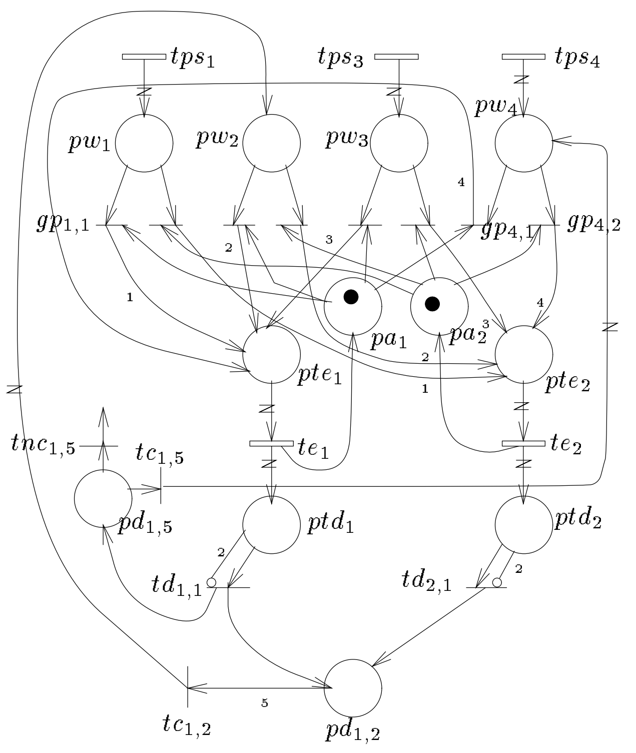
1. Transitions $tps_j, 1 \leq j \leq i-1$, if any, are retained, with the change that output place is pw_{i-1} and multiplicity is $\min\{1, M'_{i-1} - \#(pw_{i-1})\}$. Similarly, transitions $tps_j, k+1 \leq j \leq N$, if any, are retained, with the change that output place is pw_{k+1} and multiplicity is $\min\{1, M'_{k+1} - \#(pw_{k+1})\}$.

2. For each task $T_l, i \leq l \leq k$, whose creation by a parent T_p was approximated by a batch Poisson process with parameter $\lambda_{p,l}$, create a transition $tps_{p,l}$ with firing rate $\lambda_{p,l}$, output place pw_l, and multiplicity of output arc equal to $\min\{n, M_l - \#(pw_l)\}$. If $l \leq i-1$ ($l \geq k+1$), transition $tps_{p,l}$ has output place pw_{i-1} (pw_{k+1}), and multiplicity of output arc equal to $\min\{n, M'_{i-1} - \#(pw_{i-1})\}$ ($\min\{n, M'_{k+1} - \#(pw_{k+1})\}$).

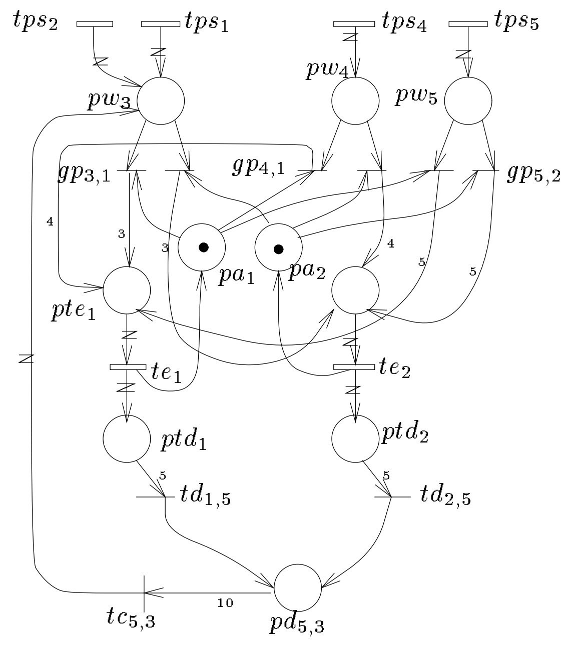
The above steps can be used to construct the series of sub-models corresponding to the complete SRN model. If these sub-models are interdependent, they should be solved iteratively.

7 Example of approximation method

We apply the task lumping technique to the example described in Section 5. We lumped the five tasks in the following way: In submodel 1 we lump tasks T_4 and T_5 into one task of priority 2 (least), while representing tasks T_1, T_2 and T_3 separately as individual tasks. In submodel 2, we lump tasks T_1, T_2 and T_3 into one task of priority 3 (highest) and represent tasks T_4 and T_5 separately as individual tasks. Note that this lumping is largely ad-hoc, and for expository purposes only. The tasks could have been grouped



(a)



(b)

Figure 4: (a) Submodel 1 (b) Submodel 2

in many other ways. Figures 4(a) and (b) depict the SRNs for the two resulting submodels. Note that in submodel 1, arrival of task T_3 is now approximated by a Poisson process, whose rate is $\Lambda_5/10$. Similarly in submodel 5, arrivals of tasks T_2 and T_5 are approximated by Poisson processes with rates $\Lambda_1/5$ and $q\Lambda_1$ respectively. The two models, are therefore interdependent, and are solved iteratively, until successive values are below a certain error tolerance level.

In all our experiments, the iteration process converged fairly rapidly, in about 5-6 iterations. The savings gained in space were enormous. Table 5 shows the number of states of the underlying CTMC generated by the same system in the complete model, and in the two sub-models.

Exact	Submodel 1	Submodel 2
6788	1085	1630
9134	1301	2170
10157	1625	1900
13676	1949	2530
17574	1533	1510
26182	2029	2230
101398	6689	4690

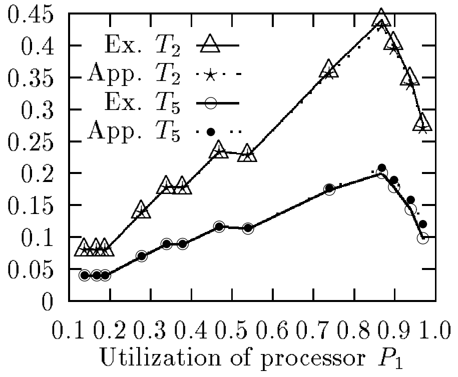
Figure 5: Savings in state space size

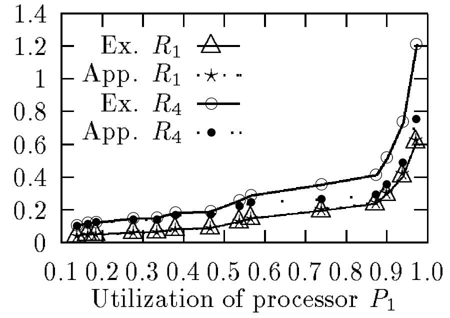
7.1 Accuracy of approximation

For system sizes where we could solve both the approximate and the complete model, we compared the approximation results with the exact results.

Figures 6(a) and 6(b) show the approximation errors for different utilization levels of processor P_1. The approximation error of high priority tasks at very high utilization levels stays quite low ($\leq 5\%$). However approximation of response times of lower priority jobs deteriorates as the utilization increases. This is because the performance of a lower priority task is more sensitive to service demands and arrival rates of tasks whose priority is higher than itself. In the aggregation, the service demand of the aggregate task is approximated by an exponential distribution, which is one source of error. On the other hand, the approximation of sporadic tasks by Poisson arrivals produces an error in the arrival rates of these tasks. The net effect is that the performance measures of a lower priority task at high utilization values are poorly approximated.

Figure 6(c) shows the exact and approximate response times of tasks T_1 and T_4 for various utilization levels, when the service demand distributions of the jobs are the same. Clearly, the approximation is much better in this case even for higher utilizations. This shows that the underestimation of variance that is a result of aggregation of tasks has a great effect on the accuracy of the approximation.





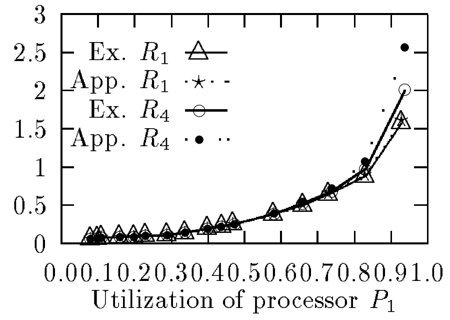


Figure 6: (a) Exact *vs* approximate throughputs (b) Exact *vs* approximate response times (c) Approximation with identical service demands

7.2 Existence of a fixed point

In our experiments with the iteration technique we saw a very good convergence behavior. In theory, we shall prove that a fixed point always exists when a decomposition of this form is made. We shall present a proof for our example model.

In the submodels of Figure 4 there were two iteration variables: Λ_1, the throughput of task T_1, and Λ_5, the throughput of task T_5. The throughputs are calculated as follows:

$$\Lambda_1 = \sum_{m \in S_1} \mu_1 (C_1 I_{1,1}(m) + C_2 I_{1,2}(m)) \; \pi_m^{(1)}(\Lambda_5) \quad (1)$$

$$\Lambda_5 = \sum_{m \in S_2} (C_1 I_{5,1}(m) + C_2 I_{5,2}(m)) \; \pi_m^{(2)}(\Lambda_1) \quad (2)$$

where in Equation (1), S_1 is the set of states of submodel 1, and in Equation (2), S_2 is the set of states of submodel 2.

Note that $\pi^{(1)}$ is a function of Λ_5, because the firing rates of the transitions tps_3 and te_j in SRN 1 are approximated using Λ_5. Similarly $\pi^{(5)}$ is a function of Λ_1, because the firing rate of transitions tps_5 and te_j in SRN 2 are approximated using Λ_1. Thus Equations (1) and (2) are the **fixed point equations** in two variables corresponding to our iterative scheme. Let the function corresponding to the above equations be G. Then we must prove that a solution exists to the fixed point equation:

$$\bar{\Lambda} = G(\bar{\Lambda})$$

where $\bar{\Lambda} = (\Lambda_1, \Lambda_5)$.

We use Brouwer's fixed point theorem [9] to prove the existence of a solution to the above equation.

Theorem 1 (Brouwer's fixed point theorem) *Let $G : S \subset R^n \to R^n$ be continuous on the compact, convex set S, and suppose that $G(S) \subset S$. Then G has a fixed point in S.*

Proof of existence of a fixed point:

• *The set S*: We first remark that we do not consider the degenerate case that the underlying CTMC of either SRN 1 or SRN 2 has only one state. (This could be achieved, for instance, by setting $M_1, \ldots, M_5 = 0$, in this example.) With this assumption and the fact that the underlying CTMCs of SRN 1 and SRN 2 are irreducible, we can say that $\forall i \in S_1,\ 0 < \pi_i^{(1)} < 1$ and $\forall i \in S_2,\ 0 < \pi_i^{(2)} < 1$. Then from Equation (1), we have,

$$\Lambda_1 < 2 \ \times \mu_1 C_{max} \times |S_1| = u_1 \ \ (say) \quad (3)$$

where $C_{max} = \max\{C_1, C_2\}$. Similarly from Equation (2),

$$\Lambda_5 < 2 \ \times \mu_5 C_{max} \times |S_2| = u_5 \ \ (say) \quad (4)$$

where $|S_1|$ and $|S_2|$ are sizes of the underlying CTMCs of SRN 1 and SRN 2 respectively when both

$\Lambda_1 > 0$ and $\Lambda_5 > 0$. It is also obvious that $\Lambda_1, \Lambda_5 > 0$. Thus we choose our domain, S, as $[0, u_1] \times [0, u_5]$. Note than whenever either $\Lambda_1 = 0$ and/or $\Lambda_5 = 0$, the underlying Markov chains corresponding to SRN 1 and 5 remain irreducible and their steady state balance equations have a unique solution. Thus Equations (1),(2),(3),(4) remain valid. Therefore, G is defined on all points in S. It is also obvious from the above discussion and Equations (3) and (4) that $G(S) \subset S$.
• Since S is a closed set in R^2, it is compact and convex.
•*Continuity of G*: The function G has two types of operations in it:

- Algebraic: Multiplication, addition. These operations are continuous.
- Solution of a linear system of equation: This is the operation required to obtain the probability vectors $\pi^{(1)}$ and $\pi^{(2)}$. The continuity of the operation at points where $\Lambda_1, \Lambda_5 > 0$ is obvious. When either or both of Λ_1 and Λ_5 are zero, we note as before, that SRNs 1 and 5 give rise to irreducible CTMCs; this property assures continuity at these points.

Thus G is a continuous function. Since G meets all the conditions of Brouwer's fixed point theorem, G has a fixed point in S. The uniqueness of the solution and convergence of the fixed point iteration method have not yet been established.

It is easy to observe that this iteration scheme will always have a fixed point as long as the underlying CTMC models are irreducible. The iteration variables that result from the use of this scheme are always throughputs of certain tasks. Therefore the proof of existence of a fixed point in the general case follows as a straightforward generalization of the proof described above.

8 Conclusions

In this paper we modeled a heterogeneous multiprocessor system with priority scheduling. We built a model which can be used to evaluate systems in which tasks arrive to the system in a very general, interdependent manner. We can conclude that the use of SRNs provides the modeler with a great degree of flexibility in experimenting with different design issues such as priority assignments.

The proposed lumping technique was shown to provide highly accurate results for higher priority tasks at all levels of processor utilization. However, a great loss of accuracy is seen in performance measures of lower priority tasks for higher processor utilization levels. We conclude that this lumping technique is useful when measures of higher priority tasks are of interest. It is also useful when the service demands of the tasks are not very different. Further, under a simple condition, existence of a fixed point is guaranteed.

Further enhancements to this model include allowing a task to be preemptive, computing response-time distribution and deterministic service demands.

References

[1] G. Ciardo, A. Blakemore, P. F. Chimento, and K. S. Trivedi. Automated generation and analysis of Markov reward models using stochastic reward nets. In *Linear Algebra, Markov Chains, and Queueing Models, IMA Volumes in Mathematics and its Applications*, volume 48. Springer-Verlag, Heidelberg, 1992.

[2] G. Ciardo, J. Muppala, and K. Trivedi. SPNP: Stochastic Petri net package. In *Proc. Int. Conf. on Petri Nets and Performance Models*, pages 142–150, Kyoto, Japan, Dec. 1989.

[3] G. Ciardo and K. S. Trivedi. A decomposition approach for stochastic Petri net models. *Perf. Eval.*, 1993. to appear.

[4] J. Daigle and C. E. Houstis. Analysis of a task oriented multipriority queueing system. *IEEE Trans. Communications*, 29(11):1669–1677, Nov. 1981.

[5] N. Jaiswal. *Priority Queues.* Academic Press, New York, 1968.

[6] L. Kleinrock. *Queueing Systems Volume II: Computer Applications.* John Wiley and Sons, New York, 1976.

[7] I. Mitrani and P.J.B. King. Multiprocessor systems with preemptive priorities. *Perf. Eval.*, 1:118–125, 1981.

[8] T. Nishida. Approximate analysis for heterogeneous multiprocessor systems with priority jobs. *Perf. Eval.*, 15:77–88, June 1992.

[9] J. M. Ortega and W. C. Rheinboldt. *Iterative Solution of Nonlinear Equations in Several Variables.* Academic Press, Inc., 1970.

[10] J. L. Peterson. *Petri Net Theory and the Modeling of Systems.* Prentice-Hall, Englewood Cliffs, NJ, USA, 1981.

[11] P.Heidelberger and K.S. Trivedi. Analytic queuing models for programs with internal concurrency. *IEEE Transactions on Computers*, 32:73–82, 1983.

[12] B. Simon. Priority queues with feedback. *Journal of the ACM*, 31(1):134–149, 1984.

[13] A. Thomasian and P.F. Bay. Analytic queuing network models for parallel processing of tasks systems. *IEEE Trans. on Computers*, 35(12):1045–1054, Dec. 1986.

[14] K. S. Trivedi. *Probability & Statistics with Reliability, Queueing, and Computer Science Applications.* Prentice-Hall, Englewood Cliffs, NJ, USA, 1982.