An Efficient On-Line Job Admission Control Scheme to Guarantee Deadlines for QoS-Demanding Applications*

Jungkeun Park[1], Minsoo Ryu[2], and Seongsoo Hong[3]

[1] Computing Lab., Samsung Advanced Institute of Technology, Korea
{jung-keun.park}@samsung.com
[2] College of Information and Communications,
Hanyang University, Korea
{msryu}@hanyang.ac.kr
[3] School of Electrical Engineering and Computer Science,
Seoul National University, Korea
{sshong}@redwood.snu.ac.kr

Abstract. Guaranteeing deadlines is essential in providing high levels of QoS (Quality of Service) in many networked applications that require timely processing and delivery of requested data. However, despite recent developments in real-time computing, current real-time scheduling theory cannot be directly applied to those applications since most real-time research has focused on the periodic task model while recent networked applications have the distinguishing characteristic that processor usage patterns include both periodic and aperiodic tasks. In this paper, we present a novel admission control scheme for a mixed set of periodic and aperiodic tasks with hard deadlines under EDF, which can achieve near optimal performance with practical utility. The proposed admission control scheme is based on a novel schedulability measure for a deadline-constrained task, called *utilization demand*, which can be viewed as a combination of the processor time demand and the utilization factor. We first show that this new schedulability measure provides a necessary and sufficient schedulability condition for aperiodic tasks. We then present an efficient schedulability test for a mixed set of periodic and aperiodic tasks under EDF. The resulting schedulability test can be implemented as an on-line admission control algorithm of $O(n)$ complexity, which in practice incurs sufficiently low overhead. Our experimental results show that the proposed admission control scheme outperforms existing approaches with respect to achievable processor utilization.

1 Introduction

Guaranteeing deadlines is essential in providing high levels of QoS (Quality of Service) in many networked applications. In general, QoS and real-time applications share the common constraint that services are required to meet timing

* The work reported in this paper was supported in part by the Korea Research Foundation Grant (KRF-2003-003-D00340) and by the research fund of Hanyang University (HY-2003).

requirements such as deadline and rate. Examples include interactive distance learning and online trading, which require timely processing and delivery of requested data. However, despite recent developments in real-time computing, current real-time scheduling theory cannot be directly applied to those applications since most real-time research has focused on the periodic task model [10, 2, 7, 9], in which task arrivals and related timing attributes are deterministic and known in advance. On the other hand, recent networked applications have the distinguishing characteristic that 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.

There have been many attempts to deal with a mix of periodic and aperiodic tasks in the context of real-time scheduling. The authors of [8, 13, 5, 6] proposed static-priority algorithms for joint scheduling of periodic and aperiodic tasks. They address the problem of minimizing the response times of soft aperiodic tasks while guaranteeing the deadlines of hard periodic tasks. Specifically, aperiodic tasks are handled in the background at a lower priority level, or at some fixed priority level by a special periodic task which serves aperiodic requests with a limited capacity. Considerable research has also focused on dynamic-priority scheduling algorithms, exemplified by the EDF (earliest deadline first) algorithm [10], and the static-priority approaches of [8, 15, 13] have been extended to dynamic-priority versions [13, 14]. Chetto and Chetto [4] and Ripoll et al. [12] proposed optimal dynamic-priority algorithms with regard to specific criteria, for example, response time or processor utilization.

However, in the above mentioned cases, it is assumed that aperiodic tasks do not have hard deadlines, and periodic tasks are given preferential treatment. Although the scheduling and analysis algorithms proposed in [4, 12, 5, 6] can be extended to handle hard aperiodic tasks, their computational complexity makes them inadequate for on-line usage in networked applications. Specifically, they require construction of a table that contains slack times present in the processor schedule for the hard periodic tasks over a specific time interval, hyperperiod (least common multiple of the task periods) [4–6] or Initial Critical Interval [12], and use the table of slack times to determine the schedulability of incoming hard aperiodic tasks. The significant computational cost of these approaches limit their applicability for on-line applications with dynamic periodic and aperiodic request arrivals.

In this paper, we attempt to provide deadline guarantees via admission control for both periodic and aperiodic tasks under EDF. Our approach differs from the approaches mentioned above in that aperiodic tasks have hard deadlines and are scheduled by the same scheduling policy as the periodic tasks, and that it can be used for on-line admission control as it incurs sufficiently low run-time overhead. Our major contribution is two-fold. First, we propose the combined use of the processor time demand [2] and the utilization factor [10] as a schedulability measure for deadline-constrained tasks. The new schedulability measure, called *utilization demand*, is defined for each task as the ratio of the processor

time required for meeting its deadline to the time remaining until its deadline expires. We show that this new schedulability measure provides a necessary and sufficient schedulability condition for aperiodic tasks. Second, we extend the utilization demand analysis to handle periodic tasks as well, and develop an efficient schedulability test for a mix of periodic and aperiodic tasks under EDF. The resulting schedulability analysis can be implemented as an on-line admission control algorithm of $O(n)$ complexity, which can achieve near optimal performance with practical utility.

2 Models and Assumptions

Consider a set of independent aperiodic tasks $Q = \{\tau_1, \tau_2, \ldots, \tau_i, \ldots\}$. Every aperiodic task $\tau_i \in Q$ has an arrival time A_i, a worst-case execution time e_i, and a relative deadline d_i that is defined from its arrival time. The absolute deadline D_i of τ_i is given by $D_i = A_i + d_i$. At any given time t, we can define $Q(t) \subset Q$ as the set of current tasks that have been admitted by t and whose deadlines have not expired by t. Thus, the current aperiodic set $Q(t)$ can be described by $\{\tau_i | \tau_i \in Q, A_i \leq t, D_i > t\}$.

We use similar notation for periodic tasks $P = \{\tilde{\tau}_1, \ldots \tilde{\tau}_i, \ldots, \tilde{\tau}_N\}$. Periodic task $\tilde{\tau}_i$ with period $\tilde{T}_i$ can be considered as an infinite sequence of aperiodic tasks. Such aperiodic tasks 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 $\tilde{d}_i$ and a common worst-case execution time $\tilde{e}_i$. We use $\tilde{A}_i$ to denote the arrival time of $\tilde{\tau}_i$, and thus the absolute deadline $\tilde{D}_{i,j}$ of $\tilde{\tau}_{i,j}$ is computed by $\tilde{D}_{i,j} = \tilde{A}_i + (j-1)\tilde{T}_i + \tilde{d}_i$.

In our discussions, we assume a generic system architecture that consists of an admission controller, admit queues, and a task scheduler, as in Figure 1. The admission controller, through admit or reject, is responsible for ensuring that the system can provide the 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 uses EDF (earliest deadline first) algorithm to select the task with the highest priority from the admit queues, and allocates the processor to it.

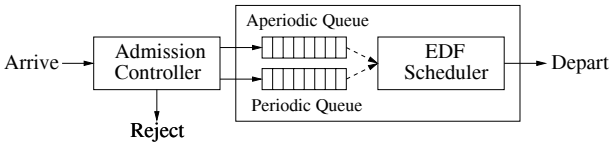

Fig. 1. System architecture composed of an admission controller, admit queues, and a task scheduler

3 Utilization Demands as Schedulability Measures

In this section we first define a point-wise version of utilization demand and then extend it to an interval-wise version, which will be used for admission tests for a mix of periodic and aperiodic tasks.

3.1 Point-Wise Utilization Demands

Consider a set of independent aperiodic tasks $Q(t) = \{\tau_1, \ldots, \tau_i, \ldots, \tau_n\}$. 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, i.e., $D_i - t$.

We 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. Let $Q(t, hp(\tau_i)) \subset Q(t)$ be the set of tasks whose priorities are equal to or higher than τ_i's priority. The utilization demand of τ_i is defined by

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

The following theorem shows that the use of utilization demands provides a necessary and sufficient schedulability condition for aperiodic task set $Q(t)$. The proof is rather straightforward, and we do not provide proofs here due to space limitation.

Theorem 1. *Aperiodic task $\tau_i \in Q(t)$ is schedulable with respect to $Q(t)$ by a work-conserving priority-based scheduler if and only if*

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

Note that the schedulability of $\tau_i \in Q(t)$ does not imply the schedulability of $\tau_i \in Q$, since Q may contain some high-priority tasks that will arrive later in the interval $[t, D_i]$ and they would preempt τ_i. Therefore, the schedulability test in Ineq.(2) is valid only until the next arrival time of a task. This necessitates the testing of the schedulability at every task arrival. In general, to check if $\tau_i \in Q$ is schedulable, it suffices to show $U_{Q(t)}(\tau_i) \leq 1$ at every arrival time until the deadline of τ_i expires. The following theorem formally states this.

Theorem 2. *Aperiodic task $\tau_i \in Q$ is schedulable with respect to Q by a work-conserving priority-based scheduler if and only if*

$$U_{Q(A_j)}(\tau_i) \leq 1 \tag{3}$$

at every arrival time A_j of $\tau_j \in Q$ such that $A_i \leq A_j \leq D_i$.

3.2 Interval-Wise Utilization Demands

We describe an extended concept of utilization demand called *interval-wise utilization demand*, that is defined over a given time interval. Let $Q(t,t') \subset Q$ be the set of tasks whose arrival times are no later than t' and absolute deadlines are later than t, i.e., $Q(t,t') = \{\tau_i | \tau_i \in Q, A_i \leq t', D_i > t\}$. In other words, $Q(t,t')$ includes the tasks that are admitted in the interval $[t,t']$ as well as the tasks of $Q(t)$. We now define the interval-wise utilization demand for $\tau_i \in Q(t,t')$ by

$$U_{Q(t,t')}(\tau_i) \stackrel{\text{def}}{=} \frac{\sum_{\tau_j \in Q((t,t'),hp(\tau_i))} e_{j,t}}{D_i - t} \tag{4}$$

where $Q((t,t'),hp(\tau_i)) \subset Q(t,t')$ represents the set of tasks that have priorities equal to or higher than τ_i. By using Theorem 1 and Theorem 2, the above definition immediately leads to the following schedulability conditions.

Theorem 3. *Aperiodic task $\tau_i \in Q(t,t')$ is schedulable with respect to $Q(t,t')$ by a work-conserving EDF scheduler if and only if*

$$U_{Q(t,t')}(\tau_i) \leq 1. \tag{5}$$

Theorem 4. *Aperiodic task $\tau_i \in Q$ is schedulable with respect to Q by a work-conserving EDF scheduler if and only if*

$$U_{Q(A_i,D_i)}(\tau_i) \leq 1. \tag{6}$$

The notion of interval-wise utilization demands allows us to take into account periodic tasks as well as aperiodic tasks. Consider a periodic task set $P = \{\tilde{\tau}_1, \tilde{\tau}_2, \ldots, \tilde{\tau}_N\}$. Essentially, all instances of periodic tasks can be considered as aperiodic tasks. That is, an instance $\tilde{\tau}_{i,j}$ of periodic task $\tilde{\tau}_i$ can be thought of as an aperiodic task τ_k that has arrival time $A_k = \tilde{A}_i + (j-1)\tilde{T}_i$ and deadline $d_k = \tilde{T}_i$. Without loss of generality, we can say that any periodic set P has an equivalent aperiodic set Q_P which consists of all task instances generated by P. We refer to this aperiodic task set Q_P as the *pseudo-aperiodic set* of P. It is obvious that P is schedulable if and only if all the tasks in Q_P are schedulable. Thus, we can use Q_P to treat periodic tasks and aperiodic tasks in a similar manner.

4 Utilization Demand Analysis for a Mixed Set

Consider a mixed set S of aperiodic set Q and pseudo-aperiodic set Q_P. Based on Theorem 4, whenever a new aperiodic task τ_x arrives, we can guarantee the schedulability of $S = Q \cup Q_P \cup \{\tau_x\}$ by ensuring that the utilization demand $U_{S(A_i,D_i)}(\tau_i)$ does not exceed 1 for any $\tau_i \in S$. However, this straightforward approach is inappropriate for on-line admission control because we have to perform schedulability checks for an unbounded number of periodic task instances in Q_P. In this section, we introduce a notion of *uniform boundedness* of utilization demands, and then use this notion to develop an $O(n)$ schedulablity test for a mixed set under EDF.

4.1 Uniform Boundedness of Utilization Demands

We first show an important property of periodic task sets. The following theorem states that if P is schedulable, then the utilization demand $U_{Q_P(t,t')}(\tau_i)$ of any task $\tau_i \in Q_P$ can never exceed the aggregate processor utilization of P.

Theorem 5. *Let* $U_P = \sum_{i=1}^{N} \frac{\tilde{e}_i}{\tilde{T}_i}$ *be the utilization of periodic task set* $P = \{\tilde{\tau}_1, \tilde{\tau}_2, \ldots, \tilde{\tau}_N\}$ *where* $\tilde{d}_i = \tilde{T}_i$. *If* P *is schedulable by EDF, then*

$$U_{Q_P(t,t')}(\tau_i) \leq U_P \tag{7}$$

for all $\tau_i \in Q_P(t,t')$ *and for all* $t < D_i$ *and* $t' \geq A_i$.

Theorem 5 demonstrates the notion of *uniform boundedness* of utilization demands. Let Ω_Q be a schedule obtained by scheduling Q with an EDF algorithm. We say that the utilization demands of Ω_Q are *uniformly bounded* if there exists a positive constant $M \leq 1$ such that $U_{Q(t,t')}(\tau_i) \leq M$ for all $\tau_i \in Q(t,t')$ and for all $t < D_i$ and $t' \geq A_i$. Thus, by Theorem 5, any pseudo-aperiodic task set Q_P has a uniform bound U_P if P is schedulable, i.e., $U_P \leq 1$.

The following theorem shows another important property. It shows that scheduling a mixed task set generates a uniformly bounded schedule if each of the individual task set has a uniform bound and if the sum of the individual bounds is no greater than 1.

Theorem 6. *For two given task sets* Q_1 *and* Q_2, *suppose that* Ω_{Q_1} *and* Ω_{Q_2} *have uniform bounds* M_1 *and* M_2, *respectively, on their utilization demands. If* $M_1 + M_2 \leq 1$, *the utilization demands of the mixed schedule* $\Omega_{Q_1 \cup Q_2}$ *are uniformly bounded by* $M_1 + M_2$.

4.2 Utilization Demand Analysis Based on Fluid-Flow Model

Theorem 5 and 6 suggest that if the processor utilization U_P of Ω_{Q_P} is no greater than 1 and the aperiodic schedule Ω_Q has a uniform utilization bound no greater than $1 - U_P$, we can guarantee all the deadlines of $Q \cup Q_P$. To do so, we need to isolate the aperiodic schedule Ω_Q from the mixed schedule $\Omega_{Q \cup Q_P}$ and bound the utilization demands of Ω_Q. This approach can be viewed as exploiting the "separation" mechanism of the fluid-flow scheduling discipline of GPS (generalized processor sharing) [11], which divides the processor bandwidth into separate flows and tasks are scheduled independently within the separate flows. Recall that we adopted EDF, not the fluid-flow GPS algorithm, for task scheduling. EDF interleaves tasks and has no separation mechanism. However, because EDF is an optimal algorithm, fluid-flow based analysis allows for safe admission tests. That is, if tasks are schedulable by the fluid-flow GPS algorithm they are also schedulable by EDF.

Suppose that a new aperiodic task τ_x arrives at time $t = A_x$. Let $Q' = Q \cup \{\tau_x\}$ and $S' = Q' \cup Q_P$. We define $U_{ad}(\tau_i)$ for any $\tau_i \in Q'$ as the pure aperiodic utilization demand that is obtained from the isolated schedule $\Omega_{Q'}$.

Here, we want to admit τ_x only if the pure aperiodic utilization demand $U_{ad}(\tau_i)$ remains uniformly bounded by $1-U_P$. It is important to note that for the mixed schedule $\Omega_{S'}$, the sum of the residual execution times of aperiodic tasks does not give the pure aperiodic demand $U_{ad}(\tau_i)$ because of the interference caused by periodic task instances. In fact, it would give a larger value than the actual $U_{ad}(\tau_i)$ since aperiodic tasks can be delayed by periodic tasks.

To compute an exact value of $U_{ad}(\tau_i)$, we first need to assume that all the aperiodic tasks are mapped to a single flow with processor bandwidth $1-U_P$, while each periodic task $\tilde{\tau}_i$ is mapped to a separate flow with processor bandwidth $\tilde{e}_i/\tilde{T}_i$. Figure 2 illustrates our fluid-flow analysis model, where three aperiodic tasks are mapped to a single flow with bandwidth $1-U_P$ and periodic tasks are mapped to separate flows collectively contributing a constant utilization demand U_P. Based on this, the exact value of $U_{ad}(\tau_i)$ can be obtained by simulating the fluid-flow GPS scheduling as proposed in [11]. But this would requires a complex algorithm like Virtual Clock algorithm [16]. Instead, in this paper we present a simpler method that can be implemented as an algorithm of $O(n)$ complexity with a very small data structure.

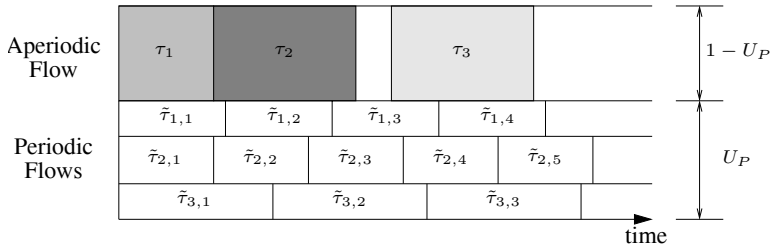

Fig. 2. Schedule of a mixed set with fluid-flow GPS discipline.

If we consider the isolated schedule $\Omega_{Q'}$, it is easy to see that the pure aperiodic demand $U_{ad}(\tau_i)$ is affected by two types of interference from other aperiodic tasks in Q'; (1) preemption by higher-priority aperiodic tasks that are admitted during the interval $[A_i, D_i]$, and (2) backlogged execution time requirements caused by previous higher-priority aperiodic tasks that were admitted before A_i but whose deadlines have not expired by A_i. If we use P_i and B_i to denote the preemption time and the backlog time, respectively, the pure aperiodic utilization demand $U_{ad}(\tau_i)$ is given by

$$U_{ad}(\tau_i) = \frac{B_i + P_i + e_i}{D_i - A_i}. \tag{8}$$

In Eq.(8), the preemption time can be easily obtained at run-time. The preemption time P_i for τ_i is initially 0 at its arrival time A_i, and is incremented at every admission of a high-priority task in the interval between its arrival time

A_i and deadline D_i. Formally, the preemption time P_i is defined by

$$P_i = \sum_{\tau_j \in PREEMPT(\tau_i)} e_j. \tag{9}$$

where $PREEMPT(\tau_i) = \{\tau_j | A_j \geq A_i \text{ and } D_j \leq D_i \text{ for } \tau_j \in Q'\}$.

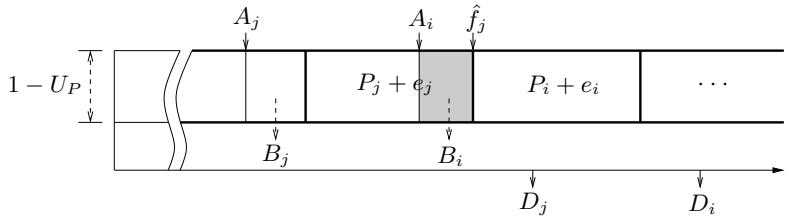

Fig. 3. Computation of backlog execution time in the isolated schedule $\Omega_{Q'}$.

We now describe how to compute backlog execution time B_i. In Figure 3, the shaded area B_i represents the backlog execution time caused by previous high-priority tasks including $\tau_{k+1}, \ldots, \tau_{j-1}, \tau_j$. Let $\hat{f}_j$ be the completion time of τ_j in the isolated schedule $\Omega_{Q'}$. Note that $\hat{f}_j$ differs from the actual completion time of τ_j in the mixed schedule $\Omega_{Q_P \cup Q'}$. We refer to $\hat{f}_j$ as the virtual completion time of τ_j. It is easy to see from Figure 3 that if $B_i \geq 0$,

$$B_i = (1 - U_P)(\hat{f}_j - A_i). \tag{10}$$

Also from Figure 3, the virtual completion time $\hat{f}_j$ of τ_j is given by

$$\hat{f}_j = A_j + \frac{B_j + P_j + e_j}{1 - U_P} = A_j + \frac{(D_j - A_j) \cdot U_{ad}(\tau_j)}{1 - U_P}. \tag{11}$$

Thus, it immediately follows that

$$B_i = U_{ad}(\tau_j) \cdot (D_j - A_j) - (1 - U_P)(A_i - A_j) \tag{12}$$

where the task τ_j is defined as the task that has the lowest priority task among the tasks in $Q((A_i), hp(\tau_i)) - \{\tau_i\}$. Since B_i is no less than 0

$$B_i = \max\{U_{ad}(\tau_j) \cdot (D_j - A_j) - (1 - U_P)(A_i - A_j), 0\}. \tag{13}$$

Eq.(8), Eq.(9), and Eq.(13) allow us to compute $U_{ad}(\tau_i)$ in an iterative manner. This means that the fluid-flow GPS simulation is not necessary in practice. Now, the following theorem summarizes our results showing that it suffices to consider only aperiodic tasks to guarantee the schedulability of a mixed set $S = Q \cup Q_P$.

Theorem 7. *A mixed set $S = Q \cup Q_P$ is schedulable by an EDF scheduler, if $U_{ad}(\tau_i) \leq 1 - U_P$ for each aperiodic task $\tau_i \in Q$.*

Using Theorem 7, one can easily determine the schedulability for a mixed task set by merely considering the utilization demands of aperiodic tasks. Note that our fluid-flow based admission test loses optimality in the sense that aperiodic tasks which may be admitted if exact EDF admission tests were done may well be rejected. However, our admission control scheme significantly reduces the complexity of exact admission test without seriously compromising the optimality. From an implementation point of view, the system has only to maintain a very small data structure for current aperiodic set $Q(t)$. On the arrival of an aperiodic task τ_x, the system only needs to compute $U_{ad}(\tau_i)$ for each $\tau_i \in Q(t) \cup \{\tau_x\}$. Thus, the admission control algorithm has a run time of $O(n)$, where n is the number of current aperiodic tasks in $Q(t)$. In the following section, we also show that the proposed scheme outperforms existing approaches with respect to achievable processor utilization.

5 Experimental Evaluation

In this section, we present simulation results to provide a performance evaluation of our admission control scheme based on utilization demand analysis (UDA) in comparison with the approaches based on synthetic utilization bounds (SYN) [1] and total bandwidth server (TBS) [14].

5.1 Experiment Setup

We implemented three admission controllers based on utilization demand analysis (UDA), synthetic utilization bounds (SYN) [1], and total bandwidth server (TBS). We also implemented a workload generator that can produce periodic and aperiodic requests with various task parameters. In our simulations, task parameters were generated as below.

- **Arrivals:** Interarrival times for aperiodic tasks were generated from a combined source of an exponential distribution and a two-state Markov Modulated Poisson Process (MMPP) to represent bursty arrival traffic. For the exponential distribution, the average interarrival times of aperiodic tasks ranged from 10 to 100 time units. The mean arrival rates of the two states of the MMPP are 10 and 1, and the mean duration periods are 100 and 10, respectively. Periods of periodic tasks were generated from a uniform distribution between 10 and 20.
- **Deadlines:** Relative deadlines of aperiodic tasks were generated from an exponential distribution with a mean value ranging from 1 and 100. Relative deadlines of periodic tasks were set equal to their periods.
- **Execution times:** Execution times of tasks were chosen from a range $[0, 50]$ such that they are less than the deadlines of the corresponding tasks.

Processor utilization was used as a performance metric in our simulations. For each simulation, we measured actual processor utilization and compared it to the generated input workload that ranged up to 150% of total processor utilization. We also used the *density* defined as $\frac{e_i}{d_i}$ to capture the tightness of deadline for task τ_i.

5.2 Results and Discussion

Figure 4 shows the measured processor utilization for mixed task sets and compares the UDA-based scheme and the SYN-based scheme. Each of mixed task set consists of 10000 aperiodic tasks and 5 periodic tasks with processor utilization $U_P = 0.1, 0.2, 0.3, 0.4$. All the task sets have the average density value of 0.1. It can be seen that the UDA-based

scheme gives a faster convergence to 100% utilization compared to the SYN-based scheme throughout the experiment, and the maximum improvement was around 13% when the average density was 0.4 and the input workload was 100%.

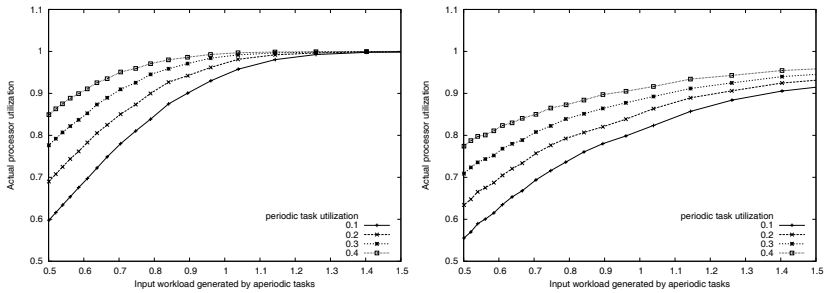

Fig. 4. Processor utilization for mixed task sets: (A) UDA-based scheme, and (B) SYN-based scheme.

Figure 5 compares the UDA-based scheme and the TBS-based scheme for mixed task sets. For this comparison, we used a common periodic utilization 0.1 for periodic tasks and varied the average density from 0.01 to 0.4. It can be seen that the UDA-based scheme achieved better processor utilization as the density value increased. The maximum improvement was around 6% when the density was 0.4 and the input workload was 100%.

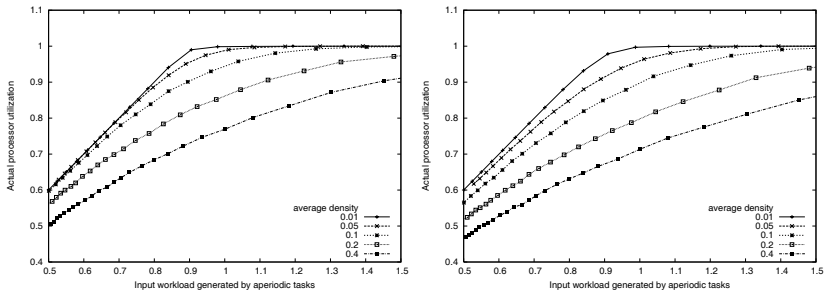

Fig. 5. Processor utilization for mixed task sets: (A) UDA-based scheme, and (B) TBS-based scheme.

6 Conclusion

In this paper, we investigated the problem of guaranteeing hard deadlines for a mixed set of periodic and aperiodic tasks. Our major contribution is two-fold. First, we introduced the notion of utilization demand, which provides a necessary and sufficient

schedulability condition for pure aperiodic task sets. Second, by combining the utilization demand analysis with the fluid-flow scheduling model, we developed an effective admission control scheme for a mixed set of periodic and aperiodic tasks.

References

1. Tarek Abdelzaher, Vivek Sharma, and Chenyang Lu. A utilization bound for aperiodic tasks and priority driven scheduling. *IEEE Transactions on Computers*, 53(3):334–350, March 2004.
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*, pages 133–137, May 1991.
3. Giorgio C. Buttazzo and Fabrizio Sensini. Optimal deadline assignment for scheduling soft aperiodic tasks in hard real-time environments. *IEEE Transactions on Computers*, 48(10):1035–1052, October 1999.
4. H. Chetto and M. Chetto. Some results of the earliest deadline first scheduling algorithm. *IEEE Transactions on Software Engineering*, 15(10):1261–1268, October 1989.
5. R. Davis, K. Tindell, and A. Burns. Scheduling slack time in fixed priority preemptive systems. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 222–231. IEEE Computer Society Press, December 1993.
6. 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*, pages 110–123, December 1992.
7. 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*, pages 166–171. IEEE Computer Society Press, December 1989.
8. J. P. Lehoczky, L. Sha, and J. Strosnider. Enhanced aperiodic responsiveness in hard real-time environments. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 261–270. IEEE Computer Society Press, December 1987.
9. J. Leung and M. Merill. A note on the preemptive scheduling of periodic, real-time tasks. *Information Processing Letters*, 11(3):115–118, November 1980.
10. C. Liu and J. Layland. Scheduling algorithm for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1):46–61, January 1973.
11. Abhay K. Parekh and Robert G. Gallagher. A generalized processor sharing approach to flow control in integrated services networks: the single-node case. *IEEE/ACM Transactions on Networking*, 1(3):344–357, June 1993.
12. Ismael Ripoll, Alfons Crespo, and Ana Garcia-Fornes. An optimal algorithm for scheduling soft aperiodic tasks in dynamic-priority preemptive systems. *IEEE Transactions on Software Engineering*, 23(6):388–400, June 1996.
13. B. Sprunt, L. Sha, and J. P. Lehoczky. Aperiodic task scheduling for hard-real-time systems. *The Journal of Real-Time Systems*, 1(1):27–60, 1989.
14. M. Spuri and G. Buttazzo. Scheduling aperiodic tasks in dynamic priority systems. *Journal of Real-Time Systems*, 10(2):1979–2012, 1996.
15. J.K. Strosnider, J.P. Lehoczky, and L. Sha. The deferrable server algorithm for enhanced aperiodic responsiveness in hard real-time environments. *IEEE Transactions on Computers*, 44(1):73–91, January 1995.
16. L. Zhang. VirtualClock: A New Traffic Control Algorithm for Packet Switching Networks. *Proc. SIGCOMM 90*, 19–29, September 1990.