Available online at www.sciencedirect.com
ScienceDirect
The Journal of Systems and Software 79 (2006) 1744–1753
The Journal of Systems and Software
www.elsevier.com/locate/jss

A faster exact schedulability analysis for fixed-priority scheduling

Wan-Chen Lu [a,*], Jen-Wei Hsieh [b], Wei-Kuan Shih [a], Tei-Wei Kuo [b]

[a] Department of Computer Science, National Tsing Hua University, 101, Kuang Fu Road, Hsinchu 300, Taiwan, ROC
[b] Department of Computer Science and Information Engineering, National Taiwan University, Taipei 106, Taiwan, ROC

Received 11 July 2005; received in revised form 16 March 2006; accepted 18 March 2006
Available online 4 May 2006

Abstract

Real-time scheduling for task sets has been studied, and the corresponding schedulability analysis has been developed. Due to the considerable overheads required to precisely analyze the schedulability of a task set (referred to as *exact schedulability analysis*), the trade-off between precision and efficiency is widely studied. Many efficient but imprecise (i.e., sufficient but not necessary) analyses are discussed in the literature. However, how to precisely and efficiently analyze the schedulability of task sets remains an important issue. The Audsley's Algorithm was shown to be effective in *exact schedulability analysis* for task sets under rate-monotonic scheduling (one of the optimal fixed-priority scheduling algorithms). This paper focuses on reducing the runtime overhead of the Audsley's Algorithm. By properly partitioning a task set into two subsets and differently treating these two subsets during each iteration, the number of iterations required for analyzing the schedulability of the task set can be significantly reduced. The capability of the proposed algorithm was evaluated and compared to related works, which revealed up to a 55.5% saving in the runtime overhead for the Audsley's Algorithm when the system was under a heavy load.

Keywords: Real-time systems; Schedulability analysis; Periodic tasks; Fixed-priority preemptive scheduling

1. Introduction

Real-time scheduling for task sets has been studied and the corresponding schedulability analysis has been developed. Most of the real-time scheduling algorithms are priority driven at the task-level and can be categorized into fixed-priority and dynamic-priority scheduling algorithms. For fixed-priority scheduling, priority of a task never changes. For dynamic-priority scheduling, priority of a task can vary with time. Tasks are composed of jobs; obviously, priorities of all jobs are fixed for fixed-priority tasks. However, dynamic-priority scheduling can be further divided at the job-level into fixed-priority job scheduling and dynamic-priority job scheduling. Various optimal algorithms for both fixed-priority and dynamic-priority scheduling have been proposed: Rate Monotonic (RM) algorithm (Liu and Layland, 1973) and *Deadline Monotonic* (DM) algorithm (Audsley et al., 1991) are well-known optimal fixed-priority algorithms, while Earliest Deadline First (EDF) algorithm (Liu and Layland, 1973) is an optimal job-level fixed-priority scheduling, and Least Slack Time First (LST) algorithm (Liu and Layland, 1973) is an optimal job-level dynamic-priority scheduling. Although dynamic-priority scheduling algorithms can achieve a better system utilization, the implementation of dynamic-priority scheduler is much more complex. Thus fixed-priority scheduling algorithms are widely used in modern real-time systems, since it can be easily implemented on top of commercial kernels that provide a limited number of priority levels (Bini and Buttazzo, 2002).

Schedulability analysis is used to verify whether every task in a given task set can meet deadline when scheduled according to the adopted scheduling algorithm. This paper focuses on the schedulability analysis for a fixed-priority scheduling algorithm due to its popularity in modern real-time systems. In this paper, task sets are scheduled

* Corresponding author. Tel.: +886 3 5742808.
E-mail addresses: wlu@cs.nthu.edu.tw, wanchen@rtlab.cs.nthu.edu.tw (W.-C. Lu).

0164-1212/$ - see front matter © 2006 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2006.03.023

according to RM algorithm, one of the optimal fixed-priority scheduling algorithms. Schedulability analysis can be categorized into three main classes according to their preciseness. *Necessary but insufficient analysis* is based on the processor utilization factor: total utilization factor of a task set shall never exceed one, or some tasks will miss their deadlines (Liu and Layland, 1973). Though such an analysis is simple and efficient, it is too optimistic to be helpful in schedulability analysis. *Sufficient but not necessary analysis* is more precise than the first (but still might provide a false-negative) while more efficient than the third (Liu and Layland, 1973; Han and Tyan, 1997; Bini et al., 2001). *Sufficient and necessary analysis*, i.e., *exact schedulability analysis*, provides an exact verification of task-set schedulability, but it also introduces higher analysis overheads and is the least efficient but the most precise.

Efficient *exact schedulability analysis* is required by various service-critical systems, such as tele-medicine systems, tele-conferencing systems, multimedia services with Quality-of-Service (QoS) requirements (Vin et al., 1994; Kuo et al., 1997), and real-time traffic scheduling over networks (Wong and Cheng, 1997; Miller and Cheng, 2000; Rao and Cheng, 2000; Cheng and Rao, 2003). It is also important for industry, e.g., HVE consumer products of Philips International, Inc., mentioned in (Bril et al., 2003). Researchers (Audsley et al., 1991; Bini and Buttazzo, 2002; Bril et al., 2003) proposed ways to improve runtime of *exact schedulability analysis*. In particular (Bini and Buttazzo (2002)) propose an adaptive schedulability-analysis framework that can be tuned through a parameter to balance the runtime and the preciseness. Based on a well-known *exact schedulability analysis* proposed by Audsley et al. (1993), referred to as the *Audsley's Algorithm*, Bril et al. (2003) modified its initial setup to jump-start the analysis process.

This paper focuses on reducing the runtime overhead of the Audsley's Algorithm. In contract to (Bril et al., 2003), improvements are imposed throughout the analysis process, not just confined to the initial setup. By properly partitioning a task set into two subsets and differently treating these two subsets during each iteration, the number of iterations required for analyzing the schedulability of the task set can be significantly reduced. Although the proposed algorithm does introduce additional cost in the partitioning of tasks in each iteration and some temporary storage in the partitioning procedure. The extra temporary storage only costs few more variables. The capability of the proposed method was evaluated and compared to the related work (Audsley et al., 1993; Bril et al., 2003), which revealed up to a 55.5% saving in the runtime overhead for the Audsley's Algorithm when the system was under a heavy load.

The rest of this paper is organized as follows. In Section 2, task model, basic concept of the *exact schedulability analysis*, and some terminologies are addressed. Section 3 presents a faster *exact schedulability analysis* for fixed-priority scheduling and proves its correctness. Section 4 provides the simulation results to evaluate the capability of the proposed method. Section 5 is the conclusion and future work.

2. Task model and definitions

This paper focuses on *exact schedulability analysis* of rate monotonic scheduling for periodic tasks over uni-processor systems. A periodic task is usually characterized by four parameters: release time, period, maximum computation time, and relative deadline. To concentrate on the proposed method, the task model is simplified by assuming that the first jobs of all tasks are released at the same time, and the period is equal to the relative deadline for each task. Since the schedulability analysis for tasks with arbitrary release times can be reduced to one for tasks with the same release time, it is admissible to assume all tasks are released simultaneously.

Definition 1 (*Task model and task set*). A periodic task τ_i comprises an infinite sequence of jobs with regular release times, where the task is a template of its corresponding jobs and one job of the task arrives at the beginning of each period. $J_{i,j}$ denotes the jth job of the task τ_i. Let $T_n = \{\tau_1 = (c_1, p_1), \tau_2 = (c_2, p_2), \ldots, \tau_n = (c_n, p_n)\}$ be a set of periodic tasks, where c_i is the maximum computation time, and p_i is the period of task τ_i. The utilization of τ_i, referred to as u_i, is defined as c_i/p_i. All jobs are ready when they arrive and should be finished before the arrival of next job. Without loss of generality, $p_1 \leqslant p_2 \leqslant \cdots \leqslant p_n$ is assumed.

As mentioned in Section 1, this paper focuses on schedulability analysis for task sets under rate monotonic scheduling. First, the concept of rate monotonic scheduling is addressed.

Definition 2 (*Feasible and schedulable*). Suppose a task set is scheduled by S scheduling algorithm, the schedule for the task set is feasible if all the jobs in each task can meet their corresponding deadlines under S scheduling. A task set is schedulable if there exists a feasible schedule.

Definition 3 (*Rate monotonic scheduling* (*Liu and Layland*, 1973)). Rate monotonic (RM) algorithm is an optimal fixed-priority scheduling algorithm, which means RM algorithm can always generate a feasible schedule if a task set is schedulable. Under RM scheduling, the individual task priorities are assigned inversely proportional to their respective periods. In other words, a task with shorter period is assigned with a higher priority (and can be scheduled earlier).

Let $T_{m-1} = \{\tau_1, \tau_2, \ldots, \tau_{m-1}\} = T_m - \{\tau_m\}$ be a subset of T_m. To analyze the schedulability of a task set T_n, a schedulability analysis is applied on T_i iteratively, for $i = 1, \ldots, n$, to check whether the lowest-priority task in T_i is schedulable. For efficiency consideration, a *sufficient but not necessary analysis* is usually applied first. When the analysis reports that the task set might be unschedulable, an *exact schedulability analysis* takes over the verification process, since the report of the *sufficient but not necessary analysis* might be a false negative. To facilitate the discussion in the remainder of this paper, we assume

that tasks in T_{n-1} are all schedulable and confine our attention to verifying whether the lowest-priority task in T_n (i.e., τ_n under RM) is schedulable.

Lehoczky et al. developed an algorithm to precisely analyze whether a task set can be feasibly scheduled under RM scheduling (Lehoczky et al., 1989). The basic idea of this algorithm is to check if the demand (total computation time demanded by the task set) is no larger than the supply (available processing time) at some integer multiples of task periods. Audsley et al. (1993) proposed an alternative for *exact schedulability analysis* by constructing a schedule and then observing whether the task set is schedulable.

Definition 4 (*Audsley's* Algorithm (Audsley et al., 1993)). Under the Audsley's Algorithm, the worst-case response time r_i of each task τ_i is derived by iteratively calculating the formula $r^{(l+1)} = c_i + \sum_{k=1}^{i-1} \left\lceil \frac{r^{(l)}}{p_k} \right\rceil \cdot c_k$ until $r^{(l+1)}$ either converges to a real number (i.e., r_i) or exceeds the deadline of task τ_i. A task is schedulable if its worst-case response time is no larger than its deadline; otherwise, the task is unschedulable. Note that tasks are sorted in a non-decreasing priority order (according to Definitions 1 and 3) and $r^{(0)} = \sum_{j=1}^{i} c_j$.

3. A faster exact schedulability analysis

3.1. Overview

This research is motivated by the needs in reducing the runtime of *exact schedulability analysis* algorithms in a tool design for hardware/software co-designs. *Exact schedulability analysis* algorithms could be applied to many system design tools, e.g., schedulability analyses in some electronic design automation (EDA) tools (Richter et al., 2003; Pop et al., 2000, 2004), where an *exact schedulability analysis* algorithm might be executed once per iteration. The improvement on the performance of *exact schedulability analysis* algorithms could significantly improve the performance of such tools. Note that although such tools are also executed in an off-line fashion, the performance of the *exact schedulability analysis* algorithms does have a scalability problem when the number of tasks is large. The needs are strong, especially when the design procedure of embedded systems goes iteratively to seek a feasible solution.

The basic idea is to attempt a larger jump in the derivation of each subsequent time instant $r^{(l)}$, such that the number of iterations required by the Audsley's Algorithm can be reduced. It is of paramount importance that the extra overhead introduced by such attempt should not be large, or the proposed algorithm loses its mission. Section 3.3 proves that the time complexity of the proposed algorithm in each iteration is $O(n)$. The overall runtime overhead is less than the Audsley's Algorithm as well, as illustrated in Section 4. To achieve this goal, the Audsley's Algorithm is modified accordingly, referred to as the Enhanced Audsley's Algorithm (EAA).

In the Audsley's Algorithm, the next time instant $r^{(l)}$ is derived by calculating the total computation requirements of all jobs that are ready before the current time instant $r^{(l-1)}$. To have a larger jump in the derivation of $r^{(l)}$, the EAA assumes that only a proportion α of processing time is available for executing real-time tasks. For the rest of the tasks, the reserved computation times are proportional to their corresponding utilizations. To be more specific, suppose task τ_i is such a task, the total computation time assigned to the jobs in τ_i before time instant t is set to $t \cdot c_i/p_i$, which is less than the actual total computation requirements, i.e., $\lceil t/p_i \rceil \cdot c_i$, of jobs in τ_i before time instant t.

The major modifications in the EAA are summarized as follows:

(1) *Partition the task set into two subsets*: During each iteration, all the tasks in the task set T_n are partitioned into two sets, namely L and R ($=T_n - L$). Note that the partitions might differ in different iterations.
(2) *Calculate the proportion of the available processing time*: Let $\alpha = 1 - \sum_{\tau_i \in L} u_i$, where $u_i = c_i/p_i$. The proportion of the available processing time for L is $1 - \alpha$, while the remaining processing time is reserved for tasks in R.
(3) *Find the next time instant*: The formula used to derive the worst-case response time in the Audsley's Algorithm is modified to

$$r^{(l)} = \frac{\sum_{\tau_i \in R} \lceil r^{(l-1)}/p_i \rceil \cdot c_i}{\alpha} = \frac{\sum_{\tau_i \in R} \lceil r^{(l)}/p_i \rceil \cdot c_i}{1 - \sum_{\tau_j \in L} u_j}.$$

The differences between the Audsley's Algorithm and the enhanced one can be illustrated by the following example: given a task set $T = \{\tau_1 = (1.5, 2), \tau_2 = (1.5, 10)\}$, $r^{(0)}$ is calculated by $\sum_{i=1}^{2} c_i = 3$. The next time instant $r^{(1)}$ determined by the Audsley's Algorithm would be $1.5 + \lceil 3/2 \rceil \cdot 1.5 = 4.5$. In the EAA, $r^{(0)}$ is also set as 3. By partitioning T into $L = \{\tau_1\}$ and $R = \{\tau_2\}$, $r^{(1)}$ is derived as $1.5/\alpha = 1.5/0.25 = 6$, which is larger than the one derived by the Audsley's Algorithm. The technical issue is how to properly partition a task set T into L and R for the EAA such that the jump in each derivation of $r^{(l)}$ can be maximized. Section 4 discusses the impact of adopting different partition rules.

3.2. Enhanced Audsley's algorithm (EAA)

Although the Audsley's Algorithm has been shown to be effective in the *exact schedulability analysis* for task sets under rate-monotonic scheduling, the needs in reducing its runtime is demanding. This section discusses in detail the enhancement over the Audsley's Algorithm. Listing 1 shows the pseudo-code of the EAA. Four input parameters are provided (Step 1) where P[ ], C[ ], and U[ ] are arrays used to store the periods, the maximum computation times, and the corresponding utilizations of tasks in the task set, and TaskNo records the number of tasks in the task set. The EnhancedAA( ) declares six variables (Steps 2 and 3)

where NextTimeInstant is used to keep the next time instant $r^{(l)}$ in the lth iteration; tmpNextTimeInstant and oldNextTimeInstant are temporal space for the possible $r^{(l)}$ and the derived $r^{(l-1)}$, respectively; diffJmp assists in exclusively partitioning the task set into two subsets, and M and UL are auxiliary for deriving $r^{(l)}$.

Listing 1: pseudo-code of the EAA

```
1  int EnhancedAA(float P[ ], float C[ ], float U[ ],
   int TaskNo) {
2     float NextTimeInstant = 0, tmpNextTimeInstant,
      oldNextTimeInstant, diffJmp;
3     float M, UL;
4
5     for (int i = 0; i < TaskNo; i++) {
6         NextTimeInstant = NextTimeInstant + C[i];
7     }
8     diffJmp = NextTimeInstant;
9     do{
10      M = UL = 0;
11      oldNextTimeInstant = NextTimeInstant;
12
13      for (i = 0; i < TaskNo; i++) {
14           if (ceil(oldNextTimeInstant/P[i])*P[i]<
15              oldNextTimeInstant + Ratio*diffJmp) {
16              UL+ = U[i];
17           } else {
18                M+ = (ceil(oldNextTimeInstant/
                  P[i])*C[i]);
19           }
20      }
21
22      tmpNextTimeInstant = (float)(M/(1.0 − UL));
23      if (tmpNextTimeInstant< = oldNextTimeInstant)
   {
24         NextTimeInstant = 0;
25         for (i = 0; i < TaskNo; i++) {
26              NextTimeInstant+ = (ceil(oldNextTimeIn-
                stant/P[i])s*C[i]);
27         }
28      } else {
29          NextTimeInstant = tmpNextTimeInstant;
30      }
31
32      diffJmp = NextTimeInstant − oldNextTimeInstant;
33      if (NextTimeInstant > P[TaskNo − 1]) {
34          return UNSCHEDULABLE;
35      }
36      if (NextTimeInstant == oldNextTimeInstant){
37         lastResponseTime = NextTimeInstant;
38         return SCHEDULABLE;
39      }
40    } while ((NextTimeInstant ! = oldNextTimeInstant)
   &&
41             (NextTimeInstant< = P[TaskNo − 1]));
42    }
```

Initially, $r^{(0)}$ is set as the summation of the maximum computation times of all tasks since they are released simultaneously (Steps 5–7); diffJmp keeps the difference between the previous two valid jumps, and diffJmp is set to $r^{(0)}$ when $l = 1$ (Step 8). In Steps 9–40, the EAA analyzes the schedulability of the task set by iteratively deriving the next time point to find out the worst-case response time of the task set. In the beginning of each iteration, M and UL are initialized to 0, and the previously derived $r^{(l-1)}$ is reserved in oldNextTimeInstant (Steps 10–11). In Steps 13–20, the task set is partitioned into two subsets L and R. For each task, if the release time of its first newly arrived job, i.e., $\lceil r^{(l-1)}/p_i \rceil \cdot p_i$, is less than the threshold, i.e., $r^{(l-1)} + Ratio \cdot diffJmp$, the task belongs to L; otherwise, the task belongs to R. A system defined parameter, *Ratio*, is a positive real number ranging from 0 to 1. When *Ratio* is set to 0, all the tasks belong to R, and the EAA operates totally the same as the Audsley's Algorithm. In Section 4, performances with different settings of *Ratio* are demonstrated, and a feasible setting is suggested. The contribution of each task to $r^{(l)}$ is counted accordingly (Step 16 or 18).

The next time instant $r^{(l)}$ can then be determined (Steps 22–30). A possible next time instant is calculated (Step 22). The newly calculated time instant might be no larger than the previously derived one, i.e., $r^{(l-1)}$, due to the task-set partition scheme (Step 23). When such situation occurs, all the tasks are put into R, and $r^{(l)}$ is re-calculated (Steps 24–27); otherwise, the value calculated in Step 22 is set as the next time instant (Step 29). When $r^{(l)}$ needs to be re-calculated, the EAA is penalized by an extra iteration count. After the next time instant $r^{(l)}$ is determined, new diffJmp for the next iteration can be set (Step 32). Finally, the schedulability of the task set is checked (Steps 33–39). If $r^{(l)}$ exceeds the deadline, UNSCHEDULABLE is returned. If the next time instant converges (and does not exceed the deadline), SCHEDULABLE is returned (Steps 36–39). Otherwise, a new iteration is started (Steps 40 and 41).

The effectiveness and efficiency of the EAA are illustrated by two examples. Example 1 demonstrates the details of the EAA. Example 2 reveals how the EAA can achieve a large improvement in the number of iterations.

Example 1. Given a task set $T_3 = \{\tau_1 = (2,4),\ \tau_2 = (1,5),\ \tau_3 = (3.3,15)\}$, the schedulability of T_3 is analyzed by the EAA. Suppose τ_1 and τ_2 have already been verified as schedulable, and *Ratio* is set as 0.5, the operation of the EAA is as follows: Initially, $r^{(0)}$ is set to 6.3 $\left(= \sum_{i=1}^{3} c_i\right)$. The release times of the newly arrived job for τ_1, τ_2, and τ_3 are 8 $(=\lceil 6.3/4 \rceil \cdot 4)$, 10 $(=\lceil 6.3/5 \rceil \cdot 5)$, and 15 $(=\lceil 6.3/15 \rceil \cdot 15)$, respectively. Since the threshold is 9.45, T_3 is partitioned into $L = \{\tau_1\}$ and $R = \{\tau_2, \tau_3\}$. The next time instant $r^{(1)} = (2 + 3.3)/(1 - 0.5) = 10.6$ is then derived accordingly. Table 1 shows the related information in deriving the worst-case response time of $J_{3,1}$, in which the third to the fifth columns indicate the release times of the newly arrived job for τ_1, τ_2, and τ_3, respectively. Note that since $r^{(4)}$ is less than $r^{(3)}$, it is re-calculated by putting all the

Table 1
The related information in deriving the worst-case response time of $J_{3,1}$

	Threshold	τ_1	τ_2	τ_3	L	R	$r^{(l)}$
$l=1$	$6.3+6.3\cdot 0.5=9.45$	8	10	15	$\{\tau_1\}$	$\{\tau_2,\tau_3\}$	10.6
$l=2$	$10.6+(10.6-6.3)\cdot 0.5=12.75$	12	15	15	$\{\tau_1\}$	$\{\tau_2,\tau_3\}$	12.6
$l=3$	$12.6+(12.6-10.6)\cdot 0.5=13.6$	16	15	15	$\emptyset$	$\{\tau_1,\tau_2,\tau_3\}$	14.3
$l=4$	$14.3+(14.3-12.6)\cdot 0.5=15.15$	16	15	15	$\{\tau_2,\tau_3\}$	$\{\tau_1\}$	13.79
$l=5$	N/A	N/A	N/A	N/A	$\emptyset$	$\{\tau_1,\tau_2,\tau_3\}$	14.3
$l=6$	$14.3+(14.3-14.3)\cdot 0.5=14.3$	16	15	15	$\emptyset$	$\{\tau_1,\tau_2,\tau_3\}$	14.3

tasks into R, and the EAA is penalized by an extra iteration count. When a termination condition, i.e., $r^{(5)}=r^{(6)}$, is satisfied, the derivation of the worst-case response time for $J_{3,1}$ is accomplished. Since the worst-case response time of $J_{3,1}$ is less than its corresponding deadline, T_3 is reported as "schedulable".

The efficiency of the EAA (compared with the Audsley's Algorithm) can be illustrated by an example in flash-memory storage systems. Flash-memory technology is becoming critical in building embedded systems applications because of its shock-resistant, power economic, and nonvolatile nature. Since flash memory is a write-once and bulk-erase medium, a garbage-collection mechanism is required to provide applications a transparent storage service. For a time-critical system, such as manufacturing systems, it is highly important to predict the number of free pages reclaimed after each block recycling so that the system will never be blocked for an unpredictable duration of time because of garbage collection. In Chang et al. (2004), a garbage-collection mechanism is modelled as a periodic task to provide a guaranteed performance for hard real-time systems. Note that the period of such task is much larger compared with read/write tasks.

Example 2. For a flash-memory storage system applied in database accesses, suppose its write task and read task can be modelled as $\tau_1=(1.6,2)$ and $\tau_2=(0.76,4)$, and the garbage-collection mechanism is modelled as a periodic task $\tau_3=(3,301)$ as well. The schedulability of such task set $T_3=\{\tau_1=(1.6,2),\tau_2=(0.76,4),\tau_3=(3,301)\}$ is analyzed. To be focused, assume τ_1 and τ_2 have been verified as schedulable. Under the EAA, the derivation of the worst-case response time for $J_{3,1}$ is as follows: initially, $r^{(0)}$ is set to 5.36 $\left(=\sum_{i=1}^{3}c_i\right)$. By partitioning T_3 into $L=\{\tau_1,\tau_2\}$ and $R=\{\tau_3\}$, $r^{(1)}=3/(1-0.99)=300$ is obtained. Since $r^{(2)}$, which is derived as 0 by partitioning T_3 into $L=\{\tau_1,\tau_2,\tau_3\}$ and $R=\emptyset$, is less than $r^{(1)}$, it is re-calculated by putting all the tasks into R, and the EAA is penalized by an extra iteration count. Finally, by partitioning $L=\emptyset$ and $R=\{\tau_1,\tau_2,\tau_3\}$, $r^{(4)}$ is equal to 300. Now that $r^{(3)}$ is equal to $r^{(4)}$ and the worst-case response time of $J_{3,1}$ is less than its corresponding deadline, the termination condition is met and the task set T_3 is reported as "schedulable". The schedulability analysis for τ_3 under the EAA is finished in four iterations. On the other hand, when the Audsley's Algorithm is applied, the number of required iterations increases dramatically to 117.

3.3. Properties

This section explores the properties of the EAA from a theoretical perspective. The correctness of the EAA is proved by Lemmas 1, 2, and Theorem 1. It is of paramount importance that the extra overhead introduced by the EAA should not be large. Theorem 2 demonstrates that the time complexity in each iteration is $O(n)$.

Lemma 1. *For every time instant $r^{(l)}$ derived by the EAA during the lth iteration, the time interval $(0,r^{(l)})$ has no idle time.*

Proof. This lemma will be proved by induction.

Induction basis. For $l=0$, $r^{(0)}=\sum_{i=1}^{n}c_i$. It is obvious that no idle time exists in $(0,\sum_{i=1}^{n}c_j)$ because the first job of each task τ_i, for $i=1$ to n, is ready for execution at time 0.

Induction hypothesis. Assume that the lemma is true for the time instant $r^{(k)}$; that is, no idle time exists in the time interval $(0,r^{(k)})$. It needs to be proved that the lemma is also true for the time instant $r^{(k+1)}$.

Induction step. The induction step is proved by a contradiction. Suppose that an idle time occurred in $(r^{(k)},r^{(k+1)})$. Since there is no idle time before the worst-case response time derived by the Audsley's Algorithm (denoted by t), t must be the first idle time in $(r^{(k)},r^{(k+1)})$, i.e., $r^{(k)}\leqslant t<r^{(k+1)}$. Without loss of generality, let $r^{(k)}=t-\varepsilon$, where $\varepsilon\geqslant 0$. According to the EAA, $r^{(k+1)}$ is derived by $\sum_{\tau_j\in R}\lceil(t-\varepsilon)/p_j\rceil\cdot c_j/\left(1-\sum_{\tau_i\in L}u_i\right)$. Because $r^{(k+1)}>t$, the following inequalities can be derived:

$$\begin{aligned}&\frac{\sum_{\tau_j\in R}\lceil t/p_j\rceil\cdot c_j}{1-\sum_{\tau_i\in L}u_i}\geqslant\frac{\sum_{\tau_j\in R}\lceil(t-\varepsilon)/p_j\rceil\cdot c_j}{1-\sum_{\tau_i\in L}u_i}>t\\&\sum_{\tau_j\in R}\lceil t/p_j\rceil\cdot c_j+t\cdot\sum_{\tau_i\in L}u_i>t\\&\sum_{\tau_j\in R}\lceil t/p_j\rceil\cdot c_j+\sum_{\tau_i\in L}(t/p_i)\cdot c_i>t\end{aligned}\qquad(1)$$

On the other hand, since t is the worst-case response time of $J_{n,1}$ derived by the Audsley's Algorithm, the following equation can be obtained:

$$\sum_{\tau_k\in T_n}\lceil t/p_k\rceil\cdot c_k=\sum_{\tau_j\in R}\lceil t/p_j\rceil\cdot c_j+\sum_{\tau_i\in L}\lceil t/p_i\rceil\cdot c_i=t\qquad(2)$$

Based on (1) and (2), the following inequality is derived: $t=\sum_{\tau_j\in R}\lceil t/p_j\rceil\cdot c_j+\sum_{\tau_i\in L}\lceil t/p_i\rceil\cdot c_i\geqslant\sum_{\tau_j\in R}\lceil t/p_j\rceil\cdot c_j+$

$\sum_{\tau_i \in L}(t/p_i) \cdot c_i > t$, which is a contradiction. Thus, it can be concluded that no idle time occurs in $(r^{(k)}, r^{(k+1)})$.

According to the induction step, the induction is completed, and hence the lemma is proved. □

Lemma 2. *When the EAA reports that the task τ_n is "schedulable", τ_n is indeed schedulable.*

Proof. The lemma is proved by a contradiction. Let $r^{(l)}$ be the derived time instant when the EAA reports "schedulable" in the lth iteration, but there is no idle time in the time interval $(r^{(l)}, r^{(l)+})$ in the RM schedule. (Here $r^{(l)+}$ means some time instant later than $r^{(l)}$.) According to the EAA, the variable diffJmp approaches 0. When the value of diffJmp is sufficiently small, the EAA will eventually put all the tasks into R and leave L empty. As a result, the inequality $r^{(l+1)} = \sum_{\tau_j \in R}(\lceil r^{(l)}/p_j \rceil \cdot c_j) = \sum_{\tau_j \in (\tau_1,\ldots,\tau_n)}(\lceil r^{(l)}/p_j \rceil \cdot c_j) > r^{(l)}$ is obtained, which means at least one new job of some task is ready at time $r^{(l)}$. Therefore, if the while-loop in the EAA executes one more time, a new time point that is larger than or equal to $r^{(l)+}$ is obtained. This is a contradiction to the "schedulable" termination condition of the EAA. □

Theorem 1. *Assume T_{n-1} is schedulable. T_n is schedulable if and only if the EAA reports T_n as "schedulable".*

Proof. ($\Rightarrow$) If T_n is schedulable, there would be an idle time in the time interval $(0, p_n)$. According to Lemma 2, the EAA will terminate and report "schedulable".

($\Leftarrow$) If T_n is unschedulable, there would be no idle time in the time interval $(0, p_n)$. According to Lemma 2, the EAA cannot find an idle time. That is, the time instant will keep increasing until exceeding p_n, and the EAA will report T_n as "unschedulable". □

Theorem 2. *The EAA requires $O(n)$ time to find the next time instant $r^{(l)}$.*

Proof. There are two parts for executing each iteration in the EAA: (1) For each task in T_n, the EAA takes constant time in determining to which subset (L or R) the task belongs. Since there are n tasks in T_n, it takes O(n) time for the EAA to partition all tasks into L and R. (2) In the lth iteration, the EAA takes O(n) time to calculate $r^{(l)}$ according to the partition result in (1). Based on the above, the EAA has O(n) time complexity in each iteration. □

4. Performance evaluation

4.1. Metrics, experimental setup, and data sets

This section assesses the proposed idea (i.e., the EAA) in relation to runtime speedup. The EAA is compared with another related *exact schedulability analysis* algorithms, i.e., the Audsley's Algorithm (Audsley et al., 1993). Both are then modified by the Initial Value Improvement method proposed in Bril et al. (2003). These two comparisons are made in terms of the runtime overhead and the number of required iterations needed for each algorithm, where each iteration derives $r^{(l)}$ based on the result of $r^{(l-1)}$.

The primary performance metric is the Runtime Ratio of the algorithms. Let x be the runtime for the EAA, and let y be the runtime for the Audsley's Algorithm. Both simulations are running schedulability analyses for 10,000 task sets. The Runtime Ratio on runtime is defined as follows: x/y. The Iteration Ratio on the number of required iterations is defined in a similar way. The comparison is repeated by modifying both algorithms by the Initial Value Improvement method proposed in Bril et al. (2003).

The task sets for performance evaluation were generated based on benchmarks and systems reported in Molini et al. (1990), Kamenoff and Weiderman (1991), Locke et al. (1991), Kim et al. (1996). A random number generator was adopted to generate task sets: the number of tasks per task set was randomly chosen between 10 and 30. The number of fundamental frequencies was a real number within a range $[1/4, 1]$ multiplied by the number of tasks in the task set. Since a task set with a utilization factor of no more than 69% would be schedulable according to the Liu and Layland bound (Liu and Layland, 1973), the data sets for the experiments were generated with a utilization factor between 75% and 100%. The experiments started with each task set with a utilization factor equal to 75%, and were repeated for sets with the utilization factor increasing in increments of 5% until 100% is reached. The utilization factor of each task was no more than 20% of the total utilization factor of its task set. Each task was assigned fundamental frequencies randomly, where the possibility of assigning i fundamental frequencies to a task was $(1/2)^{i-1}$. The period of each task was derived by the multiplication of each of its assigned fundamental frequencies. A total of 10,000 task sets were tested for each utilization factor.

4.2. Experimental results

Each *exact schedulability analysis* algorithm was evaluated experimentally in the same way. The schedulability of each task in a task set was verified in a non-increasing order of task priority. The *sufficient but not necessary analysis* proposed by Liu and Layland (1973) was applied first to verify the schedulability of tasks until some task failed the verification. If the ith task failed the verification, the schedulability of all the remaining tasks (with an index of no less than i) were verified by *exact schedulability analysis* algorithms in the experiments. Table 2 shows the average percentage of tasks in a task set being verified by *exact schedulability analysis* algorithms in the experiments.

4.2.1. EAA compared with the Audsley's algorithm

Fig. 1 demonstrates the iteration overheads for the EAA against the Audsley's Algorithm. The performance differences of the EAA under different settings of the Ratio (as indicated in the legend) are also illustrated. As mentioned

Table 2
The average percentage of tasks in a task set remaining to be verified

	Utilization factor of task set					
	75%	80%	85%	90%	95%	100%
Average percentage	14.42%	18.30%	21.59%	25.05%	28.36%	31.68%

in Section 3, different settings of Ratio result in different task-set partitions, from which the value of $r^{(l)}$ derived in the lth iteration varies. As shown in Fig. 1, a lower Iteration Ratio over the number of required iterations was achieved when the utilization factor of a task set was large. This is because the EAA tends to jump farther in such cases, compared with the Audsley's Algorithm. The simulation result shows that the performance improvement of the Iteration Ratio is maximized when the value of Ratio in the EAA was set to 0.2. The mean and standard derivation of the number of iterations for the Audsley's Algorithm and the EAA (with Ratio = 0.2) are shown in Table 3.

Fig. 2 shows the Runtime Ratio for the EAA against the Audsley's Algorithm in terms of the runtime needed for each algorithm. The measuring of runtime for each algorithm was done by taking advantage of an Intel supported instruction RDTSC, which can measure runtime in the unit of CPU cycles (Intel, 1997). The EAA produces up to a 55.5% saving in the runtime overhead for the Audsley's Algorithm when the system was in a heavy load. (The Runtime Ratio for the EAA against the Audsley's Algorithm ranged from 44.5% to 68% when the value of Ratio in the EAA was set to 0.2.) Table 4 shows the mean and standard derivation of the runtime for the Audsley's Algorithm and the EAA (with Ratio = 0.2). Note that the unit of runtime was a CPU cycle.

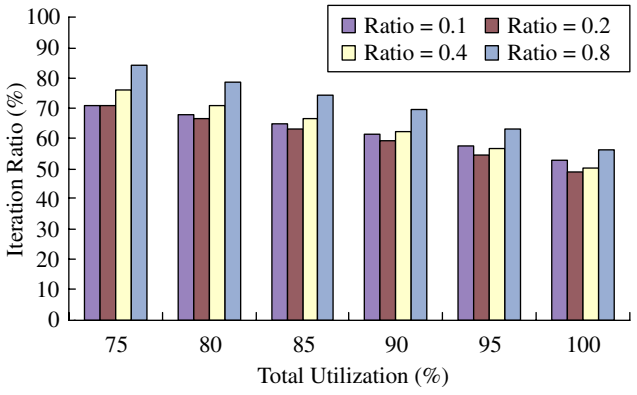

Fig. 1. Comparison in the iteration ratio.

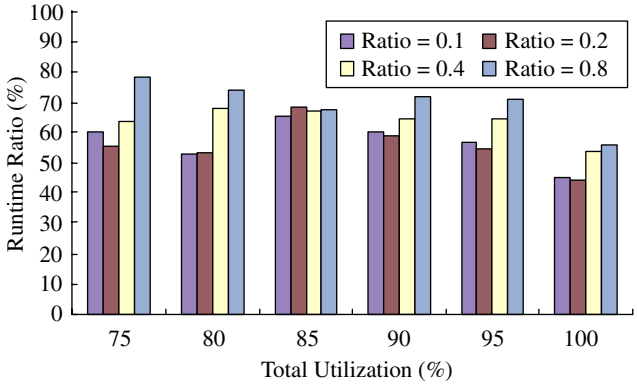

Fig. 2. Runtime ratio comparison.

Based on Figs. 1 and 2, the EAA would significantly outperform the Audsley's Algorithm when the value of Ratio in the EAA was small, e.g., 0.2. Table 5 shows the performance of the EAA. Among 10,000 Task Sets, the EAA (with Ratio = 0.2) outperformed the Audsley's Algorithm in most cases for both the number of iterations and the runtime.

4.2.2. Initial value improvement

In Bril et al. (2003), Bril et al. modified the initial setup for the Audsley's Algorithm to jump-start the analysis process: $r^{(0)}$ is set to $\max\left(c_i/\left(1-\sum_{j=1}^{i-1}u_j\right), r_{n-1}+c_i\right)$, in which r_{n-1} is the previously derived worst-case response time for $J_{n-1,1}$. To complete the study of this paper, the idea in Bril et al. (2003) is adopted in both the Audsley's Algorithm and the EAA. With the initial value improvement, Fig. 3 demonstrates the iteration overheads for the EAA against the Audsley's Algorithm. The simulation result shows that the performance improvement of the Iteration Ratio is better when the value of Ratio set in the EAA was small. The mean and standard derivation of the number of iterations with initial value improvement for both the Audsley's Algorithm and the EAA (with Ratio = 0.2) are shown in Table 6.

With the initial value improvement, Fig. 4 shows the Runtime Ratio for the EAA against the Audsley's Algorithm in terms of the runtime needed for each algorithm.

Table 3
The mean and standard deviation of the number of iterations for the Audsley's Algorithm and the EAA (with Ratio = 0.2)

		Utilization factor of task set					
		75%	80%	85%	90%	95%	100%
Audsley's Algorithm	Mean	48.66320	63.37820	79.31720	98.97070	125.86380	137.11110
	Standard deviation	36.93568	34.94621	36.58553	39.43057	48.06183	58.91121
EAA	Mean	32.34480	39.84970	47.59280	56.10830	65.90040	64.44790
	Standard deviation	26.94133	24.46076	24.13280	24.00492	26.17973	28.21729

Table 4
The mean and standard deviation of runtime (in CPU cycles) of the Audsley's Algorithm and the EAA (with Ratio = 0.2)

		Utilization factor of task set					
		75%	80%	85%	90%	95%	100%
Audsley's Algorithm	Mean	344,752.69	425,897.81	524,472.73	644,888.71	814,831.38	879,427.18
	Standard deviation	350,480.30	341,917.86	400,578.73	456,076.75	528,122.48	590,828.24
EAA	Mean	222,747.47	259,801.81	303,035.39	348,551.87	406,775.73	394,669.40
	Standard deviation	247,988.65	232,609.10	241,294.58	236,444.57	289,069.98	303,109.62

Table 5
The efficiency of the EAA (with Ratio = 0.2)

		Utilization factor of task set					
		75%	80%	85%	90%	95%	100%
Iteration	EAA is better	9993	10,000	10,000	9999	10,000	9998
	Equal	7	0	0	0	0	2
	EAA is worse	0	0	0	1	0	0
Runtime	EAA is better	9964	9986	9993	9991	9993	9982
	EAA is worse	36	14	7	9	7	18

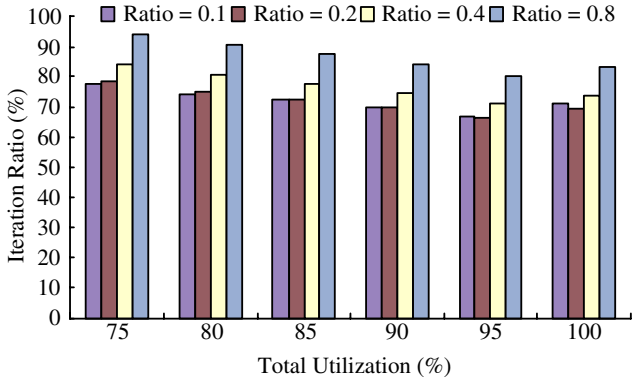

Fig. 3. Comparison in the iteration ratio with initial value improvement for both algorithms.

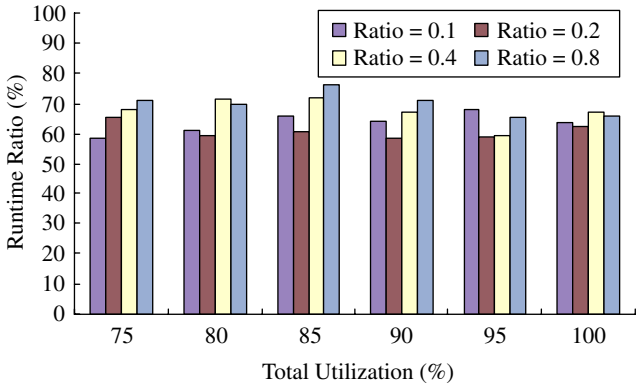

Fig. 4. Comparison in the runtime ratio with initial value improvement for both algorithms.

According to the simulation results, the EAA with Initial Value Improvement can speed up the schedulability analysis significantly. For instance, the Runtime Ratio for the EAA against the Audsley's Algorithm ranged from 59% to 66% when the value of Ratio in the EAA was set to 0.2. In other words, the EAA produces up to a 41% saving in the runtime overhead for the Audsley's Algorithm when the system was in a heavy load. Table 7 shows the mean and standard derivation of the runtime with initial value improvement for both the Audsley's Algorithm and the EAA (with Ratio = 0.2). Note that the unit of runtime was a CPU cycle.

Table 8 shows the performance of the EAA when both algorithms were enhanced by the initial value improvement. Among 10,000 Task Sets, the EAA (with

Table 6
The mean and standard deviation of the number of iterations with initial value improvement for both the Audsley's Algorithm and the EAA (with Ratio = 0.2)

		Utilization factor of task set					
		75%	80%	85%	90%	95%	100%
Audsley's Algorithm	Mean	40.16420	51.14560	62.59170	75.95960	92.68770	83.94700
	Standard deviation	32.31783	30.99649	32.56320	35.02468	41.76408	43.51630
EAA	Mean	31.46770	38.33050	45.44290	53.05790	61.60840	58.41560
	Standard deviation	26.84397	24.31287	23.84215	23.56130	25.32597	26.61234

Table 7
The mean and standard deviation of runtime (CPU cycles) with initial value improvement for both the Audsley's Algorithm and the EAA (with Ratio = 0.2)

		Utilization factor of task set					
		75%	80%	85%	90%	95%	100%
Audsley's Algorithm	Mean	308,952.03	367,796.10	439,015.05	520,728.33	632,235.65	562,065.80
	Standard deviation	407,847.57	321,222.50	342,499.34	369,328.82	460,239.15	454,742.28
EAA	Mean	215,636.46	250,299.74	287,133.03	328,916.27	374,175.44	349,675.45
	Standard deviation	245,590.66	288,908.25	232,272.77	273,497.80	250,466.09	275,220.50

Table 8
The efficiency of the EAA (with Ratio = 0.2) with initial value improvement for both algorithms

		Utilization factor of task set					
		75%	80%	85%	90%	95%	100%
Iteration	EAA is better	9283	9640	9803	9873	9923	9651
	Equal	381	197	95	56	25	121
	EAA is worse	336	163	102	71	52	228
Runtime	EAA is better	9899	9962	9979	9981	9982	9945
	EAA is worse	101	38	21	19	18	55

Ratio = 0.2) outperformed the Audsley's Algorithm in most cases in terms of the number of iterations and the runtime.

5. Conclusion and future work

The goal of this research is to improve the performance of the *exact schedulability analysis*, the Audsley's Algorithm. The basic idea is to create a larger jump in the derivation of each subsequent time instant $r^{(l)}$, such that the runtime overhead required by the Audsley's Algorithm can be much reduced. By properly partitioning a task set into two subsets, L and R, and differently treating tasks in L and R during each iteration, the number of iterations required for analyzing the schedulability of the task set can be reduced. The time complexity of the proposed algorithm in each iteration has been proved to be $O(n)$. Although the proposed algorithm does introduce additional cost in the partitioning of tasks in each iteration and some temporary storage in the partitioning procedure. The extra temporary storage only costs few more variables. The capability of the proposed algorithm was evaluated and compared to the related work (Audsley et al., 1993; Bril et al., 2003), which revealed up to a 55.5% saving (Fig. 2 with Total Utilization = 100% and Ratio = 0.2) in the runtime overhead for the Audsley's Algorithm when the system was in a heavy load. A minimum saving of 21.7% is provided (Fig. 2 with Total Utilization = 75% and Ratio = 0.8). Furthermore, EAA produces a maximum saving in iterations of 50.7% (Fig. 1 with Total Utilization = 100% and Ratio = 0.2) and a minimum saving in iterations of 6.1% (Fig. 3 with Total Utilization = 75% and Ratio = 0.8).

For the future work, the EAA will be modified to be capable of handling more general task models, e.g., allowing resource accesses among tasks. For analyzing the schedulability of a task set with synchronization requirements, the blocking time produced by priority inversion must be taken into consideration. The EAA will also be extended to the multiframe task model (Mok and Chen, 1996), in which the computation requirements of tasks in consecutive periods are modelled as regular patterns.

References

Audsley, N.C., Burns, A., Richardson, M., Wellings, A. 1991. Hard real-time scheduling: the deadline-monotonic approach. In: Proceedings of the 8th IEEE Workshop on Real-Time Operating Systems and Software, May, pp. 133–137.

Audsley, N.C., Burns, A., Richardson, M., Tindell, K., Wellings, A., 1993. Applying new scheduling theory to static priority preemptive scheduling. Software Engineering Journal 8 (5), 284–292.

Bini, E., Buttazzo, G.C., 2002. The space of rate monotonic schedulability. In: Proceedings of the 23rd IEEE Symposium on Real-Time Systems, December, pp. 169–178.

Bini, E., Buttazzo, G.C., Buttazzo, G., 2001. A hyperbolic bound for the rate monotonic algorithm. In: Proceedings of the 13th IEEE Euromicro Conference on Real-Time Systems, June, pp. 59–66.

Bril, R.J., Verhaegh, W.F.J., Pol, E.J.D., 2003. Initial values for on-line response time calculations. In: Proceedings of the 15th IEEE Euromicro Conference on Real-Time Systems, July, pp. 13–22.

Chang, L.P., Kuo, T.W., Lo, S.W., 2004. Real-time garbage collection for flash memory storage systems of real-time embedded systems. ACM Transactions on Embedded Computing Systems (TECS) 3 (4), 837–863.

Cheng, A.M.K., Rao, S., 2003. Real-time traffic scheduling and routing in packet-switched networks using a least-laxity-first strategy. Journal of VLSI Signal Processing.

Han, C.C., Tyan, H.Y., 1997. A better polynomial-time schedulability test for real-time fixed priority scheduling algorithms. In: Proceedings of the 18th IEEE Symposium on Real-Time Systems, pp. 36–45.

Intel, 1997. Using the RDTSC Instruction for Performance Monitoring. Available from: <http://developer.intel.com/drg/pentiumII/appnotes/RDTSCPM1.htm>.

Kuo, T.W., Lee, S.L., Lin, Y.S., Liu, Y.H., 1997. Providing video-on-demand services on Windows NT. In: Proceedings of International Symposium on Multimedia Information Processing.

Kim, N., Ryu, M., Hong, S., Saksena, M., Choi, C.H., Shin, H., 1996. Visual assessment of a real-time system design: a case study on a CNC

controller. In: Proceedings of the 17th IEEE Symposium on Real-Time Systems, pp. 300–310.

Kamenoff, N.I., Weiderman, N.H., 1991. Hartstone distributed benchmark: requirements and definitions. Proceedings of the IEEE Symposium on Real-Time Systems, 199–209.

Liu, C.L., Layland, J.W., 1973. Scheduling algorithms for multiprogramming in a hard-real-time environment. Journal of the ACM 20 (1), 46–61.

Lehoczky, J.P., Sha, L., Ding, Y., 1989. The rate monotonic scheduling algorithms: exact characterization and average behavior. In: Proceedings of the 10th IEEE Symposium on Real-Time Systems, December, pp. 166–171.

Locke, C.D., Vogel, D.R., Mesler, T.J., 1991. Building a predictable avionics platform in Ada: a case study. In: Proceedings of the 12th IEEE Symposium on Real-Time Systems, December, pp. 181–189.

Mok, A.K., Chen, D., 1996. A multiframe model for real-time tasks. In: IEEE 17th Real-Time Systems Symposium, December, pp. 22–29.

Miller, L., Cheng, A.M.K., 2000. Admission of high priority real-time calls in an ATM network via bandwidth reallocation and dynamic rerouting of active channels. In: Proceedings of the 21st IEEE Real-Time Systems Symposium.

Molini, J.J., Maimon, S.K., Watson, P.H., 1990. Real-time system scenarios. In: Proceedings of the 12th IEEE Symposium on Real-Time Systems, December, pp. 214–225.

Pop, P., Eles, P., Peng, Z., 2000. Bus access optimization for distributed embedded systems based on schedulability analysis. DATE.

Pop, P., Eles, P., Peng, Z., 2004. Schedulability-driven communication synthesis for time triggered embedded systems. Real-Time Systems 6 (3), 297–325.

Richter, K., Racu, R., Ernst, R., 2003. Scheduling analysis integration for heterogeneous multiprocessor SoC. In: Proceedings of the 24th IEEE International Real-Time Systems Symposium.

Rao, S., Cheng, A.M.K., 2000. Scheduling and Routing of Real-Time Multimedia Traffic in Packet-Switched Networks. In: Proceedings of the IEEE International Conference on Multimedia and Expo.

Vin, H.M., Goyal, P., Goyal, A., 1994. A statistical admission control algorithms for multimedia servers. In: Proceedings of the ACM International Conference on Multimedia.

Wong, C., Cheng, A.M.K., 1997. An Approach for imprecise transmission of TIFF image files through congested real-time ATM networks. In: Proceedings of the 22nd International Conference on Local Computer Networks.