Schedulability-Driven Performance Analysis of Multiple Mode Embedded Real-Time Systems

Youngsoo Shin‡, Daehong Kim§, and Kiyoung Choi§
‡Center for Collaborative Research and Institute of Industrial Science,
University of Tokyo, Tokyo 106-8558, Japan
§School of Electrical Engineering, Seoul National University, Seoul 151-742, Korea

Abstract

Providing multiple modes to support dynamically changing environments, standards, and new services is prevalent in embedded systems, especially in mobile radio systems. Because such a system frequently contains time-constrained tasks, it is important to analyze the temporal requirements as well as the functional correctness. This paper presents a method to analyze temporal requirements imposed on an embedded real-time system supporting multiple modes. While most performance analysis methods focus only on testing the feasibility of a task or a system, our method goes further by addressing the problem of locating hot spots of a system thereby helping the designer to choose among alternative designs or architectures. We formally define the analysis problem and show that it is very unlikely to be solved efficiently. We present a heuristic algorithm, which is accurate and fast enough to be used in iterative processes in system-level analysis and design. The analysis problem is extended to accommodate probabilistic behavior exhibited by soft real-time tasks.

1 Introduction

One way to cope with dynamically changing environments, standards, and future technologies is to introduce reconfigurability into a system. While reconfigurability can be obtained by introducing reconfigurable devices such as FPGAs, recent high performance microprocessors and DSPs have made the software approach increasingly viable alternative for implementing most parts of a complex embedded system, thereby enabling reconfigurability of a system via software. Mobile telephony [1] is one of such applications that might benefit from software reconfigurability because it faces with multiple and rapidly changing standards, new signal processing algorithms, and new services. As an example, a mobile user might reconfigure a terminal, which is configured to process only still images at one time, to handle motion images through downloading MPEG program over the air interface [2]. A set of tasks consisting of the system is altered by introducing MPEG application. The result is that a system operates in multiply defined configuration status or *mode*.

Because such a system, which we call *multimode system* in this paper, may contain hard real-time tasks such as a control program as well as soft real-time tasks such as multimedia applications, it is crucial to analyze the temporal requirements of the system as well as the functional correctness. In this paper, we propose a schedulability-driven performance analysis method for multimode systems. Compared to most conventional real-time analysis [3], our method differs in two respects. First, the conventional methods focus on the analysis for a fixed set of tasks, thus for a single mode of operation, whereas our method targets the analysis *over a set of modes*. Second, most real-time schedulability analysis makes a decision on whether or not a set of tasks can be scheduled on a processor or processors while satisfying all the timing constraints. However, it cannot do much when the task set is found to be unschedulable. This is the case where a lot of engineering approaches, including hand-crafted code tuning, reimplementing core routines, and experimenting with various timing parameters, are repeated until all the constraints are satisfied. During these time-consuming iterative processes, designers are often confronted with queries like: *Does reducing the execution time of task A improve the schedulability? If so, how much is necessary? Can the same amount of schedulability improvement be achieved by reducing the execution time of other tasks with smaller amount or less effort?* Because the amount of such optimization directly relates to designer's effort, it should be minimized. Our method automatically selects tasks and computes the amount of optimization such that it is minimized.

Figure 1 shows an example of flow of design methodology for multimode systems. The first step adheres to the conventional top-down design methodology for real-time systems. In the figure, we show the steps adopted in Design Approach for Real-Time Systems (DARTS) [4]. The system is specified by hierarchically structured data flow/control flow diagrams. It is then structured into tasks according to the task structuring criteria such as sequential, temporal, and functional cohesion criteria. Task interfaces are then defined by analyzing the tasks. Detailed design of each task follows. In the second step, we define modes under which the system operates. Each mode is represented as a fixed set of tasks, which are defined in the first step. In the third step, we analyze performance of the system from the definition of modes, timing constraints imposed on the system, and timing parameters of each task extracted from the first step. If the analysis fails, that is, one or more tasks fail to satisfy their deadlines or quality of service (QoS) requirements, we perform the detailed analysis to reveal the impact caused by each task to the performance of each mode. The analysis result suggests a designer an optimization *goal* to which design effort should be devoted. This can be a valuable information because it saves the designer from spending time to optimize tasks which have no effect or potentially little effect on the system performance. The optimization step follows by adopting various methods such as task transformation [5], aperiodic event transformation [6], or hardware-software codesign.

The remainder of the paper is organized as follows. In the next section, we briefly review related work, which focus on the performance analysis of embedded real-time systems. In section 3, we introduce the underlying model of the performance analysis, the definition of the analysis problem, and our strategy to the analy-

DAC 2000, Los Angeles, California

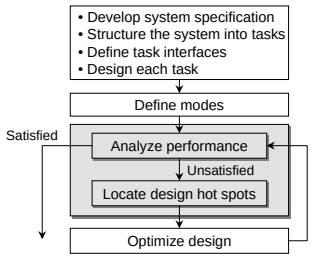

Figure 1: An example of flow of design methodology for multimode systems.

sis problem. In section 4, we discuss extensions to our strategy to the analysis problem to incorporate soft real-time tasks with QoS requirements and probabilistic execution times and present experimental results for a number of real-time system examples in section 5. We draw conclusions in section 6.

2 Related Work

Schedulability analysis for mode changes is proposed [7] for a flexible real-time system, which is similar to our definition of multimode system. Because the task set in old mode and that in new mode mutually interfere during a mode change, the method focuses on the analysis to verify whether tasks during a mode change are schedulable. A performance estimation for audio and video applications, usually called continuous media (CM) applications, is presented [8] for a networked embedded system. Because such applications are frequently associated with soft real-time constraints and probabilistic execution time variations, a stochastic approach is proposed to estimate a delay of each application and a probability that each application misses the specified deadline. While most performance analysis focus on the feasibility test for each task, a test for a system, in other words a test considering all tasks together, is proposed [9]. Each task is associated with period, deadline, and a probability distribution of execution time. A feasibility probability of a system is calculated from a probability that each task meets its deadline.

All these methods as well as classical real-time analysis focus on the determination of the schedulability only and do not provide any solutions in case when the system fails to satisfy given timing constraints, which is the main ingredient of our approach.

3 Performance Analysis Problem

3.1 Model of Multimode System

The task model of multimode system we use in this paper builds upon a widely-used real-time process model [10]. The extended task model (a mix of hard real-time and soft real-time task) to accommodate probabilistic behavior exhibited by soft real-time tasks, the definition of corresponding analysis problem, and the solution strategy are all presented in the next section.

A multimode system is defined as a set of fixed number of *modes*. Each mode is defined again as a set of fixed number of tasks, which are members of a set of tasks, denoted by τ. In other words, a mode is a subset of τ possibly with intersection with other modes. For a scheduling of each mode of multimode system, we assume, in this paper, a rate monotonic scheduling (RMS) [10] on a single processor, which is proven to be optimal fixed priority scheduling. Fixed priority scheduling has several advantages over other scheduling schemes. It is quite simple to implement in most kernels. Also, many analytical methods are available to determine whether a system is schedulable or not. However, it is not difficult to extend our approach to other scheduling methods [11] or to take into account other factors such as task dependencies, scheduler overhead [12], and access to shared resources [13] by adopting corresponding analysis methods.

Table 1: An example of task set

	T_i	D_i	C_i
τ_1	10	10	4
τ_2	16	16	8
τ_3	25	25	10
τ_4	50	50	15
τ_5	50	50	13

The set of tasks $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ is an ordered set of periodic or sporadic tasks in the descending order of priority, i.e. the priority of τ_i is higher than that of τ_j, provided that $i < j$. The parameters of each task include its period (the minimum inter-arrival time between successive requests in case of a sporadic task) T_i, deadline D_i, worst case execution time (WCET) C_i, and the maximally reducible execution time, that is, the amount of execution time that can be reduced by some kind of optimization, mre_i[1].

Example 1 *Consider an example of hard real-time task set as shown in Table 1. Rate monotonic priority assignment is a natural choice because periods are equal to deadlines. Priorities are assigned in row order in the table, that is, τ_1 has the highest priority and τ_5 has the lowest priority. Assume that we have two video applications, τ_1 and τ_4, two audio applications, τ_2 and τ_5, and a control task managing the entire system, τ_3*[2]*. Assume that we have 5 modes as given by*

Π_1		τ_2	τ_3		
Π_2		τ_2	τ_3	τ_4	
Π_3	τ_1		τ_3		τ_5
Π_4	τ_1	τ_2	τ_3		
Π_5			τ_3	τ_4	τ_5

3.2 CTA Problem

As shown in Figure 1, if the performance analysis fails, we perform the detailed analysis to reveal the impact caused by each task to the performance of each mode. We formally define the problem of this step, which we call *cut off time assignment* (CTA) problem, as the problem of computing the amount of execution time to be cut off (through optimization) from each task such that all tasks are ensured to meet their deadlines in all modes. Formally, an instance of the CTA problem is a 2-tuple (τ, Π), where

- $\Pi = \{\Pi_i | \Pi_i \subseteq \tau\}$ specifies the set of modes of a system. Each mode (Π_i) is an ordered (with respect to priorities) subset of tasks. Π_i is said to be *feasible* if and only if all of its tasks can be scheduled while their deadlines are met.

A solution to the instance of the CTA problem is a function $f : \tau \to R_0^+$, where R_0^+ denotes the set of non-negative reals, such that

[1] mre_i can be used in various ways in our methods. We can assign 0 to mre_i if τ_i has no room for further optimization. In case a degree of optimization is hard to estimate in advance, we can assign the same value of C_i to mre_i (thus make an assumption that τ_i can be optimized such that its execution time becomes 0), and then iterate the process as depicted in Figure 1.

[2] This may not be realistic assumption because video and audio tasks are frequently regarded as soft real-time tasks. The assumption is only for the illustrative purpose.

- $f(\tau_i) \leq mre_i, \forall \tau_i \in \tau$.
- Π_i is feasible, $\forall \Pi_i \in \Pi$, when C_i is reduced by $f(\tau_i), \forall \tau_i \in \tau$.

The *cost* of the solution is $\sum_{\forall \tau_i \in \tau} f(\tau_i)$. Obviously the definition of the cost may not be accurate in terms of actual cost. For example, it is not directly proportional to the hardware area when we are to implement parts of tasks with hardware to reduce the execution time of each task by $f(\tau_i)$. However, it is sufficiently informative to help the designer to choose among alternative and functionally equivalent implementations. A solution is called *optimal* if and only if it has minimum cost.

3.3 Analysis of a System with Single Mode

In this subsection, we review the performance analysis for a single set of hard real-time tasks [14], on which our approach is based. The schedulability analysis for RMS is based on the *critical instant theorem* [10] which says that if a task meets its deadline whenever the task is requested simultaneously with requests for all higher priority tasks, then the deadline will always be met for all task phasings. This implies that we need to perform the analysis from time 0 up to the least common multiple of all task periods under the assumption that all tasks are requested simultaneously at time 0. This again requires the analysis to be performed in the continuous time interval. Lehoczky et al. [15] showed that we actually need the analysis only at discrete time points instead of continuous time interval. The set of time points, called *scheduling points*, for task τ_i is defined by

$$S_i = \{kT_j | \forall \tau_j \in hep(i); k = 1, \ldots, \lfloor \frac{T_i}{T_j} \rfloor \}, \quad (1)$$

where $hep(i)$ denotes the set of tasks having priority higher than or equal to that of τ_i. τ_i can be scheduled without violating its deadline, if there *exist* one or more scheduling points $t \in S_i$, which satisfy

$$\sum_{\forall \tau_j \in hep(i)} C_j \cdot \lceil \frac{t}{T_j} \rceil \leq t. \quad (2)$$

Note that the left hand side of the inequality represents the cumulative demands on the processor imposed by $hep(i)$.

We assume that elements of S_i are sorted in ascending order. We define $S_{i,j}$ as the jth element of S_i, that is, jth scheduling point of τ_i. For each task, we try to find a scheduling point that satisfies inequality (2). If we can find no such scheduling points, we compute the time deviation by which the corresponding task misses each of its scheduling points. We define $\Delta c_{i,j}$ as the time deviation of τ_i at the jth scheduling point, which is given by

$$\Delta c_{i,j} = \sum_{\forall \tau_k \in hep(i)} C_k \lceil \frac{S_{i,j}}{T_k} \rceil - S_{i,j}. \quad (3)$$

Note that $\Delta c_{i,j}$ is positive (thus violates inequality (2)) for *all* scheduling points of τ_i that is not schedulable. It indicates the amount of time which should be cut off from *somewhere* in the execution time of the tasks in order to meet the inequality at $S_{i,j}$. This can be realized by reducing the execution time of any tasks in $hep(i)$. Therefore, for each $\Delta c_{i,j}$, we compute the amount of time, denoted by d_{ijk}, which should be cut off from the execution time of $\tau_k \in hep(i)$, given by

$$d_{ijk} = \frac{\Delta c_{i,j}}{\lceil \frac{S_{i,j}}{T_k} \rceil}. \quad (4)$$

In equation (4), the denominator, which is the number of invocations of τ_k during the interval of $S_{i,j}$, increases as T_k decreases. Therefore, d_{ijk} decreases with T_k. This leads us to the following theorem.

Theorem 1 *For a task τ_i that is not schedulable, d_{ijk} is minimum at such k that minimizes T_k.*

By Theorem 1, we select a task having the highest priority (thus having the lowest period), which has minimum value of d_{ijk}, as the candidate for execution time reduction. Let the task to be denoted by $\tau_{\hat{k}}$. We minimize $d_{ij\hat{k}}$ over j, thus obtain $\min_j d_{ij\hat{k}}$, because τ_i is schedulable if there *exists* a scheduling point that satisfies inequality (2). This process is carried out for all tasks which miss their deadlines. Then, we maximize $\min_j d_{ij\hat{k}}$ over i to obtain the minimum required time by which the execution time of $\tau_{\hat{k}}$ must be reduced in order to make all tasks schedulable. Because the time to be taken off from $\tau_{\hat{k}}$ is bounded by $mre_{\hat{k}}$, we take the minimum between $\max_i[\min_j d_{ij\hat{k}}]$ and $mre_{\hat{k}}$. For an arbitrary task τ_k, the amount of time that is to be cut off from C_k is given by

$$D_k = \min\left(\max_i[\min_j d_{ijk}], mre_k\right). \quad (5)$$

If mre_k is taken as D_k, meaning that it is not sufficient to reduce the execution time of τ_k in order to make the task set feasible, we iterate the above steps with the task having the next highest priority until the task set becomes feasible. The above process is illustrated in the following example.

Example 2 *Consider Π_2 in Example 1. From equation (1), we can find scheduling points for each task, which yields*

$$S_2 = \{T_2\},\ S_3 = \{T_2, T_3\},\ S_4 = \{T_2, T_3, 2T_2, 3T_2, T_4\}.$$

It can be shown that τ_2 is schedulable whereas τ_3 and τ_4 do not. Thus we compute D_2 with help of the following table, which yields

$$\begin{aligned} D_2 &= \min(\max[\min_j d_{3j2}, \min_j d_{4j2}], mre_2) \\ &= \min(\max[0.5, 3.7], mre_2) \\ &= \min(3.7, mre_2) \\ &= 3.7, \end{aligned}$$

if mre_2 is larger than or equal to 3.7.

τ_3	$\Delta c_{3,1} = 2$	$d_{312} = 2$	$d_{313} = 2$	
	$\Delta c_{3,2} = 1$	$d_{322} = 0.5$	$d_{323} = 1$	
τ_4	$\Delta c_{4,1} = 17$	$d_{412} = 17$	$d_{413} = 17$	$d_{414} = 17$
	$\Delta c_{4,2} = 16$	$d_{422} = 8$	$d_{423} = 16$	$d_{424} = 16$
	$\Delta c_{4,3} = 19$	$d_{432} = 9.5$	$d_{433} = 9.5$	$d_{434} = 19$
	$\Delta c_{4,4} = 11$	$d_{442} = 3.7$	$d_{443} = 5.5$	$d_{444} = 11$
	$\Delta c_{4,5} = 17$	$d_{452} = 4.3$	$d_{453} = 8.5$	$d_{454} = 17$

3.4 Analysis of a System with Multiple Modes

When we have one or more task sets (modes), the analysis becomes complicated because reducing C_k by the amount of D_k which is obtained considering only one mode, affects the feasibility of other modes to which τ_k belongs. That is, the optimality of D_k in one mode is not guaranteed across multiple modes. To obtain an optimal solution (a solution with minimum cost) to a given instance of the CTA problem, it can be shown that we should search all the possible *path*s of the problem instance in the worst case.

To illustrate the meaning of the path, consider modes Π_2, Π_3, and Π_4 in Example 1, which is repeated in Figure 2. If we consider only Π_2, the minimal cost is obtained when we compute sequentially (D_2, D_3, D_4) until Π_2 becomes feasible. However, the situation is quite different when we consider all three modes together. Reducing C_2 is not the best choice for Π_4 (see Theorem 1). Furthermore, it brings about no benefit to Π_3 because Π_3 does not

$$\begin{array}{llll} \Pi_2 & & \tau_2 & \tau_3 \quad \tau_4 \\ \Pi_3 & \tau_1 & & \tau_3 \qquad \tau_5 \\ \Pi_4 & \tau_1 & \tau_2 & \tau_3 \end{array}$$

$$(D_2(\Pi_2) = 3.7, D_1(\Pi_3) = 0.6, D_1(\Pi_4) = 1.3)$$
$$(D_3(\Pi_3) = 1.5, D_2(\Pi_2) = 2.7, D_1(\Pi_4) = 2.1)$$
$$(D_1(\Pi_4) = 2.8, D_2(\Pi_2) = 3.7)$$
$$\vdots$$

Figure 2: Paths of an instance of CTA problem.

contain τ_2. τ_3 *may* be one of the candidates because reducing C_3 affects all three modes. As a consequence, a lot of alternatives exist in order to make all three modes feasible. Three of them are shown in Figure 2. Consider the first one. For Π_2, if the execution time of τ_2 is reduced by 3.7, it becomes feasible (see Example 2) if mre_2 is assumed to be larger than or equal to 3.7. Thus, D_2 computed in a mode Π_2, denoted by $D_2(\Pi_2)$, is 3.7. Because Π_3 does not contain τ_2, it is still infeasible. Now we try to reduce the execution time of τ_1. For Π_3, $D_1(\Pi_3) = 0.6$ is required to make it feasible. While Π_4 contains both τ_1 and τ_2, it is still infeasible meaning that $(D_2(\Pi_2), D_1(\Pi_3))$ is not sufficient. Finally, $D_1(\Pi_4) = 1.3$ is needed to make Π_4 feasible and completes a path. Thus, the solution given by this path is $f(\tau_1) = 1.9, f(\tau_2) = 3.7$. That is, if the execution time of τ_1 is reduced by 1.9 and that of τ_2 is reduced by 3.7, then all three modes are guaranteed to be feasible. Similarly, we can try other paths and compute the cost. Unfortunately, however, the number of paths exponentially grows with the number of modes and the number of tasks in each mode. Furthermore, $D_i(\Pi_j)$ is not constant but depends on the previous values in the same path. In the example shown in Figure 2, $D_2(\Pi_2)$ has its values 3.7 and 2.7 in the first two paths, respectively. Consequently, the optimal solution of an instance of the CTA problem is very unlikely to be obtained efficiently.

Thus, we propose a heuristic algorithm, which is outlined in Figure 3. Before we explain the algorithm in detail, we introduce notations we use in the explanation.

- $im(i)$ denotes the set of infeasible modes to which τ_i belongs. In the example in Figure 2, $im(2) = \{\Pi_2, \Pi_4\}$.
- $D_{i,max}$ denotes the maximum D_i over all modes in $im(i)$, that is, $D_{i,max} = \max_{\forall \Pi_j \in im(i)} D_i(\Pi_j)$.
- T specifies the set of tasks whose execution time is not yet reduced.
- $hp(i)$ denotes the set of tasks with priority higher than that of τ_i.
- $\rho_{i,j}$, where $i > j$, specifies the expected number of invocations of τ_j relative to τ_i. Consider $\rho_{4,2}$ in Example 2. The scheduling points of τ_4 exhibited by τ_2 and τ_4 consist of $\{T_2, 2T_2, 3T_2, T_4\}$. The number of invocations of τ_2 at each scheduling points are 1,2,3,4 respectively while that of τ_4 is always 1 (see inequality (2)). Thus, $\rho_{4,2} = \frac{1+2+3+4}{1+1+1+1} = \frac{5}{2}$. For arbitrary τ_i and $\tau_j, i > j$, it can be shown that $\rho_{i,j} = \frac{\lceil \frac{T_i}{T_j} \rceil + 1}{2}$.

The core of the heuristic is to repeat the process of assigning a *weight* to each task and reducing the execution time of a task having the highest weight. Note that the complexity of the CTA problem stems from the fact that two factors should be taken into account in the selection of a task to reduce its execution time: its priority (recall Theorem 1) and the number of modes including the task. Thus, the weight is assigned by combining two factors together

Algorithm *Heuristic_CTA_solve*
begin
L1: Construct a set of modes Π;
L2: Reduce Π by removing $\Pi_i \subset \Pi_j, \exists \Pi_j \in \Pi, i \neq j$;
L3: Reduce Π by removing feasible modes;
L4: **for** $\alpha = 0.0 \ldots 1.0$ in step size of 0.1 **do**
L5: $\Pi' = \Pi, T = \cup_{\forall \Pi_i \in \Pi} \Pi_i$;
L6: **while** $\Pi' \neq \phi$ **do**
L7: Compute $w(i)$ for each task in T;
L8: Reduce the execution time of τ_i, which has the highest weight, by $D_{i,max}$;
L9: Remove a feasible mode from Π';
L10: $T = T - \{\tau_i\}$;
L11: **end do**
L12: Compute a $cost = \sum_{\forall \tau_i \in \tau} f(\tau_i)$ and update $min\_cost = \min(cost, min\_cost)$;
L13: Initialize C_i and mre_i for all tasks to its original value;
L14: **end do**
L15: Return min_cost and $f(\tau_i), \forall \tau_i \in \tau$;
end

Figure 3: Pseudo code of the heuristic algorithm.

such that it reflects as accurately as possible the relative *importance* in making all modes feasible. The weight of τ_i, denoted by $w(i)$, is given by

$$w(i) = \begin{cases} \alpha \left(\frac{\sum_{\forall \tau_j \in hp(i) \cap T} \rho_{i,j}}{|hp(i) \cap T|} \right)^{-1} + \beta \frac{|im(i)|}{|\Pi|}, & \text{if } hp(i) \cap T \neq \phi \\ \alpha + \beta \frac{|im(i)|}{|\Pi|}, & \text{otherwise} \end{cases} \quad (6)$$

where α and β are constant weighting factors. The first term is the inverse of the average of $\rho_{i,j}$. That is, it reflects the amount of relative effect, which is normalized with respect to a task of the highest priority, obtained when the same amount of the execution time is reduced from each task. In the example of τ_3 in Figure 2, it computes to yield $\frac{4}{7}$ meaning that reducing the execution time of τ_3 brings about roughly $\frac{4}{7}$ effects compared to reducing the same amount from τ_1 or τ_2. The second term regards the number of infeasible modes which are affected when the execution time of τ_i is reduced. It is noted that the effect of reducing the execution time of a task increases as the number of infeasible modes including it increases.

The algorithm works as follows. First, we reduce the problem size by removing modes that are subsets of some other modes (L2). This is based on the following theorem. The proof can be found in the Appendix.

Theorem 2 *If Π_i is a subset of a feasible task set Π_j, then Π_i is also feasible.*

The problem size is further reduced by removing already feasible modes (L3). For each task involved in one or more infeasible modes, we compute a weight following the equation (6) (L7). The execution time of τ_i having the highest weight is reduced by $D_{i,max}$ (L8). This may make some of modes feasible and those modes are removed (L9). τ_i is also removed from the future consideration (L10). The process is repeated until all modes are made feasible (L6). The cost obtained by the heuristic depends on the values of α and β, which control the relative importance of two factors in equation (6). Unfortunately, however, extensive experiments reveal that the values of α and β, which yield the minimum cost, depends on the application. In our heuristic, β is set to $1 - \alpha$ and we repeat the process for a range of α (L4) and select the best one.

Example 3 *Consider Π in Example 1. mre_i is assumed to be 70% of C_i for all tasks. Π_2, Π_3, and Π_4 remain after the number of*

modes are reduced (L2 and L3) as shown in Figure 2. For example of $\alpha = 0.5$*, we compute a weight for each task:* $w(1) = 0.83, w(2) = 0.67, w(3) = 0.79, w(4) = 0.38, w(5) = 0.42$*. Because* τ_1 *has the highest weight, we reduce* C_1 *by* $D_{1,max} = 2.80$*. This makes* Π_3 *feasible. We remove* Π_3 *and* τ_1 *from* Π' *and* T*, respectively. We compute weights for the remaining tasks:* $w(2) = 1.00, w(3) = 0.83, w(4) = 0.50$*. Reducing* C_2 *by* $D_{2,max} = 3.67$ *makes the remaining modes feasible and completes the process.*

4 Extensions to Performance Analysis

In this section, we present extensions to performance analysis to accommodate soft real-time tasks. Due to space limitations, we informally show the basic ideas instead of full description. While a deadline miss in hard real-time tasks leads to a system malfunction or a catastrophe, that in soft real-time tasks is generally regarded as a performance degradation. For example, a video application has timing requirements which arise due to a need for smooth display and a synchronization with audio. Unlike hard real-time tasks, tardy results only decrease the quality of video rather than lead to a malfunction. Thus, for each soft real-time task, QoS requirement $QoS(\tau_i)$[3] is defined as well as its period (T_i) and soft deadline (D_i). Regarding the execution time, we assume that each soft real-time task is associated with a random variable X_i and its probability density function (PDF), denoted by $p_{X_i}(x)$, instead of WCET. This is because soft real-time tasks such as multimedia applications usually have highly variable execution times due to the difference in the amounts of input data to be processed at each moment, data dependent execution, and so on [16]. Although a characterization of distribution for a given application is in itself a difficult work, we assume that it is given through various methods such as static analysis, profiling, or measurement. Note that the model described above is more general than the one in the previous section in that it can also model a hard real-time task by assuming 1 for QoS requirement and Dirac delta function with center located at WCET and with height of 1 for PDF.

The performance analysis method presented in the previous section becomes rather complicated when both hard real-time and soft real-time tasks co-exist because of a statistical characterization of soft real-time tasks. Specifically, the result of the summation in the left hand side of inequality (2) is not constant but a new random variable. Thus, to analyze a schedulability we compute a PDF of the new random variable, which is a convolution of PDFs of corresponding random variables summed up[4] [17].

We illustrate how the steps in the analysis in section 3 is modified when soft real-time tasks are involved using an example. Consider an example of soft real-time task set as shown in Table 2 and PDF of each task as shown in Figure 4(a). We assume that priorities are assigned rate monotonically. It can be shown that QoS requirements of both τ_1 and τ_2 are satisfied. For τ_3 we perform the analysis at each scheduling point: 10, 16, 20, and 25 (see equation (1)). As an example at 25, the PDF associated with the left hand side of inequality (2) results from the convolution of 6 PDFs; $p_{X_1} \otimes p_{X_1} \otimes p_{X_1} \otimes p_{X_2} \otimes p_{X_2} \otimes p_{X_3}$. Figure 4(b) shows graphically the result of convolution. The probability that the overall computational requirements are less than or equal to 25 (the probability that the deadline at 25 is satisfied) is the sum of probabilities below 25 at PDF, which yields 72%. Similarly for scheduling points at 10, 16, and 20, the probability computes to 0.4%, 33%, and 35%, respectively.

[3] We define $QoS(\tau_i)$ as the probability that an arbitrarily chosen instance of τ_i will meet its deadline (D_i).

[4] If Y is a sum of statistically independent random variables X_1 and X_2 ($Y = X_1 + X_2$), its PDF is a convolution of PDFs of X_1 and X_2. That is, $p_Y(y) = \sum_{x=-\infty}^{\infty} p_{X_1}(y-x) p_{X_2}(x) \triangleq p_{X_1}(y) \otimes p_{X_2}(y)$.

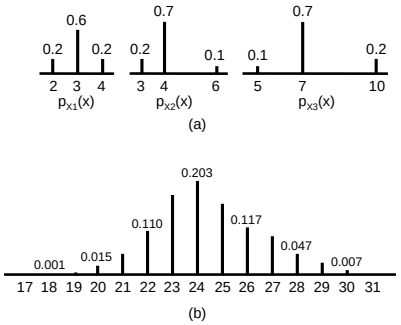

Figure 4: (a) PDF of each task and (b) Result of convolution at time 25.

Table 2: An example of soft real-time task set

	T	D	QoS
τ_1	10	10	90%
τ_2	16	16	80%
τ_3	25	25	80%

Because QoS requirement (80%) is not satisfied at any scheduling points, we compute $\Delta c_{i,j}$ ($\Delta c_{3,j}$ in this example) at each scheduling point. It is defined by

$$\Delta c_{i,j} = y_{min} - S_{i,j}, \tag{7}$$

where y_{min} is the minimum of y satisfying $Prob(Y \leq y) \geq QoS(\tau_i)$. Y is sum of random variables in the left hand side of inequality (2) at $S_{i,j}$. In the example, $y_{min} = 26$ and $\Delta c_{3,4} = 1$. Note that in this formulation we assume that the execution time of each task can be reduced uniformally along its PDF. As an example of $p_{X_1}(x)$ in Figure 4(a), reducing the execution time of τ_1 by 1 means shifting $p_{X_1}(x)$ left by 1. This corresponds to a kind of optimization which optimize portions of a task not in the conditional path of the task thereby reducing the execution time uniformally at all cases. The remaining analysis steps (equation (4), (5) and steps in subsection 3.4) are the same as those of the previous section.

5 Experiments

We have implemented our heuristic algorithm as well as an exhaustive search algorithm, which searches all the possible paths as outlined in subsection 3.4 with proper pruning. An example multimode system is constructed based on continuous media (CM) applications [8], which include two audio tasks (`PCM` and `LPC`) and two video tasks (`v7.5fps` and `v15fps`, which differ in frame rates), and 4 DECT (Digital Enhanced Cordless Telephone) protocol tasks (`RV, SLI, ISR2.1, ISR2.2`) [18]. We assume that protocol tasks are always present in the system. Audio tasks are assumed to be reconfigurable meaning that it is not allowed for two tasks to be present simultaneously in the system. Video tasks are assumed similarly. Consequently, there are 9 modes based on the combination of CM applications. Because the statistics of the actual execution times of the tasks are not available, we assume that CM applications as well as the protocol tasks behave as hard real-time tasks. For all tasks, we assume that mre_i is 70% of C_i. The cost of 1294.1 (f(`RV`) = 140.0, f(`PCM`) = 75.3, f(`LPC`) = 1023.4, f(`v15fps`) = 55.4) is obtained with heuristic algorithm in 2.1 s (on Ultra 1), which is only slightly larger than the minimum cost of 1280.2 (f(`RV`) = 140.0, f(`PCM`) = 61.4, f(`LPC`) = 1023.4, f(`v15fps`) = 55.4) obtained in 6.5 h.

To further evaluate the heuristic algorithm, we build 17 synthetic benchmarks (the most simplest one with 3 tasks and 3 modes and the most complex one with 17 tasks and 128 modes) and perform the experiment. The results reveal that the cost obtained with the heuristic is only larger than the minimum cost, which is either obtained by exhaustive search or by simulated annealing[5], by 6.6% on the average. Consequently, in view of accuracy and speed, the proposed heuristic algorithm seems to be appropriate in system-level design as shown in Figure 1, where a lot of iterative processes may be required to reach the final implementation.

6 Conclusion

We study a schedulability-driven performance analysis method for a multimode system, which can change its mode by reconfiguration. While most performance analysis methods only *test* a feasibility of each task or a system, our method goes further by locating hot spot of a system to which design efforts should be devoted thereby helping the designer to choose among alternative designs or architectures. It also saves design cost by helping designer to avoid unnecessary optimization of tasks which have no effect or potentially little effect on the overall system performance. We formulate the analysis problem into the CTA problem and show that it is very unlikely to be solved efficiently. The extensions to the CTA problem is presented to accommodate probabilistic behavior exhibited by soft real-time tasks. We present a heuristic algorithm, which is accurate and fast enough to be used in iterative processes in system-level analysis and design. As future work, we plan to apply the proposed methodology to real industrial example.

Appendix

Proof of Theorem 2 For any task $\tau_k \in \Pi_i \cap \Pi_j$, its scheduling points in each task set satisfies

$$S_k(\Pi_i) \subseteq S_k(\Pi_j), \tag{8}$$

where $S_k(\Pi_i)$ is a set of scheduling points of τ_k in Π_i. Because τ_k is schedulable in Π_j, it satisfies inequality (2) at one or more scheduling points $t \in S_k(\Pi_j)$. There are two cases. First, if those scheduling points are also in $S_k(\Pi_i)$, then τ_k is also schedulable in Π_i. Consider the second case when τ_k in Π_j satisfies inequality (2) at one or more scheduling points $t \in S_k(\Pi_j)$ but $t \notin S_k(\Pi_i)$. Assume that t_1 and t_3 is consecutive elements of $S_k(\Pi_i)$ and that t_1, t_2, and t_3 is consecutive elements of $S_k(\Pi_j)$. Let A be the cumulative computational demands on the processor (left hand side of inequality (2)) by tasks in $\Pi_j - \Pi_i$ at t_2 and B be those by tasks in $\Pi_j \cap \Pi_i$. Following shows the case.

	Π_i		Π_j
		t_1	t_1
τ_k			$A + B \leq t_2$
	B	t_3	$A' + B > t_3$

B does not change at t_3 because t_2 is some integer multiple of period of a task in $\Pi_j - \Pi_i$. Note that in the left hand side of inequality (2), the number of invocations of τ_j ($\lceil \frac{t}{T_j} \rceil$) at two consecutive scheduling points, t_i and t_j, differs only if t_i is some integer multiple of T_j. It follows then that

$$B \leq t_2 - A < t_3, \tag{9}$$

because $t2 < t3$. Thus, τ_k in Π_i satisfies inequality (2) at t_3 and is schedulable. Because $\tau_k, \forall \tau_k \in \Pi_i$ is schedulable in both case, Π_i is feasible. □

[5] We have also implemented a simulated annealing-based heuristic to obtain a possibly near-optimal results for examples for which we cannot obtain the costs in reasonable time by exhaustive search.

References

[1] J. Mitola, "The software radio architecture," *IEEE Communications Magizine*, pp. 26–38, May 1995.

[2] W. Tuttlebee, "The impact of software radio," in *Proc. Software Radio Workshop*, May 1997.

[3] C. Locke, "Software architecture for hard real-time applications: cyclic executives vs. fixed priority executives," *The Journal of Real-Time Systems*, vol. 4, pp. 37–53, Mar. 1992.

[4] H. Gomaa, *Software Design Methods for Concurrent and Real-Time Systems*. Reading, MA: Addison-Wesley Publishing Company, Inc., 1993.

[5] R. Gerber and S. Hong, "Semantics-based compiler transformations for enhanced schedulability," in *Proc. IEEE Real-Time Systems Symposium*, Dec. 1993.

[6] G. Arora and D. Stewart, "AFTER: A CASE tool to assist in fine-tuning of embedded real-time systems," in *Proc. IEEE Real-Time Systems Symposium*, Dec. 1996.

[7] P. Pedro and A. Burns, "Schedulability analysis for mode changes in flexible real-time systems," in *Proc. Euromicro Workshop on Real-Time Systems*, pp. 17–19, June 1998.

[8] A. Kalavade and P. Moghé, "A tool for performance estimation of networked embedded end-systems," in *Proc. Design Automat. Conf.*, pp. 257–262, June 1998.

[9] T. Zhou, X. Hu, and E. Sha, "A probabilistic performance metric for real-time system design." to appear in Proc. Int'l Workshop on Hardware/Software Co-Design, May 1999.

[10] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard real time environment," *J. ACM*, vol. 20, pp. 46–61, Jan. 1973.

[11] A. Atlas and A. Bestavros, "Statistical rate monotonic scheduling," in *Proc. IEEE Real-Time Systems Symposium*, pp. 123–132, Dec. 1998.

[12] D. Katcher, H. Arakawa, and J. Strosnider, "Engineering and analysis of fixed priority schedulers," *IEEE Trans. on Software Eng.*, vol. 19, pp. 920–934, Sept. 1993.

[13] L. Sha, R. Rajkumar, and J. P. Lehoczky, "Priority inheritance protocols: An approach to real-time synchronization," *IEEE Trans. on Computers*, vol. 39, pp. 1175–1185, Sept. 1990.

[14] Y. Shin and K. Choi, "Enforcing schedulability of multi-task systems by hardware-software codesign," in *Proc. Int'l Workshop on Hardware/Software Co-Design*, pp. 3–7, Mar. 1997.

[15] J. Lehoczky, L. Sha, and Y. Ding, "The rate monotonic scheduling algorithm: Exact characterization and average case behavior," in *Proc. IEEE Real-Time Systems Symposium*, pp. 166–171, Dec. 1989.

[16] L. Abeni and G. Buttazzo, "Integrating multimedia applications in hard real-time systems," in *Proc. IEEE Real-Time Systems Symposium*, pp. 4–13, Dec. 1998.

[17] T. Tia, Z. Deng, M. Shankar, M. Storch, J. Sun, L. Wu, and J. Liu, "Probabilistic performance guarantee for real-time tasks with varying computation times," in *Proc. IEEE Real-Time Technology and Applications Symposium*, pp. 164–173, May 1995.

[18] F. Slomka, J. Zant, and L. Lambert, "Schedulability analysis of heterogeneous systems for performance message sequence chart," in *Proc. Int'l Workshop on Hardware/Software Co-Design*, pp. 91–95, Mar. 1998.