論文
3-9-4

On-line Schedulability Check Algorithm for Imprecise Real-time Tasks
(부정확한 실시간태스크들을 위한 온라인 스케쥴가능성 검사 알고리즘)

송 기 현*
(Gi-Hyeon Song)

ABSTRACT

In a (hard) real-time system, every time-critical task must meet its timing constraint, which is typically specified in terms of its deadline. Many computer systems, such as those for open system environment or multimedia services, need an efficient schedulability test for on-line real-time admission contol of new jobs. Although various polynomial time schedulability tests have been proposed, they often fail to decide the schedulability of the system precisely when the system is heavily loaded. Furthermore, the most of previous studies on on-line real-time schedulability tests are concentrated on periodic task applications. Thus, this paper presents an efficient on-line real-time schedulability check algorithm which can be used for imprecise real-time system predictability before dispatching of on-line imprecise real-time task system consisted of aperiodic and preemptive task sets when the system is overloaded.

요 약

(경성) 실시간시스템에 있어서, 모든 긴급한 태스크는 만기라고 하는 시간적 제약조건을 충족시켜야만 한다. 개방시스템 환경이나 멀티미디어 서비스들을 위한 것들과 같은 많은 컴퓨터시스템들은 온라인으로 도착하는 새로운 작업들을 허용할 수 있느냐 없느냐에 대한 실시간 제어를 위한 효율적인 스케쥴가능성 검사를 필요로 한다. 비록 지금까지 여러가지의 다항식복잡도를 갖는 스케쥴 가능성 검사들이 제안되어 왔지만 이들은 시스템에 상당한 과부하가 걸릴때에는 이 시스템의 스케쥴 가능성을 종종 정확하게 판정하지 못한다. 더욱이, 온라인 실시간 스케쥴가능성검사들에 있어서의 대부분의 연구들이 주기적인 태스크 응용들에 집중되어 있다.

그래서 본 논문에서는 시스템에 과부하가 발생할 때 비주기적이며 선점가능한 태스크 집합들로 구성된 부정확한 온라인 실시간 태스크 시스템을 실행하기 이전에 스케쥴가능한지를 예측할 수 있는 효율적인 온라인 실시간 스케쥴 가능성 검사 알고리즘을 제시하였다.

* 정회원 : 대전보건대학 경영정보과 교수

논문접수 : 2002. 8. 6.
심사완료 : 2002. 12. 10.

1. Introduction

In a (hard) real-time system, every time-critical task must meet its timing constraint, which is typically specified in terms of its deadline. It is essential for every time-critical task to complete its execution and produce its result by its deadline. Otherwise, a timing fault occurs, and the result produced by the task is of little or no use. Unfortunately, many factors, such as variations in processing times of dynamic algorithms and congestion on the communication network, make meeting all timing constraints at all times difficult.

An approach to minimize this difficulty is to trade off the quality of the results produced by the tasks with the amounts of processing time required to produce the results. Such a tradeoff can be realized by using theimprecise computation technique[6]. In the imprecise computation model, each task can be divided into two parts, a mandatory part and an optional part. When system load is normal, the optional part is executed and the computation produces a precise result. On overloaded conditions, however, all or portion of optional part is skipped to conserve system resources for the mandatory parts of other tasks. The imprecise computation sacrifices accuracy to meet the deadlines of mandatory parts.

On the other hand, many computer systems, such as those for open system environment or multimedia services, need an efficient schedulability test for on-line real-time admission contol of new jobs. Efficient on-line real-time schedulability tests are also useful to various service-critical systems, such as tele-medicine systems, tele-conferencing systems, and multimedia services that have Quality-of-Service (QoS) requirements [5]. Although various polynomial time schedulability tests [1, 2, 3, 4, 8] have been proposed,they often fail to decide the schedulability of the system precisely when the system is heavily loaded. Furthermore, the most of previous studies on on-line real-time schedulability tests [1, 2, 3, 4, 8] are concentrated on periodic task applications.

Thus, this paper presents efficient on-line real-time schedulability check algorithm which can be used for real-time system predictability before dispatching of on-line real-time task system consisted of aperiodic and preemptive task sets for the aim of minimization of total error, minimization of the maximum or average error, minimization of the number of discarded optional tasks, minimization of the number of tardy tasks, and minimization of average response time when the system is overloaded.

The rest of this paper is organized as follows. Section 2 introduces related work. Section 3 provides problem formulation. Section 4 presents the on-line schedulability check algorithm for imprecise real-time systems. Section 5 and 6 show numerical example and the conclusion, respectively.

2. Related Work

A system is underloaded if there exists a schedule that will meet the deadline of every task and overloaded otherwise. Scheduling underloaded systems is a well-studied topic, and several on-line algorithms have been proposed for the optimal scheduling of these systems on a uniprocessor [10, 11]. Examples of such algorithms include earliest-deadline-first (ED) and smallest-slack-time (SL). However, none of these classical algorithms make performance guarantees during times when the system is overloaded. In fact, Locke has experimentally demonstrated that these algorithms perform quite poorly when the system is overloaded [12].

Practical systems are prone to intermittent overloading caused by a cascading ofexceptional situations. A good on-line scheduling algorithm, therefore, should give a performance guarantee in overloaded as well as underloaded circumstances.

Although various polynomial time schedulability tests [1, 2, 3, 4, 8] have been proposed, they often fail to decide the schedulability of the system precisely when the system is heavily loaded. Furthermore, the most of previous studies on on-line real-time schedulability tests [1, 2, 3, 4, 8] are concentrated on periodic task applications. Recently, [9] presents efficient on-line schedulability tests, but it is for priority driven real-time systems.

Therefore, when the system is heavily loaded, previous studies [1, 2, 3, 4, 8, 9] can not help having limitation even though they show the best possible competitive factors of the on-line schedulers. So, in this paper, the imprecise computation model is adopted.

3. Problem Formulation

Assume that at an instant, there are identically arrived N preemptive imprecise tasks running on a single processor and they are sorted according to their deadlines, $d_1 \le d_2 \le \text{K} \le d_{N-1} \le d_N$. The restrictive simultaneous-arrival assumption also was used in the top-level scheduling of the two-level approach proposed in [7]. In [7], the top-level scheduling algorithm invoked at every task arrival allocates service times to the tasks, and the low-level scheduling algorithm actually schedules the tasks and provides them with the allocated service times.

Let an imprecise task T_i be composed of the mandatory part M_i and the optional part O_i, and characterized by its arrival time r_i, deadline d_i, and computational requirement m_i and o_i for M_i and O_i, respectively. Let p_i be the sum of m_i and o_i. Let $y_{i,j}$ be the service time assigned to task i during interval j $[d_{j-1}, d_j]$. Then the total amount of service time allocated to each task T_i is represented as y_i. That is, y_i is $\sum_{j=1}^{i} y_{i,j}$ since $y_{i,j}$ becomes zero when $j \phi i$.

A schedule determines the amount of service time to be given to each task T_i during the schedulable interval which is defined as an interval between the task's arrival time and its deadline. In this paper, it is interested in schedulability test of on-line real-time tasks. In the schedulability test, only mandatory part computational requirements are considered when the real-time system is overloaded.

That is, the problem can be formulated as the following [Fig. 1].

Compute y_i, *for* $1 \le i \le N$
Subject to

1) $\sum_{k=j}^{N} y_{k,j} \le d_j - d_{j-1} \quad j = 1, \Lambda, N$

2) $y_i \ge m_i \quad i = 1, \Lambda\, N$

3) $y_i \le p_i \quad i = 1, \Lambda\, N$

4) $y_{i,j} \ge 0 \quad 1 \le j \le i \le N$

[Fig. 1] Problem formulation of on-line real-time schedulability check

The first constraint simply states that the sum of service times actually given to all schedulable task(s) in each interval is equal to or less than the length of the interval. The second constraint ensures that the mandatory part of each task be always scheduled. The third constraint implies that the service time actually given to each task can not exceed the computational requirement. The last constraint says that a non-negative amount of service time is allotted to each task in each schedule interval.

4. On-line Schedulability Check Algorithm

In this section, a two-levelschedulability check policy which is composed of top-level schedulability check and low-level scheduling is proposed. In the top-level schedulability check which is executed whenever a new task arrives, determines the amount of mandatory processing timesto be allocated to all schedulable tasks at that instant and checks whether the mandatory computational requirements of the all schedulable tasks are satisfied or not. The low-level scheduling algorithm actually schedules tasks into service.

The maximumof service time is bounded by that obtained by the top-level algorithm. The amount of mandatory execution time that each task receives actually depends on the arrival time of new tasks. In other words, the amount of service time determined by the top-level algorithm is valid only until the next new task's arrival. The on-line real-time schedulability check algorithm of this paper is proposed as following [Fig. 2].

In the above algorithm, whenever a on-line task T_i arrives, the "DetermineRunningTasks(i)" function determines schedulable tasks at T_i arrival, then the "SortTaskByDeadline()" function sorts the schedulable tasks by deadlines on ascending order. then the "ConstructIntervals (r_i)"function constructs intervals of the schedulable tasks. Next, the top-level schedulability check algorithm called as "TopLevelSchedulabilityCheck()" is performed.

```
Algorithm OnLineSchedulability()
  For i = 1 to N
    Call DetermineRunningTasks ( i )             ' determine schedulable tasks at T_i arrival.
    Call SortTaskByDeadline ()                   ' sort the schedulable tasks by deadlines.
    Call ConstructIntervals ( r_i )              ' construct intervals of the schedulable tasks.
    OnCheck = TopLevelSchedulabilityCheck()      ' perform top-level algorithm
    If (OnCheck = False) Then "Exit Algorithm"
    Call LowLevelScheduling ( r_i , r_i+1 )      ' perform low-level scheduling algorithm at
                                                     time interval [ r_i , r_i+1 ]
    Call RecomputeMi()                           ' recompute computational requirement m_i
  Next i
End Algorithm
```

[Fig. 2] Algorithm for On-line Schedulability

This algorithm computes the optimal mandatory service times allotted to N tasks. Let I_j be an interval $[d_{j-1}, d_j]$. There exists up to N mutually disjoint intervals, even though the actual number is usually smaller than N since more than one tasks share one same interval in many real world applications. This algorithm tries to assign interval I_k to the tasks that are schedulable in the interval I_k. The assignment procedure is done through backward iteration. That is, the assignment starts from the last interval I_N and finishes with the first interval I_1. A part or all of an interval is distributed to the schedulable tasks (of which deadlines are greater than or equal to d_k) as much as to allow them to execute their mandatory parts. [Fig. 3] depicts the detail of this algorithm.

In this algorithm, $y_{i,j}$ represents the amount of mandatory service time to be allocated to task T_i in interval I_j. In this [Fig. 3], variable idt, idv, u and v stand for the index of task, the index of interval, the amount of time that is necessary to execute the mandatory part of task T_{idt}, and the length of the remaining part of interval I_{idv}, respectively. Since the value of idt or idv is decreased by one in each iteration, the algorithm terminates in finite step. In this algorithm, if '*while loop*' terminates due to the condition such that $(idt \geq 1\ and\ idv \pi 1)$, a feasible schedule for the imprecise real-time task set can not be found and the imprecise real-time task set can not be scheduled.

```
y_{i,j} = 0,  1 ≤ j ≤ i ≤ N
idt = N,  idv = N
u = m_N,  v = d_N − d_{N−1}
d_0, d_{−1} = r_i
while  (idt ≥ 1  and  idv ≥ 1)  do
    if  (u > v)  then
      y_{idt,idv} = v,  u = u − v,  idv = idv − 1,  v = d_{idv} − d_{idv−1}
    elseif  u = v  then
      y_{idt,idv} = u,  idt = idt − 1,  u = m_{idt},  idv = idv − 1,  v = d_{idv} − d_{idv−1}
    else
      y_{idt,idv} = u
      if  (idt > idv)  then
        v = v − u,  idt = idt − 1,  u = m_{idt}
      else  /* for  the  case  that  idt = idv  and  the  interval  [d_{idv−1}, d_{idv}]  is  not
                fully  distributed */
        idt = idt − 1, u = m_{idt}, idv = idv − 1, v = d_{idv} − d_{idv−1}
      endif
    endif
end while

if (idt ≥ 1 and idv < 1) then the imprecise system is not schedulable
                              else we have got the optimal schedule
```

[Fig. 3] Top-level schedulability check algorithm

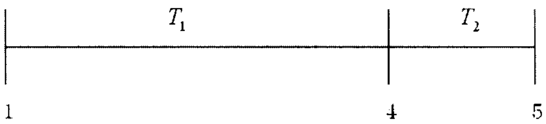
In this case, the algorithm is terminated. Otherwise, i.e., if '*while loop*' terminates due to the condition such that ($idt = 0$ *and* $idv = 0$), the low-level scheduling algorithm called as "LowLevelScheduling (r_i, r_{i+1})" is performed on time interval [r_i, r_{i+1}]. The low-level scheduling algorithm actually assign a processor to tasks according to the amounts of mandatory processing time determined by the top-level schedulability check algorithm with EDF(Earliest Deadline First) algorithm. Finally, in the "RecomputeMi()" function, the mandatory requirements of all scheduled tasks in time interval [r_i, r_{i+1}] reduced by the amount of assigned processor time. The OnLineSchedulability() algorithm repeats above processes until no on-line task arrived.

5. Numerical Example and Complexity Analysis

Consider a sample task set with four tasks shown in <Table 1>. At time 1, tasks T_1, T_2, and T_3 are arrived. Then tasks T_1, T_2, and T_3 are schedulable at time 1. Then, the 3 schedulable tasks are sorted by deadline with ascending order resulting as T_1 - T_2 - T_3. As the result of deadline sort, intervals $I_1 = [1,5]$, $I_2 = [5,10]$, $I_3 = [10,12]$ are constructed. Then, the "TopLevelSchedulabilityCheck()" algorithm is performed. The algorithm starts with idt = 3, idv = 3, u = 3, and v = 2. Since the required service time for the mandatory part of T_3 (i.e., 3) is greater than the length of interval I_3(2), $y_{3,3}$ becomes equal to 2, u is decreased by 2 (the length of I_3) and becomes equal to 1, idt remains unchanged, and idv is decreased by 1. In the next iteration with idt =3, idv =2, u = 1 and v = 5, a part of I_2 is assigned to T_3, since T_3 has not received enough amount of service time for its mandatory part in the first iteration. So $y_{3,2}$ becomes equal to 1 and idt is decreased by 1, but idv remains unchanged. In the third iteration with idt = 2, idv = 2, u = 3 and v = 4, a part of remaining length of I_2 (i.e., 3) is allocated to task T_2. So $y_{2,2}$ becomes equal to 3. Finally, in the fourth iteration, idt, idv, u and v becomes 1, 1, 3 and 4, respectively. A part of I_1 (i.e., 3) is assigned to task T_1 in order to allow the task to receive the amount of service time equal to its computational requirement for the mandatory part. Thus, $y_{1,1}$ becomes equal to 3.

Then, the algorithm is terminated normally with idt = 0, idv = 0. As the result of this assignment, the mandatory service times allotted to T_1, T_2 and T_3 become equal to 3 respectively. So, y_1, y_2 and y_3 become equal to 3 respectively. <Table 2> depicts the result. <Table 3> depicts how all variables such as idt, idv, u and v are changed. Then, as the next step of "OnLineSchedulability()" Algorithm, "LowLevelScheduling" (1,5) is performed. The low-level scheduling algorithm actually assign a processor to tasks according to the amounts of mandatory processing time determined by the top-level schedulability check algorithm with EDF(Earliest Deadline First) algorithm.

[Fig. 4] shows the result of the low-level scheduling algorithm.



[Fig. 4] Low-level scheduling result on interval [r_1, r_4]

Next, as the final step of "OnLineSchedulability()" Algorithm, "RecomputeMi()" is performed. Then, the computational requirements of m_1 and m_2 are reduced by 3 and 1 respectively thus, the computational requirements of m_1 and m_2 become equal to 0 and 2 respectively.

Next, at time 5, task T_4 is arrived. then the same processes are repeated. The result of the top-level schedulability check algorithm is represented as <Table 4>, <Table 5>, and the result of the low-level scheduling algorithm is represented as [Fig. 5]. Thus, as the result of the OnLineSchedulability() algorithm, we can conclude that the on-line real-time task set shown in <Table 1> can be scheduled normally.

<Table 1> A sample task set

Tasks	r_i	m_i	o_i	p_i	d_i
T_1	1	3	3	6	5
T_2	1	3	3	6	10
T_3	1	3	2	5	12
T_4	5	3	5	8	14

<Table 2> $y_{i,j}$ at time instant 1 for the task set in <Table 1>. (Shaded cells represent intervals in which tasks are not schedulable)

Tasks	I_1=[1,5]	I_2=[5,10]	I_3=[10,12]
T_1	3		
T_2	0	3	
T_3	0	1	2

<Table 3> Changes of variables

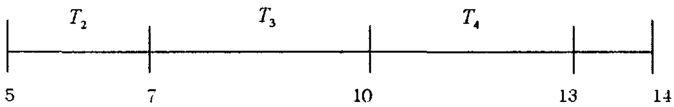
iteration	idt	u	idv	v	$y_{i,j}$	comments
initial	3	3	3	2	all $y_{i,j}$ are zero	
1	3	1	2	5	$y_{3,3}$ = 2	
2	2	3	2	4	$y_{3,2}$ = 1	
3	1	3	1	4	$y_{2,2}$ = 3	
4	0	0	0	0	$y_{1,1}$ = 3	T_1, T_2 and T_3 are schedulable

<Table 4> $y_{i,j}$ at time instant 5 for the task set in <Table 1>.
(Shaded cells represent intervals in which tasks are not schedulable)

Tasks	I_1=[5.10]	I_2=[10.12]	I_3=[12.14]
T_2	2		
T_3	2	1	
T_4	0	1	2

<Table 5> Changes of variables

iteration	idt	u	idv	v	$y_{i,j}$	comments
Initial	3	3	3	2	all $y_{i,j}$ are zero	
1	3	1	2	2	$y_{3,3}$ = 2	
2	2	3	2	1	$y_{3,2}$ = 1	
3	2	2	1	5	$y_{2,2}$ = 1	
4	1	2	1	3	$y_{2,1}$ = 2	
5	0	0	0	0	$y_{1,1}$ = 2	T_2, T_3 and T_4 are schedulable



[Fig. 5] Low-level scheduling result on interval [r_4, d_4]

In the proposed algorithm for on-line schedulability which is depicted in [Fig. 2], the number of iterations that the "For loop" is executed is bounded by O(N), where N is the total number of tasks in the imprecise task system. Next, the number of iterations that each procedure or function in the "For loop" is executed is bounded by O(logN) because the number of schedulable tasks which the "DetermineRunningTasks(i)" function determines at T_i arrival can be bounded by log N. So, the complexity of the proposed algorithm in [Fig. 2] becomes O(NlogN).

6. Conclusion

Many computer systems, such as those for open system environment or multimedia services, need an efficient schedulability test for on-line real-time admission contol of new jobs. Efficient on-line real-time schedulability tests are also useful to various service-critical systems, such as tele-medicine systems, tele-conferencing systems, and multimedia services that have Quality-of-Service (QoS) requirements [5]. Although various polynomial time schedulability tests [1, 2, 3, 4, 8] have been proposed, they often fail to decide the schedulability of the system precisely when the system is heavily loaded. Furthermore, the most of previous studies on on-line real-time schedulability tests [1, 2, 3, 4, 8] are concentrated on periodic task applications.

Thus, this paper presents efficient two-level on-line real-time schedulability check algorithm which is composed of top-level schedulability check and low-level scheduling. This algorithm can be used efficiently for real-time system predictability before dispatching of on-line real-time task system consisted of aperiodic and preemptive task sets for the aim of minimization of total error, minimization of the maximum or average error, minimization of the number of discarded optional tasks, minimization of the number of tardy tasks, and minimization of average response time when the system is overloaded. The efficient imprecise on-line real-time scheduling algorithms are expected on the basis of the proposed two-level schedulability check algorithm.

※ References

[1] A. Burchard, J. Liebeherr, Y. Oh, and S.H. Son, "New Strategies for Assigning Real-Time Tasks to Multiprocessor Systems," IEEE Transactions on Computers, Volume 44, No. 12, pp.1429-1442, December 1995.

[2] A.K. Mok and D. Chen, "A Multiframe Model for Real-Time Tasks," IEEE 17TH Real-Time Systems Symposium, December 1996.

[3] C.-C. Han, "A Better Polynomial Time Schedulability Test for Real-Time Multiframe Tasks," IEEE 19TH Real-Time Systems Symposium, pp. 104-113, December 1998.

[4] C.-C. Han and H.-Y. Tyan, "A Better Polynomial Time Schedulability Test for Real-Time Fixed Priority Scheduling Algorithms," IEEE 18TH Real-Time Systems Symposium, pp. 36-44, December 1998.

[5] H.M. Vin, P. Goyal, and A. Goyal, "A Statistical Admission Control Algorithms for Multimedia Servers," Proceedings of the ACM International Conference on Multimedia, Oct 1994.

[6] J.Y. Chung, J.W.S. Liu, and K.J. Lin, "Scheduling Periodic Jobs that Allow Imprecise Results," IEEE Transactions on Computers, Volume 19, No. 9, pp.1156-1173, September 1990.

[7] Jayanta K. Dey, James Kurose, and Don Towsley, "On-Line Scheduling Policies for a Class of IRIS," IEEE Transactionson Computers, Volume 45, No. 7, pp.802-813, July 1996.

[8] T.-W. Kuo and A.K. Mok, "Load Adjustment in Adaptive Real-Time Systems," IEEE Transactionson Computers, Volume 16, No. 12, December 1997.

[9] Tei-Wei Kuo, Yu-Hua Liu and Kwei-Jay Lin, "Efficient On-Line Schedulability Tests for Priority Driven Real-Time Systems," Proceedings of the Sixth IEEE Real Time Technology and Applications Symposium (RTAS 2000), 2000.

[10] M. Dertouzos, "Control Robotics : the procedural control of physical processes," In Proc. IFIP congress, pp.807-813, 1974.

[11] A. Mok, "Fundamental Design Problems of Distributed Systems for the Hard Real-time environment," Doctoral Dissertation, M.I.T., 1983.

[12] C. Douglass Locke, "Best-Effort Decision Making for Real-Time Scheduling," Doctoral Dissertation, Computer Science Department, Carnegie-Mellon University, 1986.

송 기 현

1985년 충남대학교 계산통계학과 졸업(이학사)
1987년 충남대학교 대학원 계산통계학과 졸업(이학석사)
1999년 아주대학교 대학원 컴퓨터공학과 졸업(공학박사)
1990년 ~ 현재 대전보건대학 경영정보과 교수
관심분야 : 실시간 스케쥴링, 웹 데이터베이스, 홈페이지 저작