Partitioned scheduling of sporadic task systems: an ILP-based approach

Sanjoy K. Baruah
The University of North Carolina
Chapel Hill, NC. USA

Enrico Bini
Scuola Superiore Santa Anna
Pisa, Italy.

Abstract— **The multiprocessor partitioned scheduling of sporadic task systems is considered. The problem of obtaining feasible partitionings under both Earliest Deadline First (EDF) and Fixed Priority (FP) scheduling is represented as integer linear programs comprised of binary (zero / one) integer variables only.**

I. Introduction

Real-time systems are often modeled as collections of recurrent tasks, each of which generates a potentially infinite sequence of jobs according to well-defined rules. One commonly-used formal model for representing such recurrent real-time tasks is the *sporadic* task model. Each sporadic task τ_i in this model is characterized by three parameters — a worst-case execution time C_i, a relative deadline D_i, and a period T_i. Such a task generates a potentially infinite sequence of jobs, with successive jobs of τ_i arriving at least T_i time units apart, each job having an execution requirement $\leq C_i$ and a deadline D_i time units after its arrival time.

In this paper, we consider real-time systems that can be modeled as collections of sporadic tasks and are implemented upon a platform comprised of several identical processors. We assume that the processors are fully *preemptive* — an executing job may be interrupted at any instant in time and have its execution resumed later with no cost or penalty.

Partitioned and global scheduling. In *partitioned* scheduling of collections of recurrent tasks, each task is assigned to a processor and all the jobs generated by the task are required to execute upon that processor. In *global* scheduling, on the other hand, different jobs of the same task may execute on different processors and a preempted job may resume execution on a processor different from the one it had been executing on prior to preemption.

Global scheduling is more general than partitioned scheduling (in the sense that every partitioned schedule is also a global schedule while the converse is not true); however, the current state of the art seems to favor partitioned scheduling. There are two major reasons for this. First, the run-time cost of inter-processor migration is unacceptably high on many platforms. Second, it has been shown [3], [4] that current schedulability tests for partitioned scheduling are superior to current schedulability tests for global scheduling. Although research and technology promises to ameliorate both these factors and render them less critical in the future, it is likely that partitioned scheduling will have a role to play in system design and implementation at present.

This research. We consider the partitioned scheduling of sporadic task systems upon multiprocessor platforms. We study two very widely-used scheduling algorithms — the *Earliest Deadline First* scheduling algorithm (EDF), [20], [11] and *Fixed Priority* scheduling (FP) [20], [2] when scheduling systems of sporadic tasks upon such preemptive platforms.

It is already known (see, e.g., [21]) that EDF- and FP-partitioning are intractable –NP-hard in the strong sense– even for task models simpler than the sporadic task model, since they are a generalization of the well-known bin-packing problem [17], [16]; hence, we do not expect to be able to obtain polynomial-time algorithms for partitioning sporadic task systems. Instead, we describe how the problem of partitioning sporadic task systems upon multiprocessors for EDF or FP scheduling may be formulated as an *integer linear programming* (ILP) problem. Although this may at first seem inconsequential — converting one intractable problem to another (since ILP is also known to be NP-hard [18]) — we believe that there is a real benefit to such conversion. This is because the optimization community has devoted immense effort to coming up with extremely efficient (although still exponential-time) algorithms for solving ILP's, and highly-optimized libraries implementing these efficient algorithms are widely available. (This is particularly true for ILP's in which each integer variable is further constrained to take in only the values zero or one – these are called *binary or zero / one integer variables*, and the corresponding ILP's *zero-one* or *binary* ILP's (BILP's); for instance, the optimization library that comes with MATLAB[1] has a very efficient solver for BILP's, called `bintprog()`). We will derive BILP representations of EDF and FP partitioning. Both BILP formulations use polynomially many integer variables. The BILP formulation of general EDF-partitioning requires exponentially many linear constraints; we will describe how a sufficient (rather than exact) EDF-partitioning algorithm can be formulated as a BILP with only pseudo-polynomially or polynomially many constraints. For FP-partitioning, the exact BILP formulation requires pseudo-polynomially many constraints, while sufficient FP-partitioning can be represented using polynomially many constraints.

[1] `http://www.mathworks.com/products/matlab/`

Organization. The remainder of this paper is organized as follows. In Section II, we briefly describe the task and machine model used in this work. In Section III, we discuss how partitioning problems may in general be represented as integer linear programs. In Section IV, we delve deeper into the ILP representation of EDF-partitioning. We describe how to represent EDF-partitioning exactly and approximately using zero-one ILP's, and discuss the tradeoffs that go into choosing the parameters for the approximation algorithms. In Section V, we describe how FP-partitioning can also be represented as an ILP. We conclude in Section VI by placing this work within the larger context of multiprocessor scheduling.

II. Model and definitions

As stated in Section I above, a *sporadic task* $\tau_i = (C_i, D_i, T_i)$ is characterized by a *worst-case execution requirement* C_i, a *(relative) deadline* D_i, and a *minimum inter-arrival separation* parameter T_i, also referred to as the *period* of the task. Such a sporadic task generates a potentially infinite sequence of jobs, with successive job-arrivals separated by at least T_i time units. Each job has a worst-case execution requirement equal to C_i and a deadline that occurs D_i time units after its arrival time. We assume a fully *preemptive* execution model: any executing job may be interrupted at any instant in time, and its execution resumed later with no cost or penalty. A *sporadic task system* is comprised of a finite number of such sporadic tasks. It is evident from the definition of sporadic tasks, and of sporadic task systems, that any given system may legally generate infinitely many different collections of jobs.

In the remainder of this paper, we will let τ denote a system of n sporadic tasks: $\tau = \{\tau_1, \tau_2, \ldots \tau_n\}$, with $\tau_i = (C_i, D_i, T_i)$ for all i, $1 \le i \le n$. Task system τ is said to be a *constrained* sporadic task system if it is guaranteed that each task $\tau_i \in \tau$ has its relative deadline parameter no larger than its period: $D_i \le T_i$ for all $\tau_i \in \tau$. *We restrict our attention here to constrained task systems*[2].

Demand bound function (DBF)**.** For any interval length t, the demand bound function $\text{DBF}(\tau_i, t)$ of a sporadic task τ_i bounds the maximum cumulative execution requirement by jobs of τ_i that both arrive in, and have deadlines within, any interval of length t. It has been shown [9] that

$$\text{DBF}(\tau_i, t) = \max\left(0, \left(\left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1\right) C_i\right) \tag{1}$$

Approximation schemes have been defined for computing the value of DBF to any desired degree of accuracy (see, e.g. [1], [12]). Equation 2 below gives such an approximation scheme for DBF; for any fixed value of k, $\text{DBF}^{(k)}(\tau_i, t)$ defines an approximation of $\text{DBF}(\tau_i, t)$ that is exact for the first k steps of $\text{DBF}(\tau_i, t)$, and an upper bound for larger values of t:

$$\text{DBF}^{(k)}(\tau_i, t) = \begin{cases} \text{DBF}(\tau_i, t) & \text{if } t \le (k-1)T_i + D_i \\ C_i + (t - D_i)U_i & \text{otherwise} \end{cases} \tag{2}$$

The following lemma provides a quantitative bound on the degree by which $\text{DBF}^{(k)}$ may deviate from DBF:

Lemma 1:

$\forall\ t \ge 0$

$$\text{DBF}(\tau_i, t) \le \text{DBF}^{(k)}(\tau_i, t) < \left(1 + \frac{1}{k}\right) \text{DBF}(\tau_i, t)\ .$$

Proof Sketch: This lemma is easily validated informally by sketching $\text{DBF}(\tau_i, t)$ and $\text{DBF}^{(k)}(\tau, t)$ as functions of t for given k (see Figure 1). $\text{DBF}(\tau_i, t)$ is a step function comprised of steps of height C_i, with the first step at $t = D_i$ and successive steps exactly T_i time units apart. The graph of $\text{DBF}^{(k)}(\tau, t)$ tracks the graph for $\text{DBF}(\tau_i, t)$ for the first k steps, and is a straight line with slope C_i/T_i after that. It is evident from the figure that $\text{DBF}^{(k)}(\tau_i, t)$ is always $\ge \text{DBF}(\tau_i, t)$, and that the ratio $\text{DBF}^{(k)}(\tau_i, t)/\text{DBF}(\tau_i, t)$ is maximized at t just a bit smaller than $kT_i + D_i$, where it is $< (k+1)C_i)/(kC_i) = (1 + \frac{1}{k})$ as claimed. ■

III. Partitioning and ILP's

In a integer linear program (ILP), one is given a set of N variables, some or all of which are restricted to take on integer values only, and a collection of "constraints" that are expressed as linear inequalities over these variables. The set of all points in N-dimensional space over which all the constraints hold is called the *feasible region* for the integer linear program[3].

In the following sections we consider a constrained-deadline task system τ comprised of n tasks that is to be partitioned among m unit-speed processors. We will let $\tau(j)$ denote the subset of τ that is assigned to the j'th processor by the partitioning algorithm, $1 \le j \le m$. Thus, $\tau(1)$, $\tau(2)$, $\ldots$, $\tau(m)$ represents a partitioning of τ into m mutually disjoint subsets and comprises the desired output of the partitioning algorithm.

To convert partitioning to a BILP we define $(n \times m)$ binary integer variables X_{ij}, with the following intended interpretation[4]

$$X_{ij} \leftarrow \begin{cases} 1 & \text{if } \tau_i \in \tau(j) \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

That is, $X_{ij} = 1$ if and only if the i'th task is assigned the j'th processor. This intended interpretation is enforced by n constraints of the following form, one for each $i \in \{1, \ldots, n\}$, encapsulating the requirement that each task must be assigned to some processor:

$$\sum_{j=1}^{m} X_{ij} = 1 \tag{4}$$

[2]This is primarily for pedantic reasons — although our techniques extend to sporadic task systems that are not constrained in much the same manner that uniprocessor EDF- and FP- scheduling results for constrained systems extend to task systems that are not constrained, the presentation of these extensions involves many grungy details and is likely to detract attention from the main ideas.

[3]One is also typically given an "objective function," also expressed as a linear inequality of the variables, and the goal is to find the extremal (maximum/ minimum) value of the objective function over the feasible region. However, for our purposes here it suffices to construct the constraints and determine whether the feasible region is empty or not.

[4]Recall that by definition, binary integer variables may take on values zero or one only.

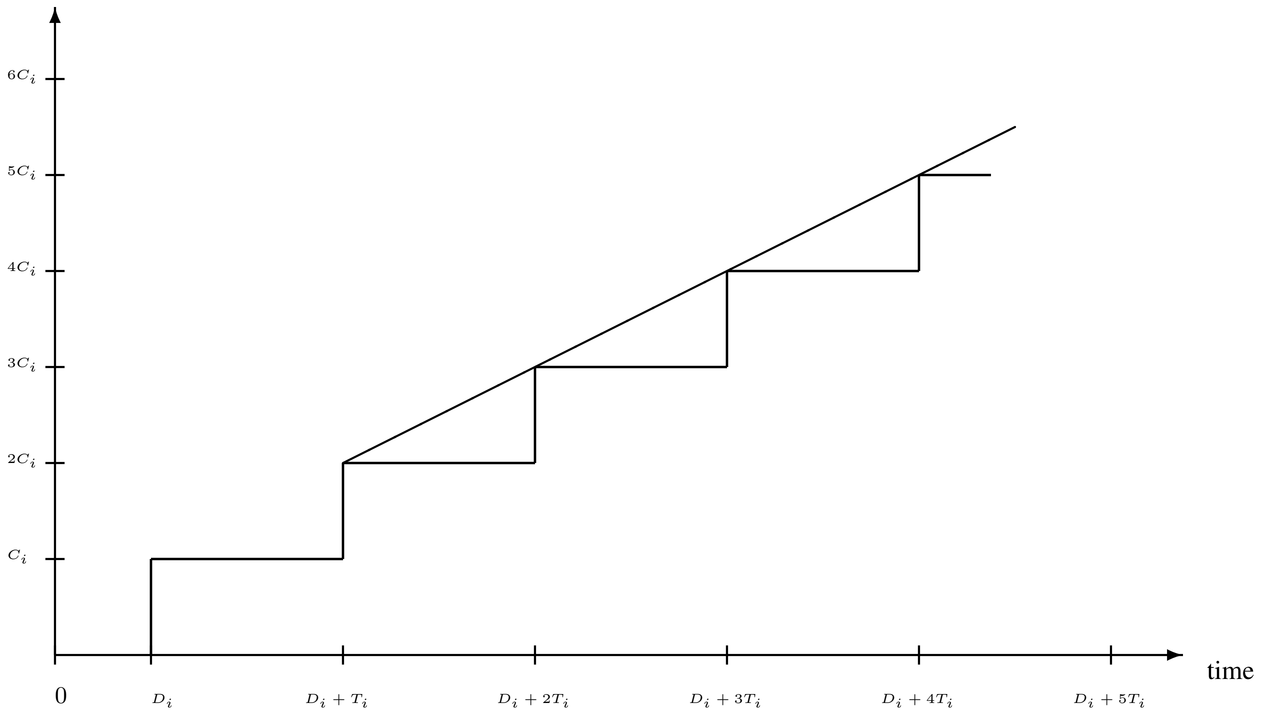

Fig. 1. Illustrating the proof of Lemma 1. The step plot represents $\text{DBF}(\tau_i, t)$. The plot for $\text{DBF}^{(k)}(\tau_i, t)$, for $k = 2$, is identical to the step plot for $t \leq d_i + T_i$, and is denoted by the straight line for larger t.

The constraints expressed in Equation 4 must be true for partitioning under any scheduling algorithm; in the following two sections, we describe the additional constraints that must be satisfied for the specific scheduling algorithms EDF (Section IV) and FP (Section V).

IV. Partitioned-EDF schedulability

In EDF scheduling [20], jobs are assigned priorities according to their absolute deadlines – the earlier the deadline of a job, the greater its priority. Under partitioned-EDF scheduling, all the tasks assigned to the j'th processor (i.e., all tasks in $\tau(j)$) must be uniprocessor EDF-schedulable, for each j, $1 \leq j \leq m$. Exact schedulability tests for uniprocessor EDF-scheduling are known [9]; these tests essentially require that the following linear constraint be satisfied

$$\sum_{\tau_i \in \tau(j)} \text{DBF}(\tau_i, t_o) \leq t_o \tag{5}$$

for each $t_o \in \mathcal{TS}(j)$, where $\mathcal{TS}(j)$ (sometimes called the "testing set") is a set of numbers defined as follows:

$$\mathcal{TS}(j) = \bigcup_{\tau_i \in \tau(j)} \big\{t | t \equiv D_i + (k-1)T_i \ \wedge \ t \leq P(j)\big\} \ . \tag{6}$$

Here, $P(j)$ denotes the least common multiple of the period parameters of all the tasks in $\tau(j)$: $P \overset{\text{def}}{=} \text{lcm}_{\tau_i \in \tau(j)}\{T_i\}$.

Observe that Equation 5 is indeed a linear constraint, since for given t_o, $\text{DBF}(\tau_i, t_o)$ evaluates to a constant. Hence Equation 5 is of the form $\sum_{i=1}^{n} A_i X_{ij} \leq t_o$ where each A_i is a constant, and the X_{ij}'s binary integer variables as stated above.

Note that $\tau(j)$ is not known when the ILP is being formulated — indeed, the goal of the ILP is to determine the $\tau(j)$'s. Let P denote the least common multiple of the period parameters of all the tasks in τ — $P \overset{\text{def}}{=} \text{lcm}_{i=1}^{n}\{T_i\}$ — and observe that $\mathcal{TS}(j) \subseteq \mathcal{TS}$, where

$$\mathcal{TS} \overset{\text{def}}{=} \bigcup_{i=1}^{n} \big\{t | t \equiv D_i + (k-1)T_i \ \wedge \ t \leq P\big\} \ . \tag{7}$$

Equation 5 is rewritten in a form that makes no reference to the unknown set $\tau(j)$ of tasks, as follows

$$\sum_{i=1}^{n} \text{DBF}(\tau_i, t_o)\, X_{ij} \leq t_o \tag{8}$$

Observe that Equations 5 and 8 are identical under the intended interpretation of the X_{ij} variables, since X_{ij} equals one for all $\tau_i \in \tau(j)$ and zero for all $\tau_i \notin \tau(j)$.

Since one such constraint must be written for each $t_o \in \mathcal{TS}$ for each processor j, there are $(m \times |\mathcal{TS}|)$ such linear constraints.

Reducing the number of constraints - I

In general, $|\mathcal{TS}|$ may be **exponential** in the representation of the task system; however, it has been shown in [9] that if the total utilization $U(\tau(j)) \overset{\text{def}}{=} \sum_{\tau_i \in \tau(j)} (C_i/T_i)$ of all the tasks assigned to the j'th processor is bounded from above by a constant c strictly less than one, then a $\mathcal{TS}$ can efficiently be identified with cardinality pseudo-polynomial in the representation of the task system. By choosing an appropriate value of c (the factors that influence this choice is discussed below) and then adding m linear constraints —one for each j, $1 \leq j \leq m$— of the form

$$\sum_{i=1}^{n} X_{ij} \frac{C_i}{T_i} \leq c \ , \tag{9}$$

we need verify Equation 8 at at most pseudo-polynomially many points t_o for each j, and hence need write at most **pseudo-polynomially** many constraints of the form given in Equation 8.

What determines the choice of c? As has been shown in [9], mandating such a restriction — that $U(\tau(j)) \leq c$ — is essentially equivalent to decreasing the computing capacity of the processors from 1 to the chosen constant c. Hence while any value of $c < 1$ will provide a pseudo-polynomial time bound, the tradeoff involved in choosing a value for c is as follows: the smaller the value of c, the smaller the value of $|\mathcal{TS}|$, but the larger the fraction of the computing capacity of the processors that remains unused by the partitioning algorithm and hence the more likely that our algorithm will fail to partition a task system that is partitionable.

Reducing the number of constraints - II

Equation 8 being true for all $j, 1 \leq j \leq m$, and all $t_o \in \mathcal{TS}$ corresponds to an exact partitioning algorithm. If we were to use $\text{DBF}^{(k)}$ instead of DBF in Equation 8:

$$t_o \geq \sum_{\tau_i \in \tau} \text{DBF}^{(k)}(\tau_i, t_o)\, X_{ij} \tag{10}$$

the partitioning algorithm is no longer exact: there may be partitionable task systems corresponding to which these constraints are no longer satisfiable (this is because, as Lemma 1 states, $\text{DBF}^{(k)}$ over-estimates the computational demand of each task). However, constructing a partitioning algorithm based on Equation 10 instead of on Equation 8 offers the advantage that the size of the testing set for which Equation 10 must be evaluated is known to be $|\tau(j)| \times k = \mathcal{O}(nk)$, which means that, for a given choice of k there are $\mathcal{O}(nm)$ such linear constraints. Furthermore, we may use Lemma 1 to quantify the degree of inaccuracy of the partitioning algorithm. In this manner, we can write an ILP representation of EDF-partitioning with the number of linear constraints **polynomial** in the representation of the task system.

To summarize the discussion above, we can transform EDF-partitioning to an binary integer linear program on the $n \times m$ binary integer variables X_{ij} with n linear constraints of the form Equation 3 (denoting that each task gets assigned to some processor), and some additional constraints generated according to one of the following three options:

1) Exponentially many linear constraints of the form Equation 8. The resulting ILP corresponds to an exact partitioning algorithm – the ILP has a feasible solution if and only if the task system is partitionable on m processors.
2) Choose a real-valued constant $c < 1$, and generate
   a) m constraints of the form Equation 9, and
   b) pseudo-polynomially many linear constraints of the form Equation 8. The resulting ILP corresponds to a sufficient partitioning algorithm – if the task system is partitionable on m processors by using no more than a fraction c of the computing capacity of each processor, then the ILP has a solution.
3) Choose an integer-valued constant of $k > 1$, and generate polynomially many linear constraints of the form Equation 10. The resulting ILP corresponds to a sufficient partitioning algorithm – if the task system is partitionable on m processors by using no more than a fraction $k/(k+1)$ of the computing capacity of each processor, then the ILP has a solution.

We can thus represent the partitioning problem exactly, or approximately to any desired degree of accuracy by generating the appropriate set of linear constraints. In this manner, we get a BILP that can be solved very efficiently (albeit still in worst-case exponential time) using highly optimized solvers (such as the `bintprog()` solver that is a part of the MATLAB optimization toolbox).

V. Partitioned-FP schedulability

In FP scheduling, each task is assigned a distinct priority and all the jobs generated by a task inherit the priority of the task that generates it. During run-time, each processor is allocated to the highest-priority job (if any) that needs to use the processor. *We will assume without loss of generality that the tasks are indexed in decreasing priority order* — task τ_1 is assigned the greatest priority, and τ_i has greater priority than τ_{i+1} for all i, $1 \leq i < n$.

Recall that we are restricting our attention in this paper to constrained-deadline sporadic task systems only. For such systems, it is known [19] that the response time of a job $\tau_i \in \tau(j)$ — i.e., assigned to the j'th processor — is maximized when the job arrives concurrently with a job of each greater-priority task in $\tau(j)$, and each such greater-priority task generates subsequent jobs as soon as allowed (i.e., with successive job arrivals separated by exactly the period parameter of the task). A sequence of job arrivals of a task systems in which each task has a job arrive at the same instant and subsequent jobs as soon as allowed is sometimes called the *synchronous arrival sequence (SAS)* of the task system.

Let us now focus on a particular task $\tau_i \in \tau(j)$ Let $\mathcal{AS}(i,j)$ denote all the time-instants in $[0, D_i)$ at which tasks that have been assigned to processor j have jobs arrive during the SAS: $\mathcal{AS}(i,j)$ contains D_i, as well as all $t < D_i$ of the form $t \equiv (\ell \times T_k)$, for $\ell = 0, 1, 2, \ldots$ and $\tau_k \in \tau(j)$. It is known [19] that τ_i meets its deadline if and only if there is a $t_o \in \mathcal{AS}(i,j)$ such that

$$t_0 \leq C_i + \sum_{\tau_k \ | \ k<i \bigwedge \tau_k \in \tau(j)} \lceil \frac{t_o}{T_k} \rceil C_k \ . \tag{11}$$

As in the EDF case (Section IV) we do not know $\tau(j)$ when we are setting up the ILP; hence, we must write m equations as follows, one for each j, $1 \leq j \leq m$, to represent the information conveyed by Equation 11:

$$t_0 \leq X_{ij} C_i + \sum_{k=1}^{i-1} \lceil \frac{t_o}{T_k} \rceil X_{kj} C_k \tag{12}$$

Since t_o, T_k, and C_k are all known, Equation 12 is a linear constraint in the X_{ij} variables. Also, it is known that the cardinality of $\mathcal{AS}(i,j)$ is pseudo-polynomial in the representation of the task system[5]. We thus see that τ_i meets its deadlines under FP-scheduling for a given priority assignment if and only if at least one of pseudo-polynomially many inequalities is satisfied. There is a standard technique in binary integer programming (see Section in the appendix) for expressing the requirement that at least one of a collection of inequalities be satisfied; by using this technique, we can thus express the FP-schedulability of τ_i as a binary integer program. By repeating this procedure for all the n tasks, we express FP-schedulability as a binary integer program on a polynomial number of variables and pseudo-polynomially many constraints.

Reducing the number of constraints

Since $\mathcal{AS}(i,j)$ may contain pseudo-polynomially many points, our BILP has pseudo-polynomially many constraints. Techniques are known [13], [14] for trading off some accuracy to reduce the number of points in $\mathcal{AS}(i,j)$ to a polynomial number; by applying these techniques, we can obtain a BILP representation of a sufficient FP-partitioning algorithm that has only polynomially many constraints.

VI. Context and Conclusions

As real-time embedded systems increasingly come to be implemented upon multiprocessor platforms, it is important that algorithms for efficiently scheduling such systems be obtained. In this work, we have considered the partitioned scheduling of sporadic task systems. Recent research [5], [6], [4], [15], [7], [8] has addressed this topic; however, all this research has been directed at obtaining approximate solutions to the partitioning problem. This work is instead directed at obtaining exact algorithms for partitioning, regardless of computational complexity. Unlike in the case of simpler task models (such as the *implicit-deadline* or "Liu and Layland" task model), in which the design of such an exact algorithm is a straightforward extension of bin-packing, obtaining exact representations of partitioning for constrained-deadline sporadic task systems turned out to not be quite as straightforward.

We have obtained a zero-one ILP representations for both EDF and FP partitioning. Such zero-one ILP's can be solved very efficiently in practice using widely-available libraries (such as MATLAB's `bintprog()` function). We have shown how the size (the number of constraints) of the ILP can be tuned depending on the degree of accuracy desired, i.e., the fraction of the computing capacity of the processing platform one is willing to "waste."

[5]Bini and Buttazzo [10] have devised a technique to identify at most 2^{i-1} points in $\mathcal{AS}(i,j)$ at which Condition 11 must be validated.

Acknowledgements

This research has been Supported in part by NSF Grant Nos. CNS-0834270, CNS-0834132, CCF-0541056, and CCR-0615197, ARO Grant No. W911NF-06-1-0425, and funding from IBM and the Intel Corporation.

References

[1] Albers, K., and Slomka, F. An event stream driven approximation for the analysis of real-time systems. In *Proceedings of the EuroMicro Conference on Real-Time Systems* (Catania, Sicily, July 2004), IEEE Computer Society Press, pp. 187–195.

[2] Audsley, N. C., Burns, A., Davis, R. I., Tindell, K. W., and Wellings, A. J. Fixed priority preemptive scheduling: An historical perspective. *Real-Time Systems 8* (1995), 173–198.

[3] Baker, T. P. Comparison of empirical success rates of global vs. partitioned fixed-priority and EDF scheduling for hard real time. Tech. Rep. TR-050601, Department of Computer Science, Florida State University, 2005.

[4] Baker, T. P. A comparison of global and partitioned EDF schedulability tests for multiprocessors. In *Proceeding of the International Conference on Real-Time and Network Systems* (Poitiers, France, 2006).

[5] Baruah, S., and Fisher, N. The partitioned multiprocessor scheduling of sporadic task systems. In *Proceedings of the IEEE Real-Time Systems Symposium* (Miami, Florida, December 2005), IEEE Computer Society Press.

[6] Baruah, S., and Fisher, N. The partitioned scheduling of sporadic real-time tasks on multiprocessor platforms. In *Proceedings of the Workshop on Compile/Runtime Techniques for Parallel Computing* (Oslo, Norway, June 2005).

[7] Baruah, S., and Fisher, N. The partitioned multiprocessor scheduling of deadline-constrained sporadic task systems. *IEEE Transactions on Computers 55*, 7 (July 2006), 918–923.

[8] Baruah, S., and Fisher, N. The partitioned dynamic-priority scheduling of sporadic task systems. *Real-Time Systems: The International Journal of Time-Critical Computing 36*, 3 (2007), 199–226.

[9] Baruah, S., Mok, A., and Rosier, L. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Proceedings of the 11th Real-Time Systems Symposium* (Orlando, Florida, 1990), IEEE Computer Society Press, pp. 182–190.

[10] Bini, E., and Buttazzo, G. Schedulability analysis of periodic fixed priority systems. *IEEE Transactions on Computers 53*, 11 (2004), 1462–1473.

[11] Dertouzos, M. Control robotics : the procedural control of physical processors. In *Proceedings of the IFIP Congress* (1974), pp. 807–813.

[12] Fisher, N., Baker, T., and Baruah, S. Algorithms for determining the demand-based load of a sporadic task system. In *Proceedings of the International Conference on Real-time Computing Systems and Applications* (Sydney, Australia, August 2006), IEEE Computer Society Press.

[13] Fisher, N., and Baruah, S. A fully polynomial-time approximation scheme for feasibility analysis in static-priority systems. In *Proceedings of the EuroMicro Conference on Real-Time Systems* (Palma de Mallorca, Balearic Islands, Spain, July 2005), IEEE Computer Society Press, pp. 117–126.

[14] Fisher, N., and Baruah, S. An approximation scheme for feasibility analysis in static-priority systems with bounded relative deadlines. *Journal of Embedded Computing* (2007). Accepted for publication.

[15] Fisher, N., Baruah, S., and Baker, T. The partitioned scheduling of sporadic tasks according to static priorities. In *Proceedings of the EuroMicro Conference on Real-Time Systems* (Dresden, Germany, July 2006), IEEE Computer Society Press.

[16] Johnson, D. Fast algorithms for bin packing. *Journal of Computer and Systems Science 8*, 3 (1974), 272–314.

[17] Johnson, D. S. *Near-optimal Bin Packing Algorithms*. PhD thesis, Department of Mathematics, Massachusetts Institute of Technology, 1973.

[18] Karp, R. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, R. Miller and J. Thatcher, Eds. Plenum Press, New York, 1972, pp. 85–103.

[19] Lehoczky, J., Sha, L., and Ding, Y. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *Proceedings of the Real-Time Systems Symposium - 1989* (Santa Monica, California, USA, Dec. 1989), IEEE Computer Society Press, pp. 166–171.

[20] Liu, C., and Layland, J. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM 20*, 1 (1973), 46–61.

[21] Sarkar, V. *Partitioning and scheduling parallel programs for execution on multiprocessors*. MIT Press, 1989.

Appendix

Expressing disjuncts as binary integer programs

In this section, we illustrate by an example the technique for expressing the requirement that at least one of a given set of inequalities be true, within the syntactic limitations of a binary integer program.

Suppose that we require that at least one of the four inequalities

$$\begin{aligned} A_1X &\leq b_1 \\ A_2X &\leq b_2 \\ A_3X &\leq b_3 \\ A_4X &\leq b_4 \end{aligned}$$

be satisfied. Let Z denote a very large positive constant. Introduce the two binary integer variables Y_1 and Y_2 (in general, the number of such variables that must be introduced is logarithmic in the number of inequalities). Replace the above 4 inequalities by the following:

$$\begin{aligned} A_1X &\leq b_1 + ((1-Y_1)+(1-Y_2))Z \\ A_2X &\leq b_2 + (Y_1+(1-Y_2))Z \\ A_3X &\leq b_3 + ((1-Y_1)+Y_2)Z \\ A_4X &\leq b_4 + (Y_1+Y_2)Z \end{aligned}$$

For any assignment of values to the newly-introduced binary variables Y_1 and Y_2, it is evident that all but one of these equalities is trivially satisfied (since the RHS is $\geq Z$), while the remaining inequality is exactly equivalent to the corresponding inequality in the original problem (i.e., prior to this transformation). Hence, all four of the latter inequalities is satisfiable if and only if at last one of the original four inequalities is satisfiable; we have thus converted the problem of satisfying at least one of the given set of four inequalities to a "conventional" problem of satisfying a set of all four inequalities.