Real-time scheduling of sporadic task systems when the number of distinct task types is small*

Sanjoy Baruah Nathan Fisher

Abstract

In some real-time application systems, there are only a few distinct kinds of tasks, each of which may be instantiated several times during runtime. The scheduling of such sporadic task systems is considered here upon both a single processor, and on multiprocessor platforms under the partitioned paradigm of multiprocessor scheduling. Algorithms that have run-time polynomial in the number of tasks in the system are presented and proved correct.

1 Introduction

Over the years, the sporadic task model [15, 3] has proven remarkably useful for the modelling of recurring processes that occur in hard-real-time systems. In this model, a *sporadic task* $\tau_i = (e_i, d_i, p_i)$ is characterized by a *worst-case execution requirement* e_i, a *(relative) deadline* d_i, and a *minimum inter-arrival separation* p_i, which is, for historical reasons, 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 p_i time units. Each job has a worst-case execution requirement equal to e_i and a deadline that occurs d_i time units after its arrival time. A *sporadic task system* is comprised of several such sporadic tasks.

A system of sporadic tasks is said to be *feasible* upon a specified platform under a specified set of constraints if it is possible to schedule the system such that all jobs' deadlines are met, for all permissible (also called *legal*) combinations of job-arrival sequences by the different tasks comprising the system. The feasibility-analysis of systems of sporadic tasks on preemptive *uni*processors has been extensively studied. On preemptive multiprocessors, however, most prior research has assumed that *all tasks have their deadlines equal to their period parameters* (i.e., $d_i = p_i$ for all tasks τ_i) — such sporadic systems are sometimes referred to in the literature as **implicit-deadline** systems.

§ **This research.** The research reported in this document is part of a larger project that aims to better understand the preemptive multiprocessor scheduling of arbitrary sporadic real-time systems. In this paper, we study the partitioned multiprocessor scheduling of arbitrary sporadic systems under the assumption that there are only a few distinct *kinds* of tasks, with each kind having multiple *instantiations* – we will refer to such task systems as *restricted* sporadic task systems. Our results concerning the analysis of restricted sporadic task systems may be summarized as follows:

- We show that straightforward implementations of the standard techniques for uniprocessor feasibility analysis — *response-time analysis* [7, 1] and the *processor demand approach* [3] — run in time pseudo-polynomial in the representation of the restricted sporadic task system.
- Using standard results from the theory of integer linear programming (ILP), we design new uniprocessor feasibility analysis algorithms for such restricted sporadic task systems that run in time polynomial in the number of tasks in the system
- We apply standard dynamic programming techniques to extend these polynomial-time uniprocessor feasibility analysis algorithms to multiprocessor platforms, thereby obtaining polynomial-time *multi*processor feasibility-analysis algorithms for restricted sporadic task systems.

§ **Organization.** The remainder of this paper is organized as follows. In Section 2, we formally specify the task model used in the remainder of this paper. In Section 3, we briefly summarize those previous results on uni- and multi-processor feasibility analysis that we will be using in the remainder of the document. We present polynomial-time algorithms for feasibility-analysis of sporadic task systems comprised of a constant number of distinct kinds of tasks implemented on uniprocessors in Section 4, and on multiprocessors in Section 5.

*This research has been supported in part by the National Science Foundation (Grant Nos. ITR-0082866, CCR-0204312, and CCR-0309825).

2 Task and machine model

In the remainder of this paper, we will assume that we have a multiprocessor platform comprised of m identical processors π_1, π_2, ..., π_m, each of unit computing capacity.

A *sporadic task* $\tau_i = (e_i, d_i, p_i)$ is characterized by a *worst-case execution requirement* e_i, a *(relative) deadline* d_i, and a *minimum inter-arrival separation* p_i. The ratio $u_i \stackrel{\text{def}}{=} e_i/p_i$ of sporadic task τ_i is often referred to as the *utilization* of τ_i; the utilization of a sporadic task system is defined to be the sum of the utilizations of all tasks comprising the system.

We assume that there are k distinct **types** of tasks for some *constant* k; each type-ℓ task has worst-case execution requirement e_ℓ, relative deadline d_ℓ, and inter-arrival separation parameter p_ℓ, for $1 \leq \ell \leq k$. These k execution requirements, relative deadlines, and inter-arrival separation parameters will be denoted by the vectors $\vec{e}_k$, $\vec{d}_k$, and $\vec{p}_k$ respectively.

Definition 1 (Restricted sporadic task system) *A* **restricted sporadic task system** *is specified by a 4-tuple of equi-sized vectors $(\vec{e}_k, \vec{d}_k, \vec{p}_k, \vec{n}_k)$ where $\vec{e}_k$, $\vec{d}_k$, and $\vec{p}_k$ denote the respective parameters of the k types of tasks, and vector $\vec{n}_k = (n_1, n_2, \ldots, n_k)$ of non-negative integers denotes that there are n_ℓ type-ℓ tasks; $1 \leq \ell \leq k$.*

§ **Static and dynamic priorities.** *Run-time scheduling* is the process of determining, during the execution of a real-time application system, which job should be executed on the shared processor at each instant in time. Run-time scheduling algorithms are typically implemented as follows: at each time instant, assign a **priority** to each active[1] job, and allocate the processor to the highest-priority job.

With respect to certain run-time scheduling algorithms, it is possible that some tasks τ_i and τ_j both have active jobs at times t_1 and t_2 such that at time t_1, τ_i's job has higher priority than τ_j's while at time t_2, τ_j's job has higher priority than τ_i's. Scheduling algorithms that permit such "switching" of priorities between tasks are known as *dynamic* priority schedulers. An example of a dynamic-priority run-time scheduling algorithm is the earliest deadline first scheduling algorithm (EDF) [12, 4], which at each instant in time executes the currently active job with the earliest deadline. It has been shown [4] that EDF is an *optimal* uniprocessor dynamic-priority run-time scheduling algorithm, in the sense that if it is possible to schedule a set of jobs such that they all complete by their deadlines, then EDF is guaranteed to do so.

By contrast, *static* priority schedulers satisfy the property that for every pair of tasks τ_i and τ_j, whenever τ_i and τ_j both have active jobs, it is always the case that the same task's jobs have priority. An example of a static-priority scheduling algorithm is the *deadline-monotonic scheduling algorithm* [11], which assigns each task a priority inversely proportional to the smaller of its period and deadline parameter, with ties broken arbitrarily but in a consistent manner.

A system of sporadic tasks is said to be *feasible* upon a specified platform if it is possible to schedule the system within the constraints imposed by the platform such that all jobs of all tasks will meet all deadlines, under all permissible (also called *legal*) combinations of job-arrival sequences by the different tasks comprising the system.

§ **Multiprocessor systems.** On multiprocessor systems, two alternative paradigms for scheduling collections of sporadic tasks have been considered: *partitioned* and *global* scheduling. In the partitioned approach, the tasks are statically partitioned among the processors, i.e., each task is assigned to a processor and is always executed on it. Under global scheduling, it is permitted that a job that has previously been preempted from one processor resume execution at a later point in time upon a different processor, at no additional cost (however each job may be executing on at most one processor at each instant in time).

Definition 2 (m-feasibility) *Restricted sporadic task system $(\vec{e}_k, \vec{d}_k, \vec{p}_k, \vec{n}_k)$ is said to be* m**-feasible** *with respect to a parcticular paradigm (i.e, static-priority or dynamic priority) if the tasks in $\vec{n}_k$ can be partitioned among m processors such that the tasks assigned to each processor are feasible with respect to the respective scheduling paradigm.*

3 Previous research

The feasibility-analysis of systems of sporadic tasks on preemptive *uni*processors has been extensively studied. It is known (see, e.g. [12, 3]) that a uniprocessor system of preemptive sporadic tasks is feasible if and only if all deadlines can be met when each task in the system has a job arrive at the same time-instant, and subsequent jobs arrive as rapidly as legal (such a combination of job-arrival sequences is sometimes referred to as a **synchronous arrival sequence** for the sporadic task system.) This fact,

[1] Informally, a job becomes *active* at its ready time, and remains so until it has executed for an amount of time equal to its execution requirement, or until its deadline has elapsed.

in conjunction with the optimality of the Deadline-Monotonic [11] and EDF [12, 4] scheduling algorithms for scheduling preemptive uniprocessor systems under static- and dynamic-priority constraints, has allowed for the design of exact preemptive uniprocessor feasibility-analysis algorithms for sporadic task systems [11, 8, 9, 15, 3]. For static-priority scheduling, feasibility-analysis algorithms are based upon the technique of *response-time analysis*[7, 1]; for dynamic priority scheduling, they are based upon the *processor demand approach* [3]. Both these techniques implicitly "simulate" the scheduling of the synchronous arrival sequence of jobs by an optimal scheduling algorithm (the deadline monotonic algorithm for static-priority scheduling, and EDF for dynamic-priority scheduling) until the end of the largest deadline for static-priority scheduling, and the least common multiple of the periods for EDF. Consequently, the run-time complexities are pseudo-polynomial and exponential-time respectively, for static-priority and dynamic-priority feasibility analysis. (For task systems in which the utilization is bounded from above by a constant strictly less than the capacity of the processor, a pseudo-polynomial feasibility analysis algorithm for EDF is presented in [3].)

Most prior research on *multiprocessor* scheduling of collections of sporadic tasks has focused on implicit-deadline systems (i.e., $d_i = p_i$ for all tasks τ_i) For such systems, feasibility-analysis is trivial under the global paradigm: [6, 17]: an implicit-deadline system τ is feasible upon a platform comprised of m unit-capacity processors if and only if **(i)** $u_i \leq 1$ for each task $\tau_i \in \tau$; and **(ii)** $\sum_{\tau_i \in \tau} u_i \leq m$. If tasks are constrained to execute on single processors (i.e., under the *partitioned* paradigm), feasibility-analysis for implicit-deadline systems is known to be NP-hard in the strong sense; sufficient feasibility tests for various bin-packing heuristics have recently been obtained [16, 14, 13].

Since the arbitrary system model is a generalization of the implicit-deadline model, the intractability result (feasibility analysis being NP-hard in the strong sense) continues to hold; to our knowledge, there are no prior non-trivial positive theoretical results concerning this problem. ("Trivial" results include the obvious ones that τ is feasible on m processors if *(i)* it is feasible on a single processor; or *(ii)* the system obtained by replacing each task τ_i by a task $\tau_i' = (e_i, \min(d_i, p_i), \min(d_i, p_i))$ is deemed feasible using the heuristics presented in [14, 13].)

Regarding *restricted* sporadic task systems, the only prior result we are aware of concerns the preemptive dynamic-priority scheduling of such systems upon uniprocessor platforms [2]. We will briefly present this result in Section 4 below.

4 Uniprocessor systems

We invite the reader to verify that the standard techniques for static-priority scheduling (response-time analysis) and for dynamic-priority scheduling (the processor demand approach) both run in pseudo-polynomial time even for task systems in which the number of distinct task types is a priori bounded by a constant. In brief, this is because both techniques essentially require a simulation of the scheduling algorithm for an interval of time that depends upon the parameters of the tasks comprising the task system. In response-time analysis of task systems in which each task's deadline is $\leq$ to the task's period, this interval is equal to the largest deadline of any task in the system; for the processor demand approach, this interval may be as large as the least common multiple (lcm) of the periods of the tasks[2].

However, we demonstrate below that feasibility analysis for such restricted sporadic task systems can in fact be converted into integer linear programming (ILP) problems in a constant number of variables. It has previously been shown [10] that integer linear programs in a constant number of variables can be solved in polynomial time; hence, we may conclude that feasibility analysis for restricted sporadic task systems can be solved in time polynomial in the representation of the task system.

In the remainder of this section, we will consider a restricted sporadic task system $(\vec{e}_k, \vec{d}_k, \vec{p}_k, \vec{n}_k)$. We describe below how the question: "Is $(\vec{e}_k, \vec{d}_k, \vec{p}_k, \vec{n}_k)$ feasible on a uniprocessor?" (equivalently, "Is $(\vec{e}_k, \vec{d}_k, \vec{p}_k, \vec{n}_k)$ 1-feasible?") under the two different paradigms of static-priority and EDF scheduling may be converted to polynomially many ILP problems each on a constant number of variables. We can then use the polynomial-time algorithm of Lenstra [10] to solve these integer linear programs, thereby obtaining a feasibility test for the restricted sporadic task system $(\vec{e}_k, \vec{d}_k, \vec{p}_k, \vec{n}_k)$ on a single processor that runs in time polynomial in the representation of the system.

§ **EDF scheduling.** It has previously been shown [2] that the problem of determining whether $(\vec{e}_k, \vec{d}_k, \vec{p}_k, \vec{n}_k)$ is feasible upon a uniprocessor under EDF-scheduling is equivalent to solving the following integer programming problem: Determine whether there are integers $x_1, x_2, \ldots, x_k$, and a positive real number t_f, such that the following $(k+1)$ linear inequalities

[2]Observe that this lcm is no more than $(p_{\max})^k$, where $p_{\max}$ denotes the largest period and k the number of distinct task types, and is hence pseudo-polynomial rather than exponential in the input, for constant k.

are satisfied:

$$\begin{aligned} d_i + (x_i - 1) \cdot p_i &\leq t_f, \text{ for } 1 = 1, 2, \ldots, k \\ \sum_{i=1}^{k} n_i \cdot e_i \cdot x_i &> t_f \end{aligned}$$

If so, then the system is infeasible; else, it is guaranteed to be feasible (see [2, Theorem 3.5] for a proof). Therefore, we can represent the uniprocessor EDF-feasibility of $(\vec{e}_k, \vec{d}_k, \vec{p}_k, \vec{n}_k)$ $(k+1)$ linear inequalities in k integer variables and one additional variable.

§ **Static-priority scheduling.** In the following, we will make the assumption that $d_\ell \leq p_\ell$ for all ℓ, $1 \leq \ell \leq k$. Under this assumption, it has been shown [11] that a sporadic task system is static-priority feasible if and only if every tasks's first job in the task system's synchronous arrival sequence meets its deadline when priorities are assigned according to the deadline-monotonic priority assignment scheme.

Without loss of generality, let us assume that task types are indexed according to non-decreasing value of the deadline parameter: $d_\ell \leq d_{\ell+1}$ for all ℓ, $1 \leq \ell < k$. (Thus, the deadline-monotonic priority assignment scheme assigns all tasks of the ℓ'th type greater priority than all jobs of the $(\ell+1)$'th type, for all ℓ.) Our procedure is iterative: Let ℓ take on the values $1, 2, \ldots, k$ in order. During the ℓ'th iteration, we suppose that it has been determined that the restricted sporadic task system $(\vec{e}_{\ell-1}, \vec{d}_{\ell-1}, \vec{p}_{\ell-1}, \vec{n}_{\ell-1})$, comprised of all tasks of the $(\ell-1)$ highest-priority types, is static-priority feasible on a uniprocessor. (Observe that this is trivially true for $\ell = 1$, since $(\vec{e}_{\ell-1}, \vec{d}_{\ell-1}, \vec{p}_{\ell-1}, \vec{n}_{\ell-1}) = (\vec{e}_0, \vec{d}_0, \vec{p}_0, \vec{n}_0)$ is the empty collection of tasks.) If we are able to show that the restricted sporadic task system $(\vec{e}_\ell, \vec{d}_\ell, \vec{p}_\ell, \vec{n}_\ell)$ is static-priority feasible on a uniprocessor as well (see below), we increment ℓ and proceed with the next iteration; else, we conclude that the input system $(\vec{e}_k, \vec{d}_k, \vec{p}_k, \vec{n}_k)$ is not static-priority on a uniprocessor feasible since its subset $(\vec{e}_\ell, \vec{d}_\ell, \vec{p}_\ell, \vec{n}_\ell)$ is not static-priority feasible on a uniprocessor.

Under the assumption that $(\vec{e}_{\ell-1}, \vec{d}_{\ell-1}, \vec{p}_{\ell-1}, \vec{n}_{\ell-1})$ is static-priority feasible on a uniprocessor, it follows from the results in [11] that $(\vec{e}_\ell, \vec{d}_\ell, \vec{p}_\ell, \vec{n}_\ell)$ is also static-priority feasible on a uniprocessor if and only if all the tasks of type ℓ meet their first deadlines when all tasks in system $(\vec{e}_\ell, \vec{d}_\ell, \vec{p}_\ell, \vec{n}_\ell)$ generate jobs according to the synchronous arrival sequence. Let t_f denote the time-instant at which the last type-ℓ job completes, and let x_i denote the number of jobs that each type-i task releases prior to time-instant t_f, for $1 = 1, 2, \ldots, \ell$. The following ℓ linear inequalities express x_i in terms of t_f:

$$x_i \cdot p_i \geq t_f, \text{ for } 1 = 1, 2, \ldots, \ell$$

The following inequality asserts that all these jobs complete by time-instant t_f:

$$\sum_{i=1}^{\ell} n_i \cdot e_i \cdot x_i \leq t_f$$

while the following inequality captures the requirement that the jobs of the type-ℓ tasks complete prior to their deadlines:

$$t_f \leq d_\ell$$

Thus, we have represented the static-priority uniprocessor feasibility of $(\vec{e}_\ell, \vec{d}_\ell, \vec{p}_\ell, \vec{n}_\ell)$, given the static-priority uniprocessor feasibility of$(\vec{e}_{\ell-1}, \vec{d}_{\ell-1}, \vec{p}_{\ell-1}, \vec{n}_{\ell-1})$, by $\ell + 2$ linear inequalities in ℓ integer variables and one additional variable.

5 Multiprocessor systems

In this section, we study the scheduling of restricted sporadic task systems upon multiprocessor platforms under the partitioned paradigm, for either static-priority or dynamic-priority scheduling. Our approach is based upon *dynamic programming*: since a partitioned multiprocessor system is essentially a collection of uniprocessor systems, we will apply dynamic programming techniques to extend the polynomial-time uniprocessor feasibility-analysis algorithms presented in Section 4 above to partitioned multiprocessor platforms.

Let us define restricted sporadic task system $(\vec{e}_k, \vec{d}_k, \vec{p}_k, \vec{\nu}_k)$ to be a **subsystem** of restricted sporadic task system $(\vec{e}_k, \vec{d}_k, \vec{p}_k, \vec{n}_k)$ if and only if there is a vector comprised of k non-negative integers $\vec{\mu}_k$ such that $\vec{n}_k \equiv \vec{\nu}_k + \vec{\mu}_k$. Observe that $(\vec{e}_k, \vec{d}_k, \vec{p}_k, \vec{n}_k)$ has a total of $\left(\Pi_{\ell=i}^{k}(n_\ell + 1)\right) = \mathcal{O}(n^k)$ distinct subsystems, including the empty one (i.e., with $\vec{\nu}_k = (0, 0, ..., 0)$), and the "full" one including all tasks in $(\vec{e}_k, \vec{d}_k, \vec{p}_k, \vec{n}_k)$ (i.e., the one for which $\vec{\nu}_k = \vec{\nu}_k$).

Given restricted sporadic task system $(\vec{e}_k, \vec{d}_k, \vec{p}_k, \vec{n}_k)$, to be scheduled upon an m-processor platform, we construct a **feasibility table**. Each column in the feasibility table corresponds to a different subsystem of $(\vec{e}_k, \vec{d}_k, \vec{p}_k, \vec{n}_k)$; hence, there are $\left(\Pi_{\ell=i}^{k}(n_\ell + 1)\right) = \mathcal{O}(n^k)$ columns. The table has m rows, and it will be filled such that the entry in the i'th row of a particular column has a "yes" if the subsystem corresponding to this column is i-feasible. Restricted sporadic task system $(\vec{e}_k, \vec{d}_k, \vec{p}_k, \vec{n}_k)$ is thus

feasible if and only if the table, when filled, contains a "yes" in the m'th row of the column corresponding to the "full" subsystem that includes all tasks in $(\vec{e}_k, \vec{d}_k, \vec{p}_k, \vec{n}_k)$. Observe that the table consists of m rows and at most $[(n/k) + 1]^k$ columns; and is hence of size polynomial in n for constant k. Our algorithm fills this table in a row at a time (beginning with the first row), and takes polynomial time to fill in each cell; consequently, the entire feasibility algorithm is polynomial in the number of task instances n.

§ **Filling in the first row.** Each cell in the first row of the feasibility table is filled with a "yes" or a "no" depending upon whether the subsystem corresponding to the cell's column is 1-feasible or not. Since these are uniprocessor scheduling problems, they can be solved in polynomial time by using the ILP-based algorithms described in Section 4.

§ **Filling in the i'th row.** We assume that all the rows $1, 2, \ldots, (i-1)$ have been filled, and describe how the i'th row is filled, for any i, $1 < i \leq m$. Our dynamic-programming algorithm for doing so is based upon the following lemma.

Lemma 1 *Restricted sporadic task system $(\vec{e}_k, \vec{d}_k, \vec{p}_k, \vec{\nu}_k)$ is i-feasible if and only if there are two vectors each comprised of k non-negative integers $\vec{\nu'}_k$ and $\vec{\nu''}_k$ such that the system $(\vec{e}_k, \vec{d}_k, \vec{p}_k, \vec{\nu'}_k)$ is $(i-1)$-feasible and the system $(\vec{e}_k, \vec{d}_k, \vec{p}_k, \vec{\nu''}_k)$ is 1-feasible, and $\vec{\nu}_k \equiv \vec{\nu'}_k + \vec{\nu''}_k$.*

Proof Sketch: We would assign the tasks in $(\vec{e}_k, \vec{d}_k, \vec{p}_k, \vec{\nu''}_k)$ to one processor, and the tasks in $(\vec{e}_k, \vec{d}_k, \vec{p}_k, \vec{\nu'}_k)$ on the remaining $(i-1)$ processors. ■

Let $(\vec{e}_k, \vec{d}_k, \vec{p}_k, \vec{\nu}_k)$ denote the subsystem associated with a cell in the i'th row of the feasibility table. For this $\vec{\nu}_k$, the subsystem associated with each cell in the $(i-1)$'th row that has been filled with a "yes" is a potential candidate for for the role of $\vec{\nu'}_k$ in the statement of Lemma 1; since there are polynomially many columns, there are polynomially many such candidates; consequently, each cell in the i'th row can be filled in polynomial time.

By using standard techniques from dynamic programming, the feasibility analysis algorithm described above is easily extended to actually obtain a feasible mapping from the set of tasks to the set of processors. In order to do so, one would need to retain some additional information during the filling in of the feasibility table. Specifically, for each cell in the feasibility table that is filled with a "yes," one would need to keep track of the identity of the cell in the *previous* row of the feasibility table corresponding to the mapping that was extended to obtain the partitioning corresponding to the current cell. Once the feasibility of the system has been determined (by having a "yes" in the cell in the last row of the feasibility table that correponds to the entire task set), one could use this information to recursively determine the tasks that go on each processor.

We have thus shown that multiprocessor partitioned feasibility-analysis of restricted sporadic task systems can be performed in time polynomial in the number of tasks in the system. This polynomial-time algorithm makes use of the polynomial-time uniprocessor feasibility-analysis algorithms we presented in Section 4, which are in turn based on Lenstra's polynomial-time algorithm for ILP in a constant number of variables [10].

Our algorithm establishes the computational complexity of partitioned feasibility-analysis for restricted sporadic task systems. In practice, however, we are not necessarily advocating that the ILP-based technique be actually used for filling in the first row of feasibility table (Section 5.1 above) — the commonly used techniques of response-time analysis and the processor demand approach have both been observed to have extremely efficient implementations in practice (despite their worst-case pseudo-polynomial time complexity), while Lenstra's ILP algorithm does not have particularly efficient implementations (although improvements to this basic algorithm have since been proposed [5]).

Also, we can exploit certain properties of the feasibility of restricted sporadic task systems to obtain speedups of our algorithm for filling in the feasibility table. These properties, and heuristics based on them, are briefly outlined below.

- *If a subsystem is i-feasible for some i, then it is j-feasible for all $j \geq i$.*

  Hence if we determine that a particular column in the i'th row is filled with a "yes,", then we may fill in the remaining entries of that column with "yes", too.

- *If a subsystem is i-feasible, then all subsystems of this subsystems are also i-feasible. Conversely, if a subsystem is not i-feasible, then neither are any subsystems of which this is a subsystem.*

  Hence, we may choose to do some form of "binary search" across each row in determining which column in the row to consider first. I.e., we should first determine (in)feasibility for subsystems of intermediate size — those that tend to have a large number of subsystems as well as being subsystems of a large number of

subsystems. Determining the (in)feasibility of such a subsystem would permit us to fill in many other entries in the same row as well, based on this property.

6 Summary and Conclusions

We have considered the uniprocessor and partitioned multiprocessor scheduling of *restricted sporadic task systems* – sporadic task systems in which there are only a few distinct kinds of tasks, and all tasks in the system are different instantiatitions of these few distinct kinds of tasks. We have shown that feasibility-analysis of such task systems can be peformed in time polynomial in the number of tasks in the system, under both the static-priority and the dynamic-priority scheduling paradigms.

While we have assumed in this paper that our multiprocessor platform is comprised of identical processors, we observe that our results are applicable to *uniform multiprocessor* platforms — platforms in which different processors have different speeds or computing capacities — as well.

In a *dynamic* real-time system, the composition of the system may change during run-time by having some tasks leave the system, and/ or new tasks joining. The design of efficient admission control algorithms for dynamic restricted sporadic task systems is an important and interesting research area that we intend to explore next.

In addition to being relevant to the analysis of restricted sporadic task systems themselves, we consider the research we have described here to form the foundations of *approximate* analysis algorithms (in particular, *sufficient* feasibility tests) for arbitrary –i.e., not restricted– sporadic task systems. Given an arbitrary sporadic task system, we may approximate it by a restricted task system by *over*estimating execution requirement, *tightening* the deadline, and *increasing* the rate (i.e., decreasing its period parameter) of every task in the system. Given a particular arbitrary sporadic task system, the challenge is to determine what the appropriate number and parameters of the few fixed kinds of tasks in an approximate system would be, to which we would approximate all the tasks in the input system. We are looking upon this as a form of optimization problem, and are working on comimg up with efficient, provable, bounds on both performance and run-time complexity.

References

[1] N. C. Audsley, A. Burns, M. F. Richardson, and A. J. Wellings. Hard Real-Time Scheduling: The Deadline Monotonic Approach. In *Proceedings 8th IEEE Workshop on Real-Time Operating Systems and Software*, pages 127–132, Atlanta, May 1991.

[2] S. Baruah, R. Howell, and L. Rosier. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-Time Systems: The International Journal of Time-Critical Computing*, 2:301–324, 1990.

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

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

[5] A. Frank and E. Tardos. An application of simultaneous diophantine approximation in combinatorial optimization. *Combinatorica*, 7(1):49–65, 1987.

[6] W. Horn. Some simple scheduling algorithms. *Naval Research Logistics Quarterly*, 21:177–185, 1974.

[7] M. Joseph and P. Pandya. Finding response times in a real-time system. *The Computer Journal*, 29(5):390–395, Oct. 1986.

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

[9] J. P. Lehoczky. Fixed priority scheduling of periodic tasks with arbitrary deadlines. In *IEEE Real-Time Systems Symposium*, pages 201–209, Dec. 1990.

[10] H. Lenstra, Jr. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8(4):538–548, November 1983.

[11] J. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation*, 2:237–250, 1982.

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

[13] J. M. Lopez, J. L. Diaz, and D. F. Garcia. Utilization bounds for EDF scheduling on real-time multiprocessor systems. *Real-Time Systems: The International Journal of Time-Critical Computing*, 28(1):39–68, 2004.

[14] J. M. Lopez, M. Garcia, J. L. Diaz, and D. F. Garcia. Worst-case utilization bound for EDF scheduling in real-time multiprocessor systems. In *Proceedings of the EuroMicro Conference on Real-Time Systems*, pages 25–34, Stockholm, Sweden, June 2000. IEEE Computer Society Press.

[15] A. K. Mok. *Fundamental Design Problems of Distributed Systems for The Hard-Real-Time Environment*. PhD thesis, Laboratory for Computer Science, Massachusetts Institute of Technology, 1983. Available as Technical Report No. MIT/LCS/TR-297.

[16] D.-I. Oh and T. P. Baker. Utilization bounds for N-processor rate monotone scheduling with static processor assignment. *Real-Time Systems: The International Journal of Time-Critical Computing*, 15:183–192, 1998.

[17] A. Srinivasan. *Efficient and Flexible Fair Scheduling of Real-time Tasks on Multiprocessors*. PhD thesis, Department of Computer Science, The University of North Carolina at Chapel Hill, 2003.