A Distributed Dispatching Scheme for Parametric Schedulers

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

Abstract

The problem of determining the parametric schedulability of a set of jobs has received quite a bit of attention. Parametric dispatching is the functional counterpart of the schedulability problem i.e. the dispatching problem is concerned with the determination of the actual start times of tasks on the time line. Existing approaches to this problem have been along sequential lines, through the use of stored function lists. Sequential approaches through function lists suffer from three major drawbacks viz. extra space requirements, $O(n)$ dispatching time and the Loss of Dispatchability phenomenon. In this paper, we present a novel distributed scheme that achieves parametric dispatching in $O(1)$ time, under reasonable assumptions. The scheme obviates the necessity for storing function lists and simultaneously eliminates Loss of Dispatchability.

1 Introduction

Real-Time systems are confronted by two major issues, that are not addressed by traditional scheduling models, viz. Parameter variability and the presence of complex relationships constraining the execution of jobs. Parametric scheduling of jobs in the context of hard real-time systems is a well-studied problem [GPS95, Cho97, Cho00, SA00a, Sub00a]. This scheduling paradigm is applicable in situations, where static scheduling techniques result in an unacceptable *Loss of Schedulability* [GPS95].

In [GPS95] and [Cho00], dispatching schemes based on function evaluation were presented. These schemes could result in the phenomenon known as *Loss of Dispatchability* (§3). Our contributions in this paper are twofold:

(a) Providing a Distributed scheme for parametric dispatching; our scheme eliminates Loss of Dispatchability, under reasonable assumptions, while simultaneously extending existing schemes to distributed environments;

(b) Decreasing the connectivity requirements of the scheme in (a), at a small loss in efficiency.

Our scheme can be interpreted in the following two ways:

1. It is the first such scheme for Distributed Parametric systems,
2. It provides a genuine tradeoff between time and resources i.e. if meeting the imposed constraints is paramount and should be achieved even at the expense of increased resource requirements (in our case, processors), then our scheme permits this. This happens to be case in many practical real-time systems [SR88, SP92, SSRB98].

The rest of this paper is organized as follows: Section §2 states the Parametric Dispatching problem, while Section §3 describes the motivation for our work. Related approaches to this problem are detailed in Section §4. We propose our architecture in Section §5 and provide an algorithm for distributed dispatching in §6. A parallel implementation of our distributed dispatching scheme is provided in §8. We summarize our contributions in §9, providing pointers for future research.

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):

$$\mathbf{A}.[\vec{\mathbf{s}}, \vec{\mathbf{e}}] \leq \vec{\mathbf{b}}, \quad \vec{\mathbf{e}} \in \mathbf{E}, \tag{1}$$

where,

- $\mathbf{A}$ is an $m \times 2.n$ rational matrix,
- $\mathbf{E}$ is an arbitrary convex set;
- $\vec{\mathbf{s}} = [s_1, s_2, \ldots, s_n]$ is the start time vector of the jobs, and
- $\vec{\mathbf{e}} = [e_1, e_2, \ldots, e_n] \in \mathbf{E}$ is the execution time vector of the jobs

We assume that $\mathbf{A}$ *is a network unimodular matrix.* Consequently, the constraint system can be represented as a network graph. The dual graph representation of the constraint system is given in Appendix §A. *A similar construction for constraint networks is also given in [DMP91], however our construction is different in that the edges have non-constant weights. We also remark that the Parametric Scheduling problem is not related in any way to the problem of Parametric Shortest Paths discussed in [Car84, YTO91] or the parameterized mathematical programming techniques discussed in [Jen90, Wil67]. In both cases, the cost vector is parameterized by a single variable; in our case, the edge-weights could be of the form* $e_1 - e_2$*, i.e. a function of more than one variable.*

2.3 Query Model

Suppose that job J_a has to be dispatched. We assume that the dispatcher has access to the start times $\{s_1, s_2, \ldots, s_{a-1}\}$ and execution times $\{e_1, e_2, \ldots, e_{a-1}\}$ of the jobs $\{J_1, J_2, \ldots, J_{a-1}\}$.

Since the actual execution times of the previously executed jobs are required, the execution time domain must perforce be an axis-parallel hyper-rectangle (`aph`). For the rest of this paper, we assume that the domain $\mathbf{E}$ in System (1) is an `aph` represented by:

$$\boldsymbol{\Upsilon} = [l_1, u_1] \times [l_2, u_2] \times \ldots [l_n, u_n] \tag{2}$$

Definition: 2.1 *A parametric schedule of an ordered set of jobs, in a scheduling window, is a vector* $\vec{\mathbf{s}} = [s_1, s_2, \ldots, s_n]$*, where* s_1 *is a rational number and each* $s_i, i \neq 1$ *is a function of the start time and execution time variables of jobs sequenced prior to job* J_i*, i.e.* $\{s_1, e_1, s_2, e_2, \ldots, s_{i-1}, e_{i-1}\}$*. Further, this vector should satisfy the constraint system (1) for all execution time vectors* $\vec{\mathbf{e}} \in \boldsymbol{\Upsilon}$*.*

The combination of the Job-model, Constraint model and the Query model constitute a schedulability specification for a real-time scheduling problem within the `E-T-C` scheduling framework [Sub00a].

The discussion above directs us to the following formulation of the parametric schedulability query:

$$\exists s_1 \; \forall e_1 \in [l_1, u_1] \; \exists s_2 \; \forall e_2 \in [l_2.u_2], \ldots \exists s_n \; \forall e_n \in [l_n, u_n] \quad \mathbf{A}.[\vec{\mathbf{s}}, \vec{\mathbf{e}}] \leq \vec{\mathbf{b}} \quad ? \tag{3}$$

In this paper, we are concerned with the dispatching problem i.e. How to compute the online dispatch times of jobs assuming that query (3) has been decided in the affirmative. *Note that this problem is the functional counterpart of the schedulability query.* (Also see [Pap94]).

2.4 Representation of Input and Output

In the `E-T-C` scheduling framework, we assume that all numerical values are represented using Θ bits; Θ is called the *resolution level* of the framework. Thus the numerical values are represented by an integer model, in that they are ordered and every value (except the smallest and largest) has a definite successor and predecessor. For the rest of this paper, the term rational number is used to mean the integer representation of that number using at most Θ bits.

3 Motivation

The motivation for parametric scheduling techniques has been provided at depth in [Sub00a, GPS95].

Example (1): Consider the two job system $J = \{J_1, J_2\}$, with start times $\{s_1, s_2\}$, execution times in the set $\{(e_1 \in)[2,4] \times (e_2 \in)[4,5]\}$ and the following set of constraints:

- *Job J_1 must finish before job J_2 commences; i.e. $s_1 + e_1 \leq s_2$;*
- *Job J_2 must commence within 1 unit of J_1 finishing; i.e. $s_2 \leq s_1 + e_1 + 1$;*

The above constraint system has no static schedule i.e. there is no pair of rational numbers that can be assigned to $[s_1, s_2]$ to guarantee the inviolability of the constraint system. (To see this, observe that the *static polytope* generated by the constraint system is empty [SA00b].)

In [GPS95] it is shown that the schedulability query for an instance of `<aph|stan|param>` can be decided in $O(n^3)$ time. (In the `E-T-C` Scheduling framework, an instance of a scheduling problem is described by $< \alpha|\beta|\gamma >$ triplet, where

1. α is the execution time domain, in this case, an axis-parallel hyper-rectangle,
2. β represents the nature of constraints, in this case, network, unimodular, and
3. γ represents the nature of the schedulability query, in this case, parametric.

For details on the `E-T-C` scheduling framework, refer [Sub00a].)

[GPS95] and [Cho97] provide dispatching schemes for this problem that perform parametric function evaluation online. The problem with online evaluation is that it could take as much as $O(n)$ time in the worst-case. A dispatching scheme with a high evaluation overhead may cause constraint violation, even if the job set is parametrically schedulable.

Consider an instance of `<aph|stan|param>` which has been declared parametrically schedulable by the scheduling algorithm in [Sak94]. Let J_a ($a > 9$) be a job in the job-set, with execution time e_a ($\in [l_a, u_a]$). Assume that the parametric dispatch functions in Example (2) constrain the execution of J_a.

Example (2):

$$\max(23, s_1 + e_1 + 8, s_2 + 7, s_3 + e_3) \leq s_a \leq \min(41, s_1 + 32, s_7 + 7, s_9 - 3) \tag{4}$$

Suppose that at the time of dispatch of J_a, all the variables in System (4) are known and that $[23, 25]$ is the interval in which J_a can execute safely. This interval is called the *safety interval* for J_a. However, if job J_{a-1} completes at time $t = 22$ and the time taken to evaluate the functions in System (4) exceeds 3 units, then J_a cannot be dispatched. This phenomenon in which a job cannot be dispatched, although it is parametrically schedulable, is termed as *Loss of dispatchability.*

A second motivating factor is the existence of Distributed Applications [Lyn96, TSK92, Mul90]. If the jobs comprising the Job Model in §2.1 are part of a distributed application in that each of them executes on a different processor, then existing sequential dispatching schemes are no longer applicable. In fact, the theory underlying sequential schemes requires that all jobs be executed on the same processor. In Section §8, we discuss a parallel implementation of our distributed scheme that succeeds in achieving essentially the same performance as the distributed version.

A third factor is the fault-tolerance provided by distributed schemes. Fault-tolerance is an important aspect of real-time system design [Bur, KV90]. While analyzing fault-tolerance or failure models is not the basic thrust of this paper, we note that our distributed scheme *enhances dependability by increased resource redundancy.* Using the definitions provided in [KV90], we observe that the redundancy provided by our scheme is active in that computations can be interleaved even when all processors are operational (See Section §5).

4 Related Work

Parametric Scheduling was introduced in [Sak94] as a technique to alleviate the inflexibility of Static Scheduling in hard, real-time tasks. In [GPS95], a sequential parametric dispatching scheme was proposed for the determining the start times of tasks online. Their scheme results in polynomial time dispatching algorithms when the constraints are "standard" i.e. network, unimodular [SA00a]. Their scheme is strictly sequential and consists of storing function lists of dispatch functions. These dispatch functions are then evaluated online and the dispatch interval for the current task is determined. Thus, in the worst-case, the system may suffer from the Loss of Dispatchability phenomenon discussed in §3. [Cho97] and [Cho00] extend the scheme in [GPS95] to include tasks with inter-period constraints. Their algorithms are once again sequential and unrealistic assumptions are made regarding the time required to compute dispatch times.

In [Sub00c], we showed that the determining the schedulability of arbitrary constraint sets is in `PSPACE`, while in [Sub00b], it is shown that the dispatching problem is Turing reducible to the schedulability problem.

This paper represents the first attempt to approach the dispatching problem for standard constraints from a distributed system perspective. We establish that the dispatching problem can be solved in $O(1)$ time with $O(n)$ processors. In fact, all that we really need are adders and comparators and thus the processor cost is not really high.

5 Proposed Architecture

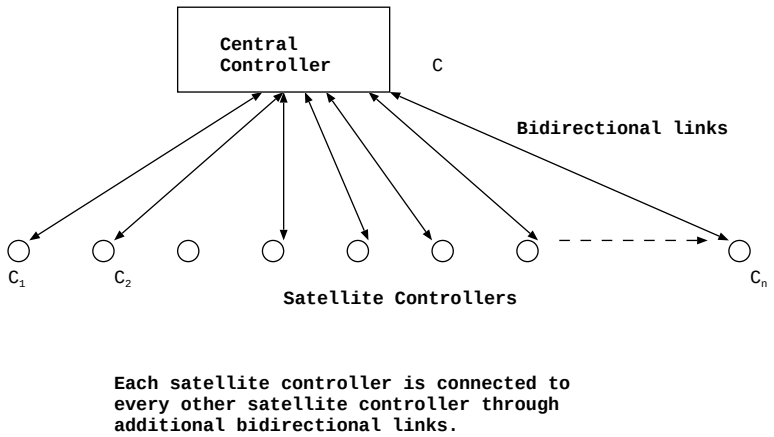

Figure 1: The Distributed Architecture for Parametric Scheduling

Figure (1) indicates the proposed architecture. There is a central controller C on which all the jobs in the job-set $\mathcal{J}$ in Section §2.1 execute. Bi-directional links are used to connect the central controller to a number of satellite controllers $\{C_1, C_2, \ldots, C_n\}$. The job of the satellite controller C_i is to compute the correct interval in which job J_i can execute. The $C_i, i = 1, 2, \ldots, n$ are themselves connected together as a complete graph, through the use of bidirectional links.

The graph representation of the constraint system is stored in each of the satellite controllers, so that each controller is aware of the exact dependencies on the other jobs.

Definition: 5.1 *Communication cost - The time taken to transmit Θ bits of information across a link in one direction.*

We assume a communication cost c_l between the links i.e. the time taken to transmit Θ bits of information in one direction is c_l. *Using the nomenclature in [Lyn96], our architecture is an asynchronous network model, without any shared memory.*

6 Distributed Dispatching Algorithm and Analysis

In the discussion below, the term processor is used to describe a satellite controller. We first consider the case in which the individual jobs are themselves executed on the central controller. We provide a distributed dispatching scheme, with one processor per job, that performs dispatching in $O(1)$ time. Assuming that the safety interval of every job is larger than the time required to make a constant number of comparisons, our scheme eliminates the Loss of dispatchability problem.

We assign one processor to each job; each processor is charged with the task of dispatching the job assigned to it. The goal is to determine the dispatch time of the current job (J_a), assuming that all jobs sequenced prior to it have been executed.

Let us study the execution of the Scheduling algorithm in [GPS95] on this problem instance at the juncture, when e_a has been eliminated. At this point, the constraints imposed on the execution of J_a by the jobs sequenced ahead of it have been resolved either into absolute constraints on s_a, or into redundant edges which are eliminated from the graph. The absolute constraints on s_a provide us with a tentative safety interval during which J_a can execute and this interval is stored in the processor responsible for the dispatching of J_a.

When a job sequenced before J_a, say J_b $(b < a)$, completes, its execution time e_b and start time s_b are immediately propagated to task J_a along the appropriate forward edges. This propagation is carried out as a broadcast from the central controller to all satellite controllers. The time spent in the broadcast is clearly $2.c_l$, since $2.\Theta$ bits are transmitted across the links to the satellite controllers. Consequently the relative constraints between J_b and J_a are transformed into absolute constraints on s_a. For instance, suppose that $s_b = 0$ and $e_b = 2$; accordingly the relative constraint $s_b + e_b + 4 \leq s_a$, becomes the absolute constraint $s_a \geq 6$.

Each resultant absolute constraint is then compared with the safety interval of J_a; the comparison results either in a narrowing of the interval (if the new constraint is non-redundant), or in the interval remaining unaltered (if the new constraint is redundant).

The above discussion is formalized as Algorithm (6.1).

Function ONLINE-DISPATCHER-FOR-J_a $(G =< V, E >)$
1: Let $[i_1, i_2], (i_1 < i_2)$ denote the current safety interval of J_a.
2: **for** ($j = 1$ **to** $a - 1$) **in parallel do**
3: When J_j finishes execution, use the value of e_j to relax the edge(s) $s_j \rightsquigarrow s_a$ into absolute constraints on s_a.
4: Compare each absolute constraint with the existing safety interval for J_a
5: **if** (new constraint is non-redundant) **then**
6: Update Safety Interval
7: **else**
8: Leave the Safety Interval unchanged
9: **end if**
10: **end for**

Algorithm 6.1: Online Dispatcher for `<aph|stan|param>`

Since there are at most 4 forward edges from any vertex $s_j (j < a)$, to s_a, Algorithm (6.1) takes at most $O(1)$ time, for each job sequenced before it. We now calculate the total time required to compute the dispatch interval of job J_a. Note that only the time taken after Job J_{a-1} has executed is relevant, since the other steps are carried out in parallel. Accordingly, relaxing 4 edges takes at most 4 additions and comparisons, i.e. $4(T_{add} + T_{comp})$, where T_{add} and T_{comp} are the times taken to perform an addition and a comparision respectively. In addition, there is

the cost of broadcasting (s_{a-1}, e_{a-1}) which is $2.c_l$. Thus the total time taken is $T_{tot} = 2.c_l + 4.(T_{add} + T_{comp})$. Assuming that the time taken to process the edges, i.e. T_{tot} is small, compared to the safety interval of a job, it is clear that Loss of Dispatchability is eliminated.

Definition: 6.1 *A dispatching scheme $\mathcal{D}$ is said to be optimal under the following conditions:*

1. *If an arbitrary scheme can dispatch the given set of jobs, then $\mathcal{D}$ can dispatch the same set of jobs,*
2. *If $\mathcal{D}$ cannot dispatch a set of jobs, then no dispatching scheme can dispatch the said set.*

Lemma: 6.1 *The dispatching scheme represented by Algorithm (6.1) is optimal to within a constant factor.*

Proof: *Clearly, we cannot hope to do better than $O(1)$ in dispatching. Our bounds are derived in terms of generic constants c_l, T_{add} and T_{comp}. The exact values of these constants are dependent upon the type of machine chosen to perform the dispatching. The Lemma follows.* □

Algorithm (6.1) and the accompanying analysis are valid when all jobs have to execute on the same processor (i.e. the central controller.) In this case, only the bidirectional links shown in Figure (1) are required; the complete graph links are not required.

Let us now consider the case, where the jobs are distributed i.e. they are required to execute on different processors. In this case the satellite controllers are no longer simple adder/comparator units, but full-fledged processors. However, the difference lies only in the communication of the start and execution times. Now, each satellite controller has to perform the broadcast of the (s_a, e_a) pair to all other satellite controllers. Once again the time taken is $T_{tot} = 2.c_l + 4.(T_{add} + T_{comp})$.

Observation: 6.1 *The requirements for the existence of links between each pair of satellite controllers can be obviated by the following technique: Instead of broadcasting the $(s_i, e_i$ to every other satellite controller, each controller broadcasts its data to the central controller, which in turn broadcasts the data to every other controller. Consequently, the update time for job J_a is now $2.c_l + 4.(T_{add} + T_{comp}) + 2.c_l$ i.e. there is a slight decrease in efficiency.*

7 Two axes of time

A useful way of looking at the distributed dispatching scheme is through Figure (2).

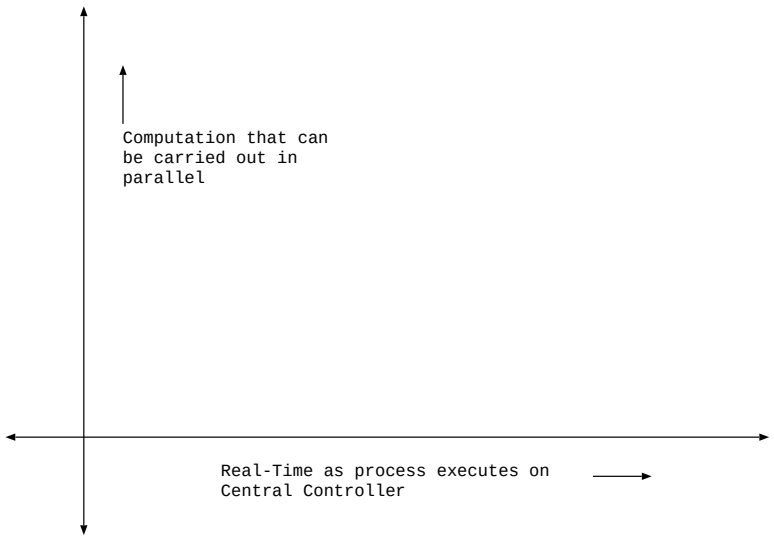

Figure 2: Interleaving Computations

Constraints on the job-set $\mathcal{J}$ are specified in real-time; when a decision algorithm (such as the one in [Sub00a] or [GPS95]) decides that a given constraint system is parametrically schedulable, it does so without taking into consideration the time involved in actually determining the dispatch interval for the individual jobs. Consequently, it is possible, depending upon the nature of constraints, for a system to be parametrically schedulable, but not have a valid dispatch schedule. Our distributed scheme exploits the fact that of all the jobs whose start time is parameterized by the execution time of the current job, only the job immediately following it, needs to perform the computation in real-time. All other jobs can perform the appropriate computations (i.e. updating safety intervals) in parallel, which is indicated by the $y-$axis in Figure (2).

8 A Parallel Implementation

In this section, we formalize the discussion in §7 into a shared-memory parallel implementation of the distributed scheme in Section §5. The parallel scheme uses the PRAM model, discussed in [Ja'92].

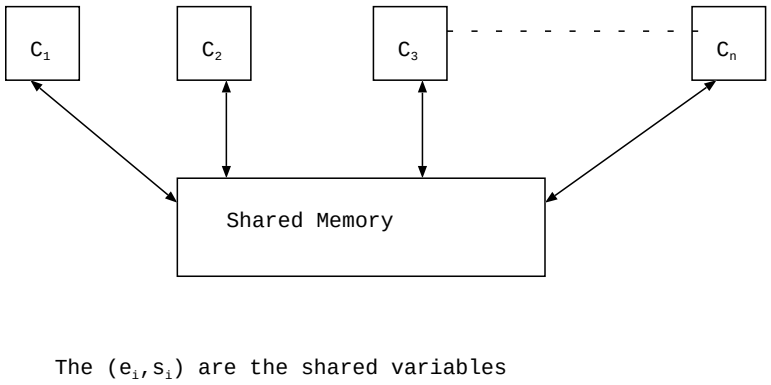

Figure 3: A Shared Memory Implementation

As in the distributed scheme, there are n processors, one for each job. However, the processors themselves communicate through the shared memory block as indicated in Figure (3). The variables (s_i, e_i) are stored in the Shared Memory. As processor C_i completes the execution of Job J_i, it writes (s_i, e_i) into the shared memory. The other processors then use this value to update their safety intervals as discussed in Algorithm (6.1). The model is thus a Concurrent Read Exclusive Write (CREW) model.

In the Parallel implementation, the communication costs become less significant as compared to the network distributed model.

9 Conclusions and Future Research

In this paper, we extended existing sequential dispatching schemes to provide a distributed dispatching scheme for parametric schedulers. Our scheme eliminates Loss of Dispatchability under the reasonable assumption that link communication costs are insignificant as compared to computation costs over $O(n)$ jobs. In the case, where all jobs are run on the same machine, i.e. sequentially, our scheme permits processors to be replaced by adders and comparators. If the constituent jobs are part of a distributed application, then techniques akin to the one we proposed are required, since sequential schemes are no longer applicable. Finally, distributed dispatching provides a degree of fault tolerance, which is crucial in real-time applications.

An open line of research is the actual implementation of our scheme across distributed systems to validate our point of view.

A Construction of Dual Graph for Standard Constraints

The purpose of this appendix is to provide a step-by-step procedure for constructing the dual graph, when the system constraints are standard.

Given a set of n jobs, with standard constraints imposed on their execution, we create a graph $G =< V, E >$, where V is the set of vertices and E is the set of edges.

1. $V =< s_0, s_1, s_2, \ldots, s_n >$, i.e. one node for the start times of each job, and node s_0 which is used for handling absolute constraints;
2. For every constraint of the form: $s_i + k \leq s_j$, construct and arc $s_i \rightsquigarrow s_j$, with weight $-k$;
3. For every constraint of the form: $s_i + e_i \leq s_j + k$, construct an arc $s_i \rightsquigarrow s_j$, with weight $k - e_i$;
4. For every constraint of the form: $s_i \leq s_j + e_j + k$, construct an arc $s_i \rightsquigarrow s_j$, with weight $e_j + k$;
5. For every constraint of the form: $s_i + e_i \leq s_j + e_j + k$, construct an arc $s_i \rightsquigarrow s_j$, with weight $e_j - e_i + k$;
6. For every constraint of the form: $s_i \leq c$, construct an arc $s_i \rightsquigarrow s_0$, with weight c;
7. For every constraint of the form: $s_i \geq c$, construct an arc $s_0 \rightsquigarrow s_i$, with weight $-c$;
8. For every constraint of the form: $s_i + e_i \leq c$, construct an arc $s_i \rightsquigarrow s_0$, with weight $c - e_i$;
9. For every constraint of the form: $s_i + e_i \geq c$, construct an arc $s_0 \rightsquigarrow s_i$, with weight $e_i - c$;
10. Finally construct arc $s_o \rightsquigarrow s_1$ with weight 0, since $s_1 \geq 0$ and arc $s_n \rightsquigarrow s_0$ with weight $L - e_n$, since all jobs have to be completed by the end of the current scheduling window.

Observation: A.1 *In the dual graph, there are $n + 1$ vertices and m edges, corresponding to a job-set with n jobs and m standard constraints on their execution.*

Observation: A.2 *There are at most 4 edges from node s_i and s_j; we classify them as:*

1. Type 1 - *An edge $s_i \rightsquigarrow s_j$ with weight k_1, representing temporal distance between the start times of J_i and J_j;*
2. Type 2 - *An edge $s_i \rightsquigarrow s_j$ with weight $-e_i + k_2$, representing temporal distance between the finish time of J_i and the start time of J_j;*
3. Type 3 - *An edge $s_i \rightsquigarrow s_j$ with weight $e_j + k_3$, representing temporal distance between the start time of J_i and the finish time of J_j;*
4. Type 4 - *An edge $s_i \rightsquigarrow s_j$ with weight $e_j - e_i + k_4$, representing temporal distance between the finish times of J_i and J_j.*

where, $k_1, k_2, k_3, k_4 \in \Re$.

Observation: A.3 *There are at most 2 edges between any job node s_i and the node s_0.*

Corollary: A.1 *In case of standard constraints, the dual graph has at most $O(n^2)$ edges.*

Proof: *Follows from the fact there are at most $(n + 1).n$ vertex pairs, with at most 4 edges between them.* □

Observation: A.4 *A forward edge, i.e. an edge $s_i \rightsquigarrow s_j, i < j$, dictates the degree of separation required between J_i and J_j , whereas a backward edge $s_j \rightsquigarrow s_i, j > i$ dictates the degree of closeness required between them.*

Example (3): We construct the dual graph for a $4-$job set $\{J_1, J_2, J_3, J_4\}$, subject to a set of standard constraints.

$$\begin{aligned} 4 \leq e_1 \leq 8,\ 6 \leq e_2 \leq 11,\ 10 \leq e_3 \leq 13,\ 3 \leq e_4 \leq 9 \\ s_4 + e_4 \leq 56 \\ s_4 + e_4 \leq s_3 + e_3 + 12 \\ s_2 + e_2 + 18 \leq s_4 \\ s_3 + e_3 \leq s_1 + e_1 + 31 \\ 0 \leq s_1,\ s_1 + e_1 \leq s_2,\ s_2 + e_2 \leq s_3, s_3 + e_3 \leq s_4 \end{aligned} \tag{5}$$

Figure (4) represents the corresponding dual graph.

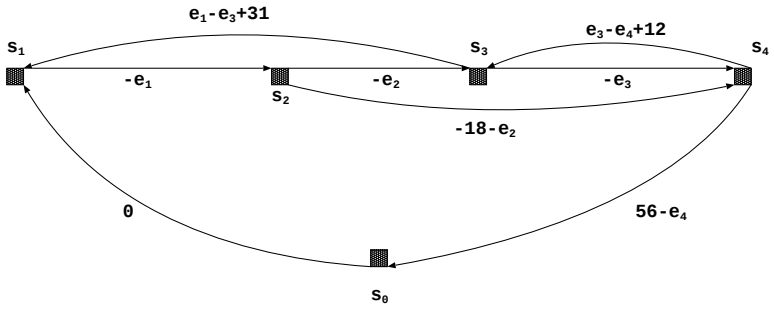

Figure 4: Dual graph of System (5)

References

[Bur] A. Burns. Distributed hard real-time systems: What restrictions are necessary?

[Car84] P. J. Carstensen. Parametric cost shortest path problems. Unpublished Bellcore memo, available from the Literaturstelle at the Inst. für Ökonometrie und Operations Research, U. Bonn, 8 May 1984.

[Cho97] Seonho Choi. *Dynamic Time-based scheduling for Hard Real-Time Systems.* PhD thesis, University of Maryland, College Park, jun 1997.

[Cho00] Seonho Choi. Dynamic dispatching of cyclic real-time tasks with relative time constraints. *JRTS*, pages 1–35, 2000.

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

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

[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.

[Jen90] L. Jenkins. Parametric methods in integer linear programming. *Annals of Operations Research*, 27:77–96, 1990.

[KV90] H. Kopetz and Paulo Verissimo. *Real-Time Dependability Concepts*, chapter 16, pages 410–445. In [Mul90], 1990.

[Lyn96] Nancy A. Lynch. *Distributed Algorithms.* Morgan Kaufmann series in data management systems. Morgan Kaufmann Publishers, Los Altos, CA 94022, USA, 1996. Prepared with LaTeX.

[Mul90] Sape J. Mullender. *Distributed Systems.* ACM Press, 1990.

[Pap94] Christos H. Papadimitriou. *Computational Complexity.* Addison-Wesley, New York, 1994.

[SA00a] K. Subramani and A. K. Agrawala. A dual interpretation of standard constraints in parametric scheduling. In *The Sixth International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, September 2000.

[SA00b] K. Subramani and A. K. Agrawala. The static polytope and its applications to a scheduling problem. 3^{rd} *IEEE Workshop on Factory Communications*, September 2000.

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

[SP92] Michael Schiebe and Saskia Pferrer, editors. *Real-Time Systems Engineering and Applications*, volume 1. Kluwer Academic Publishers, 1992.

[SR88] J. A. Stankovic and K. Ramamritham. *Hard Real-Time Systems.* IEEE Computer Society Press, Los Alamitos, 1988.

[SSRB98] John A. Stankovic, Marco Spuri, Krithi Ramamritham, and Giorgio C. Buttazzo, editors. *Deadline Scheduling for Real-Time Systems.* Kluwer Academic Publishers, 1998.

[Sub00a] K. Subramani. *Duality in the Parametric Polytope and its Applications to a Scheduling Problem.* PhD thesis, University of Maryland, College Park, July 2000.

[Sub00b] K. Subramani. Parametric dispatching is as easy as parametric schedulability. *Document in preparation*, 2000.

[Sub00c] K. Subramani. Parametric schedulability with arbitrary constraint sets is not harder than `pspace`. *Submitted to IPL*, September 2000.

[TSK92] S. Toueg, P. G. Spirakis, and L. Kirousis, editors. *Distributed Algorithms*, Berlin, 1992.

[Wil67] Leopold B. Willner. On parametric linear programming. *SIAM Journal on Applied Mathematics*, 15(5):1253–1257, September 1967.

[YTO91] N. E. Young, Robert E. Tarjan, and J. B. Orlin. Faster parametric shortest path and minimum-balance algorithms. *Networks*, 21:205–221, 1991.