A Hybrid Proactive Approach for Integrating Off-line and On-line Real-Time Schedulers

Weirong Wang[1], Aloysius K. Mok[1], and Gerhard Fohler[2]

[1] Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712-1188
`{weirongw, mok}@cs.utexas.edu`
[2] Department of Computer Engineering
Malardalen University, Sweden
`gerhard.fohler@mdh.se`

Abstract. The issue of integrating event-driven workload into existing static schedules has been addressed by Fohler's Slot Shifting method [5] [6]. Slot Shifting takes a static schedule for a time-driven workload as input, analyzes its slacks off-line, and makes use of the slacks to accommodate an event-driven workload on-line. Slot Shifting is a *reactive* method in the sense that it does not address how to produce a static schedule so that it can be successfully integrated with the event-driven workload. We shall show that the choice of the static schedule cannot be considered independent of the event-driven workload. We propose a *proactive* hybrid scheduler with both an off-line component and an on-line component. Time-driven workload and event-driven workload are modeled as periodic tasks and sporadic tasks respectively. The off-line component produces a pre-schedule. The on-line component schedules all periodic and sporadic jobs by using a variation of the EDF scheduler with one additional constraint: the sequencing of the periodic jobs in the static schedule must be as specified by the pre-schedule. The off-line component is optimal in the sense that it produces a valid pre-schedule for a given periodic task set and a given sporadic task set if and only if one exists.

1 Introduction

An embedded computer system may be required to process periodic as well as non-periodic tasks. For example, consider a node in a wireless sensor network. The wireless node may collect information from its sensors and performs signal processing transformations on the data at fixed time intervals that may be characterized by periodic tasks [8]. The node may also perform mode changes and relay control signals for state machines among the nodes, and these tasks may be characterized as sporadic tasks [9]. This paper addresses the problem of scheduling a combination of periodic tasks and sporadic tasks by introducing a concept called proactive scheduling. The novelty of this concept lies in a combination of off-line and on-line scheduling techniques that can be shown to be optimal and also preserve the ordering of jobs that may be important for the off-lined scheduled periodic tasks. A hybrid proactive scheduler will be presented for scheduling a mixed set of periodic and sporadic tasks.

The Cyclic Executive (CE) [1] is a well accepted scheduling technique for periodic tasks. A CE schedule is produced off-line to cover the length of a hyper-period, i.e., the least common multiple of the periods of all the tasks. The CE schedule is represented as a list of executives, where each executive defines an interval of time in a hyper-period to be allocated to a specific task. During on-line execution, a CE scheduler partitions the time line into an infinite number of consecutive hyper intervals, each the length of a hyper-period, and repeats the CE schedule within each hyper interval. The significant advantages of CE include the following: (1) the on-line overhead of CE is very low, $O(1)$ and can usually be bounded by a small constant. (2) a variety of timing constraints, such as mutual exclusion and distance constraints can be solved by off-line computation [12], which might otherwise be difficult to handle by typical on-line schedulers such as the Earliest Deadline First (EDF) scheduler. However, the drawback of CE is that it does not provide sufficient flexibility for handling the unpredictable arrival times of sporadic tasks. Even though sporadic tasks can be modeled as pseudo periodic tasks [9] and can therefore be scheduled by CE, this method may reserve excessive capacity whenever the deadline of a sporadic task is short compared with its mean time of arrival, as is typically the case of mode changes in state machines.

The Earliest Deadline First scheduler (EDF) is known to be optimal for scheduling periodic and sporadic tasks [8] [9]. EDF requires that at any moment during on-line scheduling, among all arrived but unfinished jobs, the job with the nearest deadline be selected for execution. However, the on-line complexity of EDF is $O(lgn)$, which is higher than that of CE ($O(1)$). Other than the potential problem with over capacity as mentioned above, the EDF scheduler does not provide strong predictability as the CE scheduler in terms of guaranteeing the ordering of jobs; the ordering of jobs to be scheduled is unknown off-line in general. This ordering may be important if programmers exploit this ordering to minimize the use of data locks at run time, as is common practice in avionics software systems.

Seeking for a balanced solution, Fohler has investigated the issue of integrating event-driven workload, modeled as aperiodic tasks into pre-computed static schedules by the Slot Shifting [5] method. Isovic and Fohler further integrated sporadic tasks into this approach [6]. In the Slot Shifting approach, the "slacks" left within the static schedule are computed off-line and then applied on-line to accommodate event-driven workload. While the Slot Shifting approach tests if a given set of aperiodic or sporadic tasks can be scheduled within a given static schedule, this method is *reactive* in the sense that it does not address how to generate a static schedule for periodic tasks to fit with the aperiodic and sporadic tasks.

In this paper we shall go further to propose a *proactive* hybrid scheduling approach, as shown in Figure 1. There is an off-line component and an on-line component in our approach. The off-line component is called Slack-Reserving Pre-scheduler (SRP), and it produces a flexible pre-schedule with slacks embedded in it. A pre-schedule consists of a list of job fragments, and it defines a fixed order by which the fragments of periodic jobs are to be scheduled. A pre-schedule is *not* a schedule, because it does not define the exact time intervals in which each fragment is scheduled. Instead, a fragment in a pre-schedule may be preempted and/or delayed on-line to accommodate the unpredictable arrivals of sporadic tasks. The on-line component schedules the pre-schedule together with the jobs of the sporadic tasks by a Constrained Earliest Deadline First (CEDF) scheduler.

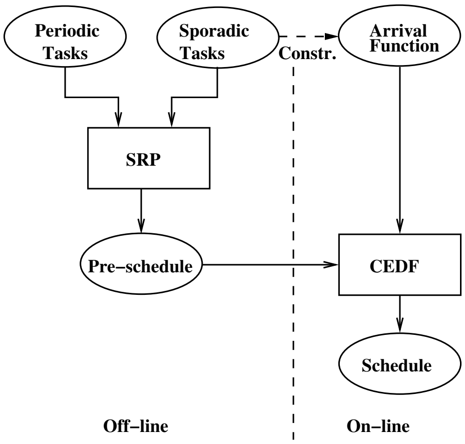

Fig. 1. Framework of the Proactive Hybrid Scheduler

The first important contribution of this paper is an optimal pre-scheduler SRP. The reactive Slot Shifting approach considers as input a given static schedule for a periodic task set and a sporadic task set; if the off-line component finds that the static schedule is "bad" because it cannot be successfully integrated together with the sporadic tasks on-line, the reactive method leaves as an unanswered question whether there exists a "good" static schedule for the periodic task set that may be integrated with the sporadic/aperiodic workload. In contrast, SRP will produce a "good" pre-schedule for the given set of periodic tasks and sporadic tasks if and only if one exists.

The second contribution of this paper is the demonstration of the non-existence of universally "good" pre-schedule in general: Given a fixed set of periodic tasks, there might exist a set Φ of sporadic task sets, and there exists a "good" pre-schedule with each specific sporadic task set in Φ, but there might not exist a single common pre-schedule "good" with all sporadic task sets in Φ. This fact implies that an optimal pre-scheduler needs the information of the sporadic task set as input.

The remainder of the paper is organized as follows. Section 2 defines the task models and the definition of schedule. Section 3 defines pre-schedule, the on-line component, and the validity of a pre-schedule. Section 4 describes and analyzes the off-line component SRP. Section 5 shows the non-existence of universally valid pre-schedule in general. Section 6 addresses related work. Section 7 summarizes this paper and points out future work.

2 Task Model

We shall adopt the usual model of real-time tasks and assume that a task can be modeled either as a periodic task or as a sporadic task with preperiod deadlines. All workloads are modeled as being made up of a periodic task set $\mathbf{T_P}$ and a sporadic task set $\mathbf{T_S}$. Each task T in either $\mathbf{T_P}$ or $\mathbf{T_S}$ is an infinite sequence of jobs. A job is defined by a triple: ready time, deadline and execution time and is written as $(r,\ d,\ c)$; the job must receive a total execution time of c between time=r and time=d at run-time. A periodic task is defined by a 4-tuple: initial offset, period, relative deadline, and execution time and is written as $(o,\ p,\ l,\ c)$. The first job of a periodic task is ready at time=o, and the subsequent jobs are ready at the beginning of each period exactly p time units apart. We shall adopt as a convention in this paper the notation $X.a$ which denotes the attribute a of entity X. For example, the j^{th} job of a periodic task T may be written as $(T.o + j \cdot T.p,\ T.o + j \cdot T.p + T.l,\ T.c)$, starting with job 0. Similarly, a sporadic task is defined by a tuple: (p, l, c), with its attributes defined the same way as a periodic task, except that the period of a sporadic task is the minimal length of the time interval between two consecutive jobs, and the actual ready time of any job of a sporadic task cannot be known until it arrives at run-time. We use an arrival function A to represent the arrival times of sporadic jobs in a particular run. A valid arrival function must satisfy the minimal arrival interval constraint: for any two consecutive jobs J_i and J_{i+1} of a sporadic task T, it must be the case that $A(J_{i+1}) - A(J_j) \geq T.p$. A job J of sporadic task T will be written as $(A(J),\ A(J) + T.l,\ T.c)$.

Let the hyper-period P be the least common multiple of the periods of all periodic tasks in $\mathbf{T_P}$ and all sporadic tasks in $\mathbf{T_S}$. For any nature number n, the time interval $(n \cdot P,\ (n+1) \cdot P)$ is a called hyper interval. Our scheduler will generate a pre-schedule for the length of one hyper-period and repeat the same sequence of jobs of the periodic tasks every hyper interval at run-time, i.e., the on-line component of our scheduler restarts the pre-schedule at the beginning of every hyper interval. For ease of discussion, let $\mathbf{J_P}$ represent the list of the jobs of the periodic tasks within one hyper interval. Every job in $\mathbf{J_P}$ occurs exactly once in every hyper interval, and will be called a *periodic job* in the remainder of this paper. Notice that in this terminology, there are as many jobs from the same periodic task in a hyper-period as the number of periods of the task in a hyper-period. Individual Jobs from the same periodic task within a hyper-period may be considered as periodic with respect to the length of the hyper-period.

For each periodic task T in $\mathbf{T_P}$, for any natural number j less than $\frac{P}{T.p}$, there is a periodic job in $\mathbf{J_P}$ which is defined by $(r,\ d,\ c) = (T.o + j \cdot T.p,\ T.o + j \cdot T.p + T.l,\ T.c)$. Job J is *before* job J' and job J' is *after* job J if and only if either (1) $J.r < J'.r$ and $J.d \leq J'.d$; or (2) $J.r \leq J.r$ and $J.d < J'.d$. Job J *contains* job J' or job J' *is contained by* job J if and only if $J.r < J'.r$ and $J'.d < J.d$. Job J is *parallel with* job J' if and only if $J.r = J'.r$ and $J.d = J'.d$. For scheduling purposes, parallel periodic jobs can be transformed into a single periodic job with their aggregate execution time. Therefore, we shall not consider parallel jobs in the remainder of the paper. We assume that $\mathbf{J_P}$ is sorted such that a job with lower index is either before or contained by a job with higher index.

Example 1. The task sets $\mathbf{T_P}$ and $\mathbf{T_S}$ defined below are used in later examples in this paper.

$$\mathbf{T_P} = \{(90, 225, 30, 5), (0, 75, 75, 15), (0, 225, 225, 135)\}; \quad \mathbf{T_S} = \{(225, 25, 25)\}$$

The hyper-period of the task sets P is 225. The set of periodic jobs $\mathbf{J_P}$ is defined as follows.

$$\mathbf{J_P} = [(0, 75, 15), (90, 120, 5), (75, 150, 15), (0, 225, 150), (150, 225, 15)]$$

Fig. 2 illustrates the periodic tasks and the periodic jobs. The vertical bars indicate the ready times and deadlines of periodic jobs, and the length of the box inside the scope of a periodic job indicates its execution time. ■

We shall assume that all tasks are preemptable and are scheduled on a single processor. A *schedule* is represented by a function S which maps each job to a set of time intervals. For instance, $S(J) = \{(b_0, e_0), (b_1, e_1)\}$ means that job J is scheduled to two time intervals (b_0, e_0) and (b_1, e_1). A schedule must satisfy the following two constraints. Firstly, there is at most one job to be scheduled at any point in time, i.e., at any time t, there exists at most one job J, by which $(b, e) \in S(J)$ and $b < t < e$. Secondly, the accumulated time allocated to

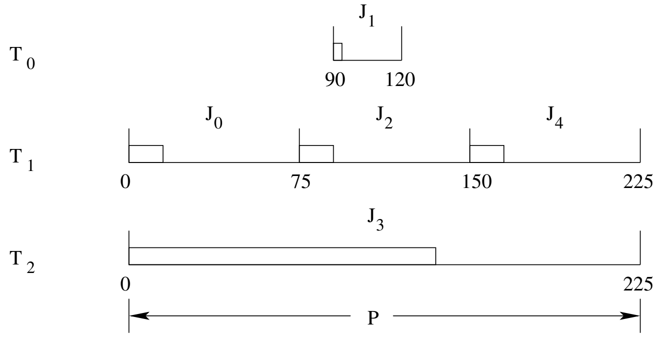

Fig. 2. Definition of $\mathbf{T_P}$ and $\mathbf{J_P}$

each job between its ready time and deadline must be no less than its specified execution time, i.e., for a job J with execution time c, $\sum_{(b,e)\in S(J)}(min(J.d,e) - max(J.r,b)) \geq J.c$.

3 Definitions of Pre-Schedule and the On-line Component

A *pre-schedule* $\mathbf{F}$ is a list of fragments. A *fragment* F is defined by a 5-tuple, (j,k,r,d,c) with the following meaning: Suppose J be the j^{th} periodic job in job list $\mathbf{J_P}$. F is the k^{th} (starting from 0) fragment in the pre-schedule of job J, and job J is scheduled for at least a total execution time of c between times$=r$ and time$=d$ in every hyper interval.

The CEDF scheduler is the EDF scheduler plus one additional constraint: the sequence of the periodic jobs must follow the pre-schedule exactly. It may be implemented as follows. At the beginning of each hyper interval, set the first fragment in the pre-schedule be marked as "current". Define $\mathbf{R}$ as the set of sporadic jobs waiting to be scheduled. The set $\mathbf{R}$ is initialized at time 0 as an empty set. When a sporadic job becomes ready, it is added into $\mathbf{R}$; when it is completely scheduled, it is removed from $\mathbf{R}$. At any time, if the deadline of the current fragment is earlier than the deadline of any job in $\mathbf{R}$, the current fragment is scheduled; otherwise, the sporadic job with the earliest deadline in $\mathbf{R}$ is scheduled. When the execution time of the current fragment is completely allocated, set the next fragment in the pre-schedule as "current", and so on.

Example 2. Pre-schedule $\mathbf{F}$ is a pre-schedule for $\mathbf{J_P}$ and $\mathbf{T_S}$ defined in Example 1.

$$\begin{aligned}\mathbf{F} = [&(0,0,0,75,15),(3,0,0,120,60),(2,0,75,120,15),\\ &(1,0,90,120,5),(3,1,90,225,90),(4,0,150,225,15)]\end{aligned}$$

Suppose that the first job of a sporadic task T_3, written as J_S arrives at different times in two different runs of the system, denoted by the arrival functions

A and A', such that $A(J_S) = 0$ and $A'(J_S) = 30$. The on-line scheduler will produce two different schedules S and S', as illustrated in Fig. 3. Each box in the schedule represents an interval of time scheduled to a job. The fragments corresponding to periodic jobs being scheduled are on the top of the boxes in the figure. Notice that the start times and the finish times of periodic jobs may vary to accommodate the arrivals of sporadic jobs, but the order defined by the pre-schedule is always followed. ■

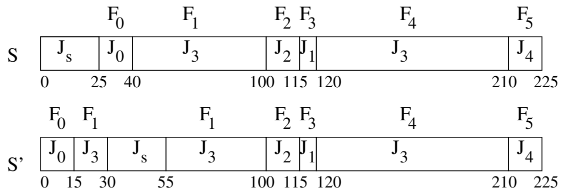

Fig. 3. Schedules Accommodating Different Arrival Functions

A pre-schedule is *valid* if and only if the following three conditions are satisfied. Firstly, under any valid arrival function A of $\mathbf{T_S}$, the on-line component always produces a valid schedule for $\mathbf{T_S}$ and $\mathbf{J_P}$. Secondly, for any fragment F, suppose J is its corresponding periodic job. Then the ready time and the deadline of F will be within the effective scheduling scope of J, i.e., $J.r \leq F.r \leq F.d \leq J.d$. Third, for any fragment F, the execution time $F.c$ is greater than or equal to 0.

4 The Off-line Component

The off-line component SRP tries to produce a pre-schedule in three steps. The first step establishes $\mathbf{F^{(j,k)}}$, a list of partially defined fragments, in which only the attributes j and k of each fragment are defined. Essentially, the first step defines a total order of fragments in the pre-schedule. The following two steps will not change the ordering of the fragments. They only define the other attributes for each fragment. The second step defines the ready time r and deadline d of each fragment and produces $\mathbf{F^{(j,k,r,d)}}$. The third step defines the execution time c of each fragment to produce $\mathbf{F^{(j,k,r,d,c)}}$, the completed pre-schedule which is also represented as $\mathbf{F}$. We shall describe the algorithms in detail step by step and show that SRP is optimal in the sense that it produces a valid pre-schedule if and only if one exists.

4.1 Step 1: Generating $\mathbf{F^{(j,k)}}$

In this step, a partially defined pre-schedule $\mathbf{F^{(j,k)}}$ is created according to the following constraint-based definition.

A periodic job in $\mathbf{J_P}$ is a *top* periodic job if and only if it does not contain any other periodic job in $\mathbf{J_P}$. Constraint 1 and 2 are about the fragments of top periodic jobs.

Constraint 1 Each top periodic job has one and only one fragment.

Constraint 2 Let J and J' be any pair of top periodic jobs, and F and F' be their corresponding fragments. Fragment F is before fragment F' if and only if periodic job J is before periodic job J'.

Constraint 3 and Constraint 4 are about the fragments of non-top periodic jobs. In the definition of these constraints, we assume that J_T and J'_T are two consecutive top periodic jobs in $\mathbf{J_P}$; i.e., J_T is before J'_T and there exists no periodic job J''_T after J_T and before J'_T. We also assume that the fragments corresponding to J_T and J'_T are F_T and F'_T.

Constraint 3 If a periodic job J contains either J_T or J'_T, then there is one and only one fragment F of J between F_T and F_T; Otherwise, there exists no fragment of J between F_T and F_T.

Constraint 4 Assume that both fragment F of periodic job J and fragment F' of periodic job J' are between and excluding F_T and F'_T. Fragment F is before fragment F' if and only if either J is before J', or J contains J' and F is not the last fragment of J.

Example 3. Assume that $\mathbf{J_P}$ is defined in Example 1. Jobs J_0, J_1, and J_4 are top jobs, while job J_2 and J_3 are not. Partially defined pre-schedule $\mathbf{F^{(j,k)}}$ is shown below.

$$\mathbf{F^{(j,k)}} = [(0,0),(3,0),(2,0),(1,0),(2,1),(3,1),(4,0)]$$

4.2 Step 2: Generating $\mathbf{F^{(j,k,r,d)}}$

This step augments $\mathbf{F^{(j,k)}}$ to $\mathbf{F^{(j,k,r,d)}}$; in other words, it defines the ready time r and deadline d for every fragment. The ready times of fragments are defined as the earliest times satisfying the following constraints: (1) the ready time of each fragment is not earlier than the ready time of its corresponding job; (2) the ready times of fragments are non-decreasing. Similarly, the deadlines of fragments are defined as the latest times satisfying the following constraints: (1) the deadline of each fragment is not later than the deadline of its corresponding job; (2) the deadlines of fragments are non-decreasing.

Example 4. Assume that $\mathbf{J_P}$ is defined in Example 1 and $\mathbf{F^{j,k}}$ is defined in Example 3. Partially defined pre-schedule $\mathbf{F^{j,k,r,d}}$ is defined as shown below.

$$\mathbf{F^{(j,k,r,d)}} = [(0,0,0,75),(3,0,0,120),(2,0,75,120),(1,0,90,120),\\ (2,1,90,150),(3,1,90,225),(4,0,150,225)]$$

4.3 Step 3: Generating $\mathbf{F^{(j,k,r,d,c)}}$

This step augments $\mathbf{F^{(j,k,r,d)}}$ to $\mathbf{F^{(j,k,r,d,c)}}$ by assigning the execution time c for every fragment. At the beginning of this step, we augment $\mathbf{F^{(j,k,r,d)}}$ to $\mathbf{F^{(j,k,r,d,x)}}$, representing the execution time of each fragment as a variable; then we solve the variables with a Linear Programming (LP) solver under three sets of constraints: non-negative constraints, sufficiency constraints and slack-reserving constraints, which are defined as follows.

Non-negative constraints require that the execution times to be non-negative; i.e., for every fragment F, $F.x \geq 0$.

Notice that a fragment may have zero execution time. A fragment with zero execution time is called a zero fragment; otherwise it is a non-zero fragment. Zero fragments are trivial in the sense that we can either delete or add them from or to a pre-schedule, and the schedule produced according to the pre-schedule will not be modified at all.

Sufficiency constraints require that for every periodic job J in $\mathbf{J_P}$, the aggregate execution time of its fragments in the pre-schedule shall be equal to the execution time of J; i.e., $\sum F.x = J.c$, where F is any fragment of J.

A slack-reserving constraint requires that the aggregate execution time of fragments and all sporadic jobs that must be completely scheduled within a time interval shall be less than or equal to the length of the time interval. An execution may start at time 0 but last infinitely, therefore the number of time intervals is infinite. In order to make the pre-scheduling problem solvable, we need to establish a finite number of critical slack-reserving constraints, such that if all critical slack-reserving constraints are satisfied by a pre-schedule, all slack-reserving constraints are satisfied. For this purpose, we define critical time intervals.

A time interval (b, e) is *critical* if and only if all of the following conditions are true. First, there exists a periodic job J in $\mathbf{J_P}$ and $J.r = b$. Second, the length of the time interval is shorter than or equal to the hyper-period; i.e., $e - b \leq P$. Third, at least one of the following cases is true: there exists a periodic job J in $\mathbf{J_P}$ and either $J.d = e$ or $J.d + P = e$; or there exists a sporadic task T in $\mathbf{T_S}$, such that $e - b = T.p \cdot n + T.d$, where n is a natural number.

In order to define slack-reserving constraints on critical intervals, we introduce two functions, $E(b, e)$ and $Slack(l)$. Function $E(b, e)$ represents the aggregate execution time of all fragments that must be completely scheduled between critical time interval (b, e), and function $Slack(l)$ represents the maximal aggregate execution time of sporadic jobs that must be completely scheduled within a time interval of length l. The slack-reserving constraint on a critical time interval (b, e) is $E(b, e) \leq e - b - Slack(e - b)$.

Function $E(b, e)$ is computed with the following two cases. For the first case, time interval (b, e) is within one hyper interval $(0, P)$; i.e. $e \leq P$. Let F_b be the first fragment with a ready time greater than or equal to b, F_e be the last fragment with a deadline less than or equal to e, then $E(b, e) = \sum F.x$, where F is between and including F_b and F_e. For the second case, time interval (b, e) straddles hyper interval $(0, P)$ and hyper interval $(P, 2P)$, i.e., $e > P$. Then let F_b be the first fragment with a ready time equal or after to b, and let F_e be the last fragment with a deadline earlier than or equal to $e - P$, function $E(b, e) = \sum F.x$, where F is any fragment including and after F_b or before and including F_e.

Function $Slack(l)$ is computed as follows. Let function $n(T, l)$ be the maximal number of jobs of T that must be completely scheduled within l. If $l - \lfloor \frac{l}{T.p} \rfloor \cdot T.p < T.d$, $n(T, l) = \lfloor \frac{l}{T.p} \rfloor$; otherwise, $n(T, l) = \lfloor \frac{l}{T.p} \rfloor + 1$. Function $Slack(l)$ is equal to $\sum_{T \in T_S} T.c \cdot n(T, l)$.

Function $E(b, e)$ is a linear function of a set of variables, and $Slack(e - b)$ is a constant; therefore, the slack-reserving constraint is a linear constraint.

Non-negative constraints, sufficiency constraints and slack-reserving constraints on critical intervals are all linear and their total number is finite; therefore, the execution time assignment problem is a LP problem, which can be solved optimally in the sense that if there exists an assignment to the execution times under these constraints, the assignment will be found in finite time. Optimal LP solvers are widely available. If an optimal LP solver returns an assignment to execution times, SRP produces a fully defined pre-schedule with the execution times; otherwise, SRP returns failure.

Example 5. Assume that $\mathbf{J_P}$ and $\mathbf{T_S}$ are defined in Example 1, and $\mathbf{F^{(j,k,r,d)}}$ is defined in Example 4.

All non-negative constraints are listed as follows.

$$F_0.x \geq 0; \quad F_1.x \geq 0; \quad F_2.x \geq 0; \quad F_3.x \geq 0; \quad F_4.x \geq 0; \quad F_5.x \geq 0$$

All sufficiency constraints are listed as follows.

$$F_0.x = 15; \quad F_3.x = 5; \quad F_2.x + F_4.x = 15; \quad F_1.x + F_5.x = 150; \quad F_6.x = 15$$

The slack-reserving constraints on many critical time intervals will be trivially satisfied. For instance, time interval $(0, 75)$ is a critical time interval, however, $E(0, 25) = F_0.x = 15$, $Slack(75 - 0) = 25$, therefore the slack-reserving constraint on this time interval is always satisfied. We list all non-trivial slack-reserving constraints below.

$$\begin{array}{ll} F_1.x + F_2.x \leq 75 & \text{critical interval } (0, 120) \\ F_4.x + F_5.x \leq 90 & \text{critical interval } (90, 225) \end{array}$$

The pre-schedule $\mathbf{F}$ in Example 2 satisfies all constraints above, so it is a valid pre-schedule. ■

4.4 Optimality and Computational Complexity

We prove the optimality and complexity of SRP in Theorem 1 and 2.

Lemma 1. *Slack-reserving constraints on all time intervals are satisfied by a pre-schedule produced by SRP.*

Assume the opposite: slack-reserving constraint is violated on time interval (b, e), which means $E(b, e) + Slack(e - b) > e - b$. Prove by two cases. Case 1: $e - b \leq P$. Let time b' be the earliest time satisfying both condition: it is between (b, e); and there exists a fragment F, and $b' = F.r + n \cdot P$, where n is a natural number; if such a time does not exist, let $b' = 0$. Let $l = e - b$. Let e' be the latest time between $(b', b' + l)$ such that (b', e') is a critical interval. Then $E(b', e') = E(b', b' + l) \geq E(b, e)$ and $Slack(e' - b') = Slack(e - b)$, and $e' - b' \leq l$, therefore $E(b', e') + Slack(b', e') > e' - b'$, which means the slack-reserving constraint must be violated on critical interval (b', e'). Contradiction.

Case 2: $e - b > P$. There exists a latest e' between (b, e), such that e' is either a deadline of a fragment F, i.e., $e' = F.d + n \cdot P$; or there exists a sporadic task T, and $e' = b + T.p \cdot n + T.d$. Then $E(b, e') = E(b, e)$ and $Slack(e' - b) = Slack(e - b)$, so $E(b, e') + Slack(e' - b) > e' - b$. Because $E(e' - P, e') + Slack(P) \leq P$ must be true, $E(b, e' - P) + Slack(e' - b - P) > e' - b - P$, which means slack-reserving constraint is also violated on interval $(b, e' - P)$. We may repeat this argument to show that the slack-reserving constraint is violated on an interval with $e - b \leq P$, where Case 1 applies. ■

Lemma 2. *If SRP produces a pre-schedule* $\mathbf{F}$*, then* $\mathbf{F}$ *is valid.*

To prove $\mathbf{F}$ is valid, we need to prove that it satisfies the three conditions as defined in Section 3. First, the ready time and deadline of each fragment in $\mathbf{F}$ are always assigned within the effective range of its corresponding job by Step 2 of SRP. Second, the non-negative constraints in Step 3 of SRP guarantee the execution time of each fragment are non-negative. Third, we prove that the online component always produces a valid schedule according to $\mathbf{F}$ by contradiction. Assume the opposite, then there are two possibilities: Case 1, CEDF fails to meet a deadline of a fragment or a sporadic job. Case 2, CEDF meets every deadline, but it does not schedule sufficient execution times for a periodic job in its effective range. Suppose Case 1 is true, then CEDF fails at a time e. Let time b be the latest ideal time before time e; if such an ideal time does not exist, let b equal to time 0. Then slack-reserving constraint on time interval (b, e) must be violated, which contradicts to Lemma 1. Suppose Case 2 is true, then at least one sufficiency constraint in Step 3 of SRP is violated. Contradiction. ■

Lemma 3. *If a valid pre-schedule exists, SRP produces one pre-schedule.*

Proof: The strategy of our proof is as follows. Assume $\mathbf{F}'$ is a valid pre-schedule. First we transform $\mathbf{F}'$ into another valid pre-schedule $\mathbf{F^1}$, which can be obtained by augmenting attributes r, d, and c to each fragment of $\mathbf{F^{(j,k)}}$, the partially defined pre-schedule generated by Step 1 of SRP. Then we further transform $\mathbf{F^1}$ to $\mathbf{F^2}$, which can be obtained by augmenting attribute c to each fragment of $\mathbf{F^{(j,k,r,d)}}$ generated by Step 2 of SRP. Because $\mathbf{F^2}$ is a valid pre-schedule, there exists a pre-schedule satisfying all constraints in Step 3 of SRP, therefore

SRP must produce a pre-schedule. We define the transformations and show the correctness of the claims below.

Transformation 1: from $\mathbf{F}'$ to $\mathbf{F^1}$

Apply the following rules to pre-schedule one at a time, until no rule can be further applied.

Rule 1: If F_b and F_e are two consecutive fragments of the same job J, merge them into one fragment F of job J, with $F.c = F_b.c + F_e.c$, $F.r = F_b.r$ and $F.d = F_e.d$.

To facilitate other rules, we first define a primitive $Swap(J_f,\ J_r,\ F_b,\ F_e)$, which swap all non-zero fragments (fragments with non-zero execution times) of job J_f before all non-zero fragments of job J_r between and including fragments F_b and F_e. It is implemented as follows. Let C_f be the aggregate execution time of all fragments of job J_f between and including F_b and F_e. Let $C_{[x,y)}$ be the aggregate execution time of all fragments of job J_f or J_r including and after F_x but before F_y. Let F_m be the latest fragment of either job J_f or job J_r, such that $C_{[b,m)}$ is less than or equal to C_f. For every fragment F of job J_r including and after F_b but before F_m, change it to a fragment of job J_f, without changing its ready time, deadline, or execution time. Similarly, for every fragment F of job J_f after F_m but before and including F_e, change it to a fragment of job J_r. If $C_{[b,m)} = C_f$, make F_m a fragment of J_r. Otherwise, split F_m into two consecutive fragments F_{mf} of job J_f and F_{mr} of job J_r, such that the ready times and deadlines of F_{mf} and F_{mr} are the same as those of F_m, but $F_{mf}.c = C_f - C_{[b,m)}$ and $F_{mr}.c = F_m - F_{mf}.c$.

Rule 2: Assume that job J_f is before job J_r. Let F_b be the first non-zero fragment of J_r, and F_e be the last non-zero fragment of J_f. If F_b is before F_e in pre-schedule, apply *Swap*.

Rule 3: Assume that job J_r contains job J_f. Let F_b and F_e be the first and the last non-zero fragments of J_f. If there exists a non-zero fragment of J_r between F_b and F_e, apply *Swap*.

Rule 4: Let F_T and F'_T be non-zero fragments of two consecutive top jobs J_T and F'_T, F_T is before F'_T and there is no other non-zero fragment of top job between them. Assume that non-zero fragment F_b of J_r is before non-zero fragment F_e of J_f, and both of them are between F_T and $F_{T'}$. If either J_f contains J_r and both J_f and J_r contain J'_T, or J_r contains J_f and only J_r contains job J'_T, apply *Swap*.

Claim 0: Transformation 1 terminates.

Cyclic swaps will not occur according to the definition of Transformation 1, therefore it must finally terminate.

Claim 1: Pre-schedule $\mathbf{F^1}$ is a valid pre-schedule.

None of the transformation rules will change the aggregate execution time of either any periodic job or any critical time interval. Also, none of transformation rules changes the ready time or deadline of a fragment beyond the valid scope of its corresponding job, or results in negative execution times.

Claim 2: Pre-schedule $\mathbf{F^1}$ can be obtained by augmenting $\mathbf{F^{(j,k)}}$.

By the transformation rules, if $\mathbf{F^1}$ does not satisfy the constraints listed in Step 1 of SRP, Transformation 1 won't terminate. Notice that if there is no fragment F of J between any consecutive pair of fragments of top jobs in $\mathbf{F^1}$, we can always plug in such a fragment F of job J with 0 execution time at proper position. Therefore, there is a one-to-one in-order mapping between $\mathbf{F^1}$ and $\mathbf{F^{(j,k)}}$, which implies the claim.

Transformation 2: from $\mathbf{F^1}$ to $\mathbf{F^2}$

Re-assign attribute r and d of every fragment according to the algorithm in Step 2 of SRP.

Claim 3: $\mathbf{F^2}$ is a valid pre-schedule.

For every fragment, its ready time and deadline are still within the valid range of its corresponding periodic job, and the execution time of each fragment remains unchanged. If CEDF produces a schedule according to $\mathbf{F^2}$, then every job, either sporadic or periodic, is sufficiently scheduled between its ready time and deadline. Therefore, if $\mathbf{F^2}$ is not valid, there must exist an arrival function A, such that CEDF fails with it at a time e, which is either the deadline of a sporadic job or a periodic job. Let b be the latest ideal time before e or time 0 if such an ideal time does not exist. Let F_b^2 be the first fragment in $\mathbf{F^2}$ with a ready time at or after time b and F_e^2 be the last fragment in $\mathbf{F_2}$ with a deadline before or at time e. Fragment F_b^2 must be the first fragment of its corresponding job J_b and $F_b^2.r = J_b.r$. Let F_b^1 be the corresponding fragment in $\mathbf{F^1}$, then $F_b^2.r \leq F_b^1.r$. Similarly, $F_e^1.d \leq F_e^2.d$. Then all fragments between and including F_b^1 and F_e^1 must be scheduled between time b and e when CEDF schedules according to $\mathbf{F^1}$. With the same arrival function A, CEDF must also fail with $\mathbf{F^1}$, which contradicts with Claim 1.

Claim 4: $\mathbf{F^2}$ can be obtained by augmenting $\mathbf{F^{(j,k,r,d)}}$.

Follows Claim 2 and the fact that the attributes r and d of fragments in $\mathbf{F^2}$ are defined by the algorithm in Step 2 of SRP.

Claim 5: A valid pre-schedule satisfies all non-negative, sufficient and slack-reserving constraints.

If a sufficiency constraint is not satisfied, CEDF will not schedule sufficient execution time for a job; If slack-reserving constraint on a critical time interval is not satisfied, CEDF fails with certain valid arrival function. In both cases, CEDF does not produce a valid schedule, which contradicts with the definition of a valid pre-schedule. Non-negative constraint directly follows the definition of valid pre-schedule.

Claim 6: SRP must produce a pre-schedule.

Because of Claim 3 and 5, $\mathbf{F^2}$ satisfies all non-negative, sufficient execution time and slack-reserving constraints. Together with Claim 4, $\mathbf{F^2}$ indicates the existence of an execution time augment to $\mathbf{F^{(j,k,r,d)}}$ that satisfies all those constraints. Therefore Step 3 of SRP must return a pre-schedule. ■

Theorem 1. *SRP is optimal in the following sense: given* $\mathbf{J_P}$ *and* $\mathbf{T_S}$*, if a valid pre-schedule exists, SRP produces a valid pre-schedule; otherwise, SRP returns failure.*

This theorem immediately follows Lemma 2 and Lemma 3. ■

Theorem 2. *The computational complexity of SRP is* $O(C(n_p^2, n_p \cdot (n_p + n_s))$*, where* n_p *and* n_s *are the number of periodic jobs and the maximal number of sporadic jobs in one hyper-period respectively, and* $C(n, m)$ *represents the complexity of LP solver with* n *variables and* m *constraints.*

Proof: The complexity of LP solver is the major factor. The maximal number of fragments is bounded by n_p^2. The number of non-negative constraints and the number of execution time constraints are bounded by n_p, and the number of slack-reserving constraints on critical intervals is bounded by $n_p \cdot (n_p + n_s)$. ■

5 The Non-Existence of Universally Valid Pre-schedule

The slack embedded in a pre-schedule by SRP specifically targets one given set of sporadic tasks. Is it possible to produce a one-size-fits-all pre-schedule? To formalize the discussion, we define the concept of *universally valid* pre-schedule. For a given periodic task set $\mathbf{T_P}$, a pre-schedule $\mathbf{F_U}$ is universally valid if and only if the following condition is true for any sporadic task set $\mathbf{T_S}$: $\mathbf{F_U}$ is a valid pre-schedule for $\mathbf{T_P}$ and $\mathbf{T_S}$ if and only if there exists a valid pre-schedule for $\mathbf{T_P}$ and $\mathbf{T_S}$. If a universally valid pre-schedule exists for every periodic task set, then a more dynamic optimal hybrid scheduling scheme could be constructed. A universally valid pre-schedule might be produced off-line without requiring the knowledge of the sporadic task set, and the sporadic task set might be allowed to change on-line, and an admission control mechanism could be implemented with the schedulability test defined in [6]. The optimality criterion for scheduling here means that if a specific sporadic task set is rejected on-line, then no valid pre-schedule exists for this sporadic task set. However, by Example 6, we can show that the one-size-fits-all pre-schedules are impossible: universally valid pre-schedule does not exist in general, so the more dynamic scheme we surmise above cannot be done.

Example 6. Suppose one periodic task set and two alternative sporadic task sets are defined as follows:

$$\mathbf{T_P} = \{(56, 100, 19, 9), (0, 100, 100, 71)\}; \quad \mathbf{T_s} = \{(50, 10, 10)\}; \quad \mathbf{T'_s} = \{(20, 4, 4)\}$$

With both sets of sporadic tasks, hyper-period $P = 100$, and the periodic job list is defined as follows.

$$\mathbf{J_P} = [(56, 75, 9), (0, 100, 71)]$$

There exists a valid pre-schedule $\mathbf{F}$ for $\mathbf{T_P}$ and $\mathbf{T_S}$, and a valid pre-schedule $\mathbf{F'}$ for $\mathbf{T_P}$ and $\mathbf{T'_S}$, defined as follows.

$$\mathbf{F} = [(1, 0, 0, 75, 46), (0, 0, 56, 75, 9), (1, 1, 56, 100, 25)]$$
$$\mathbf{F'} = [(1, 0, 0, 75, 48), (0, 0, 56, 75, 9), (1, 1, 56, 100, 23)]$$

Suppose there is a universally valid pre-schedule $\mathbf{F_U}$. Let X_b be the total execution time of all fragments of J_1 before the last fragment of J_0 in $\mathbf{F_U}$; let X_a be the aggregate execution time of all fragments of J_1 after the first fragment of J_0 in $\mathbf{F_U}$. A universally valid pre-schedule $\mathbf{F_U}$ must satisfy the following set of constraints:

$$\begin{aligned} X_b + X_a &\geq 71 && \text{sufficiency constraint for } J_1 \\ X_b &\leq 46 && \text{slack-reseving constraint on } (0, 75) \text{ for } \mathbf{T_S} \\ X_a &\leq 23 && \text{slack-reseving constraint on } (56, 100) \text{ for } \mathbf{T'_S} \end{aligned}$$

The constraints contradict with each other, therefore $\mathbf{F_U}$ does not exist. ■

6 Related Work

In the introduction section, we have reviewed the Slot Shifting scheme [5] [6], CE scheduler [1] and EDF scheduler [8]. In this section, we review other closely related work.

Gerber *et al* proposed a parametric scheduling scheme [7]. They assume that the execution order of real-time tasks is given, the execution times of tasks may range between upper and lower bounds, and there are relative timing constraints between tasks. The scheduler consists of an off-line component and an on-line component. The off-line component formulates a "calendar" which stores two functions to compute the lower and upper bounds of the start time for each task. The bound functions of a task take the parameters of tasks earlier in the sequence as variables and can therefore be solved on-line. Based on the bounds on the start time, the on-line dispatcher decides when to start scheduling the real-time task or waiting non-real-time tasks. The parametric scheduling scheme and our proactive hybrid scheduler share the following similarities: (1) Both schemes make use of off-line computation to reduce on-line scheduling overhead and yet guarantee schedulability. (2) Both schemes make use of slacks to handle a non-deterministic workload represented either as variable parameters or as sporadic tasks. Yet, there are significant differences between these two schemes. The parametric scheduling scheme assumes that the ordering of the tasks is given, but our proactive hybrid scheduler does not. The implementation techniques of these two schemes are quite different.

In terms of implementation techniques, SRP is more related to a line of research by Erschler *et al* [4] and Yuan *et al* [13], even though their works are focussed on non-preemptive scheduling of periodic tasks. Erschler *et al* introduced the concept of "dominant sequence" which defines the set of possible sequences for non-preemptive schedules. They also introduce the concept of "top job". Building upon the work of Erschler *et al*, Yuan *et al* proposed a "decomposition approach". Yuan *et al* define several relations between jobs, such as "leading" and "containing", and apply them in a rule-based definition of "super sequence", an equivalent of dominant sequence. The partially defined pre-schedule $\mathbf{F}^{(\mathbf{j},\mathbf{k})}$ in our paper is similar to the dominant sequence or the super sequence, and we adopt some of their concepts and terminology as mentioned. However, in view of the NP-hardness of the non-preemptive scheduling problem, approximate search algorithms are applied to either the dominant sequence or super sequence to find a schedule in [4] and [13]. In this paper, the execution time assignment problem on the partially defined pre-schedule can be solved optimally by LP solver.

General composition approaches have been proposed in recent years, such as the open system environment [3] by Deng and Liu, temporal partitioning [10] by Mok and Feng, and hierarchical schedulers [11] by Regehr and Stankovic. The general composition schemes usually focus on the segregation between components or schedulers, which means that the behavior of a component or scheduler will be independent to the details of other components or schedulers. In contrast, proactive hybrid scheduler focuses on putting scheduling overhead off-line and yet provides strict on-line deadline guarantee by making intensive use of the information available about all tasks.

7 Conclusion

This paper proposes a proactive hybrid scheduler that combines the advantages of CE and EDF schedulers. We present an off-line Slack Reserving Pre-scheduling algorithm which is optimal in the sense that given a periodic task set and a sporadic task set, SRP produces a valid pre-schedule if and only if one exists. We also demonstrate the non-existence of universally valid pre-schedule, which implies that every optimal pre-scheduling algorithm requires both the periodic task set and the sporadic task set as input.

For future work, the proactive hybrid scheduler may be extended to accommodate workload models other than periodic and sporadic tasks with preperiod deadlines, such as complex inter-task dependencies, We may also apply the pre-scheduling technique to other forms of hybrid schedulers, such as a combination of CE and the fixed priority scheduler.

References

1. T. P. Baker, A. Shaw. The cyclic executive model and Ada, Proceedings of IEEE Real-Time Systems Symposium, pp.120-129, December 1988.
2. G. B. Dantzig. Linear Programming and Extensions, Princeton University Press, 1963.
3. Z. Deng and J. Liu. Scheduling Real-Time Applications in an Open Environment. Real-Time Systems Symposium, pp. 308-319, December 1997.
4. J. Erschler, F. Fontan, C. Merce, F. Roubellat. A New Dominance Concept in Scheduling n Jobs on a Single Machine with Ready Times and Due Dates, Operations Research, 31:114-127.

5. G. Fohler. Joint Scheduling of Distributed Complex Periodic and Hard Aperiodic Tasks in Statically Scheduled Systems, Real-Time Systems Symposium, pp. 152-161, December 1995.
6. D. Isovic, G. Fohler. Handling Sporadic Tasks in Off-line Scheduled Distributed Real-Time Systems, the 11th EUROMICRO Conference on Real-Time Systems, pp. 60-67, York, England, July 1999.
7. R. Gerber, W. Pugh, and M. Saksena. Parametric Dispatching of Hard Real-Time Tasks, IEEE Trans. on Computers, Vol.44, No.3, pp. 471-479, Mar 1995.
8. C.L. Liu and J.W. Layland. Scheduling Algorithms for Multi-programming in Hard Real-time Environment. Journal of ACM 20(1), 1973.
9. A. K. Mok. Fundamental Design Problems of Distributed Systems for the Hard-Real-Time Environment. Ph.D. thesis. MIT. 1983.
10. A. K. Mok, X. Feng. Towards Compositionality in Real-Time Resource Partitioning Based on Regularity Bounds. Real-Time Systems Symposium, pp. 129-138, 2001.
11. J. Regehr, J. A. Stankovic. HLS: A Framework for Composing Soft Real-Time Schedulers. Real-Time Systems Symposium, pp. 3-14, 2001.
12. Duu-Chung Tsou. Execution Environment for Real-Time Rule-Based Decision Systems. PhD thesis, Department of Computer Sciences, The University of Texas at Austin, 1997. esting for Real-Time Tasks, Real-Time Systems, Vol. 11, No. 1, pp. 19-39, 1996.
13. X. Yuan, M.C. Saksena, A.K. Agrawala, A Decomposition Approach to Non-Preemptive Real-Time Scheduling, Real-Time Systems, Vol. 6, No. 1, pp. 7-35, 1994.