Journal of Embedded Computing 1 (2005) 57–71
IOS Press

Deterministic and statistical admission control for QoS-Aware embedded systems

Jungkeun Park[a,*], Minsoo Ryu[b] and Seongsoo Hong[a]
[a]*School of Electrical Engineering and Computer Science, Seoul National University, Seoul 151-742, Korea*
[b]*College of Information and Communications, Hanyang University, Seoul 133-791, Korea*

Abstract. This paper presents two classes of admission control schemes for embedded system applications that have real-time constraints but with composite characteristics in request arrivals and resource requirements. First, we present admission control tests using *utilization demands* to handle a mix of periodic and aperiodic tasks with deterministic execution times. The utilization demand is defined as the processor utilization required for a task to meet its deadline with certainty, thus for deterministic deadline guarantees. We show that the use of utilization demands eliminates the need for complicated schedulability analysis and enables on-line admission control. Second, we present statistical admission control schemes using *effective execution times* to handle stochastic execution times. Effective execution times are determined from the specified probability demanded by the application and stochastic properties of task execution times. Every task is associated with an effective execution time and is restricted to using processor time not exceeding its effective execution time. This scheme allows every task to meet its deadline with the specified probability without being interfered with, and greatly simplifies the admission control when combined with utilization demands. Our experimental results confirm the correctness of the proposed approaches. They also show that our admission tests are very accurate to achieve high-level processor utilization.

Keywords: Admission control, deterministic, statistical, schedulability analysis, utilization demands, effective execution times

1. Introduction

Meeting QoS (Quality of Service) demands on embedded systems is a key issue for future CPU and network intensive applications. Such applications can be found in networked QoS applications including multimedia-supporting handheld devices, which require real-time data delivery and processing with limited resources. However, current hard real-time technologies cannot be directly applied to those QoS applications since most real-time research has put an emphasis on the deterministic task model [1–5] in which task arrivals are periodic and execution times are bounded. On the other hand, many networked QoS applications have two distinguishing characteristics. First, processor usage patterns include both periodic and aperiodic tasks. For example, a query for continuous multimedia involves periodic tasks for delivery and processing of continuous data, and a query for static data types involves aperiodic tasks. Second, task execution times are either deterministic or stochastic, such as CBR (constant bit rate) video data versus VBR (variable bit rate) data.

In this paper, we attempt to provide deadline guarantees via admission control for real-time tasks. Such deadline guarantees can be either deterministic or statistical depending on the characteristics of task execution times. When task execution times are upper bounded and their bounds are known, deterministic deadline guarantees can be provided so that all tasks meet deadlines with certainty. The deterministic guarantee provides the highest level of deadline guarantees, however, it may be an overly conservative approach for some multimedia applications which are not greatly impacted by infrequent deadline misses. This necessitates statistical deadline guarantees. When task execution times are not bounded or exhibit great variability,

*Corresponding author. E-mail: jkpark@redwood.snu.ac.kr.

ISSN 1740-4460/05/$17.00

a statistical approach provides probabilistic deadline guarantees with a specified probability.

We present new admission control schemes for both types of deadline guarantees. First, we propose an efficient admission control test, based on *utilization demands*, to handle a mix of periodic and aperiodic tasks with deterministic execution times. The utilization demand is defined as the processor utilization required for a task to meet its deadline. We use the utilization demand to develop an admission test for deterministic deadline guarantees under EDF. We show that the use of utilization demands eliminates the need for complicated schedulability analysis and enables on-line admission control.

Second, we present two statistical schemes to ensure that tasks meet deadlines with a specified probability. The first approach uses *statistical utilization demands* which is a statistical version of utilization demands. Statistical utilization demands allow us to compute probabilities that tasks meet deadlines in the worst case, thus enabling statistical deadline guarantees for a mix of periodic and aperiodic. The shortcoming of this approach is that it leads to computationally complex algorithms since computing probabilities generally requires expensive convolution operations. The second approach improves upon the first one using *effective execution times*. Effective execution times are determined from the specified probability demanded by the application and stochastic properties of execution times. Every task is associated with an effective execution time and is restricted to using processor time not exceeding its effective execution time. If a task consumes more processor time than its effective execution time, it is immediately discarded. This approach allows every task to meet its deadline with the specified probability without being interfered with, and greatly simplifies the admission control when combined with utilization demands.

We have implemented simulators using the proposed admission control schemes and performed experiments in both deterministic and statistical settings. Our experimental results show that the proposed approaches can successfully handle a mix of periodic and aperiodic tasks as well as stochastic execution times. They also show that the proposed admission tests are very accurate to achieve high-level processor utilization.

1.1. Related work

There has been much work which has focused on handling mixed sets of periodic and aperiodic tasks [6–11]. The algorithms in [6–9] assume that aperiodic tasks are soft real-time and give preferential treatment to periodic tasks. In these approaches, aperiodic tasks are handled at a lower priority level in the background, or at a some fixed priority level by a special periodic task which serves aperiodic requests with a limited capacity. The algorithms proposed in [12,13] handle aperiodic tasks with explicit deadlines. Also, they are known to be optimal with regard to specific criteria, for example, of the response time or processor utilization. However, they not only require complete knowledge of the aperiodic tasks, but also have high computational complexities when used on-line. In our model, all aperiodic tasks have explicit deadlines and are scheduled by the same scheduling policy as periodic tasks. Moreover, our approach using utilization demands eliminates the need for complicated schedulability analysis, requiring low run-time overhead.

In the meantime, several researchers have worked on non-deterministic solutions to real-time scheduling problems with stochastic execution times. The statistical rate monotonic scheduling (SRMS) in [14] is a non-deterministic version of the classical rate monotonic scheduling. Under the assumption that the accurate execution time of a task is known when the task arrives, SRMS allows one to compute the percentage of deadline misses. Tia et al. [15] proposed two methods to handle stochastic task execution times, *probabilistic time-demand analysis* and *transform-task method*. The probabilistic time-demand analysis attempts to provide a lower bound on the probability that a periodic task meets its deadline under fixed priority scheduling. The probabilistic time-demand analysis is based on the notion of critical instant at which the first instances in all periodic tasks are released simultaneously. The critical instant leads to the worst case when all tasks complete before their deadlines, i.e., when no backlog exists. However, it has not been proven for unbounded execution times that the critical instant is the worst case. Another method, called transform-task method, divides each task into a periodic task and a sporadic task. The periodic task has the same period as the original task and has a fixed execution time that should be chosen such that all the periodic tasks in the system are schedulable. If the actual execution time of a periodic task is larger than the fixed execution time, the excessive portion of the task is modeled as a sporadic task that can be scheduled by either a sporadic server or a slack stealing algorithm.

The key idea of our approach using effective execution times is similar to that of the transform-task method

in that each task is associated with a fixed amount of execution time and its processor usage is enforced accordingly. Our contribution is to give a formal definition of effective execution times based on the notion of statistical schedulability and to combine effective execution times with utilization demands in order to provide an efficient, statistical version of admission control scheme. In fact, the use of effective execution times allows us to easily extend existing deterministic scheduling algorithms and analysis techniques to handle stochastic execution times.

The remainder of this paper is organized as follows. In Section 2, we discuss our models and assumptions. Section 3 describes schedulability tests based on utilization demands for periodic and aperiodic tasks with bounded execution times. Section 4 describes two classes of statistical approaches, one with statistical utilization demands and the other with effective execution times. Section 5 describes simulation results to evaluate the proposed approaches. Section 6 summarizes our results and discusses future research directions.

2. Models and assumptions

Consider a set of aperiodic tasks $Q = \{\tau_1, \tau_2, \ldots, \tau_i, \ldots\}$ where tasks are in arrival order, i.e., τ_i arrives earlier than τ_{i+1}. We use $Q(t) \subset Q$ to denote the current set of tasks that have been admitted by t and have not completed by t. Every aperiodic task $\tau_i \in Q$ has an arrival time A_i and a relative deadline d_i that is defined from its arrival time. The absolute deadline D_i of τ_i is then given by $D_i = A_i + d_i$. Task τ_i also has an execution time requirement. If its execution time is bounded from above, then its least upper bound is denoted by e_i. Otherwise, we use e_i^* to represent the execution time as an independent random variable. We assume that the probability density function (pdf) of e_i^* is known, and it is denoted by $g_{e_i}(e)$.

We use similar notation for periodic tasks. Periodic task $\tilde{\tau}_i$ with period $\tilde{T}_i$ can be considered as an infinite sequence of aperiodic requests. Such aperiodic requests are referred to as *periodic task instances* which are denoted by $\tilde{\tau}_{i,j}$. Each periodic task instance $\tilde{\tau}_{i,j}$ has a common relative deadline $\tilde{d}_i$ which is equal to its period, i.e., $\tilde{d}_i = \tilde{T}_i$. We use $\tilde{A}_i$ to denote the arrival time of $\tilde{\tau}_i$, and $\tilde{R}_i$ to denote the start time of the period of $\tilde{\tau}_i$ when $\tilde{\tau}_i$ is accepted. We often refer to $\tilde{R}_i$ as release time. Unless noted explicitly, we will assume that the period of $\tilde{\tau}_i$ starts immediately after $\tilde{A}_i$, i.e., $\tilde{R}_i = \tilde{A}_i$. Using release time $\tilde{R}_i$, the absolute deadline $\tilde{D}_{i,j}$ of $\tilde{\tau}_{i,j}$ is computed by $\tilde{D}_{i,j} = \tilde{R}_i + (j-1)\tilde{T}_i + \tilde{d}_i$. Finally, each periodic task instance $\tilde{\tau}_{i,j}$ has also an execution time requirement. If the execution time of $\tilde{\tau}_{i,j}$ is upper bounded for all j, then the least upper bound is denoted by $\tilde{e}_i$. Otherwise, we use $\tilde{e}_{i,j}^*$ to represent its execution time as an independent random variable. We assume that all random variables $\tilde{e}_{i,j}^*$ are identically distributed according to the same probability density function $g_{\tilde{e}_i}(e)$. Appendix IV summarizes the notation used throughout this paper.

In our discussions, we assume a simple system architecture that consists of an admission controller, admit queues, and a task scheduler, as in Fig. 1. The admission controller, through admit or reject, is responsible for ensuring that the system can provide promised deadline guarantees for all tasks accepted. Every admitted task is then put into an appropriate admit queue depending on its type. There are two separate queues, one for aperiodic tasks and the other for periodic tasks. The task scheduler then chooses the admitted task with the highest priority, and allocates processor to it. This simple architecture allows us to consider a wide variety of models for embedded system operation and configuration.

The scheduling algorithm considered here is earliest deadline first (EDF) [1]. EDF was selected for two reasons. First, EDF is known to be optimal for deterministic deadline guarantees in the sense that it can schedule any task set which is schedulable by any other algorithm. Even though optimality of EDF has not been proven in a statistical environment, it still serves as a benchmark for other scheduling algorithms. Second, the EDF algorithm allows for utilization-based schedulability tests which incur little run-time overhead. Under EDF, if the utilization of a task set does not exceed one, then the set is schedulable. In the next section, we will show that the utilization-based test and our utilization demand analysis can be combined successfully into an integrated schedulability test.

3. Deterministic admission control

In this section we introduce the notion of utilization demands and present schedulability tests for deterministic deadline guarantees. We first define utilization demands for aperiodic tasks, and derive a necessary and sufficient schedulability test for them. We then extend utilization demands for a mixed set of aperiodic and periodic tasks. Throughout this section, we assume that the execution time of every task τ_i is upper-bounded by its worst-case execution time e_i.

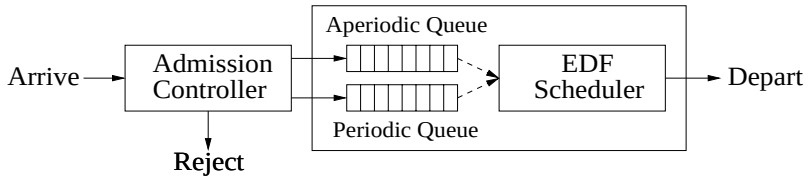

Fig. 1. System architecture composed of an admission controller, admit queues, and a task scheduler.

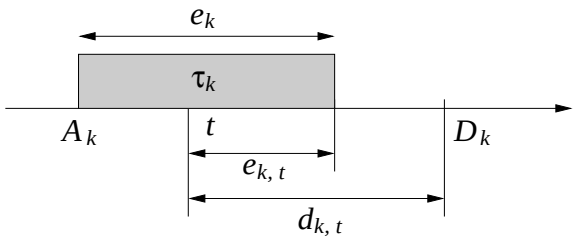

Fig. 2. Residual execution time and lead time.

3.1. Utilization demands for aperiodic tasks

Consider a set of independent aperiodic tasks $Q(t) = \{\tau_1, \tau_2, \ldots, \tau_i, \ldots\}$ processed by a work-conserving EDF scheduler. We associate each task $\tau_i \in Q(t)$ with two dynamic variables, residual execution time $e_{i,t}$ and lead time $d_{i,t}$. At time t, the residual execution time $e_{i,t}$ represents the maximum remaining processor time required to complete τ_i, and the lead time $d_{i,t}$ represents the difference between its absolute deadline D_i and the current time t [16], i.e., $D_i - t$.

Residual execution times and lead times provide us sufficient information in determining the schedulability of $\tau_i \in Q(t)$. For instance, suppose that τ_k is the highest priority task in $Q(t)$ and no other higher priority tasks will arrive until the completion of τ_k. At any time t in the interval between the arrival and completion of τ_k, we can say that τ_k is schedulable if and only if $e_{k,t} \leqslant d_{k,t}$. Figure 2 illustrates this pictorially.

We now define utilization demands using residual execution times and lead times. Formally, utilization demand $U_{Q(t)}(\tau_i)$ is defined for $\tau_i \in Q(t)$ as the processor time required to meet its deadline divided by its lead time. If the task τ_i is the highest priority task in $Q(t)$, the utilization demand $U_{Q(t)}(\tau_i)$ is simply represented by $\frac{e_{i,t}}{d_{i,t}}$. It is obvious that τ_i is schedulable if and only if $U_{Q(t)}(\tau_i) \leqslant 1$ since it has the same meaning as $e_{i,t} \leqslant d_{i,t}$ except for the case of $d_{i,t} = 0$. However, in the general case where τ_i is not the highest priority task in $Q(t)$, we need to consider its higher-priority tasks. Since τ_i can run only after the higher-priority tasks complete, $U_{Q(t)}(\tau_i)$ should also include residual execution times of higher-priority tasks in its numerator. Let $Q(t, ed(t')) \subset Q(t)$ be the set of tasks whose absolute deadlines are no later than the time t'. We define the utilization demand of τ_i by

$$U_{Q(t)}(\tau_i) \stackrel{\text{def}}{=} \frac{\sum_{\tau_j \in Q(t, ed(D_i))} e_{j,t}}{d_{i,t}}. \tag{1}$$

The following theorem shows that the schedulability analysis using Eq. (1) is a necessary and sufficient condition for an aperiodic task set.

Theorem 3.1. Aperiodic task set $Q(t)$ is schedulable by a work-conserving EDF scheduler if and only if

$$U_{Q(t)}(\tau_i) \leqslant 1 \tag{2}$$

for any $\tau_i \in Q(t)$.

Proof. We consider the "if" part first. Let f_i be the worst case finish time of $\tau_i \in Q(t)$. Since the scheduler is work-conserving, the finish time f_i will be current time plus the sum of residual execution times of higher priority tasks including τ_i's execution time. Therefore,

$$\begin{aligned} f_i &= t + \sum_{\tau_j \in Q(t, ed(D_i))} e_{j,t} \\ &= t + d_{i,t} \cdot U_{Q(t)}(\tau_i) \\ &= t + (D_i - t) \cdot U_{Q(t)}(\tau_i). \end{aligned}$$

Since $U_{Q(t)}(\tau_i) \leqslant 1$,

$$\begin{aligned} t + (D_i - t) \cdot U_{Q(t)}(\tau_i) &\leqslant t + (D_i - t) \\ &\leqslant D_i. \end{aligned}$$

Next, we consider the "only if" part. The proof is by contradiction. We assume that all the tasks in $Q(t) = \{\tau_1, \ldots, \tau_i, \ldots, \tau_n\}$ are schedulable and there exists $\tau_i \in Q(t)$ such that $U_{Q(t)}(\tau_i) > 1$. It immediately follows that

$$\begin{aligned} f_i &= t + \sum_{\tau_j \in Q(t, ed(D_i))} e_{j,t} \\ &= t + (D_i - t) \cdot U_{Q(t)}(\tau_i) \\ &> t + (D_i - t) = D_i. \end{aligned}$$

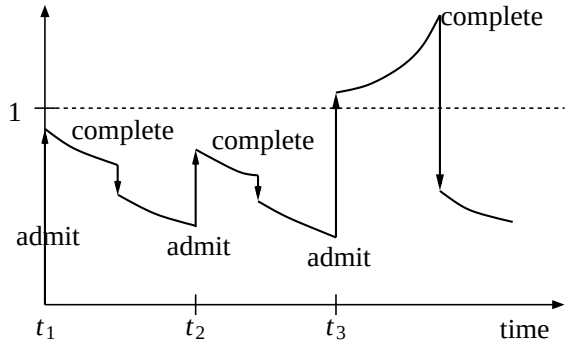

Fig. 3. Utilization demand as a function of time.

This contradicts the assumption that $Q(t)$ is schedulable. □

It is important to note that the schedulability of $\tau_i \in Q(t)$ does not imply the schedulability of $\tau_i \in Q$. This is because $Q(t)$ does not take into account future tasks that may be admitted after t. Therefore, the schedulability test Eq. (2) is valid only until the next arrival time of a new task. This necessitates testing of schedulability at every task arrival. Figure 3 illustrates the utilization demand $U_{Q(t)}(\tau_i)$ as a function of time with several admissions and completions of τ_i's higher-priority tasks. When a new higher-priority task is admitted $U_{Q(t)}(\tau_i)$ jumps up, and when a higher-priority task completes $U_{Q(t)}(\tau_i)$ jumps down. Moreover, we can see that if $U_{Q(t)}(\tau_i)$ is less than one at t, $U_{Q(t)}(\tau_i)$ decreases until the next arrival time. Otherwise, $U_{Q(t)}(\tau_i)$ can increase unboundedly. Thus, to check if $\tau_i \in Q$ is schedulable, it suffices to show $U_{Q(t)}(\tau_i) \leqslant 1$ at every arrival time until τ_i completes.

3.2. Utilization demands for periodic tasks

Consider a periodic task set $P = \{\tilde{\tau}_1, \tilde{\tau}_2, \ldots, \tilde{\tau}_N\}$. Basically, all instances of periodic tasks can be considered as aperiodic tasks. That is, a periodic task instance $\tilde{\tau}_{i,j}$ can be regarded as an aperiodic task τ_k that has arrival time $A_k = \tilde{A}_i + (j-1)\tilde{T}_i$ and deadline $d_k = \tilde{T}_i$. This interpretation gives a possibility to apply utilization demands to periodic tasks. In general, periodic set P can be associated with an aperiodic set Q_P which consists of all task instances generated by P. We refer to this aperiodic task set Q_P as *pseudo-aperiodic set* of P. Note that P is schedulable if and only if all the tasks in Q_P are schedulable. Thus, the use of Q_P allows us to treat periodic tasks and aperiodic tasks in a similar manner.

However, there is a major difference between pseudo-aperiodic set Q_P and ordinary aperiodic set Q. Pseudo-aperiodic set Q_P allows the complete knowledge of future task arrivals after current time t and admission for every such task must be guaranteed. To represent utilization demands for such pseudo-aperiodic tasks, we introduce a more general concept of utilization demand that is defined over a given time interval. Let $Q(t,t') \subset Q$ be the set of tasks that are admitted in the interval between t and t' including the tasks of $Q(t)$. We will often use $Q(t,\infty)$ to include both current tasks that have not completed by t and future tasks that will be admitted after t. We now define the utilization demand for $\tau_i \in Q(t,t')$ by

$$U_{Q(t,t')}(\tau_i) \stackrel{\text{def}}{=} \frac{\sum_{\tau_j \in Q((t,t'),ed(D_i))} e_{j,t}}{D_i - t} \tag{3}$$

where $Q((t,t'),ed(D_i)) \subset Q(t,t')$ represents the set of tasks whose absolute deadlines are no later than D_i. The above definition provides a more general schedulability condition as below.

Theorem 3.2. Aperiodic task set $Q(t,t')$ is schedulable by a work-conserving EDF scheduler if

$$U_{Q(t,t')}(\tau_i) \leqslant 1 \tag{4}$$

for any $\tau_i \in Q(t,t')$.

Theorem 3.2 is useful because of its generality since it applies to arbitrary task sets including ordinary aperiodic sets Q and pseudo-aperiodic sets Q_P. It is sufficient to let $Q(t,t') = Q(t)$ for the purpose of admission control of ordinary aperiodic tasks, thus Theorem 3.2 reduces to Theorem 3.1. On the other hand, the schedulability of pseudo-aperiodic set Q_P can be guaranteed by ensuring that $U_{Q_P(t,\infty)}(\tau_i) \leqslant 1$ for all $\tau_i \in Q_P(t,\infty)$ at any time t. This schedulability condition for Q_P can be compared to the Liu and Layland's EDF condition for P described by

$$U_P = \sum_{\tilde{\tau}_i \in P} \frac{\tilde{e}_i}{\tilde{T}_i} \leqslant 1. \tag{5}$$

In fact, we can show that $U_{Q_P(t,\infty)}(\tau_i) \leqslant U_P$ for all $\tau_i \in Q_P(t,\infty)$ at any time t if $U_P \leqslant 1$. We postpone the proof of this result to Appendix 1.

3.3. Schedulability condition for a mixed task set

We now apply utilization demands to a mixed set of periodic and aperiodic tasks. Let S be the union set of ordinary aperiodic set Q and pseudo-aperiodic set Q_P, i.e., $S = Q \cup Q_P$. Based on Theorem 3.2, we

can guarantee the schedulability of $S(t,\infty)$ by ensuring that the utilization demand $U_{S(t,\infty)}(\tau_i) \leqslant 1$ for all $\tau_i \in S(t,\infty)$ at any time t. In fact, this schedulability test is not realizable since we have to compute utilization demands for an infinite number of tasks in the unbounded set $S(t,\infty)$.

Theorem 3.2 can be refined for the mixed set S by using a *busy period* model. Specifically, the busy period model allows the utilization demand $U_{S(t,\infty)}(\tau_i)$ to be decomposed into two separate parts, an aperiodic contribution by $Q(t)$ and a periodic contribution by $Q_P(t,\infty)$. Since Eqs (4) and (5) permits schedulability tests for individual Q and Q_P, respectively, the decomposition leads to a realizable schedulability test.

A busy period is a time interval in which the processor is busy and no idle time points exist. Busy periods are often defined with respect to a given whole set S, and such busy periods are called here *level-0 busy periods*. On the other hand, we can also consider imaginary busy periods for the subsets of S. Consider a subset $S((t,\infty), ed(D_i)) \subset S$ for a given task $\tau_i \in S$. With respect to $S((t,\infty), ed(D_i))$, we can define busy periods in which only τ_i and its higher priority tasks in S are processed. We call such busy periods *level-τ_i busy periods*. Figure 4 illustrates level-0 busy periods and level-τ_i busy periods. In Fig. 4, the grey areas represent the execution of tasks whose absolute deadlines are later than D_i and the black areas represent the execution of tasks whose absolute deadlines are no later than D_i.

We choose one among level-τ_i busy periods such that the arrival time A_i of τ_i is included in the busy period. Let b_i be the beginning of the chosen one. Then, the utilization demand of $\tau_i \in S$ over the chosen busy period is given by

$$U_{S(b_i,D_i)}(\tau_i) = \frac{\sum_{\tau_j \in S((b_i,D_i),ed(D_i))} e_j}{D_i - b_i}. \quad (6)$$

In the above equation, we could use e_j instead of residual execution time $e_{j,t}$ since no tasks in $S((b_i,D_i),ed(D_i))$ could have started its execution before b_i. This also implies that all the tasks in $S((b_i,D_i),ed(D_i))$ arrive in the level-τ_i busy period.

The following lemma is an intermediate result and shows that it is sufficient to consider only busy periods for the purpose of schedulability tests.

Lemma 3.3. Given a mixed set $S = Q \cup Q_P$, $S(t,\infty)$ is schedulable by a work-conserving EDF scheduler if

$$U_{S(b_i,D_i)}(\tau_i) \leqslant 1 \quad (7)$$

for any $\tau_i \in S(t,\infty)$.

Proof. See Appendix 2. □

We proceed with $U_{S(b_i,D_i)}(\tau_i)$ in Lemma 3.3 to decompose it into a periodic contribution and an aperiodic contribution. Note that we have arbitrarily chosen τ_i from $S(t,\infty)$ and we want to guarantee the schedulability for any such one. The following decomposition is then conservatively performed to ensure the schedulability of τ_i regardless of its choice. For any $\tau_i \in S(t,\infty)$ at any time t, it follows that

$$\begin{aligned} U_{S(b_i,D_i)}(\tau_i) &= \frac{\sum_{\tau_j \in S((b_i,D_i),ed(D_i))} e_j}{D_i - b_i} \\ &= \frac{\sum_{\tau_j \in Q_P((b_i,D_i),ed(D_i))} e_j}{D_i - b_i} \quad (8) \\ &\quad + \frac{\sum_{\tau_j \in Q((b_i,D_i),ed(D_i))} e_j}{D_i - b_i} \quad (9) \\ &= U_{pc} + U_{\mathrm{ac}} \quad (10) \end{aligned}$$

where $U_{pc} = \frac{\sum_{\tau_j \in Q_P((b_i,D_i),ed(D_i))} e_j}{D_i - b_i}$ and $U_{\mathrm{ac}} = \frac{\sum_{\tau_j \in Q((b_i,D_i),ed(D_i))} e_j}{D_i - b_i}$.

We first show that the periodic contribution can be reduced into a simple form using $U_P = \sum_{\tilde{\tau}_j \in P} \frac{\tilde{e}_j}{\tilde{T}_j}$. Let τ_p be the task that has the latest absolute deadline among $Q_P((b_i,D_i),ed(D_i))$. Note again that τ_i may be in either $Q_P((b_i,D_i),ed(D_i)$ or $Q((b_i,D_i),ed(D_i)$ since we have arbitrarily chosen τ_i in $S(t,\infty)$. In the case of $\tau_i \in Q_P((b_i,D_i),ed(D_i))$, then $\tau_i = \tau_p$ and $D_i = D_p$. Otherwise, $\tau_i \neq \tau_p$ and $D_p \leqslant D_i$. In either case, we can write

$$\begin{aligned} U_{pc} &= \frac{\sum_{\tau_j \in Q_P((b_i,D_i),ed(D_i))} e_j}{D_i - b_i} \quad (11) \\ &\leqslant \frac{\sum_{\tau_j \in Q_P((b_i,D_p),ed(D_p))} e_j}{D_p - b_i} \quad (12) \end{aligned}$$

Using the periodicity of tasks in P, all the tasks of $Q_P((b_i,D_p),ed(D_p))$ can be determined for the worst case. For any $\tilde{\tau}_k \in P$, $\lfloor (D_p - b_i)/\tilde{T}_k \rfloor$ is the maximum number of its requests that arrive in the interval $[b_i, D_p]$ and have deadlines no later than D_i. Thus, it follows that

$$\begin{aligned} U_{pc} &\leqslant \frac{\sum_{\tilde{\tau}_j \in P} \lfloor (D_p - b_i)/\tilde{T}_j \rfloor \tilde{e}_j}{D_p - b_i} \quad (13) \\ &\leqslant \sum_{\tilde{\tau}_j \in P} \frac{\tilde{e}_j}{\tilde{T}_j} = U_P \quad (14) \end{aligned}$$

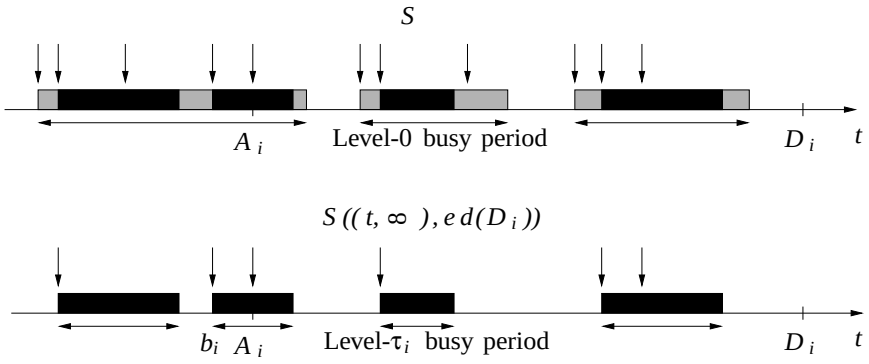

Fig. 4. Level-0 busy periods and level-τ_i busy period.

We next consider the aperiodic contribution $U_{\rm ac}$. Similarly, let τ_q be the task that has the latest absolute deadline among $Q((b_i, D_i), ed(D_i))$. It immediately follows that

$$U_{\rm ac} = \frac{\sum_{\tau_j \in Q((b_i,D_i),ed(D_i))} e_j}{D_i - b_i} \tag{15}$$

$$\leqslant \frac{\sum_{\tau_j \in Q((b_i,D_q),ed(D_q))} e_j}{D_q - b_i} = U_{Q(b_i,D_q)}(\tau_q) \tag{16}$$

The above result includes b_i which depends on the choice of τ_i from $S(t,\infty)$. We want to eliminate b_i in Eq. (17) to determine the upper bound of $U_{\rm ac}$, independent of the choice of τ_i. To do so, we use a level-0 period. Consider a level-0 period that contains the time point b_i. It is easy to see that there always exists only one such a level-0 busy period. Let $B(t)$ be the beginning of the chosen level-0 busy period. Clearly, $B(t) \leqslant b_i$ and b_i is one of arrival times of tasks that are admitted in the interval $[B(t), t]$. Thus, the upper bound of $U_{\rm ac}$ can be determined by computing $U_{Q(A_j,D_q)}(\tau_q)$ at every arrival time of $\tau_j \in S(B(t), t)$.

$$U_{\rm ac} \leqslant \max\{U_{Q(A_j,D_q)}(\tau_q) || \tau_j \in S(B(t),t)\}. \tag{17}$$

The following theorem summarizes our results and also shows that it is sufficient to consider only the aperiodic tasks that arrive in the interval $[A_j, D_i]$. Let $Q(A_j, D_i) - Q(A_j)$ be the set of tasks that arrive in the interval $[A_j, D_i]$.

Theorem 3.4. Given a mixed set $S = Q \cup Q_P$, $S(t,\infty)$ is schedulable by a work-conserving EDF scheduler if the following condition holds for any $\tau_i \in Q(t)$.

$$U_{\rm ac}^{\max} + U_P \leqslant 1 \tag{18}$$

where $U_{\rm ac}^{\max} = \max\{U_{Q(A_j,D_i)-Q(A_j)}(\tau_i) \;||\; \tau_j \in Q((B(t),t))\}$.

Proof. See Appendix 3. □

Using Theorem 3.4 one can easily determine the schedulability for a mixed task set at run-time. From an implementation point of view, the system has only to maintain a very small data structure for the current aperiodic set $Q(t)$ and periodic set P. Specifically, the system needs to compute at run-time $U_{\rm ac}^{\max}$ and U_P, which involves arrival times A_i and execution times e_i for aperiodic set $Q(t)$, and periods $\tilde{T}_i$ and execution times $\tilde{e}_i$ for periodic set P. The algorithm for the admission control has a run time of $O(n^2)$ where n is the size of current set $S(t)$.

4. Statistical admission control

In this section, we present two classes of admission control schemes to provide statistical deadline guarantees. The first one uses *statistical utilization demands* which is a statistical version of utilization demands. Statistical utilization demands allows us to easily develop schedulability tests in a similar way to what we have done in Section III, but requires a computationally complex algorithm. The second scheme associates each task with a fixed amount of processor time, *effective execution time*, that would be allocated to the task. It discards any task whose processor usage exceeds its allocated processor time. We combine effective execution times and utilization demands to develop *effective utilization demands* which greatly simplify the admission control.

Throughout this section, we assume that execution times of tasks τ_i are independent random variables de-

noted by e_i^*. We also assume that the probability density function (pdf) of e_i^* is known by the time A_i at which τ_i arrives, and it is denoted by $g_{e_i}(e)$. Let $e_{i,t}^{alloc}$ be the processor time allocated to τ_i by t. The residual execution time $e_{i,t}^*$ is then defined as $e_{i,t}^* = e_i^* - e_{i,t}^{alloc}$. Thus, residual execution times $e_{i,t}^*$ are also independent random variables. In general, the probability density function of $e_{i,t}^*$ can be obtained from the knowledge of $g_{e_i}(e)$ and $e_{i,t}^{alloc}$.

4.1. Statistical utilization demands

A statistical approach allows for infrequent deadline misses but ensures that tasks meet their deadlines with a specified probability. Specifically, probabilistic deadline guarantee is provided in the form of

$$Pr(f_i \leqslant D_i) \geqslant 1 - \varepsilon \tag{19}$$

where ε is generally small, e.g., $\varepsilon = 0.01$. Here we call ε *deadline miss probability*. We formally define the statistical version of schedulability as below.

Definition 4.1. If the probability that an aperiodic task τ_i meets its deadline is equal to or greater than $1-\varepsilon$, τ_i is said to be *statistically schedulable with probability* $1-\varepsilon$

Consider an aperiodic task $\tau_i \in Q(t,t')$. The probability that τ_i meets its deadline can be stated as

$$\begin{aligned} & Pr(f_i \leqslant D_i) \\ & = Pr(t + \sum_{\tau_j \in Q((t,t'),ed(D_i))} e_{j,A_i}^* \leqslant D_i) \qquad (20) \\ & = Pr(\frac{\sum_{\tau_j \in Q((t,t'),ed(D_i))} e_{j,A_i}^*}{D_i - t} \leqslant 1) \qquad (21) \end{aligned}$$

Equation (21) immediately motivates us to develop *statistical utilization demands* as below.

$$U_{Q(t,t')}^*(\tau_i) \stackrel{\text{def}}{=} \frac{\sum_{\tau_j \in Q((t,t'),ed(D_i))} e_{j,t}^*}{D_i - t}. \tag{22}$$

The statistical utilization demand $U_{(\cdot)}^*(\tau_i)$ differs from the utilization demand $U_{(\cdot)}(\tau_i)$ in that it involves random variables $e_{i,t}^*$ instead of $e_{i,t}$ and itself is also a random variable. Using statistical utilization demand $U_{(\cdot)}^*(\tau_i)$, we can write

$$Pr(f_i \leqslant D_i) = Pr(U_{Q(t)}^*(\tau_i) \leqslant 1). \tag{23}$$

Thus, it is straightforward to derive statistical versions of Theorem 3.2 and Lemma 3.3

Theorem 4.1. Aperiodic task set $Q(t,t')$ is statistically schedulable with probability $1-\varepsilon$ by a work-conserving EDF scheduler if

$$Pr(U_{Q(t,t')}^*(\tau_i) \leqslant 1) \geqslant 1 - \varepsilon \tag{24}$$

for any $\tau_i \in Q(t,t')$.

Lemma 4.2. Given a mixed set $S = Q \cup Q_P$, $S(t,\infty)$ is statistically schedulable with probability $1-\varepsilon$ by a work-conserving EDF scheduler if

$$Pr(U_{S(b_i,D_i)}^*(\tau_i) \leqslant 1) \geqslant 1 - \varepsilon \tag{25}$$

for any $\tau_i \in S(t,t')$.

Here, we can decompose $U_{S(b_i,D_i)}^*(\tau_i)$ in a similar way to what we have done in Section 3. Let U_{pc}^* and U_{ac}^* be random variables which represent a periodic contribution and an aperiodic contribution to $U_{S(b_i,D_i)}^*(\tau_i)$, respectively.

$$U_{S(b_i,D_i)}^*(\tau_i) = U_{pc}^* + U_{\text{ac}}^* \tag{26}$$

where

$$U_{pc}^* \leqslant \sum_{\tilde{\tau}_j \in P} \frac{\tilde{e}_j^*}{\tilde{T}_j} \quad \text{and} \tag{27}$$

$$\begin{aligned} U_{\text{ac}}^* &\leqslant \frac{\sum_{\tau_j \in Q((b_i,D_q),sd(\tau_q))} e_j^*}{D_q - b_i} \\ &= U_{Q(b_i,D_q)}^*(\tau_q). \end{aligned} \tag{28}$$

Let U_P^* be the statistical utilization $\sum_{\tilde{\tau}_j \in P} \frac{\tilde{e}_j^*}{\tilde{T}_j}$. We can obtain the following schedulability condition by using Theorem 3.4.

Theorem 4.3. Given a mixed set $S = Q \cup Q_P$, $S(t,\infty)$ is schedulable by a work-conserving EDF scheduler if the following condition holds for any $\tau_i \in Q(t)$.

$$\begin{aligned} & Pr(U_{Q(A_j,D_i)-Q(A_j)}^*(\tau_i) + U_P^* \leqslant 1) \\ & \quad \geqslant 1 - \varepsilon \end{aligned} \tag{29}$$

for all A_j of $\tau_j \in Q((B(t),t), ed(D_i))$.

By setting $Q = \phi$ in Theorem 4.3, we can derive a statistical schedulability condition for periodic task set P.

Corollary 4.4. Periodic task set P is schedulable by a work-conserving EDF scheduler if the following condition holds for any $\tau_i \in Q(t)$.

$$Pr(U_P^* \leqslant 1) \geqslant 1 - \varepsilon. \tag{30}$$

Applying the above results to admission control requires computing probabilities at run-time. We can compute the desired probabilities by convolving the sum of random variables, since task execution times are statistically independent and every desired probability is represented by a sum of random variables. For instance, the probability in Eq.(29) can be rewritten as below.

$$Pr(U^*_{Q(b_i,D_q)}(\tau_q) \leqslant 1) \tag{31}$$

$$= Pr(\frac{\sum_{\tau_j \in Q((b_i,D_q),ed(D_q))} e^*_j}{D_q - b_i} \leqslant 1) \tag{32}$$

$$= \frac{1}{D_q - b_i} \int_0^{D_q - b_i} g_{e_j}(e) * \ldots * g_{e_k}(e) de \tag{33}$$

where convolutions are performed for all $\tau_j \in Q((b_i, D_q), ed(D_q))$ and $g_{e_j}(e)$ is the pdf of e^*_j for $\tau_j \in Q((b_i, D_q), ed(D_q))$.

In fact, the admission control using Eq. (33) results in a computationally complex algorithm since it involves expensive convolution operations. For instance, the computational complexity of convolution $g * h$ is known to be $O(n^2)$ where n is the number of points in discretized functions of g and h. Although the run-time overhead can be reduced if we use FFT (Fast Fourier Transform) [18], the algorithm still requires $O(nlog_2 n)$ for $g * h$. Our next approach eliminates the need for convolutions by taking advantage of effective execution times, thus enabling efficient on-line admission control.

4.2. Effective utilization demands

Our second approach improves upon the first approach by using *effective execution times*. Every task is associated with a particular amount of processor time, called effective execution time, and the admission control is performed using effective execution times. If any task overruns its effective execution time, it is immediately discarded. By overrun, we mean that a task consumes more processor time than its effective execution time.

The objective of preventing task overruns is to isolate tasks from one another. Under this scheme, every task can independently receive processor time up to the amount of its effective execution time. Thus, the probability that a task meets its deadline is not adversely affected by other tasks. If we choose appropriate values for effective execution times for a given bound ε, tasks can be statistically schedulable with probability $1 - \varepsilon$. To choose the minimal processor time required for a given bound, we can define the effective execution time e^ε_i of τ_i as a function of the required deadline miss probability ε and probability density function $g_{e_i}(e)$.

$$\int_0^{e^\varepsilon} g_{e_i}(x) dx = 1 - \varepsilon. \tag{34}$$

Clearly, discarding overrun tasks has the implication that execution times are forced to be bounded. This enforcement needs extra run-time support from the underlying system. The system has to watch the amount of allocated processor time for each task $\tau_i \in S(t)$ and discard overrun tasks. Despite this overhead, the great benefit of preventing task overruns is that it allows us to integrate effective execution times and the deterministic techniques we developed in Section III. Using effective execution times, we define *effective utilization demand* $U^*_{Q(t,t')}(\tau_i)$ by

$$U^\varepsilon_{Q(t,t')}(\tau_i) \stackrel{\text{def}}{=} \frac{\sum_{\tau_j \in Q((t,t'),ed(D_i))} e^\varepsilon_{j,t}}{D_i - t} \tag{35}$$

where $e^\varepsilon_{j,t} = e^\varepsilon_{j,t} - e^{alloc}_{j,t}$.

Based on the above definition, the following theorem provides a statistical version of schedulability condition for a mixed set.

Theorem 4.5. Given a mixed set $S = Q \cup Q_P$, $S(t, \infty)$ is schedulable by a work-conserving EDF scheduler if the following condition holds for any $\tau_i \in Q(t)$.

$$U^{\varepsilon,\max}_{\text{ac}} + U^\varepsilon_P \leqslant 1 \tag{36}$$

where $U^{\varepsilon,\max}_{\text{ac}} = \max\{U^\varepsilon_{Q(A_j,D_i)-Q(A_j)}(\tau_i) \mid\mid \tau_j \in Q((B(t), t))\}$ and $U^\varepsilon_P = \sum_{\tilde{\tau}_j \in P} \frac{\tilde{e}^\varepsilon_j}{\tilde{T}_j}$.

In many applications, it may be unnecessarily stringent to discard overrun tasks. If the system is not overloaded, it is often advantageous to allow an overrun task as long as its further execution does not interfere with other admitted tasks. There are two other possibilities for handling overruns without affecting statistical guarantees for other admitted tasks. The first one is to give second chances to overrun tasks. Under this, the overrun task, whether it is periodic or aperiodic, is treated as a new aperiodic task. This task can receive processor time if it passes a new admission test. The other one is to provide utilization slack.

The use of utilization slack is similar to the idea of slack stealing [8,12]. By Theorem 4.5, we can determine utilization slack and estimate available processor time for an overrun task. The following corollary im-

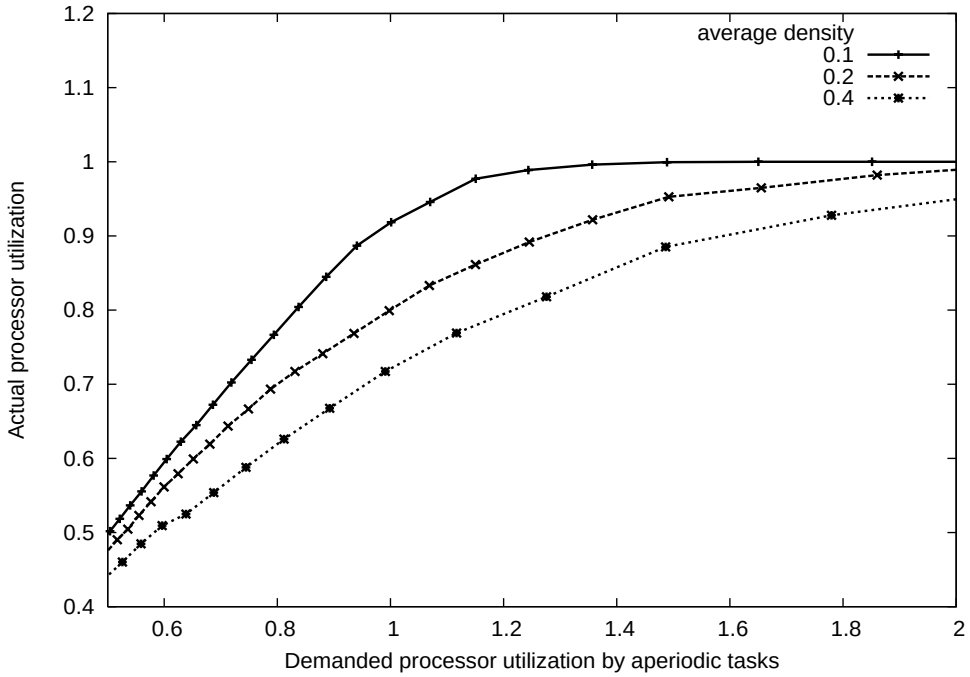

Fig. 5. Deterministic admission control for aperiodic task sets.

mediately follows from Theorem 4.5 and shows how to estimate available processor time.

Corollary 4.6. Given a mixed set $S = Q \cup Q_P, S(t,\infty)$ and a work-conserving EDF scheduler, suppose that $\tau_i \in S(t,\infty)$ overruns at time t. Let $e_i^{slack}(t)$ be the available processor time for τ_i such that every task $\tau_j \in S(t,\infty)$ is statistically schedulable with probability $1-\varepsilon$. The available processor time $e_i^{slack}(t)$ satisfies the following

$$e_i^{slack}(t) \leqslant d_{i,t} \cdot (1 - U_{\mathrm{ac}}^{\varepsilon,\max} - U_P^{\varepsilon}). \tag{37}$$

where $U_{\mathrm{ac}}^{\varepsilon,\max} = \max\{U^{\varepsilon}_{Q(A_j,D_i)-Q(A_j)}(\tau_i) \parallel \tau_j \in Q((B(t),t))\}$ and $U_P^{\varepsilon} = \sum_{\tilde{\tau}_j \in P} \frac{\tilde{e}_j^{\varepsilon}}{\tilde{T}_j}$.

5. Experimental evaluation

In this section, we present a series of simulations to show the efficacy of the proposed admission control approaches. We first describe the experiment setup including the choice of a performance metric. We then present experimental results for deterministic and statistical admission control schemes using utilization demands and effective execution times, respectively.

5.1. Experiment setup

The aim of our experiments was to provide a performance evaluation of the proposed approaches as well as a check on their correctness. Processor utilization was the major performance metric used in our experiments. For each simulation, we measured the processor utilization and compared it to the input workload that represents the demanded processor utilization by artificially generated tasks. Note that an ideal admission controller could yield almost the same processor utilization as the demanded processor utilization as long as the demanded utilization does not exceed 100%.

We implemented a simulator of the simple architecture described in Fig. 1, which consists of an admission controller, admit queues, and a task scheduler. We also implemented a task generator that provides the simulator with input workloads by generating periodic and aperiodic tasks.

Task parameters can be specified either for a deterministic simulation environment or a statistical simulation environment. The parameter setting for the deterministic environment was as follows.

– **Arrivals:** Interarrival times for aperiodic tasks were generated from an exponential distribution with a specified average. Periods of periodic tasks

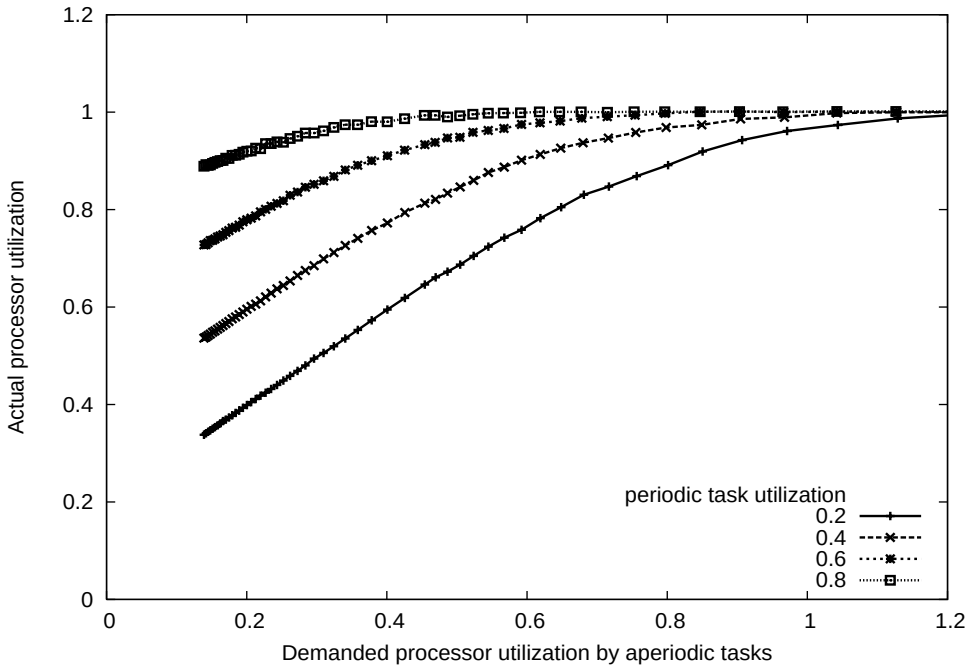

Fig. 6. Deterministic admission control for mixed sets of periodic and aperiodic tasks.

were generated from a uniform distribution between 10 and 20

– **Deadlines:** Relative deadlines of aperiodic tasks were generated from a uniform distribution between 10 and 20. Relative deadlines of periodic tasks were set equal to their periods.
– **Execution times:** Worst-case execution times of tasks were chosen to be less than their relative deadlines from the range (0, 20]. The chosen worst-case execution times were also used as actual execution times.

The statistical simulation environment used the same parameter setting as the deterministic environment except for execution times. In order to investigate how probability distributions of execution times affect the performance of our admission control schemes, we used normal distributions with various pairs of mean and standard deviation.

5.2. Deterministic admission control using utilization demands

The results for deterministic admission control can be summarized by the following. First, the processor utilization was very close to the demanded processor utilization in most cases as long as the demanded processor utilization did not exceed one. This indicates that our admission control tests were very accurate to achieve processor utilization as high as possible. Second, the achievable processor utilization deteriorates if tasks have tight deadlines compared to their execution times. We could capture the tightness of deadline by $\frac{e_i}{d_i}$, which is often called *density*. Figure 5 shows the results with various density values.

Figure 5 compares processor utilization to input workload for three task sets, each of which consists of 1000 aperiodic tasks. Each task set has a different average density. The average values are 0.1, 0.2, and 0.4, respectively. As shown in the figure, higher density gives low processor utilization. The reason is that tasks with high density values contribute more to the utilization demand $U_{\mathrm{ac}}^{\max}$ than the tasks with low density values. Thus, the admission controller is more likely to reject tasks that have high density values.

Figure 6 shows our second results for mixed sets, each of which consists of 1000 aperiodic tasks and 5 periodic tasks. The figure draws some representative results with various values of periodic task utilization $U_P = 0.2, 0.4, 0.6, 0.8$, and shows that no task misses its deadline in any case. This confirms that the proposed approach can successfully handle a mixed set of periodic and aperiodic tasks.

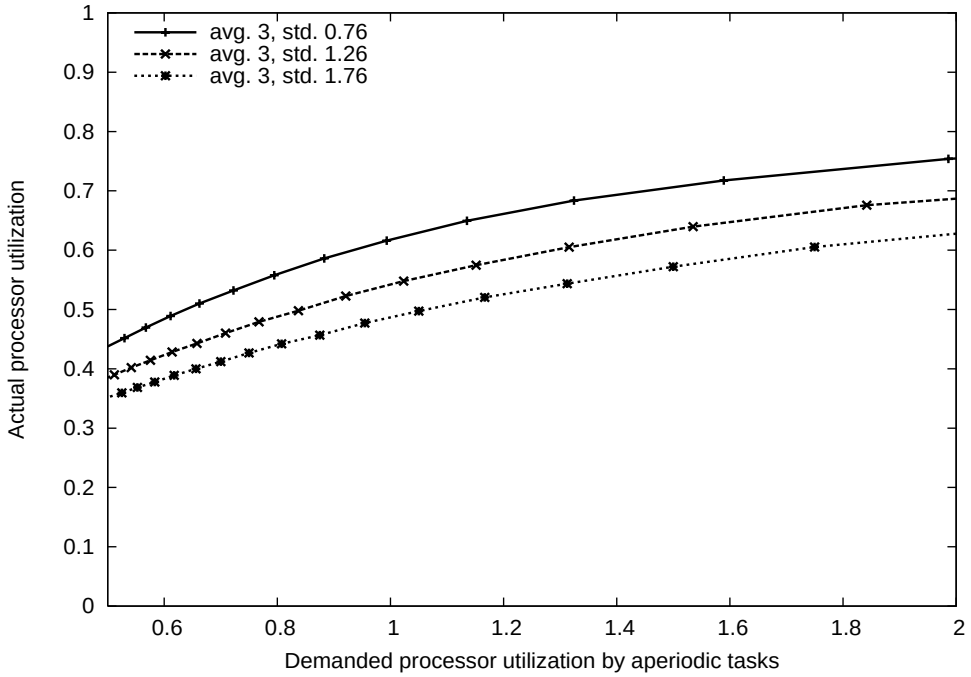

Fig. 7. Statistical admission control for mixed sets of periodic and aperiodic tasks: $N(\mu = 2, \sigma = 0.5)/N(\mu = 3, \sigma = 0.76)/N(\mu = 3, \sigma = 1.26)/N(\mu = 3, \sigma = 1.76)$.

5.3. *Statistical admission control using effective execution times*

We now consider the results for statistical admission control using effective execution times. In our preliminary experiments, we found that the characteristics of execution time distributions have a strong impact on the performance of the proposed admission control scheme. To investigate this, we generated execution times using normal distributions with various mean and standard deviations, and measured the processor utilization for each specific probability distribution. Here, we defined the demanded processor utilization using effective execution times instead of worst-case execution times.

Figure 7 shows the results for three mixed sets S_1, S_2, and S_3, each of which consists of 5 periodic tasks and 10,000 aperiodic tasks. The deadline miss probability was set to be $\varepsilon = 0.1$ for all the tasks. For periodic tasks, a common normal distribution $N(\mu = 2, \sigma = 0.5)$ was used to generate their execution times so that $U_P^{\varepsilon} = 0.4$ for the three mixed sets. For aperiodic tasks, three different normal distributions, $N(\mu = 3, \sigma = 0.76)$, $N(\mu = 3, \sigma = 1.26)$, and $N(\mu = 3, \sigma = 1.76)$ were used to generate their execution times for S_1, S_2, and S_3, respectively. This gave us three effective execution times, 3.97, 4.61, and 5.26 for S_1, S_2, and S_3 respectively. As shown in the figure, high standard deviation leads to low processor utilization. The reason is that high standard deviation gives high effective execution times which have a strong negative impact on the effective utilization demand $U^{\varepsilon,\max}$. That is, tasks with high effective execution times are more likely to be rejected. Since all the tasks had the same average execution times, achievable processor utilization deteriorates when the execution times have a high standard deviation. Figure 8 shows the similar results with different normal distributions $N(\mu = 2, \sigma = 2.04)$, $N(\mu = 3, \sigma = 1.26)$, and $N(\mu = 4, \sigma = 0.48)$.

6. Conclusion

Our major contributions include two classes of admission control schemes using utilization demands and effective execution times. First, we have presented a novel schedulability analysis using utilization demands which can be applied to a mix of periodic and aperiodic tasks with deterministic execution times. We have shown that the algorithm for this analysis requires a very small data structure and incurs low run-time over-

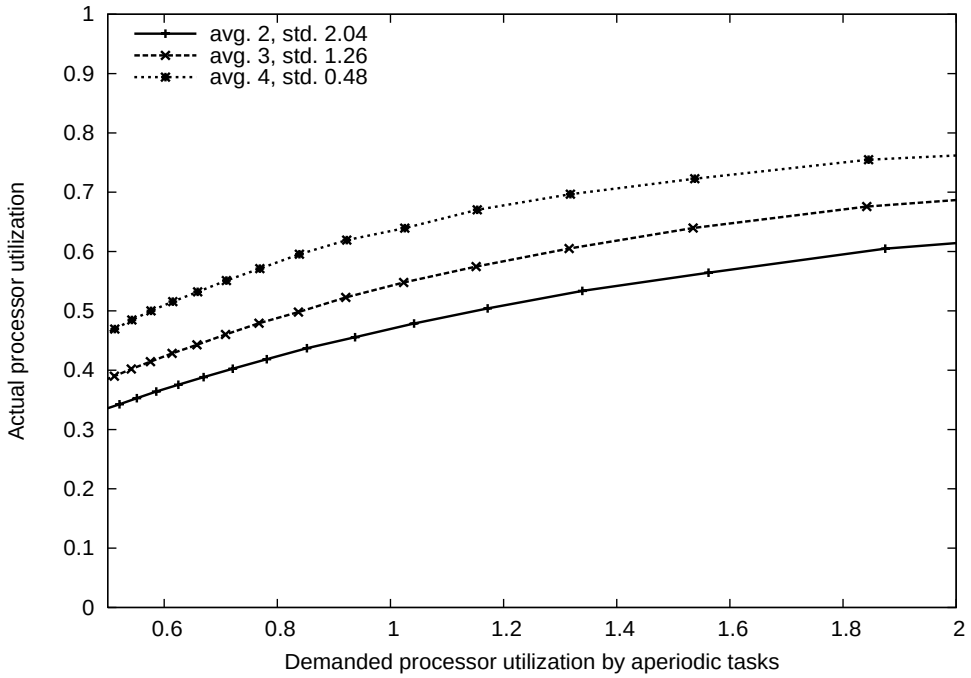

Fig. 8. Statistical admission control for mixed sets of periodic and aperiodic tasks: $N(\mu = 2, \sigma = 0.5)/N(\mu = 2, \sigma = 2.04)/N(\mu = 3, \sigma = 1.26)/N(\mu = 4, \sigma = 0.48)$.

head, and thus it enables an efficient on-line admission control. Second, we have presented an efficient statistical scheme based on effective execution times and overrun handling. By preventing overruns, effective execution times allow tasks to meet their deadlines with a specified probability without being interfered with. We have shown that the statistical admission control can be greatly simplified by using effective utilization demands which are a combination of effective execution times and utilization demands.

There are several future research directions. First, we could extend the deterministic admission control scheme based on utilization demands for fixed priority scheduling algorithms such as rate monotonic (RM) algorithm. Second, we could evaluate a tradeoff between various, often conflicting factors. For instance, deadline miss probability might have a strong influence on the achievable processor utilization. Although we have not considered this problem in this paper, the results presented here will be useful in such an evaluation.

Acknowledgments

The work reported in this paper was supported in part by the Korea Research Foundation Grant (KRF-2003-003-D00340), by the research fund of Hanyang University (HY-2003), by the National Research Laboratory (NRL) Grant M1-9911-00-0120, by the Institute of Computer Technology (ICT), and by the Automation and Systems Research Institute (ASRI).

Appendix 1

Since $U_{Q_P(t,t')}(\tau_i)$ is maximum when $t = b_i$ and $t' = D_i$, it suffices to show that $U_{Q_P(b_i,D_i)}(\tau_i) \leqslant U_P$. It immediately follows from Eqs (11), (12), (13), and (14).

$$\begin{aligned} U_{Q_P(b_i,D_i)}(\tau_i) &= \frac{\sum_{\tau_j \in Q_P((b_i,D_i),ed(D_i))} e_j}{D_i - b_i} \\ &\leqslant \frac{\sum_{\tau_j \in Q_P((b_i,D_p),ed(D_p))} e_j}{D_p - b_i} \\ &\leqslant \frac{\sum_{\tilde{\tau}_j \in P} \lfloor (D_p - b_i)/\tilde{T}_j \rfloor \tilde{e}_j}{D_p - b_i} \\ &\leqslant \sum_{\tilde{\tau}_j \in P} \frac{\tilde{e}_j}{\tilde{T}_j} \;=\; U_P \end{aligned}$$

Appendix 2: Proof of Lemma 3.3

The proof is very similar to that of Theorem 3.1. Let f_i be the worst case finish time of $\tau_i \in S(t,\infty)$. Since the scheduler is work-conserving, the finish time f_i will be no later than b_i plus the sum of execution times of higher priority tasks including τ_i's execution time. Note that there can be more than one busy periods in the interval $[b_i, D_i]$. Such cases mean that τ_i completes in the first busy period since the processor is work-conserving. This indicates that some of higher priority tasks in $S((b_i,D_i),ed(D_i))$ do not interfere with τ_i, thus $f_i < b_i + \sum_{\tau_j \in S((b_i,D_i),ed(D_i))} e_j$. If there exists only one busy period in the interval $[b_i, D_i]$, f_i will be exactly the same as $b_i + \sum_{\tau_j \in S((b_i,D_i),ed(D_i))} e_j$. In either case, we can write

$$\begin{aligned} f_i &\leqslant b_i + \sum_{\tau_j \in S((b_i,D_i),ed(D_i))} e_j \\ &\leqslant b_i + (D_i - b_i) \cdot U_{S((b_i,D_i))}(\tau_i). \end{aligned}$$

Since $U_{S((b_i,D_i))}(\tau_i) \leqslant 1$,

$$\begin{aligned} f_i &\leqslant b_i + (D_i - b_i) \\ &\leqslant D_i. \end{aligned}$$

This completes the proof. □

Note that the converse is not true in Lemma 3.3. We can show this by producing a counter example. Suppose that there are more than one busy periods in the interval $[b_i, D_i]$. As mentioned above, this means that $f_i < D_i$. Further suppose that higher priority tasks arrive after f_i and the sum of their execution times is greater than $D_i - b_i$. This leads to $U_{S(b_i,D_i)}(\tau_i) > 1$, but τ_i is schedulable.

Appendix 3: Proof of Theorem 3.4

Let $U_{\text{ac}}^{\max}$ be $\max\{U_{Q(A_j,D_q)}(\tau_q) || \tau_j \in S((B(t), t))\}$. It follows from Eqs (10), (14), and Eq. (18) that $S(t,\infty)$ is schedulable if $U_{\text{ac}}^{\max} + U_P \leqslant 1$.

Here, we can refine $\max\{U_{Q(A_j,D_q)}(\tau_q) \mid\mid \tau_j \in S((B(t),t))\}$ in two respects. First, periodic task arrivals do not affect the upper bound $U_{\text{ac}}^{\max}$. Second, $Q((A_j,D_q),ed(D_q))$ in Ineq.(18) should not contain the tasks that arrive before A_j since we only need to consider candidates for b_i which is the beginning of busy period. Thus, it is sufficient to consider only the tasks that arrive in the interval $[A_j, D_i]$ when we compute $U_{Q(A_j,D_q)}(\tau_q)$.

Appendix 4: Summary of notations

Notation	Meaning
Q	Set of aperiodic tasks
$Q(t)$	Set of aperiodic tasks that have been admitted by t and have not completed by t
$Q(t,t')$	Set of tasks that are admitted in the interval between t and t' including the tasks of $Q(t)$
P	Set of periodic tasks
Q_P	Pseudo-aperiodic set of P
$Q_P(t,t')$	Subset of pseudo-aperiodic set Q_P corresponding to $Q(t,t')$
S	Mixed set of periodic and aperiodic tasks ($= Q \cup Q_P$)
$S(t,t')$	$Q(t,t') \cup Q_P(t,t')$
$\tau_i, \tilde{\tau}_i, \tilde{\tau}_{i,j}$	Aperiodic task, periodic task, and periodic task instance
A_i, e_i, d_i, D_i	Arrivel time, execution time, relative deadline, and absolute deadline of τ_i
e_i^*	Random variable of e_i
$g_{e_i}(e)$	Pdf of e_i
e_i^e	Effective execution time of τ_i
$e_{i,t}$	Residual execution time of τ_i at t
$d_{i,t}$	Lead time of τ_i at t ($= D_i - t$)
$\tilde{A}_i, \tilde{R}_i, \tilde{T}_i$	Arrival time, release time, and period of $\tilde{\tau}_i$
$\tilde{e}_i$	Least upper bound on execution time of $\tilde{\tau}_{i,j}$
$\tilde{D}_{i,j}$	Absolute deadline of $\tilde{\tau}_{i,j}$
$\tilde{e}_{i,j}^*$	Random variable of $\tilde{e}_{i,j}$
$g_{\tilde{e}_i}(e)$	Pdf of $\tilde{e}_i$
$U_X(\tau_i)$	Utilization demand for $\tau_i \in X$
U_P	Utilization of periodic task set P
b_i	Beginning of level-τ_i busy period that includes A_i
$B(t)$	Beginning of level-0 busy period that includes b_i
$Q(t, ed(t'))$	Set of tasks whose absolute deadlines are no later than t' in $Q(t)$
$Q((t,t'),ed(D_i))$	Set of tasks whose absolute deadlines are no later than D_i in $Q(t,t')$
$Q_P((t,t'),ed(D_i))$	Set of periodic task instances whose absolute deadlines are no later than D_i in $Q_P(t,t')$
$S((t,t'),ed(D_i))$	$Q((t,t'),ed(D_i)) \cup Q_P((t,t'),ed(D_i))$

References

[1] C. Liu and J. Layland, Scheduling algorithm for multiprogramming in a hard real-time environment, *Journal of the ACM* **20**(1) (Jan., 1973), 46–61.

[2] N. Audsley, A. Burns, M. Richardson and A. Wellings, *Hard real-time scheduling: The deadline-monotonic approach*, in Proceedings of IEEE Workshop on Real-Time Operating Systems and Software, May, 1991, pp. 133–137.

[3] J.P. Lehoczky, L. Sha and Y. Ding, *The rate monotonic scheduling algorithm: Exact characterization and average case behavior*, in Proceedings of IEEE Real-Time Systems Symposium, IEEE Computer Society Press, Dec., 1989, pp. 166–171.

[4] T. Baker and A. Shaw, The cyclic executive model and ada, *The Journal of Real-Time Systems* **1**(1) (Sept., 1989), 7–25.

[5] J. Leung and M. Merill, A note on the preemptive scheduling of periodic, real-time tasks, *Information Processing Letters* **11**(3) (Nov., 1980), 115–118.

[6] J.P. Lehoczky, L. Sha and J. Strosnider, *Enhanced aperiodic responsiveness in hard real-time environments*, in Proceedings of IEEE Real-Time Systems Symposium, IEEE Computer Society Press, Dec., 1987, pp. 261–270.

[7] B. Sprunt, L. Sha and J.P. Lehoczky, Aperiodic task scheduling for hard-real-time systems, *The Journal of Real-Time Systems* **1**(1) (1989), 27–60.

[8] R. Davis, K. Tindell and A. Burns, *Scheduling slack time in fixed priority pre-emptive systems*, in Proceedings of IEEE Real-Time Systems Symposium, IEEE Computer Society Press, Dec., 1993, pp. 222–231.

[9] M. Spuri and G. Buttazzo, Scheduling aperiodic tasks in dynamic priority systems, *Journal of Real-Time Systems* **10**(2) (1996), 1979–2012.

[10] G. Fohler, *Joint scheduling of distributed complex periodic and hard aperiodic tasks in statically scheduled systems*, in Proceedings of IEEE Real-Time Systems Symposium, Dec., 1995, pp. 22–33.

[11] D. Isovic and G. Fohler, *Online handling of hard aperiodic tasks in time triggered systems*, in Proceedings of the 11th EUROMICRO Conference on Real-Time Systems, June, 1999.

[12] J.P. Lehoczky and S. Ramos-Thuel, *An optimal algorithm for scheduling soft-aperiodic tasks in fixed-priority preemptive systems*, in Proceedings of IEEE Real-Time Systems Symposium, Dec., 1992, pp. 110–123.

[13] H. Chetto and M. Chetto, Some results of the earliest deadline first scheduling algorithm, *IEEE Transactions on Software Engineering* **15**(10) (Oct., 1989), 1261–1268.

[14] A.K. Atlas and A. Bestavros, *Statistical rate monotonic scheduling*, in *Proceedings of IEEE Real-Time Systems Symposium*, Dec., 1998, pp. 123–132.

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

[16] J.P. Lehoczky, *Real-time queueing theory*, in Proceedings of IEEE Real-Time Systems Symposium, Dec., 1996, pp. 186–195.

[17] G. Buttazzo, *Value vs. deadline scheduling in overload conditions*, in Proceedings of IEEE Real-Time Systems Symposium, Dec., 1995, pp. 90–99.

[18] J.R. Johnson and R.W. Johnson, *Challenges of computing the fast fourier transform*, in Proceedings of Optimized Portable Application Libraries (OPAL) Workshop, June, 1997.