Sanjoy Baruah, Deji Chen, Sergey Gorinsky, and Aloysius Mok. Generalized multiframe tasks. Real-Time Systems 17 (1), pp. 5-22. July 1999.

Generalized multiframe tasks

Sanjoy Baruah   Deji Chen   Sergey Gorinsky   Aloysius Mok

Abstract

A new model for sporadic task systems is introduced. This model — *the generalized multiframe task model* — further generalizes both the conventional sporadic-tasks model, and the more recent multiframe model of Mok and Chen. A framework for determining feasibility for a wide variety of task systems is established; this framework is applied to this task model to obtain a feasibility-testing algorithm that runs in time pseudo-polynomial in the size of the input for all systems of such tasks whose *densities* are bounded by a constant less than one.

Keywords: Recurring multiframe tasks, preemptive uniprocessor scheduling, hard deadlines, feasibility analysis.

1 Introduction

Multiframe tasks were introduced by Mok & Chen [6], as a generalization to the well-known periodic task model of Liu & Layland [4]. A multiframe task is represented by a tuple $(\vec{E}, P)$, where $\vec{E} = [E_o, E_1, \ldots, E_{N-1}]$ is a vector of *execution times*, and P is the *minimum separation time*. The task generates an infinite succession of *frames*; the ready times of consecutive frames are at least P time units apart, the execution requirement of the i'th frame ($i \geq 0$) is $E_{i \bmod N}$, and the deadline of each frame is P time units after its ready time. Feasibility conditions were presented by Mok & Chen for the uniprocessor static-priority scheduling of systems of such multiframe tasks.

In this paper, we study a natural generalization of the multiframe task model. In our model – the **generalized multiframe (gmf) task model** – the multiframe model of Mok & Chen is further generalized in that (i) the deadlines of frames are allowed to differ from the minimum separation; further, all the frames need not have the same deadlines, and (ii) all the minimum separations need not be identical. Formally, a gmf task T is characterized by the 3-tuple $(\vec{E}, \vec{D}, \vec{P})$, where $\vec{E}, \vec{D}$ and $\vec{P}$ are N-ary vectors $[E_0, E_1, \ldots, E_{N-1}]$ of execution requirements, $[D_0, D_1, \ldots, D_{N-1}]$ of (relative) deadlines, and $[P_0, P_1, \ldots, P_{N-1}]$ of minimum separations respectively. The interpretation is as follows: The i'th frame of task T has an arrival time a_i, a deadline $a_i + d_i$, and an execution requirement of e_i, where

- $a_0 \geq 0$, and $a_{i+1} \geq a_i + P_{i \bmod N}$,
- $d_i = D_{i \bmod N}$, and
- $e_i = E_{i \bmod N}$.

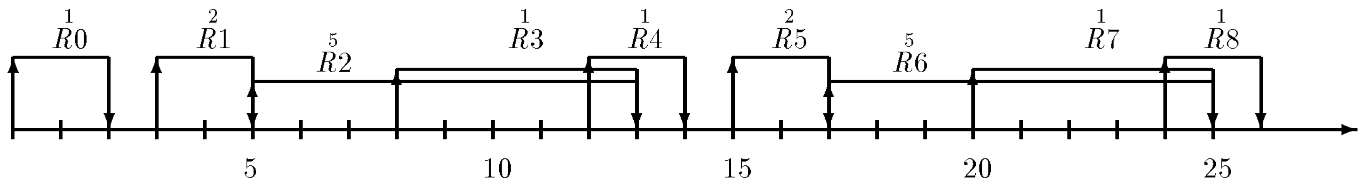


Figure 1: Example

Example 1 $T = ([1,2,5,1],[2,2,8,5],[3,2,3,4])$ is a gmf task with $N = 4$. Figure 1 depicts a legal sequence of frame arrivals, and the corresponding deadlines, for T (Ri denotes the i'th frame, and the number above Ri denotes the execution requirement of Ri). Observe that, as $R3$ and $R4$ demonstrate, it is *not* necessary that the i'th frame's deadline precede the arrival time of the $(i+1)$'th frame. (We point out that in this sequence of frame arrivals, each frame arrives at the earliest instant that it is legal for it to do so. We will revisit this scenario in Section 3.1.)

■

This research: Our focus in this paper is on determining uniprocessor feasibility conditions for systems of gmf tasks. That is, given a system of gmf tasks, how do we determine if they can always (i.e., for any legal set of frame arrival times) be scheduled to meet all deadlines on a single processor by an optimal uniprocessor scheduling algorithm such as Earliest Deadline First (EDF) [3] or Least Slack [5]? In order to answer this question, we abstract away from gmf tasks in Section 2, and study a (very general) category of task systems satisfying what we call the *task independence assumptions*. For such systems, we provide a general methodology, based upon the concept of *demand bound functions*, for determining feasibility. We apply this methodology to systems of gmf tasks in Section 3, to obtain an algorithm for feasibility determination for such systems. Perhaps somewhat surprisingly, it turns out that this problem is no more difficult, from a run-time complexity point of view, than determining feasibility in sporadic task systems [1, 2]. As a corollary to our main results, we also obtain an algorithm for determining feasibility for the multiframe model of Mok & Chen, that complements the static-priority feasibility algorithm in [6].

Significance of this research. The gmf task model is, in our opinion, the logical "next step" in the succession of models that have been developed to represent recurring tasks with minimum separation constraints. As was pointed out by Mok & Chen [6], the simplest model — each task characterized by an execution requirement e and a minimum separation p (deadlines are "implicit," i.e., they are assumed to occur p units after the frame's arrival) — is a trivial extension of the periodic task model of Liu & Layland [4]. Mok's generalization [5] explicitly added the deadline d, with the interpretation that the deadline of a frame occurs d time units after its arrival. The multiframe model of Mok & Chen [6] permitted each task to cycle through a given finite sequence of frame execution times, but did away with the

$$\begin{tikzcd}
 & \textbf{generalized multiframe} & \\
\text{sporadic, explicit deadlines [5]} \arrow[ur] & & \text{multiframe [6]} \arrow[ul] \\
 & \text{sporadic, implicit deadlines [4]} \arrow[ul] \arrow[ur] &
\end{tikzcd}$$

Figure 2: Relationship between the various sporadic task models

explicit deadline(s). The gmf-task model adds explicit deadlines to the multiframe model, and, for good measure, permits the minimum separations, too, to cycle through a given finite sequence of values. The relationship between the various models for sporadic tasks is graphically shown in Figure 2, where "A $\rightarrow$ B" denotes that the model B is a generalization of model A (the "$\rightarrow$" relation is, of course, transitive).

At first glance, the gmf model may appear to be easily analyzed for feasibility by transforming each gmf task into a set of regular sporadic tasks (in the sense of Mok [5]) with "offsets." Thus for example, a gmf task $([1,2],[2,2],[10,10])$ would be considered equivalent, for feasibility-analysis purposes, to two sporadic tasks T_1 and T_2 such that both have deadline 2 and minimum-separation 20, T_1 has execution requirement 1, and T_2 has execution requirement 2 and is "offset" from T_1 by 10 units (in the sense that the first frame of T_1 arrives at 0 and successive frames arrive exactly 20 units apart, while the first frame of T_2 arrives at 10 and successive frames arrive exactly 20 units apart). However, such an approach to feasibility-analysis is incorrect. (To see why, consider a gmf system consisting of two tasks – the one above, and the task $([1],[2],[20])$. Using the same reduction, this second gmf task would transform to a sporadic task with execution requirement 1, that first arrives at 0 and has successive arrivals exactly 20 units apart. The system would therefore be considered feasible. However, it is actually of course infeasible – consider the situation when the first frame of the second task arrives at the same instant as the second frame of the first task.) The problem lies in the fact that such a transformation fails to correctly identify the "worst-case" combination of frame arrivals: indeed, as we will see in the following sections, identifying such worst-case combinations of events is quite non-trivial, and a general methodology for doing so is one of the main new ideas developed here.

2 General framework

In this section, we consider a very abstract model of task systems. We study the feasibility problem for this abstract model, and provide a framework for determining feasibility for task systems in this model. This abstract model is defined as follows.

A *task* is defined to be an entity that generates a (possibly infinite) sequence of *jobs* or *frames*. Each job is characterized by an *arrival time*, a *deadline*, and a (worst-case) *execution requirement*. Each task is characterized by a *workload constraint*, which determines the exact nature of the sequence of jobs that a task may generate. A set of jobs generated by a task is called *legal* if it satisfies the workload constraint associated with the task. A *task system* consists of several tasks which share a resource. This research is restricted to the study of task systems having only one copy of the resource. This resource is assumed to be completely preemptable.

Task independence assumptions. We make the following assumptions regarding the various tasks in a task system:

1. *The runtime behavior of a task does not depend upon the behavior of other tasks in the system.* That is, each task is an independent entity, perhaps driven by separate external events. It is not permissible for one task to generate a job directly in response to another task generating a job. Instances of task systems *not* satisfying this assumption include systems where, for example, all tasks are required to generate jobs at the same time instant, or where it is guaranteed that certain tasks will generate jobs before certain other tasks. (However, such systems can sometimes nevertheless be represented in such a manner as to satisfy this assumption, by modelling the interacting tasks as a single task which is assumed to generate the jobs actually generated by the interacting tasks.)

2. *The workload constraints can be specified without making any references to "absolute" time.* That is, specifications such as "Task T generates a job at time-instant 3" are forbidden.

   There are several scenarios within which this assumption holds. Consider first a distributed system in which each task executes on a separate node (jobs correspond to requests for time on a shared resource) and which begins execution in response to an external event. All temporal specifications are made relative to the time at which the task begins execution, which is not *a priori* known.

   As another example, consider a distributed system in which each task (i.e., the associated process) maintains its own (very accurate) clock, and in which the clocks of different tasks are not synchronized with each other. The accuracy of the clocks permit us to assume that there is no clock drift, and that all tasks use exactly the same units for measuring time. However, the fact that these clocks are not synchronized rules out the use of a concept of an absolute time scale.

   (We observe that periodic task systems — where periodic task T is specified by the parameters *start-time* s, *computation requirement* c, and *period* p, with the intepretation that T must be scheduled for c units of time over interval $[s + kp, s + kp + p)$ for all integer p — violate the task independence assumption since the start-times are defined

	T_1		T_2
J_{11}	request$(1,5)$;		idle$(3,\infty)$;
J_{12}	(idle$(1,10)$; request$(1,2)$)	J_{21}	request$(2,4)$;
	$\vert\vert$	J_{22}	(idle$(2,8)$; request$(3,7)$)
J_{13}	(idle$(6,6)$; request$(2,3)$)		

The "request(x,y)" command issues a non-blocking request for x units of time on the shared resource with a deadline y time units from the instant the request is made. The "idle(x,y)" command indicates that the task is idle (actually, doing something that does not involve the shared resource) for an interval of time that is at least x and no more than y units long. The ";" indicates sequential composition, and the "||" indicates parallel composition. Each task may begin execution at any time.

Figure 3: Example tasks.

in terms of an absolute time scale. However, sporadic task systems [5, 1, 2], specified in terms of computation requirements, relative deadlines, and minimum separations, satisfy this assumption, as do systems of gmf-tasks.)

In terms of legal sets of jobs, the first condition above implies that a set of jobs generated by an entire task system is legal in the context of the task system if and only if the jobs generated by each task are legal with respect to the constraint associated with that task. Letting an ordered 3-tuple (a, e, d) represent the job generated by some task T with arrival time a, execution requirement e, and deadline d, the second condition implies that if $\{(a_o, e_o, d_o), (a_1, e_1, d_1), (a_2, e_2, d_2) \ldots\}$ is a legal arrival set with respect to the workload constraint for task T, then so is the set $\{(a_o - x, e_o, d_o - x), (a_1 - x, e_1, d_1 - x), (a_2 - x, e_2, d_2 - x) \ldots\}$, where x may be any real number.

The task independence assumptions are extremely general and are satisfied by a wide variety of the kinds of task systems one may encounter in practice. As described above, sporadic task systems satisfy these assumptions, as do "worst-case" periodic task systems [4] (which are periodic task systems where each task may choose any start-time — it is proved [4] that the worst-case occurs when all tasks have the same start time), even if each periodic task may specify a *deadline* in addition to computation requirment and period, and systems of multiframe tasks [6]. So do more sophisticated systems, such as, for example a teleconferencing application: "A process generates successive multi-packet video-message at least p_1 time units apart, and each video-message is followed by a multi-packet audio-message within p_2 time units ($p_2 < p_1/3$)," or the system described below in Example 2.

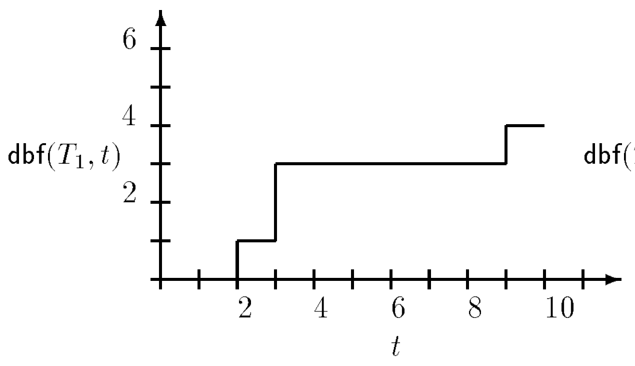
Example 2 Consider a system τ of two tasks T_1 and T_2 that share a resource (Figure 3). Task T_1 may begin execution at any time, and generates 3 jobs — J_{11} arrives at the shared resource immediately when T_1 begins execution, J_{12} arrives between 1 and 10 time units after T_1 begins execution, and J_{13} arrives exactly 6 time units after T_1 begins execution. Task T_2, too, may begin execution at any time, and generates 2 jobs with J_{21} arriving no earlier than 3 time units after T_2 begins execution, and J_{22} arriving between 2 and 8 time units after J_{21}.

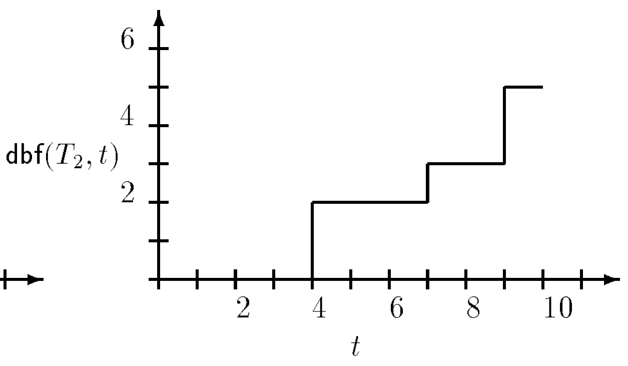
We will formally prove later in this section that this task system is in fact infeasible; meanwhile, we encourage the reader to informally verify that this is indeed so.

It is noteworthy that determining feasibility for many interesting[1] tasks systems not satisfying the task independence assumptions (such as periodic task systems with deadlines not equal to period) turns out to be computationally difficult (often NP-hard), and hence of limited interest from the perspective of efficient determination of feasibility.

***Definition*: Demand Bound Function.** Let T be a task, and t a positive real number. The *demand bound function* $\mathsf{dbf}(T, t)$ denotes the maximum cumulative execution requirement by jobs of T that have both arrival times and deadlines within any time interval of duration t.

Example 3 Consider again the example task system from Example 2. We plot the demand bound functions for tasks T_1 and T_2 below:





These functions have been determined by careful examination of the structures of the tasks; we illustrate the process by means of a few examples, and invite the reader to validate that the remainders of the functions are plotted correctly. Let t_1 denote the time at which task T_1 begins execution:

$\mathsf{dbf}(T_1, 3) = 3$: If J_{12} and J_{13} both arrive at time $t_1 + 6$, then they both have their arrival times and deadlines in the interval $[t_1 + 6, t_1 + 9)$.

$\mathsf{dbf}(T_1, 9) = 4$: If J_{12} arrives between $t_1 + 1$ and $t_1 + 7$, then J_{11}, J_{12} and J_{13} all have arrival times and deadlines in the interval $[t_1, t_1 + 9)$.

Let t_2 denote the time at which task T_1 begins execution, and let t' denote the arrival time of J_{21} ($t' \geq t_2 + 3$):

[1]By "interesting" we mean, at the least, that the tasks generate a potentially infinite sequence of jobs, and that these jobs not all be too constrained in the allowable deadlines that may be specified.

$\mathsf{dbf}(T_2, 4) = 2$: This corresponds to the interval between J_{21}'s arrival time and deadline.

$\mathsf{dbf}(T_2, 7) = 3$: This corresponds to the interval between J_{22}'s arrival time and deadline.

$\mathsf{dbf}(T_2, 9) = 5$: Suppose J_{22} arrives at the earliest possible time — i.e., at $t' + 2$. Then both J_{21} and J_{22} have their arrival times and deadlines in the interval $[t', t' + 9)$.

Theorem 1 Task system τ is infeasible if and only if $\sum_{T \in \tau} \mathsf{dbf}(T, t) > t$ for some positive real number t.

Proof: We prove the implicaton in one direction here, and outline how the other direction may be proved. The proof is similar to ones that appear in [1, 2], the interested reader is referred there for further details.

If: Suppose that $\sum_{T \in \tau} \mathsf{dbf}(T, t_o) > t_o$. Consider any time interval $[t_s, t_s + t_o)$.

For each $T \in \tau$, let $w(T) \overset{\text{def}}{=} \mathsf{dbf}(T, t_o)$. By the definition of demand bound functions, there is an interval of duration t_o during which T can generate jobs with a total execution requirement equal to $w(T)$, such that both their arrival times and deadlines lie within the interval. As a consequence of the task independence assumption, it follows that T can generate a similar set of jobs with arrival time and deadlines within the interval $[t_s, t_s + t_o)$, such that the total execution requirement of these jobs is also $w(T)$. Let $R(T)$ denote this set of jobs. It is straightforward to see that no scheduling algorithm can schedule the set of jobs $\bigcup_{T \in \tau} R(T)$, since each job has arrival time and deadline within the interval $[t_s, t_s + t_o)$, and the total execution requirement of all the jobs exceeds the length of the interval.

Only if: Let τ be an infeasible task system. Consider an infeasible set of jobs of τ, with the earliest job arriving at t_s, on which EDF misses its first deadline at t_f. There it can be shown, using techniques very similar to the ones in [1, 2], that there is an infeasible set of jobs of τ such that

- every task has its earliest job arrive at time t_s, and subsequent jobs arrive at the earliest possible times, and
- EDF never idles the processor on this set of jobs prior to missing a deadline at some time $t'_f \leq t_f$.

The total execution requirement of this set of jobs over $[t_s, t'_f)$ therefore exceeds $t'_f - t_s$; furthermore, this total execution requirement is exactly equal to $\sum_{T \in \tau} \mathsf{dbf}(T, t'_f - t_s)$. The theorem follows. ■

The statement of Theorem 1 immediately suggests a procedure for checking whether a system τ of tasks is feasible:

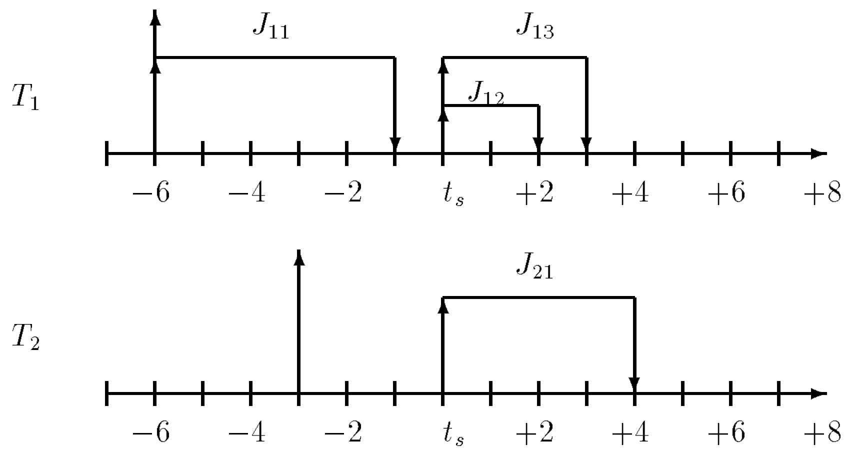
1. Determine the demand bound function for every task T in τ.
2. Determine if there exists a $t \in \mathbf{R}$ such that

$$\left(\sum_{T \in \tau} \mathsf{dbf}(T, t)\right) \quad > \quad t$$

Example 4 To verify whether task system τ from Example 2 is feasible, it suffices to compute $\mathsf{dbf}(T_1, t) + \mathsf{dbf}(T_2, t)$ at all points t such that $\mathsf{dbf}(T_1, t^-) < \mathsf{dbf}(T_1, t)$ or $\mathsf{dbf}(T_2, t^-) < \mathsf{dbf}(T_2, t)$:

t	2	3	4	7	9
$\mathsf{dbf}(T_1, t) + \mathsf{dbf}(T_2, t)$	$1 + 0 = 1$	$3 + 0 = 3$	$3 + 2 = 5$	$3 + 3 = 6$	$4 + 5 = 9$

Since $\mathsf{dbf}(T_1, 4) + \mathsf{dbf}(T_2, 4) > 4$, we conclude that τ is infeasible. An example of an unschedulable scenario is the following: Consider an interval $[t_s, t_s + 4)$. Suppose that T_1 begins execution at $t_s - 6$, and jobs J_{11}, J_{12}, and J_{13} arrive at times $t_s - 6, t_s$, and t_s respectively. Suppose further that T_2 begins execution at $t_s - 3$, and has job J_{21} arrive at time t_s . Then jobs J_{12}, J_{13} and J_{22} , with a total execution requirement of 5, all arrive and have deadlines within the interval $[t_s, t_s + 4)$, and are therefore unschedulable:



Comment: Since the demand bound function for each task T has an infinite domain (the real numbers), it is not possible to explicitly compute the demand bound functions in (1) above. Instead, an attempt must be made to implicitly determine $\mathsf{dbf}(T, t)$ for given T and t, as and when required. In order to be able to do so, we must know more about the exact nature of the tasks, and the manner in which they generate jobs.

3 Feasibility determination in gmf task systems

We now return to the problem of determining feasibility for a system of gmf tasks. Observe that gmf task systems satisfy the task independence assumptions of Section 2; hence the feasibility-determination methodology described above — determine the demand-bound function $\mathsf{dbf}(T, t)$ for every task T and every time-instant t, and then determine if there is a t such that $(\sum_{T \in \tau} \mathsf{dbf}(T, t)) > t$ — is applicable.

The l-MAD property. A gmf-task T satisfying the *localized Monotonic Absolute Deadlines (l-MAD)* property satisfies the additional constraint that its i'th frame has a deadline no later than that of its $(i+1)$'th frame; i.e., that $D_i \leq P_i + D_{(i+1) \bmod N}$ for all i. (Observe that the example task system depicted in Figure 1 satisfies the l-MAD property.)

The l-MAD property accurately captures the characteristics of a wide variety of real-time applications. Consider, for example, an application in which a remote multipurpose sensor samples several different kinds of signals in a round-robin fashion, with each kind characterized by a size (e.g., number of packets) and a latency requirement, that need to be transported over a packet-based circuit-switched network to a processing center. The sensor can be represented as a task $T = (\vec{E}, \vec{D}, \vec{P})$, with N – the dimension of the vectors – equal to the number of different kinds of data sampled, E_i and D_i representing the size and the latency requirement of the i'th kind of data, and P_i representing the time lapse between the instants a sample from the i'th and $(i+1) \bmod N$'th kind of data is obtained. With the network modelled as the shared resource, and the l-MAD property reflecting the fact that data from the sensor is sent into the network in a first-in first-out fashion, this model very accurately represents the traffic generated by the sensor and injected into the network. Several other applications (including the ones described in [6]) particularly from the domain of real-time networking, are naturally described as gmf tasks satisfying the l-MAD property.

Our goal in this section is to design an efficient feasibility-analysis test for systems of gmf-tasks, with or without the l-MAD property. For ease of exposition, we first focus on systems satisfying the l-MAD property — in Sections 3.1 and 3.2 below, we present a feasibility test for a system of gmf-tasks all of which satisfy the l-MAD property. Later (Section 3.3) we outline how to extend this feasibility test to handle systems of tasks that may not be l-MAD — it will be seen here that all the major ideas are exactly those used in the l-MAD case.

3.1 Determining the demand bound functions

Recall that a gmf task T is characterized by the 3-tuple $(\vec{E}, \vec{D}, \vec{P})$, where $\vec{E}, \vec{D}$ and $\vec{P}$ are N-ary vectors $[E_0, E_1, \ldots, E_{N-1}]$, $[D_0, D_1, \ldots, D_{N-1}]$, and $[P_0, P_1, \ldots, P_{N-1}]$, with $D_i \leq P_i + D_{(i+1) \bmod N}$ for all i.

It is not difficult to see that the worst-case workload, as quantified by $\mathsf{dbf}(T, t)$, occurs (for each value of t) when task T generates a job at some time instant t_o, and then generates subsequent jobs at the earliest possible times in order to have as many jobs as possible with deadlines at or before $t_o + t$. Unfortunately, it is not immediately obvious what this first job is, and indeed the choice of the first job for the interval which determines $\mathsf{dbf}(T, t)$ depends upon the choice of t, and is different for different t. (For example, in Figure 1 $\mathsf{dbf}(T, 2)$ is defined by job $R1$ while $\mathsf{dbf}(T, 5)$ is defined by the job $R0$ followed as soon as legally possible by the job $R1$.)

Consider the set of jobs $\rho(T) = \{R_o, R_1, R_2, \ldots\}$ such that (letting $a(R_i)$, $e(R_i)$ and $d(R_i)$ denote the arrival time, execution requirement, and deadline respectively of job R_i):

- $a(R_o) = 0$ and $a(R_{i+1}) = a(R_i) + P_{i \bmod N}$,
- $d(R_i) = a(R_i) + D_{i \bmod N}$, and
- $e(R_i) = E_{i \bmod N}$.

```
Algorithm build-list

  1. Generate ordered pairs ("workload", "interval size") as follows:
       For $i \leftarrow 0$ to $N-1$, do
         For $j \leftarrow 0$ to $N-1$ do
           generate the ordered pair $(e(R_i) + e(R_{i+1}) + \cdots + e(R_{i+j}),\quad d(R_{i+j}) - a(R_i))$.
  2. Sort the ordered pairs into an array in increasing order of interval size (within interval
     size, in decreasing order of workload).
  3. Delete all those ordered pairs whose workloads are not strictly larger than the workloads
     of all ordered pairs occurring prior to them in the sorted array.
```

Figure 4: Constructing a lookup list of $\mathsf{dbf}(T, t)$ for small t.

That is, $R_o, R_1, R_2, \ldots$ denote the jobs that are generated by T if T generates its first job — R_o — at time 0, and generates each subsequent job at the earliest possible time. (Figure 1 denotes the first few jobs in $\rho(T)$ for the example multiframe task T considered in Example 1.)

The crucial observation is that, if job R_i is to be the first job of an interval that determines $\mathsf{dbf}(T, t)$ for some t, then the jobs which contribute to this worst-case workload are exactly those jobs in the ordered sequence $[R_i, R_{i+1}, R_{i+2}, \ldots,]$ which have deadlines $\leq a(R_i) + t$. One could therefore in principle identify all candidate intervals that determine the value of the demand-bound function for different values of t by simply enumerating, for each pair of positive integers i, j, the cumulative execution requirement and interval-size if job R_i were to be the first job of a dbf-determining interval and job R_{i+j} were to be the last job in the interval. The demand bound function can be determined from this enumerated list: for each t, $\mathsf{dbf}(T, t)$ is equal to the largest execution requirement from among all the enumerated pairs which have interval-size no greater than t. Of course, such a procedure will never terminate, since the variables i and j grow without bound. (Observe, however, that we only need consider $R_i \in \{R_o, R_1, R_2, \ldots, R_{N-1}\}$; this is because every sequence of jobs $[R_{N+i}, R_{N+i+1}, \ldots, R_{N+i+j}]$ generates exactly the same ordered pair as the sequence $[R_i, R_{i+1}, \ldots, R_{i+j}]$.)

Figure 4 presents a procedure that constructs a list of execution-requirments generated by every possible sequence *of at most N jobs*, and the size of the interval within which this execution requirement must be met. It is relatively straightforward to observe that Algorithm build-list runs in time $O(N^2 \log N)$ and produces a list sorted by interval size such that, for a given t, $\mathsf{dbf}(T, t)$ can be determined from this list in $O(\log N)$ time, using binary search (provided, of course, that t is small enough to be present in this list).

Example 5 Consider once again the gmf task T of Example 1. Algorithm build-list generates one ordered pair for each combination of i and j, as i and j each range over $0, 1, 2,$ and 3:

	$j = 0$	$j = 1$	$j = 2$	$j = 3$
$i = 0$	$(1, 2)$	$(3, 5)$	$(8, 13)$	$(9, 13)$
$i = 1$	$(2, 2)$	$(7, 10)$	$(8, 10)$	$(9, 11)$
$i = 2$	$(5, 8)$	$(6, 8)$	$(7, 9)$	$(9, 12)$
$i = 3$	$(1, 5)$	$(2, 6)$	$(4, 9)$	$(9, 17)$

After sorting, the list looks as follows:
$\langle(2,2),(1,2),(3,5),(1,5),(2,6),(6,8),(5,8),(7,9),(4,9),(8,10),(7,10),(9,11),(9,12),(9,13),$
$(8,13),(9,17)\rangle$.

Upon deleting redundant pairs, the remaining ordered pairs are:

$$\langle(2,2,)(3,5),(6,8),(7,9),(8,10),(9,11)\rangle$$

■

Consider now an interval $[t_s, t_f]$ encompassing *more* than N jobs. Let R_s be the first, and R_f the last, job contained entirely within this interval:

$$s \stackrel{\text{def}}{=} \min \quad i \quad \ni \quad a(R_i) \geq t_s; \quad f \stackrel{\text{def}}{=} \max \quad i \quad \ni \quad d(R_i) \leq t_s \ .$$

Observe that the total execution requirement of all the tasks in $\rho(T)$ over the interval $[t_s, t_f]$ is $\sum_{s \leq i \leq f} e(R_i)$.

Suppose that there are $qN + r$ jobs in $\rho(T)$ in the interval $[t_s, t_f]$, where $r < N$. That is, r and q are defined thus:

$$r \stackrel{\text{def}}{=} (f - s + 1) \bmod N; \quad q \stackrel{\text{def}}{=} (f - s + 1 - r)/N \ .$$

These $qN + r$ jobs in $\rho(T)$ that contribute to the workload over $[t_s, t_f]$ can be grouped in the following manner:

$$\begin{array}{llllll}
R_s & R_{s+1} & \cdots & \cdots & \cdots & R_{s+N-1} \\
R_{N+s} & R_{N+s+1} & \cdots & \cdots & \cdots & R_{s+2N-1} \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
R_{(q-1)N+s} & R_{(q-1)N+s+1} & \cdots & \cdots & \cdots & R_{s+qN-1} \\
R_{qN+s} & R_{qN+s+1} & \cdots & R_f & &
\end{array}$$

The jobs in each row in the above grouping together have a total execution requirement of $E \stackrel{\text{def}}{=} \sum_{i=o}^{N-1} E_i$; the total execution requirement of all the tasks together is therefore

$$qE + \sum_{s+qN \leq i \leq f} e(R_i) \ . \tag{1}$$

Since $a(R_{s+qN}) = t_s + qP$, it follows that the total execution requirement over $[t_s, t_f]$ is equal to qE plus the total execution requirement over $[t_s + qP, t_f]$; furthermore, since the interval $[t_s + qP, t_f]$ contains $r < N$ jobs, the cumulative execution-requirement over this interval can be obtained from the list generated by Procedure build-list. **Algorithm** compute-dbf (Figure 5) is based upon this observation. $D_{\min}$ denotes the smallest relative deadline. The function-call get-from-list(t) uses the sorted list constructed by Algorithm build-list to return dbf(T, t) — as discussed above, this can be accomplished in $O(\log N)$ time.

```
compute-dbf($t$)
/*$E \overset{\text{def}}{=} \sum_{i=o}^{N-1} E_i$;  $P \overset{\text{def}}{=} \sum_{i=o}^{N-1} P_i$;  $D_{\min} \overset{\text{def}}{=} \min_{o \le i \le N-1} \{D_i\}$ */
if $t < D_{\min}$ $\longrightarrow$ return 0
$\square$         $\longrightarrow$ return $\left(\left\lfloor \frac{t - D_{\min}}{P} \right\rfloor E + \text{get-from-list}(D_{\min} + (t - D_{\min}) \bmod P)\right)$
```

Figure 5: Algorithm compute-dbf

Example 6 Consider once again the task T from Example 1, upon which we had simulated the behavior of Algorithm build-list in Example 5. Suppose now that we wish to determine dbf$(T, 15)$ by Algorithm compute-dbf. Noting that $P = 12$, $D_{\min} = 2$, and $E = 9$, we have

$$\text{dbf}(T, 15) = \left(\left\lfloor \frac{15 - 2}{12} \right\rfloor \cdot 9 + \text{get-from-list}(2 + (13 \bmod 12))\right) = 9 + 2 = 11 .$$

$$\text{dbf}(T, 100) = \left(\left\lfloor \frac{100 - 2}{12} \right\rfloor \cdot 9 + \text{get-from-list}(2 + (98 \bmod 12))\right) = 72 + 2 = 74 .$$

$$\text{dbf}(T, 11) = \left(\left\lfloor \frac{11 - 2}{12} \right\rfloor \cdot 9 + \text{get-from-list}(2 + (9 \bmod 12))\right) = 0 + 9 = 9 .$$

■

3.2 Feasibility determination

In Section 3.1 we have seen how Algorithm build-list preprocesses a given gmf task T in $O(N^2 \log N)$ time (where N is the dimension of the vectors representing T) to generate data-structures that allow dbf(T, t) to be computed in $O(\log N)$ time for any t.

Let $T = (\vec{E}, \vec{D}, \vec{P})$ be any gmf task; $\vec{E} = [E_o, \ldots, E_{N-1}]$; $\vec{D} = [D_o, \ldots, D_{N-1}]$; and $\vec{P} = [P_o, \ldots, P_{N-1}]$. We now define a reduction from T to a set $\gamma(T)$ of "regular" sporadic tasks with deadlines in the sense of [5][2].

Let $\langle (w_1, t_1), (w_2, t_2), \ldots, (w_m, t_m) \rangle$ denote the sorted list of (workload, interval-size) ordered pairs generated by Algorithm build-list on task T. (Observe that $w_i < w_{i+1}$, and $t_i < t_{i+1}$, for all i, $1 \le i < m$; and that $w_m \equiv E$, where E is as defined in Figure 5.) We define the set $\gamma(T)$ as follows:

$$\begin{aligned} \gamma(T) \overset{\text{def}}{=} \{ & (w_o, t_o, P), (w_1 - w_o, t_1, P), (w_2 - w_1, t_2, P), \ldots, \\ & \ldots, (w_{m-1} - w_{m-2}, t_{m-1}, P), (w_m - w_{m-1}, t_m, P) \} \end{aligned} \tag{2}$$

where $P \overset{\text{def}}{=} P_o + P_1 + P_2 + \cdots + P_{N-1}$.

[2]Recall that such tasks are represented by a 3-tuple (e, d, p), where e represents the execution requirement of each frame (job) generated by the task, d the time interval between the arrival-time and the deadline of each frame, and p the minimum time interval between the arrival instants of successive frames.

Example 7 In Example 5, we had traced the behavior of Algorithm build-list on the task T of Example 1. Using the sorted list of ordered pairs generated by Algorithm build-list, $\gamma(T)$ is seen to equal

$$\{(2,2,12),(1,5,12),(3,8,12),(1,9,12),(1,10,12),(1,11,12)\} \ .$$

■

It is not hard to see that the workload generated by $\gamma(T)$ when each task in $\gamma(T)$ generates its first job at time 0, and each subsequent job as soon as legal (i.e., exactly at times $k \cdot P$, for all $k \in \mathbf{N}$) is exactly the same as that generated by $\rho(T)$:

Lemma 1 For all gmf tasks T and all time intervals t

$$\mathsf{dbf}(T,t) = \sum_{T' \in \gamma(T)} \mathsf{dbf}(T',t) \ .$$

Theorem 2 follows.

Theorem 2 A system of gmf tasks τ is feasible on a single processor if and only if the system of sporadic tasks

$$\bigcup_{T \in \tau} \gamma(T)$$

is feasible on a single processor[3].

Thus, we have reduced the problem of determining feasibility of a set of gmf tasks to the problem of determining feasibility of a set of "regular" sporadic tasks. This reduction consists of calls to Algorithm build-list, followed by a simple multiset union operation, and is easily seen to take $O(n^2 \log n)$ time, where n is the length of the representation of the gmf task system.

The problem of determining feasibility of a system of sporadic tasks has been previously studied [1, 2]. The major result is that the sporadic task system $\{(e_1,d_1,p_1),\ldots,(e_n,d_n,p_n)\}$ can be tested for feasibility in time $O(\log n \cdot \frac{\rho}{1-\rho} \cdot \max_{1 \le i \le n}\{p_i - d_i\})$, where $\rho \stackrel{\text{def}}{=} \sum_{i=1}^{n} \frac{e_i}{p_i}$. As a consequence, we conclude that a gmf task system τ consisting of n gmf tasks[4] be tested for feasibility in time

$$O(\log n \cdot \frac{\rho}{1-\rho} \cdot \max_{T \in \tau}\{(P_o + P_1 + \cdots + P_{N-1}) - D_o\}) \ ,$$

where $\rho \stackrel{\text{def}}{=} \sum_{T \in \tau}(E_o + E_1 + \cdots + E_{N-1})/(P_o + P_1 + \cdots + P_{N-1})$ is the *density* of the task system. No feasible task system can have a density greater than one; for task systems with density *a priori* bounded from above by some constant $c < 1$, Theorem 2 suggests a pseudopolynomial-time algorithm for determining feasibility for a system of gmf tasks.

[3]Here the "union" operator – $\bigcup$ – defines a "multiset" union; i.e., duplicate tasks are not removed from the resulting system.

[4]Recall that each task $T \in \tau$ is represented by three vectors of length N each; $T \stackrel{\text{def}}{=} ([E_o,\ldots,E_{N-1}],[D_o,\ldots,E_{N-1}],[P_o,\ldots,P_{N-1}])$.

3.3 GMF-task systems without the l-MAD property

In the preceding, we have assumed that the gmf-task system satisfied the l-MAD property —- that each task T satisfied $D_i \leq P_i + D_{(i+1) \bmod N}$ for all i. We had argued that such task systems are likely to be the ones that arise most frequently in practice, and had hence developed an efficient feasibility test for these systems. For the sake of completeness, we briefly outline how the preceding results may be extended to handle systems of tasks that do *not* satisfy the l-MAD property.

For each gmf task T, Algorithm build-list will once again build a table containing $\mathsf{dbf}(T,t)$ for small enough t, and an analog of Equation 1 will form the basis of using this table to compute $\mathsf{dbf}(T,t)$ for larger t. When T satisfied the l-MAD property, we saw that it was sufficient to have Algorithm build-list only consider intervals whose workload was comprised of at most N jobs. For the general case, however, this is not sufficient: Consider a gmf task $T = ([91,1],[100,1],[5,5])$. This task is clearly infeasible in itself (consider its density); however, the smallest value of t for which $\mathsf{dbf}(T,t) > t$ is $t = 100$, and there will be a total of 11 jobs with both arrival times and deadlines within this interval with a cumulative execution requirement of 101.

Recall that Equation 1 permitted us to compute $\mathsf{dbf}(T,t)$ for arbitrarily large t, given the base values explicitly computed by Algorithm build-list. Let $D_{\max}$ be defined as follows:

$$D_{\max} \overset{\text{def}}{=} \max_{0 \leq i < N} \{D_i\}.$$

The following theorem provides the analog of Equation 1 for tasks that do not necessarily satisfy the l-MAD property.

Theorem 3 $\mathsf{dbf}(t + mP) = dbf(t) + mE$, where $D_{\max} \leq t < D_{\max} + P$ and $m \in \mathbf{N}$.

Proof: Similar to the derivation of Equation 1; details are omitted here. ■

As a consequence of this theorem, we know what Algorithm build-list must compute —- a lookup table of values of $\mathsf{dbf}(T,t)$ for all $t < D_{\max} + P$.

To summarize:

- Algorithm build-list, using a strategy very similar to the one depicted in Figure 4, generates all non-redundent ordered pairs ("workload", "interval size") for all intervals of size $< D_{\max} + P$. This can be done in time $O((1 + D_{\max}/P)N^2 \cdot \log((1 + D_{\max}/P)N))$. And, each lookup of this table would take time $O(\log((1 + D_{\max}/P)N))$. Observe that, when $D_{\max} \leq P$ — as is the case with l-MAD tasks — these reduce to the complexities of the corresponding operations in Section 3.1.
- Algorithm compute-dbf is modified as follows:

```
compute-dbf($t$)
$/* E \overset{\text{def}}{=} \sum_{i=o}^{N-1} E_i;\; P \overset{\text{def}}{=} \sum_{i=o}^{N-1} P_i;\; D_{\min} \overset{\text{def}}{=} \min_{o \leq i \leq N-1} \{D_i\};\; D_{\max} \overset{\text{def}}{=} \max_{o \leq i \leq N-1} \{D_i\} */$

if $t < D_{\max}$ + P $\longrightarrow$ return  0
□                      $\longrightarrow$ return  $\left(\left\lfloor \frac{t - D_{\max}}{P} \right\rfloor E$ + get-from-list$(D_{\max} + (t - D_{\max}) \bmod P)\right)$
```

- Defining $\gamma(T)$ from the table generated by Algorithm build-list as in Equation 2, we conclude that Theorem 2 holds for general gmf tasks as well, and can once again reduce the problem of feasibility determination to one of feasibility determination for a set of "regular" sporadic tasks. And, this system of sporadic tasks can be analyzed in pseudo-polynomial time, provided the density of the task system is *a priori* bounded from above by some constant $c < 1$,

4 Conclusions

Tasks that generate a potentially infinite sequence of frames, with consecutive frame arrivals separated by a specified minimum time interval, arise frequently in real-time systems. Starting with the seminal work of Liu and Layland [4], several increasingly more sophisticated models have been proposed for such task systems. These include the model devised by Mok [5], and the more recent multiframe model of Mok and Chen [6]. We have described here what we believe is the next logical generalization, presenting a model which unifies the incompatible models in [5] and [6]. Somewhat surprisingly, while this new model buys us considerably more expressive prower, it turns out that feasibility determination in this generalized model is no more difficult (from a run-time complexity point of view) than the earlier, simpler models — this we show by actually designing a feasibility testing algorithm for systems of gmf tasks.

References

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

[2] Sanjoy Baruah. *The Uniprocessor Scheduling of Sporadic Real-Time Tasks.* PhD thesis, Department of Computer Science, The University of Texas at Austin, 1993.

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

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

[5] Aloysius K. Mok. *Fundamental Design Problems of Distributed Systems for the Hard Real-Time Environment.* PhD thesis, MIT Laboratory for Computer Science, May 1983. Technical Report MIT/LCS/TR-297.

[6] Aloysius K. Mok and Deji Chen. A multiframe model for real-time tasks. In *Proceedings of the 17th Real-Time Systems Symposium*, Washington, DC, 1996. IEEE Computer Society Press.

Procedure feasibility(τ)

- Call Algorithm build-list on each $T_i \in \tau$.
- Construct a priority queue Q of 3-tuples (time, task-id, demand), with the 3-tuple with the smallest value of time having greatest priority.
- For each task T_i, let t_i be the smallest t such that $\mathsf{dbf}(T_i, t) > 0$ — t_i is easily determined from the list constructed by Algorithm build-list. Insert $(t_i, T_i, \mathsf{dbf}(T_i, t_i))$ into Q
-

```
$S \leftarrow 0$
repeat {
    $(t_o, T_o, d) \leftarrow$ deletemin($Q$)
    $S \leftarrow S + d$; if ($S > t_o$) return "infeasible"
    determine the smallest $t' > t_o$ such that $\mathsf{dbf}(T_o, t') > \mathsf{dbf}(T_o, t_o)$
    insert $(t', T_o, \mathsf{dbf}(T_o, t') - \mathsf{dbf}(T_o, t_o))$ into $Q$
    }
```

Figure 6: Feasibility determination for a system of gmf tasks

---

Appendix

In Section 3.2, we described how a gmf task system could be tested for feasibility by reducing each gmf task T to a set $\gamma(T)$ of "regular" sporadic tasks. We now briefly outline how feasibility determination of a system of gmf tasks may actually be perfromed starting from first principles, *without* first reducing to regular sporadic tasks. Given a system τ of gmf tasks to be scheduled on a single processor, Procedure feasibility (Figure 6) determines if τ is feasible.

Each task is initially preprocessed by Algorithm build-list, and a priority queue is constructed that will contain future time-instants at which the sum of the demand bound functions – $\sum_{T_i \in \tau} \mathsf{dbf}(T_i, t)$ – is to be incremented, and by how much. (Note that it is straightforward to determine, for any given t_o, the smallest $t' > t_o$ at which $\mathsf{dbf}(T, t)$ increases again — from Algorithm compute-dbf (Figure 5), it follows that this occurs at the earliest $t' > t_o$ at which either(i) get-from-list($t' \bmod P$) > get-from-list($t_o \bmod P$), or (ii) $\lfloor (t' - D_{\min})/P \rfloor > \lfloor (t_o - D_{\min})/P \rfloor$.) The variable S contains this sum of the demand bound functions at the "current time," which is intitally zero. Upon each iteration of the loop, the current time is updated to the value of time returned by the priority queue, which is when the next increment in S is to occur. S is updated to reflect this increase in $\mathsf{dbf}(T_o, t)$ at this new current time t_o, and a new 3-tuple representing the next increment to $\mathsf{dbf}(T_o, t)$ — at time t' — is inserted into the priority queue.

This loop iterates until a t is found for which $\sum_{T_i \in \tau} \mathsf{dbf}(T_i, t) > t$; if τ is feasible, it will never terminate. However, using techniques very similar to those used in [1, Lemma 6] (see also [2]), it can be shown that if τ is infeasible, then there is a t for which Procedure **feasibility** reports "infeasible" which is no greater than[5]

$$\frac{\rho}{1-\rho} \cdot \max_{T \in \tau}\{(P_o + P_1 + \cdots + P_{N-1}) - D_o\} ,$$

where $\rho \overset{\text{def}}{=} \sum_{T \in \tau}(E_o + E_1 + \cdots + E_{N-1})/(P_o + P_1 + \cdots + P_{N-1})$ is the density of the task system.

[5]Recall that each task $T \in \tau$ is represented by three vectors of length N each; $T \overset{\text{def}}{=} ([E_o, \ldots, E_{N-1}], [D_o, \ldots, E_{N-1}], [P_o, \ldots, P_{N-1}])$.