Non-Preemptive Fixed-Priority Uniprocessor Scheduling where the Execution Time of a Job Depends on the Scheduling of Jobs that Executed Before it

Björn Andersson, Dionisio de Niz and Sagar Chaki
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA, USA
{baandersson, dionisio, chaki}@sei.cmu.edu

Abstract— We consider non-preemptive fixed-priority scheduling of a set of constrained-deadline sporadic tasks on a single processor. We assume that the execution time of a job J depends on the sequence of jobs executed before J, that is, the execution time of a job of a task is not a constant. We raise the following two open problems (i) given a priority assignment, can the response time of a task be computed in pseudo-polynomial time? and (ii) how to create an optimal priority-assignment scheme?

I. MOTIVATION

With today’s processors, the execution time of a task is heavily dependent on whether a memory operation (load/store) results in a cache hit or a cache miss. Even with non-preemptive scheduling and even with a single processor, the execution time of a job of a task depends on the scheduling of other tasks. As an illustration, consider tasks T1, T2 and T3 which are scheduled non-preemptively on a single processor. Jobs of task T1 start their execution by reading a variable x, where x is an array. Ditto for the jobs of task T2; jobs of task T2 start their execution by reading a variable x. However, jobs of task T3 never access variable x. For each of the tasks T1 and T2, it holds that its jobs have the execution time 5 milliseconds if variable x was not in the cache. If a job of task T2 executed immediately after a job of task T1, then T2’s job will experience a cache miss when referencing the variable x but we may be able to prove that when the job of task T2 references variable x, it results in cache hits when T2 references x and hence the execution time of T2’s job becomes 4 milliseconds.

Therefore, we need a scheduling theory which takes into account the fact that the execution time of a job may depend on which jobs executed just before it. Unfortunately, the current research literature offers no such scheduling theory.

II. MODEL

Task and platform characterization. We consider a system comprising a task set $\tau$ composed of $n$ constrained-deadline sporadic tasks. A task $\tau_i \in \tau$ is characterized by integers $D_i$ and $T_i$ with the interpretation that the task generates a (potentially infinite) sequence of jobs where the arrival times of jobs by $\tau_i$ are separated by at least $T_i$ time units and a job of task $\tau_i$ must finish its execution within $D_i$ time units after its arrival.

The execution time of a job depends on the job executing before it, and we therefore define the following concepts for task $\tau_i$. The symbol $n histories_i$ is an integer greater than or equal to one. $history length_i^h$ is an integer greater than or equal to one and it is defined for $1 \leq h \leq n histories_i$. The symbol $history item_{i,k}$ is an integer in $\{1, 2, 3, \ldots, n\}$ and it is defined for $1 \leq h \leq n histories_i$ and $1 \leq k \leq history length_i^h$.

We say that the for job $J$ generated by task $\tau_i$, the execution time $C_i^h$ is historyallowed if it holds for the $history length_i^h$ jobs that executed before $J$ that for each $j \in \{1, 2, \ldots, history length_i^h\}$, the $j$th job before $J$ is the generated by the task with index $history item_{i,j}$. The execution time of a job is the minimum among all its historyallowed execution times. Note that $C_i^h$ is historyallowed if $history item_{i,h}=0$. We assume that for each task $\tau_i$ there is one $h$ such that $history length_i^h=0$.

Figure 1a shows an example task set in this model.

Scheduling. We assume that each task $\tau_i$ is assigned a priority $prio_i$ and each job generated by task $\tau_i$ is given the priority of the task that generated the job. We say that a job $J$ is eligible for execution at time $t$, if (i) job $J$ arrives at $t$ or earlier and (ii) job $J$ finishes execution later than $t$.

We assume non-preemptive scheduling, that is, if a job has started to execute then it will continue to execute until it finishes. When a job finishes, the job selected for execution is the one with the highest priority among the jobs that are eligible for execution at that time. Figure 1(b) and Figure 1(c) show examples of schedules.
n=3,

\[ T_1=50, D_1=11, \quad T_2=150, D_2=14, \quad T_3=500, D_3=500, \]
\[ nhistories=2, \quad nhistories=2, \quad nhistories=1 \]
\[ C_1=5, historylength_1=0, \quad C_1=5, historylength_1=0, \quad C_1=5, historylength_1=0 \]
\[ C_2=4, historylength_2=1, historyitem_2=2, \quad C_2=4, historylength_2=1, historyitem_2=2 \]

(a) An example of a task set. This task set models that task \( \tau_1 \) and \( \tau_2 \) share some variables and therefore executing one of them just before the other reduces the execution time of the other.

(b) A schedule generated for a specific arrival pattern for the task set in (a).

(c) A schedule generated for another specific arrival pattern for the task set in (a).

Figure 1. An example of a task set and two examples of schedules that can be generated for different arrival patterns.

**Response time and schedulability.** The response time of a job is the time that the job finishes execution minus the arrival time of the job. The response time of a task \( \tau_i \) (denoted \( R_i \)) is the maximum response time that a job of \( \tau_i \) can experience. We say that a task set is *schedulable* with respect to priority assignment \( P \) if \( \forall i: R_i \leq D_i \). We say that a task set is *non-preemptive fixed-priority feasible* if there exists a priority assignment such that the task set is schedulable with respect to this priority assignment. We say that a priority-assignment scheme \( A \) is *optimal* if for each task set that is non-preemptive fixed-priority feasible, the task set is schedulable with respect to the priority given by the priority-assignment scheme \( A \).

Note that in Figure 1(b), the job of task \( \tau_2 \) has execution time of four time units because it executes after a job of task \( \tau_1 \). Hence the job of task \( \tau_2 \) meets its deadline. Classical non-preemptive analysis however, which does not consider that the execution time of a job depends on the job that executes before it, would calculate an upper bound on the response time being one time unit longer than the one in the example in Figure 1(b) and hence classical non-preemptive analysis would deem the task set in Figure 1(a) unschedulable.

Figure 1(c) shows an example of a schedule for another arrival pattern. Note here that the job by \( \tau_2 \) does not execute directly after a job of task \( \tau_1 \) and hence the execution time of the job of task \( \tau_2 \) is five, that is, one time unit more than it was in Figure 1(b).

III. OPEN PROBLEM FORMULATION

We propose the following two open problems:

**OP1.** Is it possible to create an algorithm, with pseudo-polynomial time-complexity, which computes \( R_i \)?

**OP2.** How to create an optimal priority assignment scheme.

We believe these two problems are interesting because both of them have (affirmative/positive) answers/solutions for the case that the execution time of a job does not depend on its history. But for our model they are unresolved.

**ACKNOWLEDGMENT**

We are grateful to discussions with Russell Kegley (Lockheed Martin), Brian Dougherty (Virginia Tech) and Jules White (Virginia Tech) who brought to our attention this problem.