Constructive Feedback Turns Failure into Success for Pre-Run-Time Scheduled Systems

Björn Allvin, Kristian Sandström and Christer Eriksson
Mälardalen Real-Time Research Centre
Mälardalen University, Sweden
http://www.mrtc.mdh.se
Email: { bai,ksm,cen }@mdh.se

Abstract

When applying pre run-time scheduling in industrial projects, engineers are faced with a problem that only to a very limited degree has been attacked by the real-time research community, namely how to provide constructive feedback to the user in cases when a feasible schedule can not be found. In this paper we describe a method for analysing the specification file. From this we derive graphs that the designer can use to pinpoint the areas in the specification that renders the system unschedulable.

1: Introduction

When applying pre run-time scheduling in industrial projects, engineers are faced with a problem that only to a very limited degree has been attacked by the real-time research community, namely how to provide constructive feedback to the user in cases when a feasible schedule can not be found. We realised the importance of this problem in a joint project with Volvo Construction Equipment AB when developing the control system for a wheel loader.

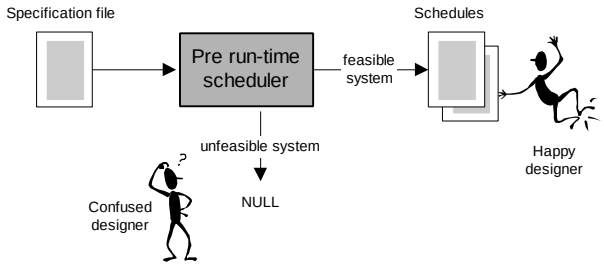

Figure 1a: Pre run-time scheduling: present situation

This *limited feedback problem* leads to confused designers (as illustrated in Figure 1a), which more or less at random have to optimise and modify the specification. However, to help the designer to come up with a specification for which the pre-run-time scheduler can find a feasible schedule, there is a need for heuristics that analyse the specification for semantic problems and give constructive feedback to the user (Figure 1b). That is, the user should be provided hints to how the problem can be overcome, i.e., how the specification can be modified to allow the generation of a feasible schedule.

This kind of feedback information is difficult to provide since the scheduling problem is computationally intractable when complex task models are used. Nevertheless, we have to come up with heuristics, otherwise it will be difficult to convince industry to use scheduling tools developed by the real-time research community.

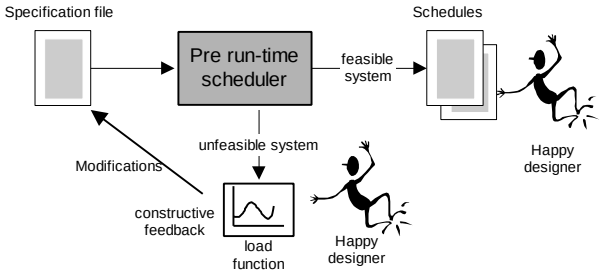

Figure 1b: Pre run-time scheduling: utilising feedback

This paper presents a method to provide feedback to the user by calculating a load function for the system. By identifying bottlenecks in the system specification we can guide the designer in modifying the input to the pre-run-time scheduler. The underlying hypothesis is that there is a correlation between the points in time when the load function has a high value, and the locality of the bottlenecks in the specification that leads to an infeasible schedule.

The problem we are facing has some relations to the sensitivity analysis provided by e.g. Punnekkat et.al [PUN97], Yerrabali et.al[YER95], and Vestal [VES94] for fixed priority scheduling. These works try to analyse the effects an increase of the worst case execution time has on the schedulability. However, none of these techniques cover complicated attributes like release times or precedence constraints.

The sequel of this paper is organised as follow. Next section presents the task model used by the scheduler. In Section 3, the system average load function and blocked intervals are presented together with a simple example. In Section 4, we present some modifications that can be done on the specification to make the bottlenecks in the specification more visible when calculating the system average load function. Finally in Section 5 we will present some preliminary results, draw some conclusions and discuss further work.

2: System model

Our task model allows a number of requirements to be expressed. For each task the following temporal requirements have to be specified:

- Period time
- Release time (relative the start of the period)
- Deadline (relative the start of the period)
- Worst case execution time
- Pre-emptive or non-preemptive

Relations between tasks can be specified by:

- Precedence relationships
- Mutual exclusion relationships (shared resources)
- Communication
    I. Synchronous communication (communication between task that have same period time)
    II. Asynchronous communication, i.e., communication between tasks running with different period times. The semantics of asynchronous communication is that messages are transferred with the lowest frequency of the sender and the receiver

In addition, all interrupts have to be specified by their minimum inter arrival time and worst case execution time of the interrupt routine.

The used pre-run-time scheduler is based on a heuristic tree search algorithm. Before scheduling starts, a comprehensive graph is constructed, to which the search algorithm is applied. The comprehensive graph contains instances of the tasks in the specification. Every instance is unique. The scheduler handles interrupts by combining exact analysis [JOS86] and the search algorithm in [SAN98]. The scheduler also takes system specific information into account, e.g., number of preemption levels allowed, communication costs, and operating system overhead.

In the sequel of this paper all references to tasks are to unique instances of tasks in the comprehensive graph.

3: System specification analysis

The feedback problem is addressed by employing two different methods: calculating the *system average load function* and finding time intervals that have to be allocated to specific tasks (denoted blocked intervals since other tasks cannot be scheduled in these intervals). The system average load function will indicate where on the timeline there is a higher risk that the scheduling will fail. Both methods will be described in detail below.

By presenting the system average load function as a graph over the complete LCM the designer can locate potential bottlenecks on the timeline and find out which tasks cause these bottlenecks. This will enable the designer to quickly focus his attention on corresponding areas in the specification file.

3.1: System Average Load (SAL)

Average load of a task is a measure of how its WCET relates to the interval in which the task can be scheduled. Task average load (TAL) is defined as follow:

$$(1)\quad TAL(T,t) = \begin{cases} \dfrac{WCET(T)}{D(T) - RT(T)} & RT(T) \le t \le D(T) \\ 0 & otherwise \end{cases}$$

Where T is a task and t is time, RT is the release time, D the deadline, and $WCET$ the worst case execution time.

When analysing the system we are, however, more interested in the system average load (SAL) function. This is easily obtained by adding up all TAL.

$$(2)\quad SAL(t) = \sum_{\forall T} TAL(T,t)$$

3.2: Blocked intervals

A blocked interval in a system is a time interval in which it is certain that a specific task has to be scheduled. A prerequisite for the existence of a blocked interval for a task is that the task has a maximum average load that is greater than 50% and that the task is non-preemptive. The 50% limit can easily be seen in Figure 2.

This means that for tasks with blocked intervals latest possible start time (*lpst*(*T*)) for the task to complete before its deadline is less than its earliest possible completion time (*epct*(*T*)). Thus, the blocked interval for task *T* is defined as [*lpst*(*T*), *epct*(*T*)], formally defined as follow:

(3)

$$lpst(T) = D(T)\text{-}WCET(T)$$
$$epct(T) = RT(T) + WCET(T)$$
$$\text{Fixed interval(T)} = \begin{cases} [lpst(T), epct(T)] & lpst(T) < epct(\text{T}) \\ NULL & otherwise \end{cases}$$

The existence of blocked intervals in the specification indicates an inflexible system, which in turn could lead to problems when trying to schedule the system.

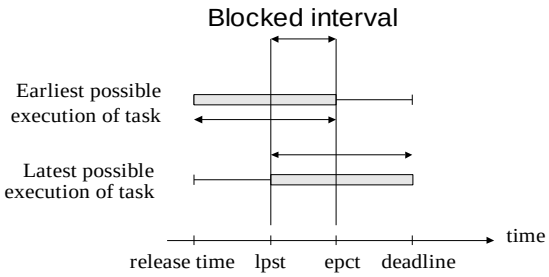

Figure 2: Definition of a blocked interval

3.3: Example

To illustrate the analysis introduced above we will present a small system containing 4 tasks. The task set is presented in Table 1.

Task	RT	D	WCET	TAL
A	0	12	3	0.25
B	2	11	2	0.222
C	7	15	5	0.625
D	9	17	2	0.25

Table 1: The task set

Apart from these temporal constraints there is also a precedence relation between task A and B, requiring task A to complete before task B can start.

Performing system average load analysis we get a SAL graph (Figure 3) which indicates a peek in average load in the interval [9,10] and a blocked interval between [10,12].

The blocked interval is caused by task C. The maximum average load is roughly 1.3.

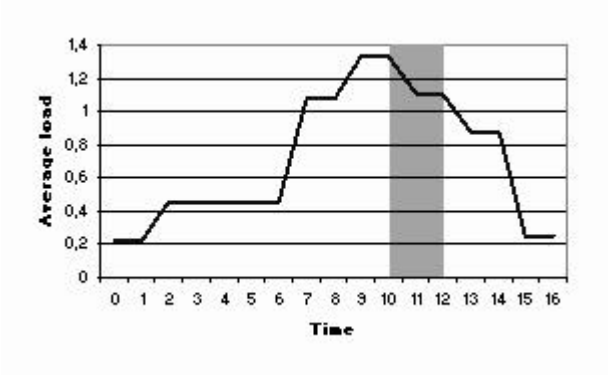

Figure 3: The System Average Load graph

The graph in Figure 3 indicated that the problem lies somewhere in the interval [8,11]. In a real system with hundreds of tasks, this interval could contain many tasks. If we could narrow down the peek in average load the process of finding bottlenecks could be facilitated.

4: Derivation of tighter constraints

An enhancement that narrows the peeks in the load function is to derive tighter constraints from the original specification. This will give the designer a clearer picture of where the actual bottlenecks can be located. The derived constraints can also reveal that the system is in fact not schedulable at all. We propose two different methods to derive tighter constraints. The first method takes advantage of the blocked interval analysis we presented earlier. The second method uses the precedence relationships given in the specification. At the end of this section we apply both methods to the example task set presented in Section 3.

4.1: Tighter constraints based on the occurrence of blocked intervals

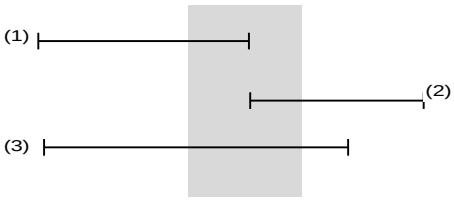

Figure 4: Modifications due to a blocked intervals created by some task in the system

Without affecting the schedulability, all tasks (except the task that causes the blocked interval) which have either deadline or release time inside the blocked interval can be modified (case (1) and (2) in Figure 4). Deadlines inside the interval are moved to the beginning of the blocked interval and release times to the end of it. In the case when a task spans over a blocked interval (case (3) in Fig.4) we have to check if the WCET of the task is small enough to execute on either side of the blocked interval. If for example the complete WCET could fit before the blocked interval but not after, we move the deadline of the task to the start of the blocked interval. In the case when the WCET fits on both sides, we can not do anything. Case (3) is of course only applicable to non-preemptive tasks.

4.2: Tighter constraints based on precedence relations

The used task model supports specification of precedence relationships between tasks. This precedence relationship information can be used to further derive an enhanced specification. We will in this paper only present modification of release times, but the basic idea also applies to deriving tighter deadlines.

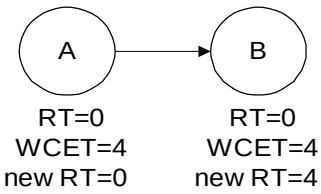

Figure 5: Precedence relationship between task A and B

For instance, in Figure 5 the start of task B requires that task A has to be finished. We can investigate whether it is actually possible for task A to be finished before the release time of B (i.e. if epct(A)≤RT(B)).

Here, the earliest completion time of A (epct(A)) is 4. So in this case we can move the release time of B to 4.

However, not many systems are as simple as the example above. In general all tasks that execute before the tasks we are examining have to be considered. All of these tasks can possibly affect, indirectly or directly, the release time of the task we are considering. The complete algorithm for finding these tasks is described below.

Algorithm for modifying release time of a task T.

1. List M = {All tasks that immediately precedes task T}
2. S = Min (Release times of tasks in M)
3. I = The interval [S, RT(T)]
4. M = M + All tasks with deadline in interval I
5. Do step 2 – 4 until no tasks are added to M
6. Sort M by ascending order of release time.
7. Initialise variable t to the smallest release time of the tasks in M.
8. A = First task in M (i.e. task in M with smallest release time). B = second task in M.
9. If t + WCET(A) > RT(B) then t = t + WCET(A) else t = RT(B)
10. Remove task A from M
11. Do step 8 to 10 until M is empty.
12. If t > RT(T) then RT(T) = t, if not then the preceding tasks of T has no effect on the release time of T

4.3: Example of enhancing a system specification

Let us revisit the example in Section 3.3 and apply the enhancement methods. Firstly we focus on any blocked intervals in the system. In the average load column in Table 2 we see that task C has an average load exceeding 0.5, this mean there exists a blocked interval. The blocked interval is visible in the graph in Figure 2 as a grey area from 10 to 12. All tasks with deadlines or release times in this interval will get modified deadlines or release times when applying the derivation method from Section 4.1.

Task	RT	D	WCET	TAL	RT_{new}	D_{new}	TAL_{new}
A	0	12	3	0.25	0	10	0.3
B	3	11	2	0.222	3	10	0.286
C	7	15	5	0.625	7	15	0.625
D	9	17	2	0.25	12	17	0.4

Table 2: The task set and the task set after optimisation.

From Table 2 it can be seen that the following modifications has been made:

Task A: Deadline shortened to 10 due to the blocked interval.

Task B: Deadline shortened to 10 due to the blocked interval and release time moved forward due to the precedence relation to task A.

Task C: No changes.

Task D: Release time moved to 12 due to the blocked interval.

If we now create a graph over the modified system (Figure 6) we can see how the peeks in average load has become higher, but the duration of the peeks has become smaller. This is related to the fact that the area under the graph represents the total WCET of all tasks in the system. Hence the area under the two graphs are the same.

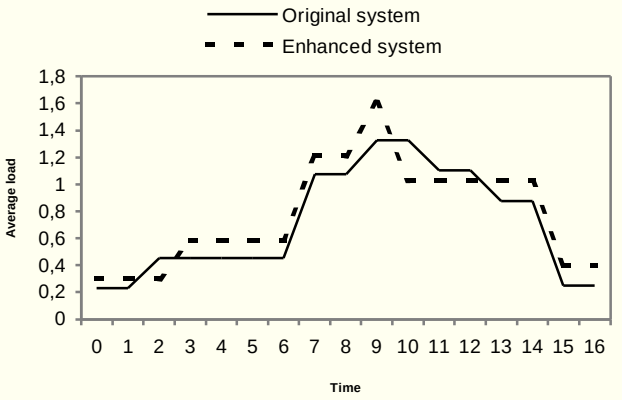

Figure 6: Average load before and after enhancement

The graph generated from the modified specification gives a clearer picture of where potential bottlenecks can be found. From being in the interval [8,11] we have now narrowed the problem down to be at time 9 on the timeline.

5: Conclusion

During an industrial project where a pre-run-time scheduler was used we identified the need of providing constructive semantic feedback to the user when the scheduling failed.

We have in this paper presented some ideas that can be used to give feedback to the user when a system is not schedulable. We see this as a first step towards giving "constructive" semantic feedback to the user.

The tool we have developed has been used for analysing a real specification that includes about 600 task instances. In this experiment we found a substantial correlation between the tasks causing peeks in the SAL function and the tasks that the designers had to alter to get a feasible schedule. This is an indication of the appropriateness of our approach.

Our current work is to further validate our approach as well as identifying other criteria for finding bottlenecks. This is done by generating infeasible specifications, passing them through the tool, and thereafter studying the correlation between load peeks and the locality of the needed changes in the specification.

6: Acknowledgement

We would like to thank the members of the MRTC group, especially Jukka Mäki-Turja, Hans Hansson, and Sasikumar Punnekkat for valuable comments on earlier versions of this paper.

7: References

[JOS86] M. Joseph and P.K. Pandya. Finding response times in a real-time system. Comp. J., 29(5). 1986.

[SAN98] Kristian Sandström, Christer Eriksson, and Gerhard Fohler. Handling Interrupts with Static Scheduling in an Automotive Vehicle Control System. RTCAS98, Hiroshima, Japan 1998.

[PUN98] Sasikumar Punnekkat, Rob Davis, and Alan Burns. Sensitivity Analysis of Real-Time Task Sets. Asian Computing Science Conference, Kahlmandu December 1997.

[YER95] R.Yerraballi et.al. Issues in Schedulability Analysis of Real-Time Systems. Proceedings of Seventh Euromicro Workshop on Real-Time Systems, June 1995.

[VES94] S. Vestal. Fixed Priority Sensitivity Analysis for Linear Compute Time Models. IEEE Transaction on Software Engineering, April 1994.