The Impact of Variability on Soft Real-Time System Scheduling

Abstract

Soft real-time systems sometimes operate under uncertain and unpredictable environmental conditions which makes event arrival times unreliable and variable. Input to such systems also change from time to time making event processing times variable. Due to such variations, traditional techniques using worst case times to estimate system performance deviate far from actual expected behavior. It would be more accurate to analyze such systems by considering the variance present in the system. More realistic performance prediction will help in decision making while deploying a system on to given hardware.

This paper presents a Method of Stages based Analysis of soft Real Time systems (MoSART). MoSART takes into account variance in both the arrival and execution time and can model the performance of different scheduling algorithms. Sensitivity analysis, experimental validation, and the discovery of state dependent algorithms that outperform popular algorithms are demonstrated using MoSART. The results bring up several insights such as variability causes deadline miss, no algorithm is uniformly optimal and choice of scheduling algorithm becomes more important for higher utilization systems. In one scenario, a state dependant algorithm devised using MoSART and optimization techniques gives around 8% and 21% less deadline miss than Earliest Deadline First and Rate Monotonic respectively.

1 Introduction

Soft real time systems such as live audio-video or wireless sensor networks are becoming increasingly popular in our society. With the advent of cyber-physical systems, this trend will increase. Such systems can tolerate some number of deadline misses and still provide some value as long as deadline misses are within an acceptable bound. These systems typically function in an uncertain environment where the underlying infrastructure, such as wireless communication, is unpredictable. This causes data exchange between distributed entities of the system to be somewhat unpredictable, which in turn causes inter-arrival times of events to be variable. Because input to such systems (*e.g.*, events captured by a sensor or frame sizes of audio/video signals) are variable, the processing time of each event may vary across non-trivial ranges.

When such a system is deployed onto a given set of hardware, the goal is to keep the deadline misses at a minimum while achieving other goals such as good response time and optimized resource utilization. Therefore, in light of such variability, it is helpful to develop techniques that can enable an administrator to analyze a particular deployment in terms of deadline misses, response times, and resource utilization. Such a technique can also help in understanding key system insights as also help design better scheduling algorithms.

This paper presents a formal technique of modeling and analyzing soft real time systems. The method of stages is used to model the variability and uncertainty present in arrival and execution times of all tasks in the system. This technique, Method of Stages based Analysis of soft Real Time systems (MoSART), models a soft real time system and estimates the deadline misses and other important metrics such as response time, throughput, and resource utilizations for each task. Such a technique is useful to predict various system parameters given a particular deployment of the tasks onto a set of nodes.

The remainder of this paper is organized as follows: Section 2 illustrates the effect of variability on a representative system; Section 3 formulates the problem that MoSART solves; Section 4 presents the modeling approach and applies the method to rate monotonic, earliest deadline first, and least laxity; Section 5 illustrates the methodology via a complete example; Section 6 illustrates the broader benefits by providing a sensitivity analysis, experimental validation and new scheduling algorithms; Section 7 compares related work in the area of probabilistic analysis of soft real-time applications; Section 8 discusses limitations; and Section 9 presents concluding remarks.

2 Motivating Example

This section presents two motivating examples which illustrates how variability in task arrival and execution times can impact system performance. Such variability is a direct result of the uncertainty/non-determinism present in the environment in which such systems operate.

A highly loaded system: Consider a highly loaded system consisting of two tasks, one which is highly variable (*i.e.*, Task 2) in its arrival times, while the other task (*i.e.*, Task 1) has less variance in its arrival times. Both tasks have the same average arrival and execution times. Subsequent arrivals are assumed to impose a deadline for the previous task arrivals. This system is analyzed and the deadlines met

for each task is obtained when each task is given scheduling priority. The task details are given in Table 1.

	Inter-Arrival Time		Execution Time			
Tasks	Mean (secs)	C.V.	Mean (secs)	C.V.	Deadline Met % Priority Task 1	Deadline Met % Priority Task 2
Task 1	10.00	0.32	6.0	0.32	86.91	44.60
Task 2	10.00	1.00	6.0	0.32	27.66	55.83
Total	-	-	-	-	57.28	50.22

Table 1. A highly loaded system

Although the tasks are identical except in their second moment (*i.e.*, coefficient of variation (CV) = standard deviation/mean), giving priority to the less variable Task 1 improves the number of met deadlines by over 14%. This indicates that scheduling based on the variance in the tasks impacts the number of deadlines met/missed.

A lightly loaded system: Now consider the exact same set of tasks but with one exception. The mean inter-arrival time of the tasks is increased to make the system lightly loaded. As before, the two tasks are identical except for their second moment. The results are shown in Table 2. In this case the number of deadlines met is better when priority is given to the task with higher variance. This result is opposite to what was seen in the earlier, highly loaded, system. Though this is a simple example, it clearly indicates that both the variance and the system load should be considered when making scheduling decisions. Simplistic techniques (*e.g.*, RM, EDF, LL), that only consider the mean and ignore the variance and the system load, are suboptimal. Therefore, it is necessary to develop a technique capable of modeling the variance in the tasks as well as the variability in the system load. Such a technique can then be used to estimate crucial system parameters such as the deadline miss rate, response time, and resource utilization of a given system.

	Inter-Arrival Time		Execution Time			
Tasks	Mean (secs)	C.V.	Mean (secs)	C.V.	Deadline Met % Priority Task 1	Deadline Met % Priority Task 2
Task 1	30.00	0.32	6.0	0.32	99.96	98.80
Task 2	30.00	1.00	6.0	0.32	77.43	82.03
Total	-	-	-	-	88.69	90.42

Table 2. A lightly loaded system

3 Problem Formulation

In this paper a real-time system is viewed as a set of N independent tasks. Each task T_i consists of a series of job arrivals. $J_{i,j}$ represents the jth instance of task T_i and has a deadline that corresponds to the next arrival of the task, along with an expected execution time. Both the inter-arrival time and the execution time are variable. and are considered to be random variables with a probability distribution. It is assumed that the distribution of both is available from historical data. The problem is to estimate the percentage of deadlines being missed/met, the expected task response time, and the resource utilization when a set of tasks is assigned to a processor, given a specified scheduling algorithm (*e.g.*, rate monotonic, earliest deadline first, least laxity).

4 Modeling Approach

This section describes the methodology underlying MoSART. In its simplest form, the system is viewed as a single processor (*i.e.*, resource) that is shared by several tasks. For simplicity, and without loss of generality, it is assumed that the deadline of a job coincides with the subsequent job arrival from the task, (*i.e.*, each job $J_{i,j}$ is expected to finish before the next job $J_{i,j+1}$ of task T_i arrives). This assumption simplifies the presentation here and does not limit the generality of the modeling methodology. This assumption is also realistic in many applications (*e.g.*, the previous weather report is due before new weather data is downloaded).

4.1 Workload Modeling

Workload modeling consists of three parts: the inter-arrival time, the deadline, and the execution time. For simplicity reasons, since the deadline is assumed to coincide with the next arriving task, only the inter-arrival time and the execution time processes need to be modeled explicitly.

As illustrated in Section 2, its important to model the variance present in task parameters. Thus both the inter-arrival time and the execution time of tasks are treated as random variables with a probability distribution. Phase-type distributions [8] are used to approximate these distributions. An Erlang distribution using the method of stages (MOS) is a special form of a phase type distribution which helps in modeling distributions with a coefficient of variation (CV, which is the ratio of the mean to the standard deviation) less than 1. In the case of a distribution having a CV greater than 1, an analogous technique using a hyper-exponential method of stages can be used. However, in this paper, the discussion is restricted to distributions with CVs less than 1.

The MOS is composed of a series of exponential stages. Each of the stages has the same mean (λ). The CV is directly related to the number of stages (k) used. Varying these two parameters, it is possible to accurately model distributions whose CVs are less than one. The probability density function (pdf), mean, variance, and coefficient of variance (CV) for a k-stage Erlang distribution are given in the following table.

PDF	Mean	Variance	CV
$\frac{(k\lambda)^k}{(k-1)!}x^{k-1}e^{-k\lambda x}$	$\frac{1}{\lambda}$	$\frac{1}{k}\left(\frac{1}{\lambda}\right)^2$	$\frac{1}{\sqrt{(k)}}$

Table 3. Erlang distribution parameters

When k is equal to 1, the pdf reduces to that of an exponential distribution, which has a CV of 1. Increasing the

value of k decreases the variance of the distribution. A larger number of stages increases the granularity and provides more detailed information on the task's arrival and execution but at a cost of increased computation overhead. If more moments of the arrival and execution distribution are desired to be matched, a more general phase type distribution could be used, but with added complexity. Other authors [16, 14, 17, 6] demonstrate how various distributions can be modeled effectively using phase type distributions.

4.2 System Modeling

This section describes the task representation, system model parameters, resource allocation, task arrivals, task service, and task deadlines. Together with a specified scheduling algorithm, a comprehensive modeling framework is provided.

4.2.1 Task Representation

The system workload is viewed as a collection of tasks. Jobs belonging to a particular task are statistically identical to each other, but the jobs of different tasks may have different arrival and service (*i.e.*, execution) characteristics. Each task is modeled by a set of arrival stages and a set of service stages.

The arrival stages model the time that the next job in a particular task arrives to the system. Completion of all the arrival stages indicates an arrival of a new job. The service stages represent the execution time of a job in a particular task and completion of the last service stage indicates job completion. A job will meet its deadline if its service stages complete before the arrival stages complete.

Figure 1 shows the representation of a particular task stream. In this example, there are two arrival stages and four

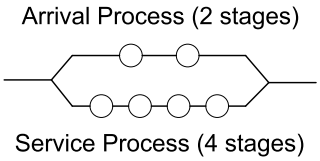

Figure 1. Task Representation

service stages. The arrival stages are statistically identical to each other. Similarly, the service stages are statistically identical to each other.

A detailed description and example of the state space model is developed in subsequent sections. Since each stage within a particular arrival or service process is statistically identical, and the time spent in a stage is assumed to be exponentially distributed, the completion time of each process has an Erlang distribution. Therefore, the underlying state space model is a Markov chain. Specifying the current stage of each process for each task completely describes the current system state.

4.2.2 Model Parameters

The state space model depends on the following parameters.

- N - number of workload tasks.
- A_i - number of arrival stages for task i jobs.
- S_i - number of service stages for task i jobs.
- μ_i - execution rate for task i jobs. Thus, $1/\mu_i$ is the average execution time for a job of task i to complete all of its service stages. Similarly, $1/(S_i\mu_i)$ is the average time that a job of task i spends at each of its service stages.
- λ_i - arrival rate for task i jobs. This represents the rate at which the arrival process for task i completes all its stages, triggering a new arrival (and signalling a deadline). Thus, $1/\lambda_i$ is the average inter-arrival time between jobs in task i and $1/(A_i\lambda_i)$ is the average time that a job of task i spends at each of its arrival stages.
- P - number of service resources (*e.g.*, processors) available for allocation.
- scheduling algorithm - algorithm used to determine the resource allocation among multiple concurrent tasks. MoSART assumes that task allocation can change at each stage boundary. Popular algorithms like Rate Monotonic (RM), Earliest Deadline First (EDF), Least Laxity First (LLF), or any new algorithm can be modeled.

4.2.3 State Space Model

This section describes the state space model in the context of an example system of tasks. It is assumed that the system is pre-emptive and that a scheduling decision is made at every stage completion. Without loss of generality, a single processor system is assumed. The set of tasks given in Table 4 is used. The arrival events of task 1 are modeled using 2 stages while the execution is modeled using 4 stages. This is because the CV of each event is $1/\sqrt{(\#of stages)}$. The stage values are shown in the CV column in Table 4. Similarly, the arrival of task 2 is modeled using 3 stages and execution with 5 stages. Thus, $N = 2$, $A_1 = 2$, $A_2 = 3$, $S_1 = 4$, $S_2 = 5$, and $P = 1$.

	Inter-Arrival Time			Execution Time		
Tasks	Mean(secs)	CV	Rate (Jobs/min)	Mean(secs)	CV	Rate (Jobs/min)
Task 1	10.0	$0.7(1/\sqrt{(2)})$	6.0	4.0	$0.5\ (1/\sqrt{(4)})$	15.0
Task 2	15.0	$0.57(1/\sqrt{(3)})$	4.0	5.0	$0.45(1/\sqrt{(5)})$	12.0

Table 4. Task Parameters

Figure 2 shows a portion of the underlying state space model that shows the possible state transitions due to an arriving task. The arrival process for task 1 has a 2-stage (*i.e.*, the top series of stages) Erlang distribution with an arrival rate of $\lambda_1(6.0)$. Thus, the rate leaving each stage is $2\lambda_1(12.0)$. Job 2 tasks have a 3-stage (*i.e.*, the third series of stages) Erlang arrival distribution with an arrival rate of $\lambda_2(4.0)$. Thus, the rate leaving each stage is $3\lambda_2(12.0)$.

The second and fourth series of stages represents the execution process for task 1 and task 2, respectively. The solid

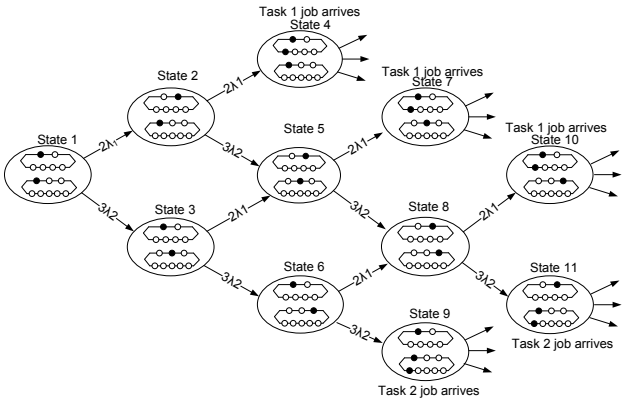

Figure 2. Task Arrivals

black circles (stages) in Figure 2 represent the currently active stages. For instance, in state 1, the arrival processes of the two tasks are both executing their first stage and no jobs are executing in the system .

The current stage of each process determines the current state of the job. Collectively, the state of each task represents the overall system state. For example, $\{(1,1),(2,0)\}$ represents the state at state 7. This means the state of the first task is $(1,1)$ while that of the second is $(2,0)$. The first task arrival process is less than 50% complete and execution is less than 25% complete. Similarly second task arrival is between 33.3% and 66.6% complete while the execution has not started yet. The completion of any stage results in a change in the system state.

Task Arrival: The arc labels in Figure 2 indicate the rates at which the various state transitions occur. Initially, in state 1, neither task has arrived yet and both of the tasks are in their first stage of arrival. Only the arrival processes are active and none of the service processes are active. The arrival process of task 1 can complete its first arrival stage at rate $2\lambda_1$, causing the system to move to state 2. Alternatively, the arrival process of task 2 can complete its first arrival stage at rate $3\lambda_2$, resulting in a transition to state 3. From state 2, the system can either move to state 4 if the arrival process of task 1 completes its 2nd stage, or to state 5 if the arrival process of task 2 completes its 1st stage. In state 4, the arrival process of task 1 has just completed both its arrival stages, which represents the arrival of a new task 1 task to the system. Consequently, the service process of task 1 becomes active (indicated by the black circle in task 1's service process) and the subsequent arrival/deadline process for task 1 is restarted at stage 1.

Task Completion and Missed Deadlines - A race between stages: Figure 3 shows another portion of the underlying state space model. In state 2, three transitions are possible: (1) the arrival/deadline process of task 1 can complete a stage with rate $2\lambda_1$ which models a missed deadline due to the arrival of a new task 1 and the system enters state 5, (2) the service process of task 1 can complete a stage with rate $4\mu_1$ and the system enters state 6, or (3) the arrival/deadline process of task 2 can complete a stage with rate $3\lambda_2$ and the system enters an unshown state indicated by the tiny arrow. If transition 1 occurs before 2 or 3, the task misses a deadline (as shown in the figure). Similarly, the transition from state 6 to state 11 represents an execution completion for task 1. This figure also shows the same

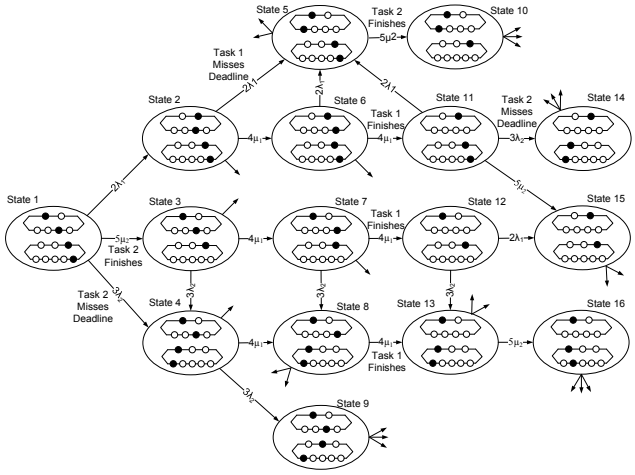

Figure 3. Task Execution/Deadline - EDF

for task 2 (*i.e.*, a met deadline in going from state 1 to 3 and a missed deadline in going from state 1 to 4).

In state 7, if task 1 completes its final service stage (with rate $4\mu_1$), the service process completes, showing that task 1 met its deadline. In this case, the task's service process terminates, but its arrival process continues. Task 2 goes through similar situations. In state 1, if the arrival process of task 2 now completes one more stage before the service process can complete its stage, then a new task 2 will arrive (*i.e.*, state 4) and the previous job of task 2 will miss its deadline. Again, if the service stage (with rate $5\mu_2$) completes before the arrival stage, the task will successfully meet its deadline (state 3). The other states and state transitions are similarly understandable.

A job can terminate in two ways, either the last stage of its arrival completes or the last stage of its execution process completes. If the service stages completes first, then the job meets its deadline. On the other hand, if the arrival stages completes first, then the job misses its deadline. The model can therefore be thought of as a horse race. At task arrival time, two "horses" (*i.e.*, an arrival/deadline horse and a service horse) begin racing. Each goes at its own pace and begins running through its successive stages. The first horse crossing the finish line (*i.e.*, its last stage) wins. If the service process horse wins, the task successfully meets its deadline. If the arrival/deadline horse wins, the task misses its deadline.

Modeling Scheduling Algorithms: The scheduling algorithm modeled in Figure 3 is earliest (expected) deadline first. In the given set of tasks, the arrival/deadline stages have the same expected time in both the tasks. The same is true for the execution stages of each task. This allows the reader to simply count the number of remaining arrival/deadline stages to determine which task has the earliest expected deadline.

For example, in state 1, task 1 has 1 remaining arrival stage until its next arrival/deadline and task 2 has no remaining arrival stages until its next arrival/deadline. Thus, the expected deadline for task 2 is earlier than that of task 1 and for this reason, task 2 is allocated the processor to execute. Since task 1 has no allocated resources, the service process for task 1 cannot proceed. Task 2's execution can continue as shown by the transition from state 1 to state 3 with a rate of $5\mu_2$. In state 4, however, task 1's deadline is 1 stage away, while task 2's deadline is 2 stages away. Thus, the processor is given to task 1. When the resource is allocated to task 1 (task 2), the average service time for each stage is $1/4\mu_1$ ($1/5\mu_2$). Note that the arrival/deadline process for a task always continues to advance regardless of any allocated resources. The various states and transitions in the state diagram are self-evident. In state 1, if the arrival process of task 1 completes a stage with rate $2\lambda_1$ (*i.e.*, transitioning to state 2), the expected deadline of task 1 is now smaller than that of task 2. Task 2 is therefore pre-empted and the resource is allocated to task 1. In the similar way, other scheduling algorithms like rate Mmonotonic (RM) or least laxity first (LLF) can also be modeled. For example, if RM is modeled, since task 1 has fewer arrival stages, it has a smaller inter-arrival time, and thus it will always receive priority. Thus, under RM, in state 1, priority will be given to task 1 instead of to task 2.

5 A Complete Example

This section presents a complete solution to a simple example to illustrate the modeling methodology. Table 5 presents the task parameters. We deliberately keep the set of parameters simple to illustrate the full solution of the example. Section 6 presents results for more complex sets of tasks.

	Inter-Arrival Time					Execution Time				
Tasks	Mean (secs)	Rate (jobs/min)	CV	Stages	Rate/stage (jobs/min)	Mean (secs)	Rate (jobs/min)	CV	Stages	Rate/stage (jobs/min)
Task 1	10.0	6.0	0.7	2	12.0	5.0	12.0	0.57	3	36.0
Task 2	6.0	10.0	1.0	1	10.0	2.0	30.0	0.7	2	60.0

Table 5. Task parameters

Table 5 also gives the arrival and execution rate of each task as well as the rates/stage. The inter-arrival time for Task 1 jobs is 10 seconds (*i.e.*, an arrival rate of 6 jobs/minute). The arrival/deadline process of Task 1 is modeled as a two stage Erlang process since the CV is 0.7 (i.e., $CV = 0.7 = 1/\sqrt{2} = 1/\sqrt{number of stages}$).

Each stage thus has a rate of 12 jobs/minute. Similarly for task 2, the job arrival rate is 10 jobs/minute and is modeled using a single stage. The job execution rates for Task 1 and Task 2 are similarly derived to be 36 jobs/minute/stage and 60 jobs/minute/stage, respectively. Figure 4 presents the complete Markov chain state space for this example.

The earliest deadline first algorithm is used to schedule jobs in this system, as evident in State 12, where the arrival process for Task 1 is in its 2nd stage while the Task 2 arrival process is in its only stage. The estimated arrival time of task 1 is $10/2 = 5$ secs, while that of task 2 is 10 secs. The earliest deadline is expected to be for Task 1, meaning that it should be scheduled. This is shown by an outgoing arc from State 12 with rate 36 (Task 1's rate/stage) to state 17.

Model analysis. Once the complete state space is enumerated, the model can be solved analytically to provide various performance metrics including resource utilization, the average number of jobs of each task in the system, and the deadline miss/met ratio of each task. These metrics follow directly from the steady-state probabilities of being in each system state. is detailed in Appendix **??**. Once probabilities are computed, the various other system performance metrics can be derived. The rate at which a task meets its deadlines is obtained by finding the product of the probability of the execution process being in the last stage and the rate at which the execution process completes a stage. The sum of this product for all states gives the overall deadlines met estimation.

Similarly, the rate at which jobs miss their deadlines can be obtained by considering all states in which an arrival/-deadline process is executing in its last stage. The missed deadline job rate can be computed by taking the product of the probability of finding the system in that state and the rate at which the arrival/deadline process completes a stage, and summing across all such states. The missed deadline job rate for each task is thus given by:

$$R_{d1} = 12\times \quad (P_7 + P_{11} + P_{12} + P_{16} + P_{17} + P_{20} + P_{21} + P_{23} + P_{24})$$

$$R_{d2} = 10\times \quad (P_3 + P_5 + P_6 + P_9 + P_{10} + P_{12} + P_{14} + P_{15} + P_{17} + P_{18} + P_{19} + P_{20} + P_{21} + P_{22} + P_{23} + P_{24})$$

where R_{d1} and R_{d2} are the rates at which the two tasks miss their deadlines, respectfully. P_i is the steady state probability of being in State i. Similarly, the rate at which the two tasks meet their deadlines is given by

$$R_{m1} = 36\times \quad (P_{13} + P_{16} + P_{21} + P_{24})$$

$$R_{m2} = 60\times \quad (P_6 + P_{10} + P_{15} + P_{19} + P_{22})$$

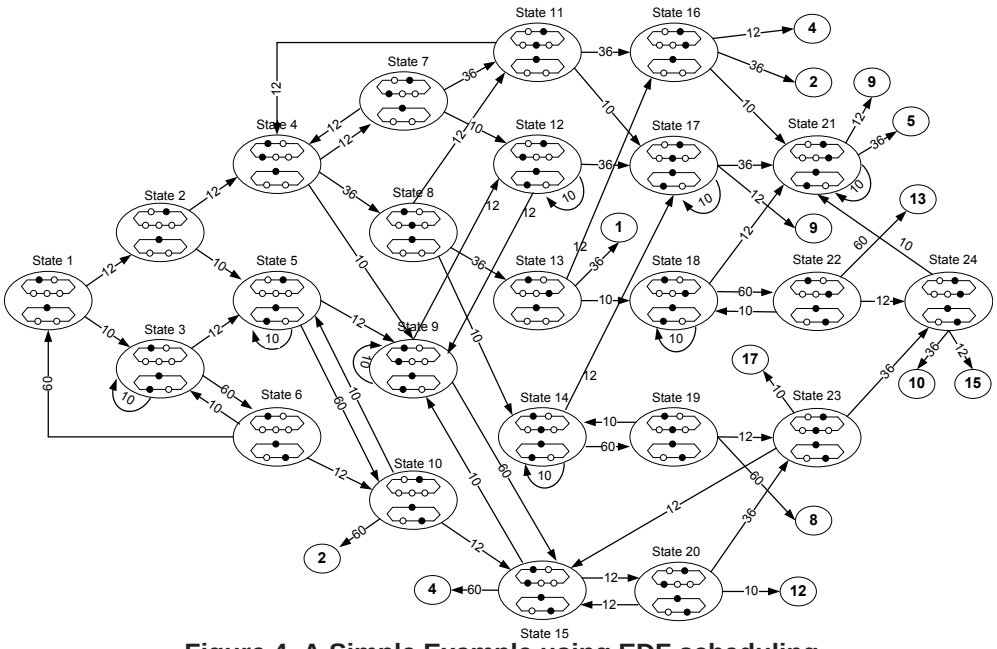

Figure 4. A Simple Example using EDF scheduling

Other performance metrics such as the utilization of the resource (*i.e.*, processor) can also be obtained easily. For example, the processor utilization of the system is given by the sum of the probabilities of the states in which at least one task is executing. This utilization calculation is equivalent to subtracting the probability that the processor is idle from one, which is $1 - \sum_i P(s_i|s_i \in S_o)$, where S_o is the set of states in which no tasks are executing. The utilization of the processor is therefore:

$$U \quad = \quad 1 - P_1 + P_2 \tag{1}$$

The utilization of the processor by each task is calculated similarly. Table 6 shows the resulting performance metrics. The data obtained while using the rate monotonic and least laxity first scheduling algorithms is also included.

Earliest Deadline First			
Tasks	Deadline Misses/min	Deadlines Met/min	Utilization
Task 1	1.97 (33%)	4.03 (67%)	0.39 (39%)
Task 2	3.30 (33%)	6.70 (67%)	0.25 (25%)
Total	5.27 (33%)	10.73 (67%)	0.64 (64%)
Rate Monotonic			
Task 1	2.32 (39%)	3.68 (61%)	0.37 (37%)
Task 2	2.66 (26.6%)	7.34 (73.4%)	0.27 (27%)
Total	4.97 (31%)	11.02 (69%)	0.64 (64%)
Least Laxity First			
Task 1	1.91 (32%)	4.09 (68%)	0.40 (40%)
Task 2	3.88 (38.8%)	6.12 (61.2%)	0.24 (24%)
Total	5.79 (36%)	10.21 (64%)	0.64 (64%)

Table 6. Example performance metrics

The results in Table 6 show that the tasks have a high percentage of deadline misses, though the utilization of the processor is not high. The reason for this result is the high variability of the inter-arrival times and the execution times for each task in this example. The CV of the arrival and execution processes vary between 1.0 and 0.57. The variation observed in practice is typically lower. Section 6 presents an analysis showing how the deadline miss rate of each task diminishes with a decrease in variability in the arrival and execution rates.

6 Broader Methodology Benefits

In addition to providing a common framework for uniformly comparing a broad spectrum of scheduling algorithms, the modeling methodology offers a convenient way to analyze the effects of variability with respect to the arrival/deadline process, the execution process, and the system load. It gives a platform to (1) conduct comprehensive sensitivity analysis, (2) perform experimental validation studies, and (3) search for new, optimal scheduling algorithms.

6.1 Sensitivity Analysis

The modeling approach presented in Section 4 is relatively simple to understand, yet powerful enough to explore a wide range of workload and system parameters. The parameters are described in Section 4.2.2 and include the number of tasks, the number of arrival/deadline stages for each task, the mean arrival/deadline rate for each task, the number of service stages for each task, the mean service rate for each task, the number of servers, and the scheduling algorithm. By holding certain parameters constant while varying others, new insights and rules-of-thumb are possible.

Consider the effect that the workload intensity has on the percentage of deadline misses under various algorithms and with various numbers of stages. Workload intensity is captured by the ratio of the aggregate arrival rate of tasks to the required service rate of the tasks. In a single server environment, this intensity is equivalent to the processor's utilization. In this sensitivity analysis, three systems are evaluated: (1) system A with a high utilization of 83%, (2) system B with a medium utilization of 67%, and (3) system C with a low utilization of 47%. Two tasks are modeled, with parameters shown in Table 7. The variability in the

	Task 1		Task 2		
System	Arr. Rate (jobs/min)	Exec. Rate (jobs/min)	Arr. Rate (jobs/min)	Exec. Rate (jobs/min)	Expected Util
A	5	15	6	12	83.34
B	5	30	10	20	66.67
C	5	30	6	20	46.67

Table 7. Task parameters

inter-arrival times and the execution times is captured by varying the number of stages from 1 (with a high CV = 1) to 8 (with a low CV = 0.35). Three scheduling algorithms (*i.e.*, rate monotonic, earliest deadline first, and least laxity first) are compared. For each set of parameters, the MoSART technique provides the percentage of missed deadlines. The results of applying this sensitivity analysis to systems A, B, and C are shown in Figure 5, 6, and 7.

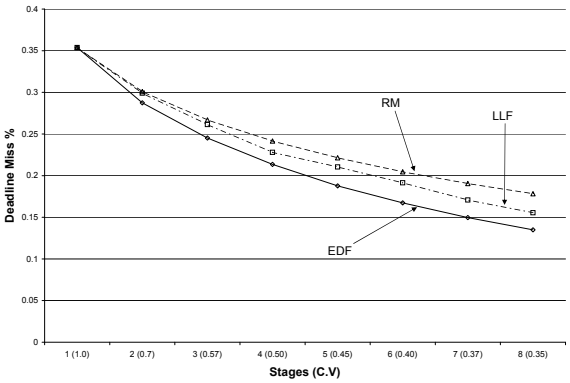

Figure 5. System with 83% Utilization

The following conclusions can be derived from these figures:

• **Lower variability improves performance.** The performance of the system improves as the variability within the system decreases. As the number of stages increases, the inter-arrival times and the execution times become more deterministic and the deadline miss percentages are reduced. For systems with a high utilization, the number of missed deadlines is reduced by a factor of two as the CV decreases from 1.00 to 0.35 (see figure 5). For systems with low utilization, the effect is even more dramatic, where the number of missed deadlines is cut by a factor of eight (see figure 7). Even systems with low utilization (*e.g.*, 47%) can expect

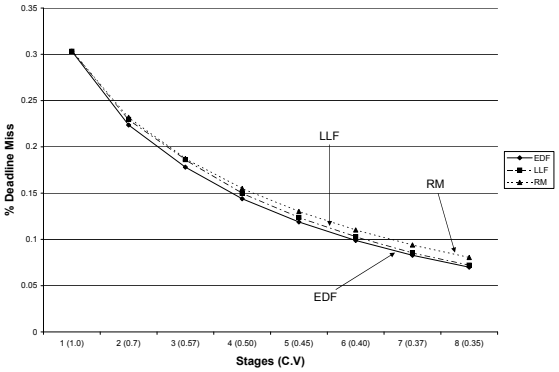

Figure 6. System with 67% Utilization

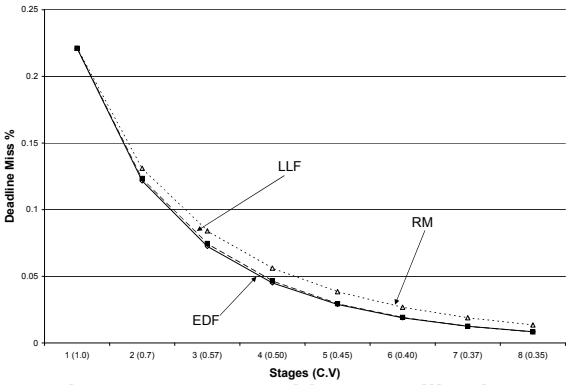

Figure 7. System with 47% Utilization

relatively high deadline miss ratios (e.g., above 20%) if the arrival and service time variability is high (*e.g.*, CV = 1.0).

• **No algorithm is uniformly optimal.** A cursory glance at Figures 5, 6, and 7 indicates that EDF, outperforms LLF, which outperforms RM. Moreover, this performance difference widens as the system becomes more deterministic and more heavily utilized. For instance, in a system with high utilization and low variance (*i.e.*, the right side of Figure 5), EDF has more than 10% fewer deadline misses than LLF, and more than 20% fewer deadline misses than RM. Under the workload setting in the previous section (*e.g.*, see Table 6), however, RM exhibited 6% fewer deadline misses than EDF and 10% fewer deadline misses than LLF, where system utilization was 64%. A closer look at Figure 7 reveals that the RM line crosses both the EDF and LLF lines around stage 2. The best algorithm depends on the specific system parameters. MoSART provides a useful and comparative sensitivity analysis given specific system parameters.

• **The scheduling algorithm choice is more important for higher utilized systems.** When system utilization is low (*e.g.*, see Figures 6 and 7), all algorithms perform similarly. Figure 5, however, demonstrates a more distinct difference between the different algorithms when the utilization is higher. It is therefore more important to choose the proper algorithm as the system utilization increases. In sys-

tems with lower utilizations, the choice depends more on which algorithm is easier to implement. For instance, RM is normally easier to implement than EDF or LLF.

• **Very low utilized systems also miss deadlines.** In hard real-time (*i.e.*, deterministic) systems, the maximum "schedulable" utilization (*i.e.*, below which all deadlines will be met) of RM is given by $n(2^{1/n} - 1)$ [12], which has a value of 0.83 when the number of tasks n = 2. The systems in the sensitivity analysis therefore have utilizations within this bound. Given soft real-time assumptions (*i.e.*, arrival and service time variability), all systems miss some deadlines. As the system becomes more deterministic, the number of deadline misses approaches 0.

• **Missed deadlines lower the processor utilization.** The ratio of the arrival rate to the service rate gives the processor utilization, which is the theoretical maximum utilization assuming all tasks meet their deadlines and none terminate prematurely for missing their deadlines. As more deadlines are missed (*e.g.*, as the variability in the arrival and service processes increases), the actual processor utilization is lower than its theoretical maximum, as shown in Figure 8. This figure shows the actual processor utilization

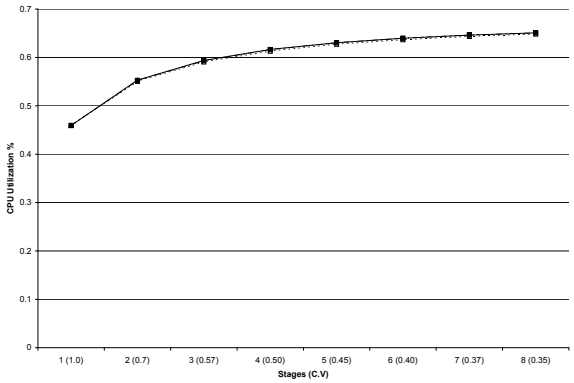

Figure 8. Utilization versus Variance

of system B (given in Table 7) as the variability decreases. This figure shows that the processor utilization approaches its maximum value of 67% as the arrival and service times become more deterministic.

The primary purpose of this sensitivity analysis is to demonstrate the effects that variability (*i.e.*, the second moment as opposed to the first moment) has on system performance. Also, MoSART is a uniform modeling methodology that can assist in examining system design alternatives before deployment.

6.2 Experimental Validation

Although MoSART is an analytical technique, it has been validated experimentally. Benchmarks were constructed and executed by emulating multiple periodic tasks on a real-time kernel. The experiments were performed on a testbed of nodes and bridges arranged in a variety of configurations. The server machines used consisted of 2.8 GHz Xeon processors, 1GB of memory, and 40GB hard disks. Each machine was running Fedora Core with real-time patches.

A typical experiment is reported here. The experiment consisted of a scheduler and two tasks. These were implemented as threads within a single process. The scheduler had the highest priority while the priority for the threads were modified by the scheduler to implement EDF.

The run-time experimental data is shown in Figure 9 along with the model predictions using the EDF scheduling algorithm. The results indicate that the modeling methodology accurately mimics the actual system behavior.

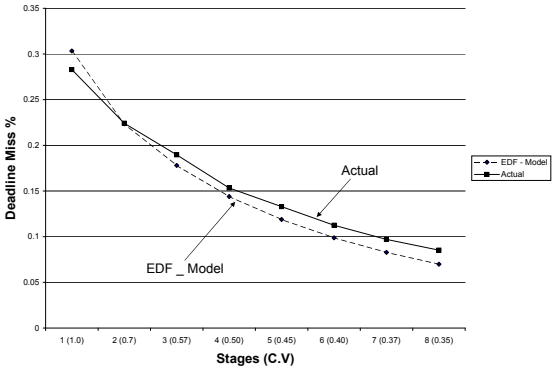

Figure 9. Experimental Validation

6.3 New Optimal Scheduling Algorithms

The modeling approach presented here not only analyzes existing scheduling algorithms, but can also be used to search for new algorithms. The modeling methodology leads to a state space model, where in any particular state, the scheduling algorithm dictates which of the competing tasks receives service from the processor.

For example, if two tasks are competing for the processor, under EDF the task with the fewest number of arrival/deadline stages left is assigned the processor, which may not be optimal. If the first task's deadline is imminent—but several service stages have not completed—the probability that the task will ultimately meet its deadline may be quite low. In this case, if the processor is allocated to the first task according to the EDF policy, not only will it likely miss its deadline, but because the processor wasted its time on a lost cause, the second task might also miss its deadline. It would be better to "sacrifice" the first task to save the second task. Such a scheduling policy is heavily state dependent, and depends upon the state (*i.e.*, the variability of the arrival and services processes, as well as the overall system load) represented by each of the competing tasks.

The state space methodology therefore lends itself directly to searching for new, optimal, scheduling algorithms. Abstractly, in every state, an unknown probability can be assigned to how much of the processor is allocated to each of the competing tasks. In a two-task system, this leads to a p-vector, with one element per system state. Each scheduling algorithm has a unique p-vector. For example, in Figure 4, the 24-element p-vector representing EDF is given by $[xx01001100111001100110 11]$, where x represents a state where no task is in the system (and, thus, the processor is not allocated to either task), 1 represents a state where the processor is allocated to Task 1, and 0 represents a state where the processor is allocated to Task 2. Elements in the p-vector could be any number between 0 and 1, which represents the fraction of the processor allocated to Task 1. A p-vector value of 0.5 therefore represents a state where processor sharing occurs between the two tasks.

The goal of the scheduling algorithm is to make the best scheduling decision at each state of the system so that the given objective is optimized. By finding (or calculating) a p-vector that optimizes a particular objective function (*e.g.*, the minimal number of missed deadlines), and realizing that each p-vector corresponds to a particular scheduling algorithm, new optimal scheduling algorithms can be discovered.

To demonstrate the finding of such a p-vector, consider the specific example in Section 5. The Matlab optimization toolkit is used to compute an optimal p-vector (*i.e.*, an optimal algorithm) for the system. This algorithm makes scheduling choices at each state of the system such that the total deadline miss percentage is minimized. The deadline miss percentage for this optimal algorithm is plotted in Figures 10 and 11.

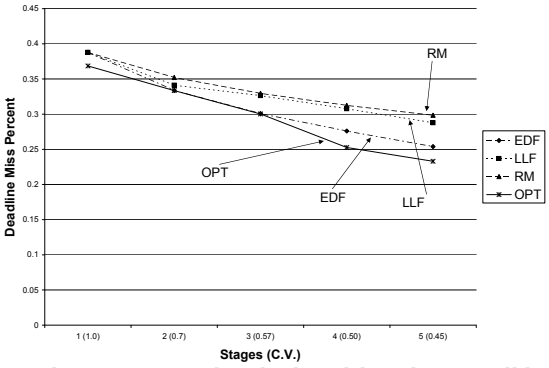

Figure 10. Optimal Algorithm (93% Util.)

The optimal p-vector algorithm is computed for two systems, one with system utilization of 83% and one with 93%. The optimal algorithm performance is plotted with that of other popular algorithms.

These figures demonstrate that there exists state dependent scheduling algorithms that outperform EDF, LLF, and RM. In general, the higher the system utilization—and the more deterministic the system—the higher the potential for improvement by using an optimal state dependent scheduling algorithm. For example, the right side of Figure 10 demonstrates that the optimal algorithm outperforms EDF by 8.2%, which outperforms LLF by 11.8%, which outperforms RM by 3.5%. By examining the p-vectors for the optimal algorithm, EDF, LLF, and RM, some (but not all) of the differences can be attributed to the sacrificing of one task to save the other task. Such investigations for better state dependent scheduling algorithms is a topic of continuing research.

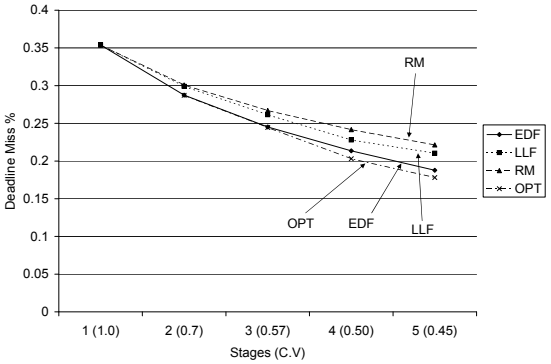

Figure 11. Optimal algorithm (83% Util.)

7 Related Work

This section compares prior work with MoSART. Tia et al. [15] extends *time demand analysis* [9, 12] by substituting the sums of fixed execution time with convolutions of probabilistic distributions. This approach allows a probabilistic expression for the response time of each task. Semi-periodic tasks are considered, where each task is released regularly but can have varying computation times. Tia et al. also presents a *transform task method* that transforms each task into a periodic task followed by a sporadic task. Burns et. al. [2] follow a similar approach to PTDA and present a probabilistic framework for analyzing faults in soft real-time systems. Cucu and Tovar [3] model the response times of fixed priority tasks with random variables. They calculate the response time of a job as the sum of response times of all higher priority tasks and its own response time. Their approach is limited to fixed priority tasks, and it is also hard to obtain other system metrics, such as device utilizations. Luca and Buttazzo [1] implement a bandwidth reservation strategy to schedule tasks with variations in task parameters. However, they do not handle variations in both arrival and execution at the same time.

Diaz [4] and Kim [7] present a derivative method for analytical and numerical solutions that calculates the response time distributions of each task in the system. The method is more complex when extended to include both fixed priority and dynamic priority scheduling. It is also hard to

measure other performance measures of the system such as resource utilization. Manolache [13] presents an analytic method based on Concurrent Generalised Petri Nets (CGPN) however the arrival times are assumed deterministic. Florescu [5] models probabilistic arrival and execution but it is entirely based on simulation. Lehoczky [10, 11] extends classical queuing theory by including customer timing requirements into traditional queuing models. The state variables of such a queuing system are continuous and unbounded. Lehoczky solves this problem under the heavy traffic case where the solution becomes simple.

The MoSART methodology is similar to Lehoczky's approach. The main difference is that tasks are modeled using Erlangian arrival and service times. Instead of a continuous state space, therefore, a series of exponential stages is used to model the passage of time in discrete steps. The advantage in this approach is that a wide variety of arrival, service, deadline, and scheduling characteristics can be modeled within a uniform framework.

8 Arrival Process Modeling

Accurately modeling the arrival process using a series of stages approach is challenging. As shown, depending on the scheduling algorithm, the series of stages is used to determine task preemption. One could think of this as a simple way for the scheduler to model time. The arrival stages predict the arrival instant of the next job. This can be used effectively by a scheduler. This models an "progress estimator" for the arrival of the next job, and is actually predicting the remaining deadline at each instant of time. For example, if a large number of stages are left before the next arrival, the scheduler will estimate the deadline to be far off. However, if there are only a few stages left, the scheduler may assume the deadline to be very close. Such information can be used by the scheduler to efficiently schedule the next job. The implementation of an EDF or LLF scheduler using such estimates would be better than using a simple constant arrival time.

9 Concluding Remarks

Various soft real time systems such as wireless sensor networks function under severe, unpredictable, and uncertain environments. Such conditions cause the arrival rates and processing times of events to be variable. It is important to model this variability to accurately estimate the various characteristics within soft real time systems. This paper presents a novel technique, MoSART, for modeling and analyzing soft real time systems with variability in the inter-arrival and execution time of events.

MoSART uses the method of stages to model the inter-arrival, deadline, and execution times. It also presents an intuitive "race" between arrival and deadline stages to model the number of deadlines met or missed for any particular job. MoSART is used to evaluate common scheduling algorithms, including earliest deadline first, least laxity first, and rate monotonic, using various sensitivity analysis comparisons. MoSART has also been experimentally validated and can be used to discover improved scheduling algorithms in a variety of contexts.

A limitation of the proposed technique is state-space explosion. Moreover, approximations, bounds, and simulations can be directly applied. Addressing such limitations, applying the methodology to a wider set of system parameters, evaluating a richer set of scheduling algorithms, conducting a more extensive experimental validation, and using the methodology to discover new scheduling algorithms are promising directions for future research.

References

[1] L. Abeni and G. Buttazzo. QoS guarantee using probabilistic deadlines. In *Real-Time Systems, 1999. Proceedings of the 11th Euromicro Conference on*, pages 242–249, 1999.

[2] A. Burns, G. Bernat, and I. Broster. A probabilistic framework for schedulability analysis. *Proceedings of the Third International Embedded Software Conference, EMSOFT, number LNCS*, 2855:1–15, 2003.

[3] L. Cucu and E. Tovar. A framework for the response time analysis of fixed-priority tasks with stochastic inter-arrival times. *SIGBED Rev.*, 3(1):7–12, 2006.

[4] J. L. Díaz, D. F. García, K. Kim, C.-G. Lee, L. L. Bello, J. M. López, S. L. Min, and O. Mirabella. Stochastic analysis of periodic real-time systems. In *RTSS '02: Proceedings of the 23rd IEEE Real-Time Systems Symposium (RTSS'02)*, page 289, Washington, DC, USA, 2002. IEEE Computer Society.

[5] O. Florescu, M. de Hoon, J. Voeten, and H. Corporaal. Probabilistic modelling and evaluation of soft real-time embedded systems. In *SAMOS*, pages 206–215, 2006.

[6] M. Johnson. An empirical study of queueing approximations based on phase-type distributions. *Stochastic Models*, 9(4):531–561, 1993.

[7] K. Kim, J. L. Diaz, and J. M. Lopez. An exact stochastic analysis of priority-driven periodic real-time systems and its approximations. *IEEE Trans. Comput.*, 54(11):1460–1466, 2005. Member-Lucia Lo Bello and Member-Chang-Gun Lee and Member-Sang Lyul Min.

[8] G. Latouche and V. Ramaswami. *Introduction to Matrix Analytic Methods in Stochastic Modeling*. Society for Industrial Mathematics, 1999.

[9] J. Lehoczky, L. Sha, and Y. Ding. The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior. In *RTSS' 89*, pages 166–171, 1989.

[10] J. P. Lehoczky. Real-time queueing theory. In *RTSS '96: Proceedings of the 17th IEEE Real-Time Systems Symposium (RTSS '96)*, page 186, Washington, DC, USA, 1996. IEEE Computer Society.

[11] J. P. Lehoczky. Using real-time queueing theory to control lateness in real-time systems. In *SIGMETRICS '97: Proceedings of the 1997 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 158–168, New York, NY, USA, 1997. ACM.

[12] C. Liu and J. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-time Environment. *JACM*, 20(1):46–61, Jan. 1973.

[13] S. Manolache, P. Eles, and Z. Peng. Schedulability analysis of applications with stochastic task execution times. *Trans. on Embedded Computing Sys.*, 3(4):706–735, 2004.

[14] R. Marie. Calculating equilibrium probabilities for λ(n)/ck/1/n queues. *SIGMETRICS Perform. Eval. Rev.*, 9(2):117–125, 1980.

[15] T.-S. Tia, Z. Deng, M. Shankar, M. Storch, J. Sun, L.-C. Wu, and J. W.-S. Liu. Probabilistic performance guarantee for real-time tasks with varying computation times. In *RTAS '95: Proceedings of the Real-Time Technology and Applications Symposium*, page 164, Washington, DC, USA, 1995. IEEE Computer Society.

[16] H. Tijms. *Stochastic models: an algorithmic approach*. Wiley, 1994.

[17] W. Whitt. Approximating a Point Process by a Renewal Process, I: Two Basic Methods. *Operations Research*, 30(1):125–147, 1982.