A Probabilistic Performance Metric for Real-Time System Design

Tao Zhou Xiaobo(Sharon) Hu Edwin H.-M. Sha
Dept. of Computer Science and Engineering
University of Notre Dame
Notre Dame, IN 46556
Email:{tzhou, shu, esha@cse.nd.edu}
Tel: (219)631-6015 Fax: (219)631-9260

Abstract

At the system level design of a real-time embedded system, a major issue is to identify from alternative architectures the best one which satisfies the timing constraints. This issue leads to the need of a metric that is capable of evaluating the overall system timing performance. Some of the previous work in the related areas focus on predicting the system's timing performance based on a fixed computation time model. These approaches are often too pessimistic. Those that do consider varying computation times for each task are only concerned with the timing behavior of each individual task. Such predictions may not properly capture the timing behavior of the entire system. In this paper, we introduce a metric that reflects the overall timing behavior of RTES. Applying this metric allows a comprehensive comparison of alternative system level designs.

1 Introduction

A real-time embedded system(RTES) must respond to certain external events under timing constraints. If the system cannot function on time, the system performance will degrade. Or even worse, a system failure will occur, which could lead to a catastrophe. Therefore, guaranteeing that a real-time embedded system meets all its timing constraints is a key issue in designing such a system. Many industrial examples show that the system level design plays a dominating role in determining the performance of a final product [14]. Hence, a major effort at the system level design is to identify from alternative architectures the best design which satisfies the timing constraints.

Many algorithms for predicting the timing performance of a real-time embedded systems exist [1–4, 6–9, 13]. A classical paper by Liu and Layland [9] presented an upper-bound on the overall processor utilization in order for each independent periodic task to complete on or before its deadline. Lehoczky *et al*. [6] extended the result by characterizing the exact behavior of each task meeting its timing requirement. Other algorithms [1, 4] either improve the above mentioned results or consider somewhat different task compositions. One common assumption used by all these papers is that each task only has a single computation time.

In most real-world applications, the computation time of a task can vary dramatically due to a number of factors. Imposing a fixed computation time for each task often leads to overly pessimistic erroneous performance estimations. A few papers published recently have studied performance estimation for tasks with varying computation times. [2, 3, 13] All of these algorithms aim at finding the probability of *each task* meeting its timing constraint. Tia *el at* [13] proposed a way to find the probability of any request of a task meeting its timing constraint by checking the processor demands for completing the request as well as all requests of higher priority tasks. They assume that the computation time of each task is described by a probability density function. Tasks are scheduled according to a fixed-priority, preemptive policy. (The paper is not consistent in handling requests missing their deadlines, which introduces an error in performance estimation. We will discuss this later.) Kalavade and Moghê [3] studied the timing performance of networked embedded systems, which may have precedence constraints. Tasks are scheduled by a fixed-priority, non-preemptive policy. The authors model the processing load of such a system by a semi-Markov process. According to this theme they developed a tool, which outputs an exact distribution of the processing delay of each task. Hou and Shin [2] derived the probability of each task meeting its timing constraint within a planning cycle, which is defined to be the least common multiple (LCM) of all task periods.

In the system level design, the goal of evaluating timing performance is to compare and perhaps rank alternative designs. Computing the probability of each task meeting its deadline by the above mentioned algorithms may not be sufficient. Timing performance of any individual task may not represent that of the entire task set, unless it is the only task of the system. Through several experiments, we have noticed that a high-level correlation often exist among tasks in the same system. Hence, the overall system timing performance cannot be evaluated by a simple average of probabilities of each task meeting its deadline. We believe that there is a need for a single system-wise parameter to measure the probability of an entire system meeting all the timing constraints. Intuitively, it is a measure of the probability of any task missing its deadline at any instant. Such a parameter would reflect the quality of the corresponding system timing implementation. In this paper we introduce a new concept which can be used to define the probability of a system meeting its timing constraints. By using this probabilistic measurement one can easily evaluate timing performance of alternative designs as the timing performance could be measured and compared based on a single value just like cost and power consumption of a system. To the authors' knowledge, this is the first paper that presents a metric of the overall timing performance for a real-time embedded system, with varying computation times.

Section 2 introduces some necessary notations and analyzes several different methods that could give a system-wise probability and their flaws. In section 3 we introduce the state concept to define the probability of a system being feasible, and use it to compute the system-wise timing behavior of a RTES. Section 4 proposes a method to approximate this system-wise probability. Experimental results are presented in section 5. Section 6 concludes the paper.

2 Assessing System-wise Feasibility Probability

The system we consider consists of n independent periodic tasks, $\boldsymbol{\tau} = \{\tau_1, \tau_2, \ldots, \tau_n\}$. They are to be scheduled on a single processor. If a RTES contains multiple processors, we assume that tasks are statically assigned to each processor. Associated with each task τ_i are three parameters **period(T_i)**, **deadline(d_i)**, and **computation time(C_i)**. Without loss of generality, we assume that C_i is a discrete random variable which takes values $c_{i1}, c_{i2}, \ldots, c_{iK_i}$. The jth request of task τ_i, r_{ij}, requires C_{ij} time to finish, where the probability of C_{ij} equal to c_{il} is q_{il}. i.e., $P(C_{ij} = c_{il}) = q_{il}$, $1 \le l \le K_i$. We assume that $P(C_{ij} = x)$ is independent of the computation times of any other task requests. We have the following assumptions on the system:

1. The first request of each task is initiated at the same time, called critical instant [9]. This assumption makes the first request of any task has the largest response time.
2. Tasks are scheduled according to a fixed priority, preemptive scheduling algorithm. A higher priority task can always preempt a lower priority one. We assume that tasks are arranged in the decreasing order of priority, that is τ_i has a higher priority than τ_j, if $i < j$.

We would like to study the ability of such a RTES meeting its deadlines, i.e., the feasibility of this RTES, which is defined as follows:

Definition 1 *A task τ_i is feasible if each of its requests meets the deadline. A task set is feasible if every task is feasible.*

Since the computation times of tasks are probabilistic, both the feasibility of each task and that of a system also become probabilistic. We use $F(x) = 1$ (resp. $F(x) = 0$) to describe the event of x being feasible (resp. infeasible), where x is either a task or a system. For example $P(F(\tau_i) = 1)$ is the probability of τ_i being feasible.

As discussed in the introduction, $P(F(\tau_i) = 1)$ can be calculated by algorithms in [2, 3, 13]. However, such probabilities may not correctly capture the overall system behavior and hence cannot be directly applied to compare different system designs. For example, consider two alternative designs of a system consisting of two periodic tasks. In design I, task τ_1 has a probability of 0.9 to be feasible, the probability of τ_2 being feasible is 0.8. In design II, these two probabilities are 0.8 and 0.9, respectively. It is difficult to decide which is a better design by only comparing each $P(F(\tau_i) = 1)$. One may argue that the relative importance of each task can be used in conjunction with their feasibility probabilities. Yet in some systems, such importance comparison may not exist. Furthermore, the scenarios in which timing violations in each design may not be the same. Consider another example in Table 1 which shows two task sets both of size two. For the ease of illustration, we assume in this example that whenever there is a request misses its deadline, it will be terminated and removed from the job queue. The feasibility probability of each set is given by the last column of Table 1. As Figure 1 shows, in set I, if jth request of τ_1, r_{1j} cannot finish on time, nor can the jth request of τ_2, r_{2j} be feasible. On the other hand in set II such a dependence of two requests of different tasks in missing timing constraints does not happen as frequently as in set I. A request of τ_2 may finish on time when a request of τ_1 misses its deadline, or vise versa. Given a choice, a designer would prefer set I to Set II, since the overall system is infeasible for a shorter duration. If simply comparing each $P(F(\tau_i) = 1)$'s, there would be no difference in the timing performance of the two task sets.

	τ_i	T_i	d_i	c_{i1}	q_{i1}	c_{i2}	q_{i2}	$P(F(\tau_i) = 1)$
set I	τ_1	5	5	1	0.8	6	0.2	0.8
	τ_2	5	5	1	1.0			0.8
set II	τ_1	5	5	1	0.8	6	0.2	0.8
	τ_2	10	10	1	0.95	9	0.05	0.8

Table 1: Each task of these two sets has a same $P(F(\tau_i) = 1)$, but these two sets have different timing performance.

We introduce a single metric for measuring the timing performance of an entire system. When tasks computation times are random variables, the feasibility of the task set can be measured by a probability, $P(F(\boldsymbol{\tau}) = 1)$. The value of this probability can be considered as the probability of every task in the system is feasible, that is,

$$P\Big(F(\boldsymbol{\tau}) = 1\Big) = P\Big(F(\tau_1) = 1 \bigcap F(\tau_2) = 1 \bigcap \ldots F(\tau_n) = 1\Big). \quad (1)$$

Note that formula (1) can also be written as:
$P\Big(F(\tau_n) = 1 | F(\tau_1\tau_2\ldots\tau_{n-1}) = 1\Big) * P\Big(F(\tau_{n-1}) = 1 |$
$F(\tau_1\tau_2\ldots\tau_{n-2}) = 1\Big) \ldots * P\Big(F(\tau_1) = 1\Big)$
It follows that a system with higher $P(F(\boldsymbol{\tau}) = 1)$ has a better timing performance with respect to satisfying deadlines.

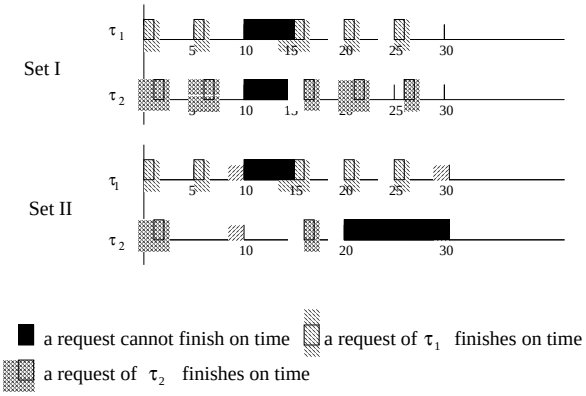

Figure 1: Two task sets have the same feasibility probability of each task, yet have different timing performance.

The challenge now is how to obtain $P(F(\boldsymbol{\tau}) = 1)$. Let us first consider some intuitive approaches and show their problems. Recall that the probability of an event can be considered as the percentage of the number of events over the total number of experiments when the number of experiments is "very large" [12]. One way to model the events of all tasks being feasible is to consider the feasible requests from all tasks. That is, let $P(F(\boldsymbol{\tau}) = 1)$ equal to the percentage of the number of feasible requests over the total number of requests as the number of total requests becomes infinite. Requests from different tasks are treated equally, even though they have different periods. (Due to the page limit, we omit further discussion on this.) Though this formulation is straightforward, the $P(F(\boldsymbol{\tau}) = 1)$ value obtained may not correctly reflect the actual

timing behavior.

Yet another way to determine $P(F(\tau)=1)$ is to approximate equation (1) by $\tilde{P}(F(\tau)=1)$, where

$$\tilde{P}\Big(F(\tau)=1\Big)=P\Big(F(\tau_1)=1\Big)P\Big(F(\tau_2)=1\Big)...P\Big(F(\tau_n)=1\Big). \quad (2)$$

This formula was used by Hou and Shin in [2]. However, since the tasks in a system are executed on a single processor, events τ_i being feasible and τ_j being feasible are dependent in the sense of scheduling even though we assume that the tasks are independent of one another. Examining the example illustrated in Figure 1, one can easily see that in task set I, τ_2's feasibility is fully dependent on that of τ_1. The conditional probability, $P(F(\tau_i)=1|F(\tau_j)=1)$ represents the probability of a request of τ_i meets its deadline when a request of τ_j is feasible. Approximating $P(F(\tau_i)=1|F(\tau_j)=1)$ by $P(F(\tau_i)=1)$ alone as did in [2] does not always properly capture the system timing behavior.(Several other approaches were studied but we omit these discussions due to the page limit.) In the following sections, we introduce a rather novel concept to evaluate $P(F(\tau)=1)$.

3 State Based Probability

The difficulty in evaluating $P(F(\tau)=1)$ lies in how to count the event of a system being feasible. Notice that some interval must be used in considering whether a system is feasible and evaluating $P(F(\tau)=1)$ as given in formula (1). If an interval is specified, the feasibility probability is simply the ratio of the expected value of the total number of intervals within which the system is feasible over the total number of intervals, for which the system is scheduled. Obviously, if a task set consists of only one task, the task period can be chosen as such an interval. When the size of a task set is greater than 1, a proper time interval is needed in order to account for the differences in task periods.

One way to define such a time interval is to set it to the least common multiple(LCM) of all task periods, $R=LCM(T_i)$, for $i=1,2,\ldots,n$. The entire task set within R is feasible if all requests submitted during this time span meet their deadlines. Otherwise, it is infeasible. Therefore, $P(F(\tau)=1)$ can be evaluated by the percentage of the number of feasible R over the total number of R during the time span the system is running. Unfortunately, the $P(F(\tau)=1)$ value computed based on R is usually very low. For instance, if the period of one task is much less than R, the task will submit many requests within R. The probability of at least one of the requests misses the deadline can be rather high, and hence the probability of the interval being feasible can be very low. This, in turn, results in low $P(F(\tau)=1)$. However, the system can actually satisfy most of the requests deadlines. In an extreme case, when R is sufficiently large(T_i's are relatively prime), within each R the system is always "infeasible". Such an estimation would result $P(F(\tau)=1)=0$, even though the system is capable of meeting some deadline requirements.

We now propose a new concept to help define the event of a system being feasible. Let a *state cycle* be the time interval between any two consecutive requests. For example, given a task system with three tasks as shown in Figure 2, assume $T_1=2,\ T_2=3,\ T_3=4$. Then, within $R=LCM(T_1,T_2,T_3)=12$, there are 8 state cycles. In general, for a task system with n tasks, each state cycle corresponds to a unique combination of n requests (since the request pattern will repeat after every R interval), with one request from each task. We denote such a combination as a *state*, s_q. In Figure 2, $s_1=\{r_{11},r_{21},r_{31}\}$, $s_2=\{r_{12},r_{21},r_{31}\}$,$s_3=\{r_{12},r_{22},r_{31}\}$, and so on.

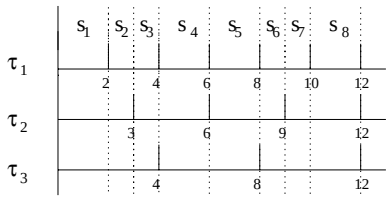

Figure 2: A state is defined by a combination of n requests, one from each task.

A state is said to be feasible if every request in the state is feasible. Since the computation time of each task is stochastic, the event of state s_q being feasible is also stochastic. We denote the probability of s_q being feasible by $P(F(s_q)=1)$ and we have:

$$P\Big(F(s_q)=1\Big)=P\Big(F(r_{1q_1})=1\bigcap F(r_{2q_2})=1\bigcap...F(r_{nq_n})=1)\Big).$$

According to the definition of a state, s_q, one can see that $P(F(s_q)=1)$ represents the timing behavior of the entire system during the state cycle when the system is in state s_q. Different states may have different probabilities of being feasible. Note that at any time instant within the state cycle, the system exhibits the same timing behavior, because the same requests are executed. If one of the request misses its deadline during the state cycle, the entire duration is considered to be infeasible. Hence, by using the state concept, we can effectively "count" the events of system being feasible. The average of the probabilities of states being feasible would be a representation of the timing performance of the overall system. Hence, we define:

$$P\Big(F(\tau)=1\Big)=\frac{1}{n_s(R)}\sum_{q=1}^{n_s(R)}P(F(s_q)=1). \quad (3)$$

where $n_s(R)$ is the total number of states. Since the same sequence of states repeats after each interval R, $n_s(R)$ can be computed by:

$$n_s(R)=1+\sum_{i=1}^{n}(\frac{R}{T_i}-1)-\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}(\frac{R}{LCM(T_i,T_j)}-1). \quad (4)$$

The proof is omitted due to the page limit.

To determine the timing performance of a system becomes finding $P(F(s_q)=1)$, for $q=1,2,\ldots,n_s(R)$. Let $N_f(t)$ be the number of feasible states within $[0,t]$. Note that $N_f(t)$ is a random variable. Furthermore, denote the expected value of $N(t)$ by $E(N_f(t))$. From the probability theory [12], we know that:

$$E(N_f(R))=\sum_{q=1}^{n_s}P(F(s_q)=1). \quad (5)$$

Combining (3) & (5), we obtain:

$$P(F(\tau)=1)=\frac{1}{n_s}E(N_f(R)). \quad (6)$$

To see if (6) gives a good indication of the system timing performance, we apply the above formula to the examples discussed in Table 1. For set I $P(F(\tau)=1)=0.8$ and for set II it is 0.76, which as analyzed in section 2, are good measurements.

4 Estimating the State Based Probability

Notice that using simulation to determine $E(N_f(R))$ and hence $P(F(\tau)=1)$ can be extremely time consuming. We present a

method to approximate $P(F(\tau)=1)$.

The value of $E(N_f(R))$ can be computed directly by combining techniques from the probability theory [12] and from scheduling theory [6]. Consider a system with n tasks. For a given computation time of each task request during $[0,R]$, we can determine the feasibility of each state by applying the algorithm in [6]. Since the task computation times are random variables, different combinations of these must be considered. Let the total number of combinations of request computation times within $[0,R]$ be $n_{cr}(R)$. Then $n_{cr}(R)$ can be calculated as:

$$n_{cr}(R)=\prod_{i=1}^{n}(K_i)^{R/T_i}$$

Where K_i is the total number of computation time variation of τ_i. The probability of a particular combination, Q_k, occurring is:

$$Q_k=P(\bigcap_{i=1}^{n}\bigcap_{j=1}^{R/T_i}(C_{ij}=c_{il}))=\prod_{i=1}^{n}\prod_{j=1}^{R/T_i}q_{il}$$

where C_{ij} is the computation time of jth request of τ_i, and takes the value c_{il} (c_{il} is any one of the possible values of C_i), with the probability q_{il}. It follows that the expected value of the number of feasible states in $[0,R]$ can be computed by:

$$E(N_f(R))=\sum_{k=1}^{n_{cr}(R)}n_f(R,k)*Q_k$$

where $n_f(R,k)$ is the number of feasible states for the combination corresponding to Q_k in $[0,R]$.

To determine $n_f(R,k)$, the algorithm presented in [6] may be used. They found that the processor time demanded to complete τ_i as well as all the higher priority tasks is

$$W_i(t)=\sum_{j=1}^{i}C_j\lceil t/T_j\rceil.$$

The idea is to find the minimum t, such that $W_i(t)\le t$. If there exists such a t, τ_i is feasible. The value of t is from the set U, where $U=\{T_k*m|\lceil(j-1)*T_i/T_k\rceil\le m\le\lceil j*d_i/T_k\rceil,k=1,2,\ldots,i\}$. One problem in directly applying the algorithm is that the algorithm only considers the first request of each task, since it assumes that all requests have the same computation time and that each task initiates its first request at the critical instant [9]. In our case, different requests of the same task may have different computation time, thus each task request must be checked to determine its feasibility in order to obtain $n_f(R,k)$. We can extend the algorithm as follows. Let $W_{ij}(t)$ be the processor time demanded to finish r_{ij} as well as requests from r_{i1} to $r_{i,j-1}$ and all the higher priority requests released during interval $I_{ij}(t)=[0,(j-1)T_i+t]$. Then $W_{ij}(t)$ is

$$W_{ij}(t)=\sum_{q=1}^{j}C_{iq}+\sum_{k=1}^{i-1}\sum_{h=1}^{z_k(t)}C_{kh}\qquad(7)$$

where $z_k(t)=\lceil((j-1)T_i+t)/T_k\rceil$. By the proofs given in [6], a request r_{ij} is feasible if there exists a value t such that

$$W_{ij}(t)\le(j-1)T_i+t\qquad(8)$$

Though formula (7) can be used to compute $n_f(R,k)$, it can be quite time consuming when the number of requests in R is very large. In the following, we consider two approaches to reduce the computing time of formula (7).

The first approach focuses on reducing the interval used for determining if r_{ij} is feasible. Tia, *et.al.* [13] proposed to use interval $I'_{ij}(t)=[(j-1)T_i,(j-1)T_i+t]$. They used $W'_{ij}(t)\le t$ to check if r_{ij} is feasible, where $W'_{ij}(t)$ is the processor time demanded by all the hinger priority tasks and the request r_{ij} since the release of r_{ij}. $W'_{ij}(t)$ is

$$W'_{ij}(t)=C_{ij}+\sum_{k=1}^{i-1}\sum_{h=z'_k}^{z_k(t)}C_{kh}\qquad(9)$$

where $z'_k=\lceil(j-1)*T_i/T_k\rceil$. They adjusted it by the following formula considering that some higher priority requests that are released earlier than r_{ij} may not have completed at r_{ij} and still demand processor time after r_{ij} is released.

$$W'_{ij}(t)=C_{ij}+\sum_{k=1}^{i-1}\sum_{h=1}^{z_k(t)}C_{kh}+\sum_{k=1}^{i-1}\sum_{h=1}^{\infty}(1-M_k)^hC_{k,-h}$$

Their formulation implies that any request of τ_i from r_{i1} to $r_{i(j-1)}$ when missing its deadline will be terminates. While requests of higher priority when missing deadlines are executed continuously. Hence there is an inconsistency in the use of scheduling policies in [13]. We propose an approach based on a consistent scheduling policy, called terminate policy. Under the terminate policy, a request is terminated and removed from the processor when missing its deadline. Such a policy would be advantageous when the possible computation times of a task vary dramatically and the task's deadline is the same as or longer than its period. Notice that at the instance the request r_{ij} is released, if $T_i \bmod T_k\neq 0$, $k<i$, a higher priority task τ_k has submitted a request r_{kz_k}, which may have finished or only partly been executed. In formula (9), C_{kz_k} is fully accounted. This introduces an error to the value of $W'_{ij}(t)$ calculated by formula (9), i.e., makes the value greater than it actually is. According to the terminate scheduling policy, the request r_{ij} will not be executed until r_{kz_k} is finished. Let t_{ij} be the instance that r_{ij} starts to be scheduled. We observe that t_{ij} is the earliest time instant at which r_{kz_k} (for all $k=1,2,\ldots,i-1$) are finished (The finishing time for r_{kz_k} is $V_{kz_k}=W_{kz_k}(t)+t_{kz_k}$). That is,

$$t_{ij}=\max\{V_{kz_k}(t)|k=1,2,\ldots,i-1,(j-1)T_i\}\qquad(10)$$

It turns out that V_{kz_k} are already computed when determining if r_{kz_k} is feasible. If we arrange the requests r_{ij} in the increasing order of both i and j, we can simply record V_{kz_k}) as needed and obtain t_{ij} by formula (10). Now, we only need to calculate $W_{ij}(t)$ in the interval $[t_{ij},t_{ij}+t]$ by

$$W''_{ij}(t)=C_{ij}+\sum_{k=1}^{i-1}\sum_{h=z''_k}^{z_k(t)}C_{kh}\qquad(11)$$

where $z''_k=\lceil t_{ij}/T_k\rceil+1$. To determine if r_{ij} is feasible, we check if there exists a t such that $W''_{ij}(t)\le t$. Thus applying a consistent terminate policy we can reduce the computing time for formula (7) without an error. Yet we don't have a way to achieve this reduction for a system not applying a terminate scheduling policy.

We now briefly discuss another approach to approximate it. Instead of calculating the probability in the interval $[0,R]$, one could apply the method described above within the interval $[0,T_n]$. The penalty

Experiments	set I	set II	set III	set IV	set V	set VI	set VII	set VIII
$P(F(\boldsymbol{\tau})=1)$ by $[0,R]$	0.8	0.76	0.62	0.71	–	0.8	–	–
$P(F(\boldsymbol{\tau})=1)$ in $[0,T_n]$	0.8	0.76	0.66	0.67	0.81	0.8	0.22	0.756
$P(F(\boldsymbol{\tau})=1)$ by simulation	0.8	0.76	0.60	0.69	0.87	0.8	0.3	0.724
multiplication	0.64	0.64	0.55	0.46	0.77	0.41	0.14	0.65

Table 2: Experiment results of eight task sets

is that such an approximation introduces an error to the result, since the states in the interval $[T_n, R]$ are omitted. Let $n_f(T_n,k)$ be the total number of states within the interval $[0,T_n]$. We have

$$P\left(F(\boldsymbol{\tau})=1\right)\Big|_{[0,T_n]} = \frac{1}{n_s(T_n)} * \sum_{k=1}^{n_{cr}(R)} Q_k * n_f(T_n,k) \tag{12}$$

where $n_s(T_n)$ is the total number of states within $[0,T_n]$. The relative error is,therefore,

$$\frac{P\left(F(\boldsymbol{\tau})=1\right) - P\left(F(\boldsymbol{\tau})=1\right)\Big|_{[0,T_n]}}{P\left(F(\boldsymbol{\tau})=1\right)} = 1 - \frac{n_s(R)}{n_s(T_n)} * \frac{\sum_{k=1}^{n_{cr}(R)} Q_k * n_f(T_n,k)}{\sum_{k=1}^{n_{cr}(R)} Q_k * n_f(R,k)}$$

5 Experimental Results

We have used our proposed metric to evaluate a number of systems. Table 2 shows 8 example task sets, for which we have obtained the probability of each set being feasible. The data of Set VIII is from [3], which is an audio and video networked embedded system. We use SES/Workbench as our simulator to get data of the third row. Three data are not available for the first row because the computing time is big. The second row is the approximation of $P(F(\boldsymbol{\tau})=1)$ within $[0,T_n]$. Formula (11) is used to calculate the processor time demanded. We use SES/Workbench as our simulator to get the data of the third row. The data in the last row are calculated according to formula (2).

Task set I and II are the same with those in Table 2. As analyzed in section 2, even though they have the same $P(F(\tau_i)=1)$, set I actually have a better timing performance. As shown in the table, $P(F(\boldsymbol{\tau})=1)$ of set I is 0.8, while that of set II is 0.76. Also in Section 2, we pointed out that formula (2) does not always capture the timing performance of a system, which may cause some errors in evaluating $P(F(\boldsymbol{\tau})=1)$. Such an error is quite significant in several of the task sets shown in Table 2. For example, for task set VI, the feasibility probability is supposed to be 0.8, but the multiplication method estimates it as 0.41. The experiments show that the probability of a system being feasible based on a *state cycle* correctly reflects the probabilistic timing behavior of a system and $P(F(\boldsymbol{\tau})=1)|_{[0,T_n]}$ is a good approximation of $P(F(\boldsymbol{\tau})=1)$.

6 Discussion

In this paper, we show that the overall timing performance is not properly represented by any of the individual task feasibility probability $P(F(\tau_i)=1)$s. We then introduce the concept of states and propose to use *state cycle* to measure the probability of an entire system being feasible. In addition, we provide an algorithm to calculate this probability based on the *state cycle*. As our analysis and examples show, the state based probability well reflects the system timing behavior. Applying this probability allows a straightforward comparison of timing performance among different system level designs by comparing a single value. Taking into consideration the variation in computation times, this probability is more flexible than the timing performance prediction based on a fixed computation time model.

Acknolowedgement

We are grateful to Dr. Asawaree Kalavade for providing data for some of our expriments. This work is supported in part by NSF under grant numbers MIP97-96162, MIP97-01416, MIP95-01006, MIP97-04276 and by an External Research Program Grant from Hewlett-Packard Laboratories, Bristol, England.

References

[1] S.C. Cheng *et. al.* "Scheduling algorithms for hard real-time systems - a brief survey", In Tutorial Hard Real-Time Systems,page 150-173,IEEE,1988.

[2] C.J. Hou, K.G. Shin "Allocation of Periodic Task Modules with Precedence and Deadline Constraints in distributed Real-Time Systems", IEEE Transaction on Computers, Vol 46, No.12, Wec.1997.

[3] A. Kalavade,*et. al.* "A Tool for Performance Estimation of Networked Embedded End-Systems", Proceeding of DAC, 1998.

[4] W.W. Leinbaugh "Guaranteed response time in a hard real-time environment" IEEE Transaction on Software Engineering, Jan. 1980

[5] R. Jain "The Art of Computer Systems Performance Analysis", 1991.

[6] J. Lehoczky,*et. al.* "The Rate Monotonic Scheduling Algorithm: Exact Characterization And Average Case Behavior", Real-Time Systems Symposium, Dec. 1989.

[7] J.Y.-T. Leung, M.L. Merrill "A note on preemptive scheduling of periodic, real-time tasks", Information Proceedings Letters,11(3),page 115-118, Nov. 1980.

[8] J. Y.-T. Leung, J. Whitehead "On the complexity of fixed-priority scheduling of periodic, real-time tasks. Performance Evaluation 2, page 237-250, 1982.

[9] C.L. Liu, J. Layland "Scheduling Algorithms for Multiprogramming in a Hard Real Time Environment", Proceeding of IEEE Real-Time Systems Symposium, page 252-260, 1973.

[10] G.K. Manacher "Production and Stabilization of Real-time Task Schedules", J. ACM14'3(July 1967), page 439-465.

[11] W. Gautschi "An Introduction to Numerical Analysis", page 74-105.

[12] S. M. Ross "Introduction to Probability Models", 4th edition, 1989.

[13] T.-S. Tia, *et. al.* "Probabilistic Performance Guarantee for Real-Time Tasks with Varying Computation Times", Proceeding of Real-Time Technology and Applications Symposium, page 164-173, May 1995. IEEE.

[14] W. Wolf "Hardware-software co-design of embedded systems", Proceeding of the IEEE, vol 82, No7,page 967-989, July 1994.