Comment on "A Pre-Run-Time Scheduling Algorithm for Hard Real-Time Systems"

Tarek F. Abdelzaher, and Kang G. Shin

Real-time Computing Laboratory
Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, Michigan 48109–2122
{*zaher,kgshin*}@*eecs.umich.edu*
313-763-0391 (voice); 313-963-4617 (fax)

ABSTRACT

In [1] a branch-and-bound implicit enumeration algorithm is described to generate a feasible schedule, if any, for each processor on a multiprocessing node running hard real-time processes. The optimization criterion is to minimize process lateness defined as the difference between the process completion time and deadline. We show in this correspondence that this algorithm does not always succeed in finding a feasible solution, and describe the reason why the algorithm might fail.

Key Words — Pre-run-time scheduling, hard real-time systems, branch and bound scheduling algorithm, multiprocessors, precedence constraints.

1 Introduction

The authors of [1] describe a pre-run-time branch-and-bound implicit enumeration algorithm which determines a feasible schedule for a set of hard real-time processes subject to precedence and exclusion constraints. Processes are assumed to be statically assigned to processors on a multiprocessing node.

Compared to *ad hoc* hand-crafted scheduling methods, the application of this pre-run-time scheduling algorithm to hard real-time systems should reduce the resources required for run-time scheduling and context switching. Unless the application violates its load specification, pre-run-time scheduling also provides a means to verify and guarantee the observance of timing constraints throughout the life cycle of the system. In addition, the scheduling model used in [1] provides a systematic and tractable approach to modifying application software. Namely, when the software is modified, the applicable scheduling model parameters may be altered, and a new set of run-time schedules can be generated. This approach eliminates the need for fine tuning the code via repeated stochastic simulations and may reduce the cost of software maintenance. An important contribution of [1] is applying their algorithm to a model of a *real* system, namely the F$-$18 Mission Computer Operational Flight Program [2]. Due to the importance of automated pre-run-time scheduling to the design and maintenance of hard real-time systems, we would like to clarify in this correspondence the fact that the algorithm in [1] is not guaranteed to find a feasible solution (where no process misses its deadline), when one exists. We also provide the reason why it occasionally fails.

The general idea of the branch-and-bound algorithm in [1] is to minimize *schedule lateness* defined as the maximum process lateness over all scheduled processes. At each search vertex an earliest deadline first (EDF) schedule is found for each processor (subject to the defined constraints between processes), a process with the maximum lateness (the *latest process*) is identified, and an attempt to reduce its lateness is made. The authors of [1] argue that "to improve the lateness of the latest process in a schedule, that process must be forced to precede or preempt another process presently preceding it on the same schedule" (p. 673). This is accomplished by appropriately introducing an extra precedence or preemption constraint. A search vertex is expanded by generating a child vertex for each possible lateness improvement; that is, for each possible way the latest process may be forced to precede or preempt some earlier process on the same schedule (by introducing an extra constraint) such that schedule lateness is reduced. The algorithm terminates when a feasible solution is reached, or when no further improvements (over the non-feasible schedules obtained thus far) are possible, i.e., no more vertices can be expanded. As will be demonstrated below, in the latter case a feasible solution may actually exist.

2 Counter-example

Consider a set of four processes T_1, T_2, T_3, and T_4, with a constraint T_3 *precedes* T_2. No exclusion constraints are defined (specifically, no exclusion constraint exists between T_1 and T_2). Each process T_i has an arrival time r_i, a computation time c_i, and a deadline d_i as shown in Table 1. To account for precedence constraints, task arrival times and deadlines are modified as described in [1]. Thus, the arrival time r_2 becomes $r_2 = 0 + c_3 = 2$. Similarly, the deadline d_3 becomes $d_3 = 12 - c_2 = 7$.

Assume the system has two processors, P_1 and P_2. Let processes T_1 and T_2 be assigned to processor P_1, and processes T_3 and T_4 be assigned to processor P_2. Figure 1-a depicts the EDF schedule obtained at the root of the search. The latest process is T_2 which is scheduled after process T_1 on P_1. The schedule is not feasible because T_2 misses its deadline. According to [1], since T_1 has an earlier deadline than T_2, forcing T_2 to precede or preempt T_1 can only increase schedule lateness. Thus, the root vertex cannot be expanded and the algorithm terminates with no feasible solution. However, a feasible solution does exist! If process T_3 is scheduled before T_4 on P_2, all processes meet their deadlines as depicted in Figure 1-b.[1] The resulting schedule, however, is no longer EDF.

The reason why the algorithm failed to find a feasible solution is that it attempts to reduce schedule lateness by modifying only the schedule of the processor running the latest task. In general, if the latest process has predecessors on other processors it is possible to improve lateness by shifting these predecessors earlier in their schedules. This aspect was apparently overlooked by the authors of [1].

[1] To obtain the schedule in Figure 1-b we also need to permit T_1 to preempt T_2 by introducing a corresponding *preemption constraint* [1].

Task	c_i	r_i	d_i
T_1	4	4	11
T_2	5	0	12
T_3	2	0	12
T_4	3	0	6

Table 1: Process data.

It is conjectured that one way to fix this flaw is to extend the notion of a *contiguous set*, used in [1], to span over multiple processors[2]. The contiguous set, as defined in [1], is the set of all processes immediately preceding the latest process on the same processor in a continuous execution sequence (including the latest process itself). The branching function considers rescheduling only the processes in the contiguous set, all of which are on the same processor with the latest process. A pre-run-time multiprocessor scheduling algorithm which uses the extended notion of the contiguous set is proposed in [3]. It defines it as a set $Z[l]$ of *all* processes in the period preceding and including the latest process such that within that period there does not exist any time where *all* processors are idle. This allows the branching function to modify schedules of other processors when rescheduling processes in the contiguous set. Further intuition behind this definition is described in [3].

It should be noted that [3] does not supersede the algorithm in [1] because it attempts to solve a slightly different multiprocessor scheduling problem. While [3] allows processes to run/resume on any processor, [1] assumes a static process-to-processor assignment which is an extra constraint. Also, [3] considers non-preemptive scheduling, while [1] allows preemption.

References

[1] T. Shepard and M. Gagne, “A pre-run-time scheduling algorithm for hard real-time systems,” *IEEE Transactions on Software Engineering*, vol. 17, no. 7, pp. 669–677, July 1991.

[2] T. Shepard and M. Gagne, “A model of the f-18 mission computer software for pre-run-time scheduling,” in *IEEE 10th International Conference on Distributed Computing Systems*, Paris, France, May 1990.

[3] J. Xu, “Multiprocessor scheduling of processes with release times, deadlines, precedence and exclusion relations,” *IEEE Transactions on Software Engineering*, vol. 19, no. 2, , February 1993.

[2] The application of this idea to the algorithm in [1] was suggested by one of the referees reviewing an earlier version of this correspondence.

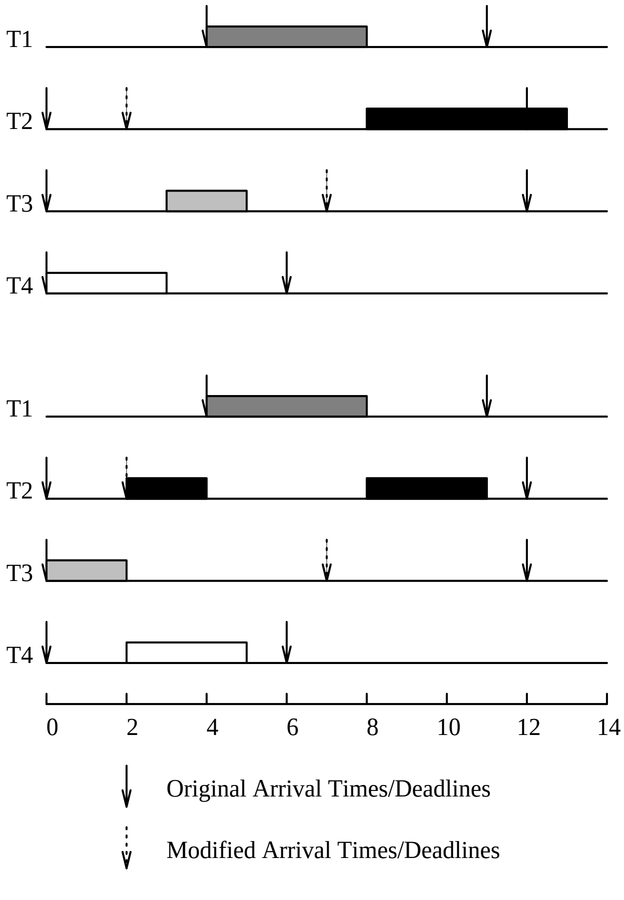

Figure 1-a.
The best schedule obtained
by the algorithm in [1].

Figure 1-b.
The optimal schedule for
the given task set.

Figure 1: The process schedule.