An Analysis of Zero-Clairvoyant Scheduling

K. Subramani

Lane Department of Computer Science and Electrical Engineering,
West Virginia University,
Morgantown, WV
ksmani@csee.wvu.edu

Abstract. In the design of real-time systems, it is often the case that certain process parameters, such as its execution time are not known precisely. The challenge in real-time system design is to develop techniques that efficiently meet the requirements of impreciseness. Traditional models tend to simplify the issue of impreciseness by assuming *worst-case* values. This assumption is unrealistic and at the same time, may cause certain constraints to be violated at run-time. In this paper, we study the problem of scheduling a set of ordered, non-preemptive jobs under non-constant execution times. Typical applications for variable execution time scheduling include process scheduling in Real-time Operating Systems such as Maruti, compiler scheduling, database transaction scheduling and automated machine control. An important feature of application areas such as robotics is the interaction between execution times of various processes. We explicitly model this interaction through the representation of execution time vectors as points in convex sets. Our algorithms do not assume any knowledge of the distributions of execution times, i.e. they are *zero-clairvoyant*. We present both sequential and parallel algorithms for determining the existence of a zero-clairvoyant schedule.

1 Introduction

Scheduling strategies for real-time systems confront two principal issues that are not addressed by traditional scheduling models viz. parameter variability and the existence of complex timing constraints between constituent jobs. Impreciseness in problem data is of both theoretical and practical significance. From an empirical perspective, system designers have used *worst-case* values in order to address non-determinism of execution time values [Pin95]. However, the assumption that every job will have an execution time equal to the maximum value in its allowable range is unrealistic and at the same time, may cause constraint violation at run-time. In this paper, we study the problem of scheduling a set of ordered, non-preemptive jobs with non-constant execution times, with the goal of obtaining a single, rational, start time vector, such that the constraints on the jobs are satisfied. We explicitly model execution time non-determinism through convex sets. To the best of our knowledge, our work represents the first effort in studying this generalization of execution time domains. Our algorithm

J.-P. Katoen and P. Stevens (Eds.): TACAS 2002, LNCS 2280, pp. 98–112, 2002.

is Zero-Clairvoyant in that it makes no assumptions about the distribution of execution times; we present both sequential and parallel algorithms for determining the existence of such a schedule. Zero-clairvoyant schedules are also called *Static schedules*, since the schedule in every single window is the same (with appropriate offsets)[1].

We shall be concerned with the following problems:

(a) Determining the static schedulability of a job set in a periodic real-time system (defined in Section §2),
(b) Determining the dispatch vector of the job set in a scheduling window.

The rest of this paper is organized as follows: In Section §2, we detail the static scheduling problem and pose the static schedulability query. The succeeding section, viz. Section §3, motivates the necessity for Static Scheduling, while Section §4 describes related approaches to this problem. Section §5 commences the process of answering the static schedulability query posed in Section §2 through the application of *convex minimization* algorithms. The algorithm we present is very general, in that it is applicable as long as the execution time vectors belong to a convex set and the constraints on the system are linear. A straightforward parallelization of the algorithm is provided, subsequent to the complexity analysis. Section §6 specializes the algorithm in Section §5 to a number of interesting restrictions. We conclude in Section §7 by tabulating the results discussed in this paper.

2 Statement of Problem

2.1 Job Model

Assume an infinite time-axis divided into windows of length L, starting at time $t = 0$. These windows are called *periods* or *scheduling windows*. There is a set of non-preemptive, ordered jobs, $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ that execute in each scheduling window.

2.2 Constraint Model

The constraints on the jobs are described by System (1):

$$\mathbf{A}.[\boldsymbol{s} \;\; \boldsymbol{e}]^{\mathbf{T}} \leq \boldsymbol{b}, \quad \boldsymbol{e} \in \mathbf{E}, \tag{1}$$

where,

- $\mathbf{A}$ is an $m \times 2.n$ rational matrix, $\boldsymbol{b}$ is a rational $m-$vector; $(\mathbf{A}, \boldsymbol{b})$ is called the constraint matrix;
- $\mathbf{E}$ is an arbitrary convex set;

[1] We shall be using the terms Static and Zero-Clairvoyant interchangeably, for the rest of this paper

- $\boldsymbol{s} = [s_1, s_2, \ldots, s_n]$ is the start time vector of the jobs, and
- $\boldsymbol{e} = [e_1, e_2, \ldots, e_n] \in \mathbf{E}$ is the execution time vector of the jobs

In this work, we consider generalized linear constraints among jobs i.e. those that can be expressed in the form: $\sum_{i=1}^{n} a_i.s_i + b_i.e_i \leq k$, for arbitrary rationals a_i, b_i, k.

The convex set $\mathbf{E}$ serves to model the following situations:

1. Execution time variability - In real-time systems such as Maruti [MKAT92], a number of statistical runs are executed on jobs, to determine upper and lower bounds on their running time under various conditions. These intervals provide a stronger confidence factor than that provided by assuming constant values for execution times. Accordingly, the convex set $\mathbf{E}$ can be restricted to an axis-parallel hyper-rectangle (`aph`) represented by: $[l_1, u_1] \times [l_2, u_2] \ldots \times [l_n, u_n]$.
2. Execution time interaction - In power applications, the execution of processes are constrained through requirements such as: The total power consumed in a cycle is bounded by k. Note that the power consumed is proportional to the square of the execution time. Consequently, this situation can be modeled by restricting $\mathbf{E}$ to the sphere represented by: $e_1^2 + e_2^2 + \ldots + e_n^2 \leq r^2$, for suitably chosen r.

Remark: 21 *To the best of our knowledge, this work is the first attempt at modeling execution time interaction within a scheduling paradigm.*

2.3 Query Model

Before we state the schedulability query, a few definitions are in order.

Definition 1. *Static (Zero Clairvoyant) Schedule - A schedule for a set of jobs that assumes no knowledge of their execution times prior to the execution of the complete job set.*

Observe that a static schedule is perforce a rational vector, i.e. no online computation of schedules is permitted.

In this paper, we are concerned with the following problems:

1. Determining whether there exists a single rational vector $\boldsymbol{s} \in \Re_+^{\mathbf{n}}$, such that the set of constraints represented by System (1) is satisfied. `This corresponds to deciding the static schedulability query`, which is carried out by the offline schedulability analyzer,
2. Computing the dispatch vectors for the current scheduling window, which is carried out by the online dispatcher. In Static scheduling, the online dispatching is obviated by the fact that deciding the static schedulability query coincides with the generation of the dispatch schedule.

We are now ready to state the static schedulability query formally:

$$\exists \boldsymbol{s} = [s_1, s_2 \ldots s_n] \quad \forall \boldsymbol{e} = [e_1, e_2, \ldots e_n] \in \mathbf{E} \qquad \mathbf{A}.[\boldsymbol{s}, \boldsymbol{e}] \leq \boldsymbol{b}, \quad ? \tag{2}$$

The combination of the Job Model, the Constraint Model and the Query Model, together constitute an instance of a scheduling problem in the `E-T-C` scheduling framework [Sub01][2].

3 Motivation

One of the fundamental aspects of real-time scheduling is the recognition of interdependencies among jobs [DMP91,Sak94] and the conversion of event-based specifications into temporal distance constraints between jobs [Das85,JM86]. For instance, the event-based requirement: *Wait 50 ms after the first message has been dispatched before dispatching the next message* spawns the following temporal distance constraint: $s_i + 50 \leq s_j$, where s_i and s_j denote the dispatch times of successive invocations of the message generating job. Real-Time Operating Systems, such as Maruti [LTCA89,MAT90,MKAT92] and MARS [DRSK89], permit interaction of jobs through linear relationships between their start and execution times. The Real-Time specification Language MPL (Maruti Programming Language) [SdSA94] explicitly includes programmer constructs that specify temporal constraints between processes (jobs). These constructs are easily transformed into linear relationships between the start and execution times of the jobs. Real-time database applications involve the scheduling of transactions; the execution of these transactions is constrained through linear relationships [BFW97]. In database transactions, temporal constraints are used to specify when processes can access a particular data item or write out a particular data value; these constraints are also easily expressible as relationships between the start times of these processes. *Static Scheduling is the only refuge of real-time systems which do not permit the online computation of schedules.*

Traditional models use *worst-case* values for execution time variables. The following example establishes that using worst-case values for execution times will not necessarily provide a valid solution.

Example 1. Consider the following constraint system imposed on a job set with two jobs viz. $\{J_1, J_2\}$.

1. J_1 finishes before J_2 commences: $s_1 + e_1 \leq s_2$,
2. J_2 commences within 1 unit of J_1 finishing: $s_2 \leq s_1 + e_1 + 1$,
3. J_2 starts at or before time $t = 6$: $s_2 \leq 6$
4. $e_1 \in [4, 6]$.

[2] In the `E-T-C` real-time scheduling framework, a scheduling problem is completely described, by describing the execution time domain, the nature of the constraint set and the type of schedulability query as a triplet.

Substituting the worst-case time for e_1, i.e. 6 in the constraints, we obtain:

$$s_1 + 6 \leq s_2, \tag{3}$$
$$s_2 \leq s_1 + 7 \tag{4}$$
$$s_2 \leq 6 \tag{5}$$

It is not hard to see (graphically) that the only solution to the above system is:

$$\begin{bmatrix} s_1 \\ s_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 6 \end{bmatrix}$$

However, during actual execution, suppose $e_1 = 4$. Then, we have,

$$s_2 > s_1 + e_1 + 1$$

thereby violating the second constraint. In Section §5.2, we will show that the above constraint system is infeasible, i.e. there do not exist start times that can guarantee the meeting of all constraints, for all execution times.

4 Related Work

Scheduling in real-time systems has received considerable attention in system design research [Sak94,SB94,HG93]. Variations of the problem that we are studying have been studied in [Sak94], [HL89], [HL92b] and [HL92a]. This problem is briefly mentioned in [Sak94] as part of *parametric scheduling*, however no algorithm is presented for the general case. In [HL89,HL92b], the problem of scheduling real-time tasks under distance and separation constraints is considered, but the execution times are regarded as constant. To the best of our knowledge, our work represents the first attempt at studying the Static Scheduling problem, in its generality. Here, we focus on the problem of scheduling a set of jobs, in which the ordering sequence is known (and supplied as part of the input), but there exist complex inter-job dependencies, captured through linear relationships between their start and execution times. Although we restrict ourselves to addressing the feasibility of the job system, the judicious use of objective functions can be used to improve the quality of our solutions. The determination of a feasible schedule coincides with the generation of a *static dispatch-calendar* that contains the dispatching information for each job: e.g. $s_1 = 2; s_2 = 15; s_3 = 24$, is a *dispatch-calendar* for a 3-job system.

5 The Static Scheduling Algorithm

We can interpret the static schedulability query (2) as asking whether there exists a rational start time vector $\boldsymbol{s}$, without any dependencies on the execution times. The static approach is to work individually with each constraint and find the execution times that make the constraint tight (or *binding*). We then argue

in Section §5.2 that the strategy is correct, inasmuch as the goal is to produce a single start time vector $\boldsymbol{s}$ that holds for all execution time vectors $\boldsymbol{e} \in E$.

We formalize the ideas discussed above into Algorithm (5.1), which decides the static schedulability query for arbitrary, convex-constrained execution time vectors and arbitrary constraint sets between the start and execution times of the jobs.

Function STATIC-SCHEDULER $(\mathbf{E}, \mathbf{A}, \boldsymbol{b})$
1: {$\mathbf{E}$ is the execution time domain and $\mathbf{A}[\boldsymbol{s}, \boldsymbol{e}] \leq \boldsymbol{b}$ is the constraint system}
2: Rewrite the constraint matrix as: $\mathbf{G}.\boldsymbol{s} \leq \boldsymbol{b} - \mathbf{H}.\boldsymbol{e}$.
3: Set $\boldsymbol{r} = [r_1, r_2, \ldots, r_m]^T = \boldsymbol{b} - \mathbf{H}.\boldsymbol{e}$
{ each r_i is an affine function of $\boldsymbol{e} = [e_1, e_2, \ldots, e_n]$}
4: **for** (i=1 **to** m) **do**
5: Let $\rho_i = \min_{\mathbf{E}} r_i$ { ρ_i is a rational number}
6: **end for**
7: **if** $(\boldsymbol{s} : \mathbf{G}.\boldsymbol{s} \leq \boldsymbol{\rho} \neq \phi)$ **then**
8: **return**(`System has static schedule` $\boldsymbol{s}$)
{ $\mathbf{G}.\boldsymbol{s} \leq \boldsymbol{\rho}$ is the **Static Polytope**}
9: **else**
10: **return**(`System has no static schedule`)
11: **end if**

Algorithm 5.1: Static Scheduling Algorithm

The principal step in the algorithm is the reduction of execution time variables in the constraints to rational numbers, through convex minimization (Step 5). Once all constraints are so reduced, we get a simple linear system in the start time variables. This linear system is called the *Static Polytope*. We declare that System (1) is statically schedulable (i.e. query (2) is true) if and only if the Static Polytope is non-empty.

We note that Algorithm (5.1) is the offline schedulability analyzer. In case of Static Scheduling, online computation during dispatching is unnecessary, since the determination of feasibility coincides with the generation of the dispatch schedule.

5.1 Example

Before proceeding with proving the correctness of the STATIC-SCHEDULER() algorithm, we present an example to demonstrate our approach.

Example 2. Consider the two job set $\{J_1, J_2\}$, with execution times $\{e_1, e_2\}$, constrained through the following convex domain:

- $e_1 \in [0, 6], e_2 \in [0, 6]$
- $e_1 + e_2 \leq 4$

Figure (1) describes the domain.
Let the system have the following constraints:

1. J_1 finishes execution at or before job J_2 commences: $s_1 + e_1 \leq s_2$.
2. J_2 finishes at or before 12 units: $s_2 + e_2 \leq 12$.

Expressing the constraints in matrix form, we get :

$$\begin{bmatrix} 1 & -1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s_1 \\ s_2 \\ e_1 \\ e_2 \end{bmatrix} \leq \begin{bmatrix} 0 \\ 12 \end{bmatrix}$$

We first rewrite this system to separate the $\boldsymbol{s}$ and the $\boldsymbol{e}$ vectors:

$$\begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} e_1 \\ e_2 \end{bmatrix} \leq \begin{bmatrix} 0 \\ 12 \end{bmatrix}$$

Moving the $\boldsymbol{e}$ variables to the RHS, we get

$$\begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \end{bmatrix} \leq \begin{bmatrix} 0 \\ 12 \end{bmatrix} - \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} e_1 \\ e_2 \end{bmatrix}$$

which is equivalent to:

$$\begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \end{bmatrix} \leq \begin{bmatrix} -e_1 \\ 12 - e_2 \end{bmatrix}$$

Minimizing $-e_1$ over the constraint domain in Figure (1), we get $\rho_1 = -4$. Likewise, minimizing $12 - e_2$ over the constraint domain, we get $\rho_2 = 8$. Thus, the static polytope is determined by:

$$\begin{pmatrix} 1 & -1 \\ 0 & 1 \end{pmatrix} \cdot \begin{bmatrix} s_1 \\ s_2 \end{bmatrix} \leq \begin{bmatrix} -4 \\ 8 \end{bmatrix}$$

as shown in Figure (2).

Using a Linear Programming solver [Ber95], we solve the above system, to get:

$$\begin{bmatrix} s_1 \\ s_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 6 \end{bmatrix}$$

5.2 Correctness

Lemma 1. *If the final polytope*[3] *in algorithm (5.1) is non-empty, then any point on it serves as a suitable vector of start times* $\boldsymbol{s} = [s_1, s_2, \ldots, s_n]$.

[3] The resultant polyhedron will always be bounded because the jobs are ordered, i.e. $s_1 < s_2 < \ldots s_n$ and the last job has a deadline, i.e. $s_n + e_n \leq L$, where L represents the time at which the current scheduling window expires.

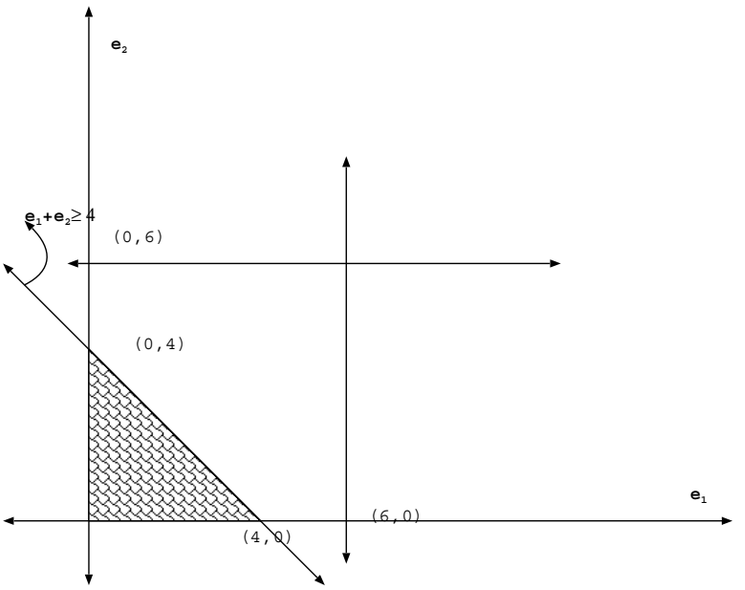

Fig. 1. Convex constrained execution times for Example (2)

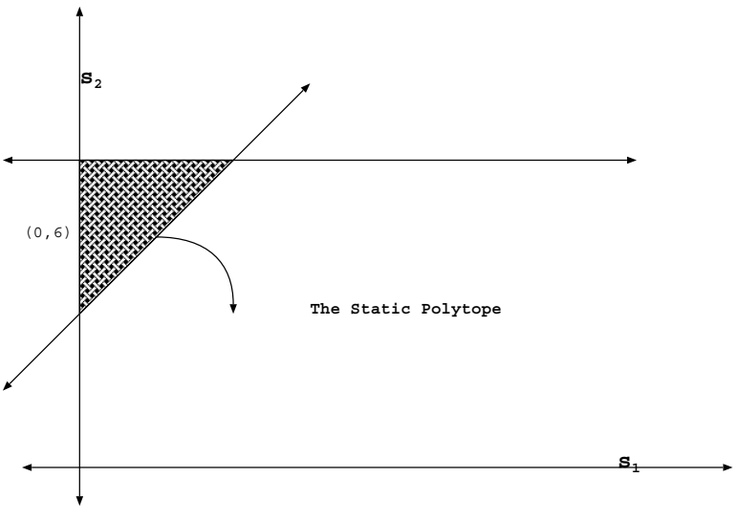

Fig. 2. The Static Polytope for Example (2)

We use a_i to denote the i^{th} row of $\mathbf{A}$, g_i to denote the i^{th} row of $\mathbf{G}$ and b_i to denote the i^{th} element of $\boldsymbol{b}$.

Proof: *Given a point* $\boldsymbol{p} = [p_1, p_2, \ldots, p_n]$ *of the final non-empty static polytope* $\mathbf{G}.\boldsymbol{s} \leq \boldsymbol{\rho}$*, let us assume the contrary and that indeed one or more of the input constraints (of the input constraint matrix) has been violated at a particular execution time vector* $\boldsymbol{e'} = [e'_1, e'_2, \ldots, e'_n] \in \mathbf{E}$*. Pick one such violated constraint, say* $\boldsymbol{a_i}.[\boldsymbol{s}, \boldsymbol{e}] \leq b_i$*. The violation of this constraint at* $(\boldsymbol{p}, \boldsymbol{e'})$ *implies that* $\boldsymbol{a_i}.[\boldsymbol{p}, \boldsymbol{e'}] > b_i$ *or after rewriting and separating the variables* $\boldsymbol{g_i}.\boldsymbol{p} > (\boldsymbol{b} - \mathbf{H}.\boldsymbol{e'})_i$*. But from the construction of the* static polytope *(See Algorithm (5.1)), we know that*

$$\rho_i = \min_{\mathbf{E}}(\boldsymbol{b} - \mathbf{H}.\boldsymbol{e})_i$$

and

$$\boldsymbol{g_i}.\boldsymbol{p} \leq \rho_i$$

which provides the desired contradiction.

Hence no constraint can be violated by choosing a point on the Static polytope. □

Lemma 2. *A point not on the Static Polytope cannot guarantee a static schedule.*

Proof: *Consider a point* $\boldsymbol{s'} = [s'_1, s'_2, \ldots, s'_n]$ *such that* $\boldsymbol{s'} \notin \{\boldsymbol{s} : \mathbf{G}.\boldsymbol{s} \leq \boldsymbol{\rho}\}$*. Let* $\boldsymbol{g_i}.\boldsymbol{s} \leq \rho_i$ *be one of the constraints that has been violated*[4]*, i.e.* $\boldsymbol{g_i}.\boldsymbol{s'} > \rho_i$*. Clearly, if the execution time vector* $\boldsymbol{e'} = [e'_1, e'_2, \ldots, e'_n] \in \mathbf{E}$ *is such that* $(\boldsymbol{b} - \mathbf{H}.\boldsymbol{e'})_i = \rho_i$*, then the point* $\boldsymbol{s'}$ *leads to violation of the constraint System (1).* □

Theorem 1. *There exists a static schedule if and only if the Static Polytope is non-empty.*

Proof: *Follows from Lemma (1) and Lemma (2).* □

5.3 Complexity

We observe that the elimination of each vector r_i and its replacement by a rational number ρ_i involves a call to a Convex minimization Algorithm. m such calls are required to eliminate all the r_i from the constraint system giving a computation time of $O(m.\mathcal{C})$, where $\mathcal{C}$ is the running time of the fastest Convex minimization algorithm ([HuL93]). One final call has to be made to a Linear Programming algorithm to verify the feasibility of the resultant static polytope. Thus, the total running time of the algorithm is $O(m.\mathcal{C} + \mathcal{L})$, where $\mathcal{L}$ is the running time of the fastest Linear Programming algorithm [Vai87]. We point out that we are using Linear Programming in the sense of determining feasibility of a linear system, as opposed to the standard optimization version. Since Linear Programming is a special case of convex minimization, we have $\mathcal{L} \leq \mathcal{C}$ and hence the complexity of our algorithm is $O(m.\mathcal{C})$.

[4] At least one such constraint exists; otherwise, $\boldsymbol{s'}$ belongs to the Static Polytope!

5.4 Parallelization

The ρ_i are created independently for each constraint; thus the steps involving the determination of the ρ_i can be carried out in parallel. This suggests the parallel[5] implementation of the **for** loop, described in Algorithm (5.2).

Function PARALLEL-STATIC-SCHEDULER($\mathbf{E}, \mathbf{A}, \boldsymbol{b}$)
1: Carry out the initialization steps as in Sequential Algorithm
2: **for** ($i = 1$ **to** m) **pardo**
3: Let $\rho_i = \min_{\mathbf{E}} r_i$
4: **end for**
5: Perform feasibility check as in Algorithm (5.1).

Algorithm 5.2: Parallel version of Static Scheduling Algorithm

Clearly the steps associated with creating the static polytope have a parallel running time of $O(\mathcal{C})$ with a total work of $O(m.\mathcal{C})$. An additional $O(\mathcal{L})$ sequential time is required for the final feasibility check.

Observation: 51 *The dispatch calendar obtained by Algorithm (5.1), can be used in every scheduling window with the appropriate offset.*

6 Special Case Analysis

We now analyze the complexity of a number of restrictions to the execution time domain $\mathbf{E}$ and the constraint matrix $\mathbf{A}$.

6.1 E Is an Axis-Parallel Hyper-Rectangle

In this case, the execution time domain can be represented by: $\boldsymbol{\Upsilon} = [l_1, u_1] \times [l_2, u_2] \times \ldots \times [l_n, u_n]$ (say). The static schedulability query (2) becomes:

$$\exists \boldsymbol{s} = [s_1, s_2 \ldots s_n] \forall \boldsymbol{e} = [e_1, e_2, \ldots e_n] \in \boldsymbol{\Upsilon} \qquad \mathbf{A}.[\boldsymbol{s}, \boldsymbol{e}] \leq \boldsymbol{b} \quad ? \tag{6}$$

We can apply the same algorithm as in Section §5, in which case we solve $(m + 1)$ linear programs to give a total running time of $O(m.\mathcal{L})$.

However, we can do much better, as we shall show shortly.

Lemma 3. *The minimum of an affine function on an axis-parallel hyper-rectangle (aph) is reached at a vertex of the* `aph`.

Proof: *From [Sch87], we know that the lemma is true over all polyhedral domains and axis-parallel hyper-rectangles are restricted polyhedral domains.* □

[5] We are using the PRAM Model, see [Ja'92]

Lemma 4. *When the domain is an* `aph`*, an affine function can be minimized by minimizing over each dimension individually.*

Proof: *See [Sch87].* □

Lemma (4) gives us the following strategy to minimize an affine function $f = a_1.e_1 + a_2.e_2 + \ldots + a_n.e_n + c$ over the `aph` $\Upsilon = [l_1, u_1] \times [l_2, u_2] \times \ldots \times [l_n, u_n]$:

- $\forall i$, if $a_i > 0$, set $e_i = l_i$
- $\forall i$, if $a_i < 0$, set $e_i = u_i$

A simple summation gives the minimum value of f over the `aph`.

Using this strategy it is clear that the STATIC-SCHEDULER algorithm runs in $O(m.n + \mathcal{L})$ sequential time.

6.2 At Most 2 Start-Time Variables per Constraints

We now consider the case, in which the relationships among the jobs can be expressed through *network constraints*, i.e. constraints in which there are at most 2 start-time variables and the execution time domain is an `aph`. Using the techniques from Section §6.1, we know that we can eliminate the execution time variables from the system in $O(m)$ time. The elimination results in a network linear system of constraints in the start time variables, i.e. in each constraint at most two variables have non-zero coefficients. [HN94] presents a fast implementation of the Fourier-Motzkin procedure to solve network linear systems in time $O(m.n^2.\log m)$. Thus, we have

Lemma 5. *Static Schedulability can be decided in time* $O(m.n^2.\log m)$*, if the execution time domain is an* `aph` *and the constraints are* network*.*

Proof: *From the above discussion.* □

Lemma 6. *If the constraints are strict relative timing constraints, statit schedulability can be decided in time* $O(m.n)$*.*

Proof: *When the constraint system consists of strict relative constraints only [GPS95,Cho00], we can represent it as a network graph (Single Source). Since the execution time domain is an* `aph`*, the execution time variables can be eliminated in* $O(m)$ *time. We can then use the Bellman-Ford algorithm, which takes* $O(m.n)$ *time, to check if the resultant network has a negative cost cycle. The existence of such a cycle coincides with the infeasibility of the input system [CLR92]. Likewise, non-existence of a negative cost cycle implies that the constraint system is feasible.*

□

7 Summary

In this paper, we analyzed the complexity of static schedulability specifications in the `E-T-C` scheduling model, which is described at depth in [Sub00]. The model

finds applicability in a number of domains. We demonstrated the existence of polynomial time algorithms for the general case and presented faster algorithms for a number of special cases. Table (1) summarizes our contributions in this paper.

Table 1. Summary of Results for Static Scheduling

	<arb|arb|stat>	<aph|stan|stat>	<aph|net|stat>
Schedulability	$O(m.\mathcal{C})$	$O(m.n)$	$O(m.n^2.\log m)$
Online Dispatching	$O(1)$		

The principal advantages of Static Scheduling are:

1. *No online computation* - As mentioned in Section §5, static scheduling obviates the need for an online computing during the dispatching phase. The start time vector computed by Algorithm (5.1) is used in every scheduling window.
2. *Efficient decidability* - We demonstrated that the static schedulability query can be decided in polynomial time, irrespective of the execution time domain (as long as it is a convex set) or the constraint matrix. This feature is particularly useful, when the real-time system is constrained through power-equations which can be approximated through convex sets.

An interesting open project is the integration of our work within the kernel of existing real-time operating systems and studying its performance in a more complete setting.

Acknowledgements. We wish to thank Michael Bond of the WVU libraries for his contributions towards the implementation.

A Implementation

We implemented our static scheduling algorithm on a Linux box, with Red Hat Linux. We used *lp-solve*[6] [Ber95] for all our algorithms. Table (2) details the machine characteristics, while table (3) tabulates our results[7]:

Most of the constraints were chosen from the boeing data-set, provided as part of [LTCA89], while some of them were randomly generated. The execution time domain **E** for most inputs was an axis-parallel hyper-rectangle, although we did use general polyhedra for some constraint sets.

[6] Version 2.1. This software is available free of cost at the URL in [Ber95].

[7] We have more detailed implementation statistics which are part of an extended version of this paper

Table 2. Machine Characteristics

Speed	500 Mhz
Processor	Pentium III
Memory	128 Mb RAM
Cache	L2
Operating System	Redhat Linux 6.0
Kernel	2.2.16
Language	Perl 5.005-03
Software	*lp-solve*

Table 3. Summary of Results for Static Scheduling

Number of jobs	Number of Constraints	Time (seconds)
5	10	0.27
10	20	0.42
15	30	0.54
20	40	1.14
25	50	1.82
30	60	2.74
35	70	4.49
40	80	6.45
45	90	7.45
50	100	10.91

A.1 Interpretation

Table (3) demonstrates that even for for fairly large job and constraint sets, static schedules can be computed quickly. Although we used an open-source software product viz. *lp-solve*, our performance did not degrade. We are confident that using a commercial product such as AMPL should decrease the computation time significantly.

References

[Ber95] M. Berkelaar. Linear programming solver. *Software Library for Operations Research, University of Karlsruhe*, 1995.

[BFW97] Azer Bestavros and Victor Fay-Wolfe, editors. *Real-Time Database and Information Systems, Research Advances*. Kluwer Academic Publishers, 1997.

[Cho00] Seonho Choi. Dynamic time-based scheduling for hard real-time systems. *Journal of Real-Time Systems*, 2000.

[CLR92] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press and McGraw-Hill Book Company, 6th edition, 1992.

[Das85] B. Dasarathy. Timing Constraints of Real-Time Systems: Constructs for Expressing Them, Methods of Validating Them. *IEEE Transactions on Software Engineering*, SE-11(1):80–86, January 1985.

[DMP91] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49:61–95, 1991.

[DRSK89] A. Damm, J. Reisinger, W. Schwabl, and H. Kopetz. The Real-Time Operating System of MARS. *ACM Special Interest Group on Operating Systems*, 23(3):141–157, July 1989.

[GPS95] R. Gerber, W. Pugh, and M. Saksena. Parametric Dispatching of Hard Real-Time Tasks. *IEEE Transactions on Computers*, 1995.

[HG93] S. Hong and R. Gerber. Compiling real-time programs into schedulable code. In *Proceedings of the ACM SIGPLAN '93 Conference on Programming Language Design and Implementation*. ACM Press, June 1993. *SIGPLAN Notices, 28(6):166-176.*

[HL89] C. C. Han and K. J. Lin. Job scheduling with temporal distance constraints. Technical Report UIUCDCS-R-89-1560, University of Illinois at Urbana-Champaign, Department of Computer Science, 1989.

[HL92a] C. C. Han and K. J. Lin. Scheduling Distance-Constrained Real-Time Tasks. In *Proceedings, IEEE Real-time Systems Symposium*, pages 300–308, Phoenix, Arizona, December 1992.

[HL92b] C. C. Han and K. J. Lin. Scheduling real-time computations with separation constraints. *Information Processing Letters*, 12:61–66, May 1992.

[HN94] Dorit S. Hochbaum and Joseph (Seffi) Naor. Simple and fast algorithms for linear and integer programs with two variables per inequality. *SIAM Journal on Computing*, 23(6):1179–1192, December 1994.

[HuL93] J. B. Hiriart-urruty and C. Lemarechal. *Convex Analysis and Minimization Algorithms*. Springer-Verlag, 1993.

[Ja'92] Joseph Ja'Ja'. An introduction to parallel algorithms (contents). *SIGACTN: SIGACT News (ACM Special Interest Group on Automata and Computability Theory)*, 23, 1992.

[JM86] F. Jahanian and A.K. Mok. Safety analysis of timing properties in real-time systems. *IEEE Transactions on Software Engineering*, SE-12(9):890–904, September 1986.

[LTCA89] S. T. Levi, S. K. Tripathi, S. D. Carson, and A. K. Agrawala. The `Maruti` Hard Real-Time Operating System. *ACM Special Interest Group on Operating Systems*, 23(3):90–106, July 1989.

[MAT90] D. Mosse, Ashok K. Agrawala, and Satish K. Tripathi. Maruti a hard real-time operating system. In *Second IEEE Workshop on Experimental Distributed Systems*, pages 29–34. IEEE, 1990.

[MKAT92] D. Mosse, Keng-Tai Ko, Ashok K. Agrawala, and Satish K. Tripathi. Maruti: An Environment for Hard Real-Time Applications. In Ashok K. Agrawala, Karen D. Gordon, and Phillip Hwang, editors, *Maruti OS*, pages 75–85. IOS Press, 1992.

[Pin95] M. Pinedo. *Scheduling: theory, algorithms, and systems*. Prentice-Hall, Englewood Cliffs, 1995.

[Sak94] Manas Saksena. *Parametric Scheduling in Hard Real-Time Systems*. PhD thesis, University of Maryland, College Park, June 1994.

[SB94] A. Stoyenko and T. P. Baker. Real-Time Schedulability Analyzable Mechanisms in Ada9X. *Proceeding of the IEEE*, 82(1):95–106, January 1994.

[Sch87] Alexander Schrijver. *Theory of Linear and Integer Programming*. John Wiley and Sons, New York, 1987.

[SdSA94] M. Saksena, J. da Silva, and A. Agrawala. Design and Implementation of Maruti-II. In Sang Son, editor, *Principles of Real-Time Systems*. Prentice Hall, 1994. Also available as CS-TR-2845, University of Maryland.

[Sub00] K. Subramani. *Duality in the Parametric Polytope and its Applications to a Scheduling Problem*. PhD thesis, University of Maryland, College Park, July 2000.

[Sub01] K. Subramani. Modeling clairvoyance and constraints in real-time scheduling. In *Proceedings of the* 6^{th} *European Conference on Planning*, September 2001.

[Vai87] P. M. Vaidya. An algorithm for linear programming which requires $O(((m+n)n^2 + (m+n)^{1.5}n)L)$ arithmetic operations. In Alfred Aho, editor, *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, pages 29–38, New York City, NY, May 1987. ACM Press.