Summary of Dynamic Scheduling of Real-Time Tasks under Precedence Constraints

Joakim Fredriksson, Jakob Lindskog, Fredrik Mårtensson, Mattias Hallefält

2019-10-21

Abstract

In this paper, we give a summary of the report Dynamic Scheduling of Real-Time Tasks under Precedence Constraints.

Introduction

This paper discusses different dynamic scheduling models for periodic tasks in conjunction with sporadic tasks. The tasks were run on a real-time computing system. The scheduling models are tested on a single processor system, the system can also receive sudden increases in processor workload from unpredictable changes in the input. The authors main concern is during runtime being available to determine if a group of sporadic tasks that has to be executed will have permission to run.

Methodology, Results and Analysis

General model

The structure of the general model is the following for periodic and sporadic.

Periodic:

For periodic tasks J a given task is defined by the following.

$J = \{T_i(s_i, C_i, R_i, P_i), i = 1 \; to \; n\}$

T	Task
s	Start time
C	Execution time
R	Deadline
P	Period

Because of the periodic nature a given task T_i executes on time $s_i + kP_i$ where $k \to \infty$

Sporadic:

For sporadic tasks $\mathbb{S}$ a given task is defined by the following.

$\mathbb{S} = \{S_i(r_i, C_i, d_i), i = 1 \; to \; n\}$

s	tASK
r	Release time
C	Execution time
d	Deadline

The sporadic tasks may need access to the system at any given time as long as its r_i have been achieved. If S_i and $i \geq 1$ then it is presumed that S are in a partial order. Where each task have successor and each successor have a predecessor. Defined by $S_i < S_j$ where S_j is the direct successor of S_i.

Lemma 1

A sequence $\mathbb{S}$ of aperiodic tasks can be scheduled by Earliest Deadline **iff** $\forall j, i \in [1, m]$ such that $r_i \leq r_j, d_i \leq d_j$:

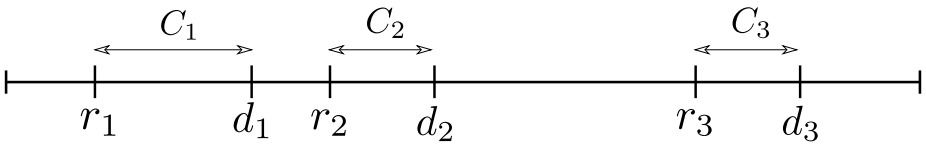
$$\sum_{r_k \leq r_i d_k \leq d_j} C_k \leq d_j - r_i$$

This lemma is illustrated in the figure below:

$j = 1..m$
$i = 1..m$
$r_i \leq r_j, d_i \leq d_j$

Lemma 1

$$\sum_{\substack{r_k \leq r_i \\ d_k \leq d_j}} C_k \leq d_j - r_i$$



$i = 1, j = 2$ $\qquad$ $i = 2, j = 3$

$C_1 \leq d_2 - r_1$ $\qquad$ $C_1 + C_2 \leq d_3 - r_2$

$i = 1, j = 3$

$C_1 \leq d_3 - r_1$

```
def lemma_1():
    for i in range(n_tasks-1):
        for j in range(1, n_tasks):
            if not sum(C[:i]) <= d[j] - r[i]:
                return False
    return True
```

The lemma was originally stated in Chetto and Chetto 1987.

Lemma 2

$$\sum_{i=1}^{n} \frac{C_i}{R_i} \leq 1$$

Equation 2 applied on a sequence $\mathbb{J}$ of periodic tasks implies that said sequence can be scheduled by the Earliest Deadline strategy, but thus is only a sufficient condition (Liu and Layland 1973) and (Labetoulle 1974).

According to (Leung and Merril 1980) a necessary and sufficient condition can be obtained by forming a ED schedule in the period of time T.

If $\mathbb{J}$ is a synchronous task set ($s_i = s_j \, \forall i \geq 1 \, \wedge j \leq n$), then $T \in [0, P]$.

If $\mathbb{J}$ is an asynchronous task set ($s_i \neq s_j \, \exists i \geq 1 \wedge j \leq n$), then $T \in [0, s_{max} + 2P]$.

The acceptance problem, that is if we should accept a certain aperiodic task at a given time τ, was recently proposed as solution in (Chetto and Chetto 1989), which says that one can use a common optimal strategy to schedule the tasks, such as ED. An aperiodic task that occurs at time τ with deadline d should be accepted if a valid schedule within $[d, D + P]$ can be found, where D represents the latest deadline of aperiodic tasks supported by the machine at time τ; usually the end of this period.

In other words, the newly occurring can be accepted if there is no-op time in the existing schedule and period, which is described by the feasibility test in equation (1).

Scheduling dependent tasks

A partial order $<$ on $\mathbb{S}$ is defined as $S_i < S_j$ such that $\mathbb{S}$ can be ordered to some valid schedule.

If $\mathbb{S}$ is not preemptable (the tasks aren't interrupted) and all tasks are ready at $t = 0$ ($\forall i,\ r_i = 0$), there exists an optional scheduling algorithm according to (Lawler and Moors 1969). There exists no optional schedule if $\exists i,\ r_i \neq 0$.

If $\mathbb{S}$ *is* preemptable though, there exists an optional scheduling algorithm for tasks that become ready at different times ($\forall i, j\ \ r_i \neq r_j$), which calculated a modified deadline d_i^* for each task in $\mathbb{S}$.

$$d_i^* = \min(d_i, \min(d_j; S_i \leq S_j))$$

This modified deadline d_i^* has time complexity $O(m^2)$. From the modified deadlines, a list of ready ($\forall t, S_i \in \mathcal{L}_\nabla \iff r_i \leq t$) tasks $\mathcal{L}_r$ is constructed. Another list for available ($S_i \in \mathcal{L}_a$ at $t \iff S_i \in \mathcal{L}_r \wedge d_j \leq t,\ \forall j \leq i$) tasks is also constructed, which contains all tasks that are waiting to be executed. $\mathcal{L}_a$ is ordered according to the modified deadlines.

Precedence between tasks are specified in one of two ways.

- Synchronizations are explicit by the task program. This abstracts nothing from a task's perspective.
- Synchronization is done by the scheduler, using a number of private variables that are updated upon task completion.

Conclusion

is usetd for seeing if ED have enif time to finch all the tasks, but it dose not check if it is schedule by ED. to make shore that ED can schedule the periodic tasks exists a strategy that can be just. it works by seening if the ED can construct a schedule that contains all the tasks (and have time to finche them), the strategy looks different if you are deling with synchronous tasks or asynchronous tasks. lets begin with synchronous task, it recuaers that all tasks begin at the same

time. and white this recoermat is it sufficient that the schedule can make a working schedule with in the time intervalt of [0,P] where P is the least common multiple with all the period times of the tasks. and thanks to the periodic nature of the tasks, if it works betwine [0,P] it works for all[(0+k)P,(1+K)P] where K = 0,1,2 .... asynchronous tasks doesn't have the same start time, this results in that it is not psoibol to just see from [0,P] becoose tasks with a later start time

Lemma 3

git

Dependent tasks

Lemma 1 - 2 are used for tasks which are non-dependent meaning tasks that are preemptable(tasks that can interupt at any given time $t \geq 0$) and have neither a precedence nor a timing constraint. However for tasks which are dependent on other tasks the theorem presented is not suitable. The dependent tasks which are non preemptable but have a timing constraint there is a scheduling algorithm. If a task do not have a timing constraint no possible schedule is possible if that tasks only run in polynomial time meaning $t \geq 0$.

For tasks that are preemptable and $r_i \neq r_j$ for any task and $i < j, i \to \infty$ there is a optimal scheduling algorithm. The algorithm uses a different deadline

$$d^* = min\{d_i, min(d_j; S_i < S_j)\}$$

The algorithm sorts by EDF but uses the modified deadline

Conclusion

Scheduling dependent tasks

New results about dependent task scheduling

In order to deal with scheduling dependent tasks together with independent tasks. The algorithm confists of two parts, calculation and creating a priority list. Both precedence and timing constrains are obeyed for all scheduable task sets. The basic idea is that the priority of a task is decided on the current deadline and upcomming deadlines and how the starting time of a task depends on the release and completion time of its predecessors.

// Some random explaination on code/math