Appeared in the Proceedings of 11th IEEE Workshop on Real-Time Operating Systems and Software, 1994.

An End-to-End Approach to Schedule Tasks with Shared Resources in Multiprocessor Systems

Jun Sun Riccardo Bettati Jane W.-S. Liu
Department of Computer Science
University of Illinois, Urbana-Champaign
Urbana, IL 61801

Abstract

In this paper we propose an end-to-end approach to scheduling tasks that share resources in a multiprocessor or distributed systems. In our approach, each task is mapped into a chain of subtasks, depending on its resource accesses. After each subtask is assigned a proper priority, its worst-case response time can be bounded. Consequently the worst-case response time of each task can be obtained and the schedulability of each task can be verified by comparing the worst-case response time with its relative deadline.

1 Introduction

Tasks in real-time systems often share resources, and semaphore-like operations are necessary to guarantee their mutual-exclusive access to critical sections. A previous study shows that careless use of semaphore operations can cause uncontrolled priority inversion, which occurs when a high-priority task is blocked by some low-priority tasks for an unpredictable amount of time [1]. We refer to the total length of time a task is delayed by lower-priority tasks due to resource contention as its *blocking time*. To ensure predictability, it is imperative to bound the blocking time of each task, as shown in [2]. Several effective solutions have been proposed for single processor systems; two well-known examples are the *Priority Ceiling Protocol* (PCP) [1] and the *Stack Based Protocol* (SBP) [3].

In multiprocessor and distributed systems concurrency and distribution complicate the resource contention problem. A task T_i can be blocked not only by a local task on the same processor due to local resource contentions, but also by a remote task that needs some global resources also needed by T_i. Rajkumar, et al. [4] extended PCP for single processor systems to multiprocessor systems and provided an initial solution for this problem. The extended protocol is known as the *Multiprocessor Priority Ceiling Protocol* (MPCP). According to MPCP, a resource needed by remote tasks on other processors is a *global resource*, and the processor on which a global resource resides is called its *synchronization processor*. When a task T_i gains access to a global resource, a *Global Critical Section* (GCS) server runs on the resource's synchronization processor on behalf of T_i. On each processor PCP is used to schedule both local tasks and GCS servers. Consequently, for each task, the total blocking time due to both local resource contention and global resource contention can be bounded, and whether each task can meet its deadline can be determined based on this blocking time by using the schedulability condition for the single-processor PCP.

However, the performance of MPCP is sometimes poor, especially for tasks on synchronization processors. One reason is that GCS servers on each synchronization processor always have higher priorities than local tasks. The priority inversion problem is reintroduced when a high-priority local task is delayed by GCS servers executing on behalf of lower-priority tasks.

In this paper we propose an end-to-end approach to scheduling tasks with shared resources and to analyzing their schedulability in multiprocessor systems. Section 2 gives an informal description of this approach and compares and contrasts it with MPCP. Section 3 presents in detail the procedure used in the end-to-end approach. Future work is discussed in section 4.

2 The End-to-End Scheduling Approach

From the viewpoint of end-to-end scheduling, a task that needs remote resources is viewed as a chain of subtasks in the following way. Each critical section associated with a remote resource is a subtask that executes on the synchronization processor of the remote resource. A segment that requires no resources or only local resources is also a subtask, and this subtask executes on the local processor. Subtasks of the same task collectively inherit the task's release time and deadline, and they execute in turn. Specifically, if task T_i has n subtasks, subtask $T_{i,1}$ is ready for execution at the release time of T_i, and subtask $T_{i,j}$ is ready for execution when subtask $T_{i,j-1}$ completes, for $j = 2, 3, \ldots, n$. The last subtask $T_{i,n}$ must complete by the deadline of T_i. If task T_i is a periodic task, this precedence relation holds for every instance of T_i.

The precedence relation among the subtasks of each task can be easily satisfied by using the phase-modification method proposed in [5]. Let $c_{i,j}$ be the worst-case response time of $T_{i,j}$. According to the phase-modification method, once we know $c_{i,k}$ for $k = 1, 2, \ldots, j-1$, we postpone the phase of the subtask $T_{i,j}$ by $\sum_{k=1}^{j-1} c_{i,k}$. This modification allows us to enforce the precedence relation between subtasks while treating the subtasks in each task as if there is no precedence relation between them. We will return to discuss how to bound the worst-case response times of subtasks on each processor using the schedulability condition in [5], provided that the subtasks are assigned fixed priorities and some single-processor synchronization protocol is used to control priority inversion. By summing up the worst-case response times of all its subtasks, we can determine the worst-case response time of each task, and therefore whether the task can meet its deadline.

Similar to MPCP, we allow nested resource accesses. However, we impose an additional restriction that all resources accessed in one nested critical section must reside on the same processor. In other words accesses to resources on different processors cannot be nested. One consequence of the end-to-end scheduling approach is that there is no need to control the accesses to remote, global resources differently from local resources. Each subtask that is a GCS server in MPCP model is local to its synchronization processor. All resource contentions are resolved locally and separately on each processor.

Table 1 gives an example, Example 1. In the table, T_i denotes a task; column *proc* lists the processor T_i is assigned to; ϕ_i is T_i's priority; p_i denotes T_i's period; and τ_i stands for T_i's processing time. The smaller the value of ϕ_i, the higher T_i's priority. The system in this example has two processors P_1 and P_2. There are two periodic tasks, T_1 and T_2, and one resource R. The deadline for each task is the end of its period. T_1 is assigned to P_1; T_2 and R are on P_2. The table lists the parameters of the tasks. Specifically, T_1 has three segments. The first and the last segments need no resource; they are executed on P_1, each with processing time 2. The middle segment requires the resource R; its processing time is 2. (The notation $t(R)$ in the *Segments* column indicates that the segment is a critical section that has duration t and accesses the resource R.) We note that the tasks can not be scheduled according to MPCP. Since T_1 needs to access R on P_2, there is a GCS server running on P_2 on behalf of T_1. This server has a higher priority than T_2. Since the processing time for this server is as long as T_2's period and T_2 will be blocked by the GCS server whenever the server executes, T_2 can not meet its deadline.

T_i	proc	ϕ_i	p_i	τ_i	Segments		
T_1	P_1	2	20	6	2	$2(R)$	2
T_2	P_2	1	2	1	1		

Table 1: Example 1 - A Simple System

In the end-to-end scheduling model, task T_1 is divided into three subtasks, $T_{1,1}$, $T_{1,2}$ and $T_{1,3}$. $T_{1,1}$ and $T_{1,3}$ execute on processor P_1 and need no resource, while $T_{1,2}$ executes on P_2 and needs resource R. $T_{1,1}$, $T_{1,2}$ and $T_{1,3}$ are dependent: the kth instance of $T_{1,1}$ (i.e., the instance of $T_{1,1}$ in its kth period) must complete before the kth instance of $T_{1,2}$ can begin execution. Similarly, the kth instance of $T_{1,3}$ cannot start execution until the kth instance of $T_{1,2}$ completes. Table 2 shows the parameters of the subtasks. $\tau_{i,j}$ is the processing time of subtask $T_{i,j}$, $f_{i,j}$ denotes the modified phase of $T_{i,j}$, and $\beta_{i,j}$ denotes the blocking time $T_{i,j}$ can experience.

$T_{i,j}$	proc	$\phi_{i,j}$	$p_{i,j}$	$\tau_{i,j}$	$\beta_{i,j}$	$c_{i,j}$	$f_{i,j}$
$T_{1,1}$	P_1	2	20	2	0	2	0
$T_{1,3}$	P_1	2	20	2	0	2	8
$T_{2,1}$	P_2	1	2	1	0	1	0
$T_{1,2}$	P_2	2	20	$2(R)$	0	6	2

Table 2: Example 1 - Using the End-to-End Approach to Schedule the Simple System

In this example, there is only one critical section, and therefore there is no blocking. The priorities of the subtasks are assigned on rate-monotonic basis. We see that the worst-case response time C_1 of the task T_1 is $c_{1,1} + c_{1,2} + c_{1,3} = 10$, which is less than 20, and the worst-case response time of T_2 is 1, and it is less than 2. We can therefore conclude that the deadlines of both tasks are always met.

Input :

1. Task set $\{T_i\}$. For each task T_i, the deadline D_i, period p_i, processing time τ_i, and resource accesses;
2. The task assignment mapping task set $\{T_i\}$ to processor set $\{P_k\}$;
3. The resource set $\{R_j\}$ and the resource assignment mapping $\{R_j\}$ to $\{P_k\}$.

Output : The conclusion whether the system can be scheduled and the priorities assigned to subtasks on each processor in the case the system is schedulable.

Step 1 : Map the given task set $\{T_i\}$ to a end-to-end task set $\{T_{i,j}\}$.

Step 2 : Assign priorities to subtasks.

Step 3 : Obtain the worst-case response time for each subtask.

Step 4 : Based on the results obtained in Step 3, analyze the schedulability for the whole system.

Figure 1: Pseudo-Code of the End-to-End Scheduling Procedure

3 Schedulability Analysis

We now describe how to choose the priorities for subtasks and determine their worst-case response times. We confine our attention to the case where tasks are periodic and their subtasks are assigned fixed priorities. However, the subtasks of each task may be assigned different priorities.

Figure 1 gives the pseudo-code description of the end-to-end scheduling procedure.

Step 1 : Map the given task set to an end-to-end task set

Following the rules below, Step 1 breaks up each task T_i in the given task set into a chain of n_i subtasks $T_{i,j}$ in the corresponding end-to-end task set :

1. Each subtask $T_{i,j}$ is either a critical section that requires some remote resources or a segment that requires no resource or only local resources. If a task has nested resource accesses, each outermost critical section is mapped to a subtask.
2. A subtask that requires no resource or only local resources is on the local processor of T_i. A subtask that requires remote resources is on the synchronization processor of the remote resources.
3. For every $j = 1, 2, \ldots, n_i - 1$, consecutive subtasks $T_{i,j}$ and $T_{i,j+1}$ are on different processors.

Rule 3 is not necessary for the correctness of the later discussion. However it allows us to obtain a tighter upper bound for the response time of each subtask.

Example 2 illustrates the rules described above. In this example there are four resources and three processors. Resource R_1 is assigned to processor P_1; R_2 and R_3 to P_2; and R_4 to P_3. Task T_1 is a periodic task. It has 10 segments, as shown by Figure 2. The shaded segments denote that T_1 requires some resources during those time intervals.

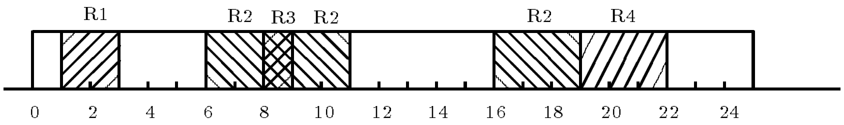
According to Step 1, T_1 is mapped into 6 subtasks, as shown by Table 3. The segment from time 0 to time 6, denoted as (0,6], is mapped onto one subtask $T_{1,1}$ because during this time interval, T_1 either does not require any resources or only requires local resources. According to rule 3, we map it onto one subtask, and it runs on the local processor, P_1. Similarly, segment (6,10] is mapped onto the subtask $T_{1,2}$ because the accesses to R_2 and R_3 are nested and only the outmost critical section becomes a subtask. This subtask runs on processor P_2. Segments (16,19] and (19,22] are two different subtasks, $T_{1,4}$ and $T_{1,5}$, because they access different remote resources. They run on P_2 and P_3 respectively. The segments (10,16] and (22,24] are mapped onto $T_{1,3}$ and $T_{1,6}$. They are both on P_1.

T_i	proc	p_i	$\tau_{i,j}$	Segment		
$T_{1,1}$	P_1	50	6	1	$2(R_1)$	3
$T_{1,2}$	P_2	50	5	$2(R_2)$	$1(R_2, R_3)$	$2(R_2)$
$T_{1,3}$	P_1	50	5	5		
$T_{1,4}$	P_2	50	3	$3(R_2)$		
$T_{1,5}$	P_3	50	3	$3(R_4)$		
$T_{1,6}$	P_1	50	3	3		

Table 3: Example 2 - Subtasks Assignment

Step 2 : Assign priorities to subtasks

Several methods can be used to assign priorities. Rate-monotonic assignment is a possible choice. Other choices include :



T_i	proc	p_i	τ_1	Segments									
T_1	P_1	50	25	1	$2(R_1)$	3	$2(R_2)$	$1(R_2, R_3)$	$2(R_2)$	5	$3(R_2)$	$3(R_4)$	3

Figure 2: Example 2 - Task T_1

- Global-deadline-monotonic assignment: the priority of a subtask is based on the global relative deadline, D_i, the deadline of the task T_i; the shorter D_i is, the higher priority $T_{i,j}$ has.
- Effective-deadline-monotonic assignment: the priority of a subtask $T_{i,j}$ is chosen based on subtask's effective relative deadline. The effective relative deadline $ED_{i,j}$ of $T_{i,j}$ in a task T_i with n_i subtasks is:

$$D_i - \sum_{k=j+1}^{n_i} \tau_{i,k}$$

  $T_{i,j}$ must complete at $ED_{i,j}$ units of time after T_i is released in order for T_i as a whole to complete in time.

Table 4 lists the priorities of subtasks in Example 3 with their priorities assigned based on their effective relative deadlines.

T_i	proc	ϕ_i	p_i	$\tau_{i,j}$
$T_{1,1}$	P_1	31	50	6
$T_{1,2}$	P_2	36	50	5
$T_{1,3}$	P_1	41	50	5
$T_{1,4}$	P_2	44	50	3
$T_{1,5}$	P_3	47	50	3
$T_{1,6}$	P_1	50	50	3

Table 4: Example 2 - Priority Assignment Based on Subtasks' Effective Deadlines

Step 3 : Determine the worst-case response times for subtasks

After Step 2 we have a set of subtasks on each processor, in which (1) every subtask requires either no resource or local resources and (2) every subtask has a fixed priority. Resource-access-control protocols for single-processor systems can be used to prevent deadlocks and uncontrolled priority inversion. Both PCP and SBP can be used in this case. Furthermore, we can obtain the worst-case blocking time $\beta_{i,j}$ for each subtask $T_{i,j}$. Consequently the worst-case response time $c_{i,j}$ for each subtask can be computed according to the following equation. The derivation for this equation can be found in [5].

$$c_{i,j} = \frac{\sum_{T_{k,l} \in H_{i,j}} \tau_{k,l} + \beta_{i,j}}{1 - \sum_{T_{k,l} \in H'_{i,j}} u_{k,l}} \qquad (1)$$

In this equation $H_{i,j}$ is the set of subtasks that (1) are on the same processor as $T_{i,j}$, (2) are of different tasks than T_i, and (3) have priorities equal to or higher than $T_{i,j}$. $H'_{i,j}$ is a subset of $H_{i,j}$ in which every subtask has a higher priority than $T_{i,j}$. $u_{i,j}$ is the processor utilization factor of $T_{i,j}$. Again, $\beta_{i,j}$ is the maximum blocking time $T_{i,j}$ can experience. For both PCP and SBP, $\beta_{i,j}$ can be approximated by $MAX(S_{k,l})$, where $S_{k,l}$ is the maximum duration of critical sections for all possible $T_{k,l}$ that (1) is on the same processor as $T_{i,j}$ and (2) has lower priorities than $T_{i,j}$.

Step 4 : Check schedulability for the whole system

From the results obtained in previous step, the worst-case response time for T_i can be obtained by summing up all response times of its subtasks :

$$C_i = \sum_j c_{i,j} \qquad (2)$$

If $C_i > D_i$, where D_i is the relative deadline of task T_i, we report failure for this task set. If all tasks pass this test, we report success.

4 Conclusions

In the previous section we present a procedure for applying the end-to-end approach to scheduling tasks with shared resources in a multiprocessor system and analyzing the schedulability. In order to make this approach practical, some formulas need to be improved and problems which may arise in practice need to be addressed. For example, the upper bound for worst-case response time given by Eq. (1) sometimes is not satisfactory, especially for subtasks with low priorities. A method based on time-demand analysis has been developed to give a much tighter bound and will be presented in a future paper.

Another practical problem arises when we fix the subtasks' phases to enforce the execution precedence among them. In order to make the modified phases consistent and meaningful in a multiprocessor or distributed system, clocks on all processors have to be strictly synchronized, which can be difficult to achieve in practice. We can allow some clock drift among processors, provided that the drift is within a maximum limit of δ time units. Extra δ time units can be added to the worst-case response time for each subtask obtained in the previous section, and the execution precedence relations among subtasks will be safely enforced.

Another solution to this problem is to use dynamic phasing for subtasks instead of static phasing used in this paper. In other words, a subtask can be triggered to start as soon as its previous subtask finishes. We are currently working on the schedulability analysis for such systems.

An alternative way to map tasks to subtasks is to map all critical sections, both for local resources and for remote resources, into subtasks. The resultant task system has end-to-end processing not only across processors but also within each processor. A study in [6] has shown that schedulability analysis for end-to-end processing within a processor is possible and promising. We are currently studying the schedulability analysis for such systems.

In this paper we assume that all resources accessed in one nested critical section must be on the same processor. This assumption in general can be overly restrictive. We will address this problem from the point of view of both resource access control and task/resource assignment. Ideally we want to assign resources to processors to minimize the number of nested critical sections that access resources on more than one processor.

In many ways, the end-to-end scheduling approach can be viewed as a divide-and-conquer approach: it divides the problem by mapping the given task set onto an end-to-end task set where each processor becomes relatively independent. It then resolves the local resource contention on each processor. Finally combines the results to obtain a global solution. This merit leads to a reduction in the complexity of the resource contention problem.

Acknowledgements

This work was partially supported by NSF contract No. NSF MIP 92-22408 and US Navy ONR contract No. N0001492J1815. The authors thank all members in Real-Time Systems Laboratory at University of Illinois for many informative and inspiring discussions, in particular, Too-Seng Tia for many in-depth discussions about MPCP model.

References

[1] L. Sha, R. Rajkumar and J. P. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-Time Synchronization". *IEEE Transactions on Computers*, Vol. 39, No. 9, September 1990.

[2] R. Rajkumar, *Task Synchronization In Real-Time Systems*, Kluwer Academic Publishers, Boston 1991.

[3] T. P. Baker, "A Stack-Based Resource Allocation Policy for Real-Time Processes". *Proceeding of the 11th Real-Time Systems Symposium*, pp. 191-200, 1990.

[4] R. Rajkumar, L. Sha and J. P. Lehoczky, "Real-Time Synchronization Protocols for Multiprocessors". *Proceeding: Real-Time Systems Symposium*, pp. 259-269, 1988.

[5] R. Bettati, "End-to-End Scheduling to Meet Deadlines in Distributed Systems". *Ph.D. thesis, Department of Computer Science, University of Illinois at Urbana-Champaign*, March 1994.

[6] M. G. Harbour, M. H. Klein, and J. P. Lehoczky, "Timing Analysis for Fixed-Priority Scheduling of Hard Real-Time Systems", *IEEE Transactions on Software Engineering*, Vol. 20, No. 1, pp. 13 - 28, January 1994.