The Feasibility Analysis of Multiprocessor Real-Time Systems *

Sanjoy Baruah Nathan Fisher
The University of North Carolina at Chapel Hill

Abstract

The multiprocessor scheduling of collections of real-time jobs is considered. Sufficient tests are derived for determining whether a given collection of jobs can be scheduled to meet all deadlines upon a specified multiprocessor platform — these tests may be applied even when the collection of jobs is incompletely specified. The applicability of these tests to the scheduling of collections of jobs that are generated by systems of recurrent real-time tasks is discussed.

Keywords: Multiprocessor platforms; Feasibility analysis; Sufficient conditions; Recurrent tasks.

1 Introduction

Given the specifications of a hard-real-time system, *feasibility analysis* is the process of determining whether the system can be executed in such a manner that all deadlines are met. In this paper, we assume that a hard-real-time instance I is represented as a collection of jobs. Each job is characterized by three parameters — an *arrival time*, an *execution requirement*, and a *deadline* — with the interpretation that it needs to execute for an amount equal to its execution requirement between its arrival time and its deadline. We permit that an executing job may be preempted at any instant and have its execution resumed later, at no additional cost or penalty.

Upon multiprocessor platforms, two alternative paradigms for scheduling real-time systems have been considered: *non-migratory* and *migratory* scheduling. Under non-migratory scheduling, each job must execute entirely on a single processor, while in migratory scheduling it is permitted that a job that has previously been preempted on some processor may resume execution at a later point in time upon a different processor, at no additional cost.

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

Given a real-time instance I, determining whether I is non-migratory feasible upon a given number of processors is at least as hard as bin-packing, and so is NP-hard in the strong sense. Exponential-time algorithms are known for solving it. By applying network flow techniques, migratory feasibility analysis can be performed in time polynomial in the number of jobs in instance I.

However, both the exponential-time non-migratory feasibility analysis and the polynomial-time migratory feasibility analysis algorithms require that all three parameters of all the jobs in instance I be known beforehand, prior to performing feasibility analysis. This may not always be possible since many real-time application systems are comprised of processes that may execute repeatedly. In real-time scheduling theory, such processes are formally modelled by **real-time tasks**, each of which generates a sequence of jobs. Different formal models for real-time tasks place different constraints upon the permissible values for these job attributes; for example, the *sporadic task model* [12, 6] mandates that the arrival times of successive jobs of a task be at least a specified time-interval apart, and that all jobs have the same execution-requirement, and a deadline that is the same amount of time after the ready-time. Many other models for real-time tasks have also been proposed, including the *Liu and Layland* model [11], the *multiframe* model [13, 14], the *generalized multiframe* model [4], and a more general DAG-based model [3]. In their most general forms, these formal models together permit the abstract representation of reasonably general conditional real-time processes as well as simpler "straight-line" code, while attempting to retain some tractability of analysis. While each such task modelled in these formal models can guarantee minimum inter-arrival separations between successive jobs, the exact arrival times of individual jobs is only revealed during run-time. (Furthermore, the number of jobs in such real-time instances is typically unknown prior to runtime, and is potentially infinite or in any event very large.)

The research reported in this document is aimed at

obtaining characterizations of real-time workloads that permit for feasibility analysis to be performed prior to run-time, and that can be efficiently determined given system representations as collections of recurrent tasks. Note that feasibility-analysis concerns an *existential* question: does there exist a schedule? The related question of how one would actually generate such a schedule during run-time for systems deemed feasible is an important one, but is not addressed in this paper. Hence, the contributions of the results presented here should be considered to be rather more theoretical than pragmatic; although these theoretical results can serve as building-blocks for actually constructing real-time systems, this aspect is not explored further in this document. (See, for example, [17] for a sampling of the kinds of results that relate the theoretical concept of feasibility to actual run-time scheduling for specific multiprocessor scheduling algorithms.)

The remainder of this paper is organized as follows. In Section 2, we formally describe our job and processor models. In Section 3, we obtain sufficient conditions for multiprocessor feasibility analysis of real-time instances. In Section 4, we describe how certain system parameters, upon which our feasibility-analysis algorithm depends, may be computed for real-time systems that are specified using various popular models for recurrent processes, such as the Liu and Layland or the sporadic task models.

2 Workload and processor models

We characterize a real-time **job** J by three parameters: an *arrival time* $J.A$, an *execution requirement* $J.E$, and a *relative deadline* $J.D$, with the interpretation that J must execute for $J.E$ time units over the time interval $[J.A, J.A + J.D)$. Without loss of generality, we assume that execution requirement parameters of jobs are normalized with respect to processor speed; i.e., $J_i.E$ denotes the amount of time for which J_i must execute. We assume that our scheduling model allows for *processor preemption*: an executing job may be preempted at any instant and have its execution resumed later, at no additional cost or penalty. We will model a real-time **instance** I as a finite or infinite collection of such jobs: $I = \{J_1, J_2, \ldots\}$. We consider the migratory and non-migratory scheduling of real-time instances upon multiprocessor platforms: in non-migratory scheduling, each job may execute on one processor only, while in migratory scheduling a job that has been preempted on a processor may resume execution on a different processor at a later point in time. (However each job may be executing on at most one processor at each instant in time.) Clearly, the migratory scheduling paradigm is more general than non-migratory scheduling.

Let us define the demand of a real-time instance over a time interval to be the sum of the execution requirements of all jobs in the instance, that have both their arrival times and their deadlines within the interval:

$$\mathsf{demand}(I, t_1, t_2) \stackrel{\text{def}}{=} \sum_{(J \in I) \wedge (t_1 \leq J.A) \wedge (J.A + J.D \leq t_2)} J.E \tag{1}$$

For any real-time instance I, we define its $\mathsf{density}$ and load as follows:

$$\mathsf{density}(I) \stackrel{\text{def}}{=} \max_{J \in I} (J.E/J.D) \tag{2}$$

$$\mathsf{load}(I) \stackrel{\text{def}}{=} \max_{t_1 < t_2} \frac{\mathsf{demand}(I, t_1, t_2)}{t_2 - t_1} \tag{3}$$

Intuitively, $\mathsf{density}(I)$ represents the maximum computational demand of any *individual* job, and $\mathsf{load}(I)$ the maximum *cumulative* computational demand of any subset of the set of jobs, in the real-time instance I.

As stated in Section 1, algorithms for both migratory and non-migratory feasibility-analysis are known, when all characteristics of all the jobs in an instance are known during the time that feasibility analysis is being performed. We refer to such instances as **completely specified** instances; i.e., a completely specified real-time instance is one in which each job's arrival time, worst-case execution-requirement, and deadline parameters are all known prior to run-time.

For many real-time systems, particularly those that are modelled as collections of recurrent (such as Liu and Layland or sporadic) tasks, it is not possible to know beforehand what real-time instance will be generated by the system during run-time. In fact, different runs of the same system may result in different real-time instances being generated; in general, any given collection of recurrent real-time tasks may legally generate infinitely many different real-time instances, each of which satisfies the constraints placed on their generation by the recurrent tasks. Also, each such real-time instance may have infinitely many jobs, which means that feasibility-analysis algorithms that need to examine all the jobs in the instance cannot possible be applied.

In this paper, we present algorithms for the analysis of real-time instances I for which complete specifications need not be known, provided upper bounds on $\mathsf{density}(I)$ and $\mathsf{load}(I)$ are a priori known. We will refer to such real-time instances as **partially specified** instances; as we will see in Section 4, tight bounds on load and $\mathsf{density}$ can be efficiently determined for all instances generated by collections of recurrent real-time tasks represented in a wide range of formal models.

Lemma 1 below relates feasibility to the load and demand parameters:

Lemma 1 *If real-time instance I is feasible on an identical multiprocessor platform comprised of m unit-capacity processors, it must be the case that*

$$\textbf{(i)}\ \mathsf{density}(I) \leq 1 \ \text{ and } \ \textbf{(ii)}\ \mathsf{load}(I) \leq m.$$

Proof: The first condition follows from the observation that on a unit-capacity processor, a job that meets its deadline by executing continually between its arrival time and its deadline has a density of one; hence, one is an upper bound on the density of any job. Taken over all jobs in I, this observation yields the first condition.

For the second condition, the requirement that $\mathsf{load}(I) \leq m$ is obtained by considering a set of jobs of I that defines $\mathsf{load}(I)$; i.e., the jobs over an interval $[t_1, t_2)$ such that all jobs arriving in, and having deadlines within, this interval have a cumulative execution requirement equal to $\mathsf{load}(I) \times (t_2 - t_1)$. The total amount of execution that all these jobs may receive over $[t_1, t_2)$ is equal to $m \times (t_2 - t_1)$; hence, $\mathsf{load}(I) \leq m$. ■

If the converse of Lemma 1 were to hold, we would have an exact necessary and sufficient condition for migratory feasibility analysis. Unfortunately, the converse of Lemma 1 does *not* hold, as is illustrated by the following example:

Example 1 Consider the real-time instance I consisting of the three jobs J_1, J_2, and J_3. All three jobs arrive at time-instant 0; jobs J_1 and J_2 have execution requirement one and deadline one; and J_3 has an execution requirement equal to two and a deadline at time-instant two.

$\mathsf{density}(I)$ equals $\max\{1/1, 1/1, 2/2\}$, which is equal to 1.

Since the only arrival-times and deadlines are at time-instants 0, 1, and 2, $\mathsf{load}(I)$ can be computed by considering the intervals $[0, 1)$, $[0, 2)$, and $[1, 2)$:

$$\mathsf{load}(I) = \max\left(\frac{1+1}{1}, \frac{1+1+2}{2}, \frac{0}{1}\right) = 2\ .$$

Thus, instance I satisfies the conditions of Lemma 1 for $m = 2$; however, it is easy to see that all three jobs cannot be scheduled to meet their deadlines on two (since $m = 2$) unit-capacity processors.

■

Lemma 1 and Example 1 tells us that while every instance I that is feasible upon a platform comprised of m unit-capacity processors has $\mathsf{load}(I) \leq m$ and $\mathsf{density}(I) \leq 1$, not every instance I' with $\mathsf{load}(I') \leq m$ and $\mathsf{density}(I') \leq 1$ is feasible on such a platform.

A different characterization of real-time workload that has been proposed is the *synthetic utilization* [1] of a task instance. Lemma 1 also helps explain why we have chosen to not use this characterization rather than the load, density abstraction. Lemma 1 demonstrates that m is an upper bound on the load of any instance feasible upon m unit-capacity processors; however, there is no corresponding upper bound on the synthetic utilization of feasible instances[1]. In obtaining a bound (as we do in Theorem 1 below) such that any instance with load no larger than this bound is guaranteed feasible, we also obtain some measure of the relative efficacy or "goodness" of this bound since we know that all instances with load greater than m are guaranteed infeasible. Since there is no upper bound on synthetic utilization of feasible instances, no such relative goodness can be deduced for a synthetic utilization bound.

3 Sufficient feasibility conditions

We now derive sufficient conditions for determining whether a given real-time instance I is non-migratory feasible upon a specified number of processors. Since migratory scheduling is more general than non-migratory scheduling (in the sense that every non-migratory schedule is also a migratory schedule in which no migrations happened to occur), these sufficient conditions are sufficient conditions for migratory feasibility analysis, too. Our sufficient conditions involve the parameters $\mathsf{density}(I)$ and $\mathsf{load}(I)$ of the real-time instance I being analyzed; for any formal task model for which these parameters can be computed efficiently, our condition will translate into an efficient sufficient condition for feasibility-testing.

Suppose that a given real-time instance $I = J_1, J_2, \ldots$ is infeasible upon m unit-capacity processors. Without loss of generality, let us assume that the jobs are *indexed by non-decreasing order of relative deadline* — i.e., $J_i.D \leq J_{i+1}.D$ for all $i \geq 1$. We now describe an (off-line) multiprocessor algorithm for scheduling this collection of jobs; since the collection of jobs has no schedule (by virtue of being infeasible), it is necessary that at some point during its execution this algorithm will report failure.

Our algorithm considers job in the order $J_1, J_2, J_3, \ldots$, i.e., in non-decreasing order of relative deadline. In considering a job J_i, the goal is to assign it to a processor in such a manner that all the jobs assigned to each processor are preemptive uniprocessor feasible.

We now describe in detail the algorithm for assigning

[1] Consider an instance I comprised of n jobs, all arriving at time-instant 0 and having an execution-requirement of 1, and with the i'th job's deadline at time-instant i. This instance is feasible upon a single unit-capacity processor, yet has synthetic utilization equal to $\sum_{i=1}^{n}(1/i)$, which increases with n.

```
JOBASSIGN
    ▷ There are m unit-capacity processors, denoted π_1, π_2, ..., π_m
    ▷ I(π_k) denotes the jobs already assigned to processor π_k
1   for i ← 1, 2, ...
2       if there is a processor π_k such that (I(π_k) ⋃ {J_i}) is preemptive uniprocessor feasible
3         then
                ▷ assign J_i to π_k
4               I(π_k) ← I(π_k) ⋃ {J_i}
5         else return ASSIGNMENT FAILED
```

Pseudo-code for job-assignment algorithm.

job J_i:

- For each processor $\pi_k, 1 \leq k \leq m$, let $I(\pi_k)$ denote the jobs that have already been assigned to this processor; our assignment algorithm ensures that $I(\pi_k)$ is preemptive uniprocessor feasible.
- Assign J_i to any processor π_k such that doing so retains feasibility on that processor; if no such π_k exists, declare failure and exit.

A pseudo-code representation of this job-assignment algorithm is presented in Figure 1.

3.1 Algorithm analysis

Now, we will examine the circumstances under which Algorithm JOBASSIGN may fail. That is, we will derive conditions that must be satisfied by any real-time instance on which Algorithm JOBASSIGN returns ASSIGNMENT FAILED. By ensuring that this condition is never satisfied, we can then obtain a sufficient schedulability test for feasibility analysis of real-time instances.

Theorem 1 *Let I denote a real-time instance satisfying*

$$\mathsf{load}(I) \leq \frac{1}{3}\left(m - (m-1)\mathsf{density}(I)\right) \ . \qquad (4)$$

Algorithm JOBASSIGN *successfully assigns every job to some processor, on a platform comprised of m unit-capacity processors.*

Proof: Let us assume that Algorithm JOBASSIGN does not successfully assign all jobs in I, and let J_i denote the first job for which Algorithm JOBASSIGN returns ASSIGNMENT FAILED.

For each processor π_k, $I(\pi_k)$ is feasible by construction. Let W_k denote the minimum amount of execution that is performed over the interval $[J_i.A, J_i.A + J_i.D)$ in any preemptive uniprocessor schedule for $I(\pi_k)$ that meets all deadlines. Since J_i cannot be accommodated on any processor, it must be the case that $W_k > (J_i.D - J_i.E)$ on each processor; summing over all m processors, we conclude that

$$\sum_{k=1}^{m} W_k > m \cdot (J_i.D - J_i.E) \qquad (5)$$

Clearly, all of this execution belongs to jobs that arrive, and / or have their deadline in, the interval $[J_i.A, J_i.A + J_i.D)$.

Let J_s denote the job with earliest arrival time that has already been assigned to some processor, such that $J_s.A + J_s.D > J_i.A$; and let J_f denote the job with latest (absolute) deadline that has already been assigned to some processor, such that $J_f.A < J_i.A + J_i.D$ (see Figure 2).

Since jobs are considered in order of their relative deadline and J_s and J_f were both assigned prior to job J_i being considered, it must be the case that $J_s.D$ and $J_f.D$ are both no larger than $J_i.D$:

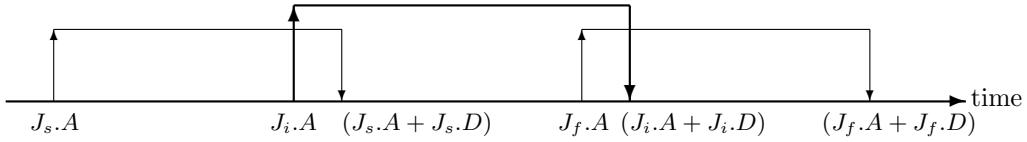
$$J_s.D \leq J_i.D \ \text{ and } \ J_f.D \leq J_i.D \ . \qquad (6)$$

From Figure 2, it is immediately evident that the interval $[J_s.A, J_f.A + J_f.D)$ is of size no more than $3 \times J_i.D$:

$$(J_f.A + J_f.D) - J_s.A \leq 3 \cdot J_i.D \ . \qquad (7)$$

Hence, all the work contributing to the expression on the right-hand side of Inequality 5 was generated by jobs that both arrive in,and have their deadline within, the $\leq 3 \times J_i.D$ interval $[J_s.A, J_f.A + J_f.D)$. By definition of $\mathsf{load}(I)$, the maximum amount of work arriving in, and having deadlines within, an interval of this length is at most $(J_f.A + J_f.D - J_s.A) \times \mathsf{load}(I)$, of which an amount $J_i.E$ (corresponding to the execution requirement of J_i) does not contribute to the RHS of Inequality 5. Hence

$$(J_f.A + J_f.D - J_s.A)\mathsf{load}(I) - J_i.E > m(J_i.D - J_i.E)$$



(Proof of Theorem 1.) Job J_i is being considered. Jobs J_s and J_f are the previously-assigned jobs with earliest arrival time and latest absolute deadline respectively, whose "intervals" overlap with that of J_i.

$$\Rightarrow \quad \text{(By Inequality 7 above)} \qquad (8)$$
$$3 \cdot J_i.D \cdot \mathsf{load}(I) - J_i.E > m(J_i.D - J_i.E)$$
$$\equiv \quad \mathsf{load}(I) > \frac{m - (m-1)\ \frac{J_i.E}{J_i.D}}{3} \qquad (9)$$

Note that the right-hand side decreases as $\frac{J_i.E}{J_i.D}$ increases; i.e., this condition is more likely to be satisfied for larger values of $\frac{J_i.E}{J_i.D}$. Since a *sufficient* condition for feasibility is that the negation of Condition 9 always hold, this is ensured by requiring that the negation of Condition 9 hold for the largest possible value of $\frac{J_i.E}{J_i.D}$, i.e., for $\frac{J_i.E}{J_i.D} = \mathsf{density}(I)$:

$$\mathsf{load}(I) \leq \frac{1}{3}\Big(m - (m-1)\ \mathsf{density}(I)\Big)$$

which is exactly Inequality 4 of the statement of the theorem. ■

Recall that our goal has been to obtain sufficient conditions for preemptive multiprocessor feasibility. Specifically, we had set out to obtain a *feasibility region* in the two-dimensional space $[0,1] \times [0,m]$, such that any instance I with $\mathsf{density}(I)$ and $\mathsf{load}(I)$ lying in this region is guaranteed to be feasible. Theorem 1 yields such a feasibility region: for given m, any instance I satisfying Equation 4 is guaranteed to be feasible upon m unit-capacity processors. In fact since it is also known from uniprocessor scheduling theory that any instance I satisfying ($\mathsf{load}(I) \leq 1$ and $\mathsf{density}(I) \leq 1$) is feasible upon a single unit-capacity processor and hence upon m unit-capacity processors for all $M > 1$, we can modify Inequality 4 to come up with the following sufficient condition for feasibility:

$$\mathsf{load}(I) \leq \max\left(1, \frac{1}{3}\Big(m - (m-1)\mathsf{density}(I)\Big)\right) \qquad (10)$$

This feasibility region is depicted visually in Figure 3.

By Lemma 1, recall that a necessary condition for instance I to be feasible on m unit-capacity processors is that $\mathsf{load}(I) \leq m$ and $\mathsf{density}(I) \leq 1$; Inequality 10 provides sufficient conditions. The following corollary to Theorem 1 formalizes this fact:

Corollary 1 *Any real-time instance I satisfying the following two conditions*

$$\mathsf{load}(I) \leq \frac{m^2}{4m-1} \qquad (11)$$
$$\mathsf{density}(I) \leq \frac{m}{4m-1} \qquad (12)$$

is feasible upon an m-processor unit-capacity multiprocessor platform under non-migratory multiprocessor scheduling[2]*.*

Proof: In order that instance I be feasible on m unit-capacity processors it is, by Equation 4, sufficient that

$$\mathsf{load}(I) \leq \frac{1}{3} \times (m - (m-1)\mathsf{density}(I))$$
$$\Leftarrow \quad \text{(From Condition 12)}$$
$$\mathsf{load}(I) \leq \frac{1}{3} \times \left(m - (m-1)\frac{m}{4m-1}\right)$$
$$\equiv \quad \mathsf{load}(I) \leq \frac{m^2}{4m-1}$$

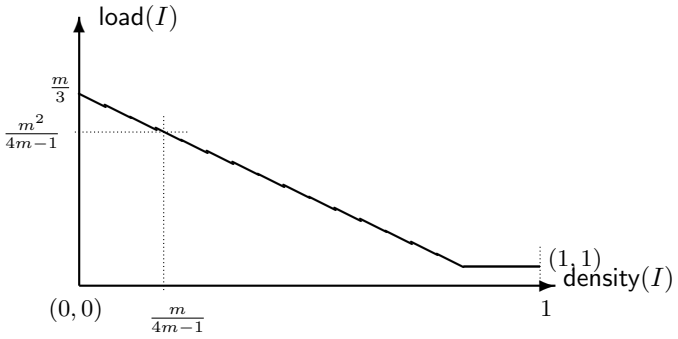
which is true, (by Condition 11 above). ■

The result in Corollary 1 above can be considered to be an analog of a "utilization" test, or a "processor demand criteria" test, for uniprocessor systems. According to Lemma 1, a necessary condition for real-time instance to be feasible on m unit-capacity processors is that $\mathsf{load}(I) \leq m$ and $\mathsf{density}(I) \leq 1$; by Corollary 1 above, it is sufficient that $\mathsf{load}(I) \leq m^2/(4m-1)$ and $\mathsf{density}(I) \leq m/(4m-1)$. Hence to within a constant factor of less than four, we have obtained bounds on the values of $\mathsf{load}(I)$ and $\mathsf{density}(I)$ that are needed (and suffice) for feasibility.

3.2 How tight is this bound?

As stated above, Corollary 1 asserts that our algorithm exhibits behavior that is, in a certain sense, no

[2] An alternative statement of this corollary could be: *the point* $(\frac{m}{4m-1}, \frac{m^2}{4m-1})$ *lies in the feasibility region of Figure 3* — this point is depicted by the dotted lines in the figure.



The feasibility region, as defined by Equation 10. All real-time instances I for which $(\mathsf{density}(I), \mathsf{load}(I))$ lie beneath the solid line are guaranteed feasible on m unit-capacity processors.

more than a factor of approximately four off optimal behavior. In fact, if we restrict our attention only to real-time systems that are characterized exclusively by their load and density parameters, we can show that our algorithm is actually within a factor of $2\frac{2}{3}$ of optimal behavior in this same sense. We do this by proving that a necessary condition for instance I to be feasible is in fact tighter than assumed above: there are instances I with $\mathsf{density}(I) = \frac{2}{3} + \epsilon$ and $\mathsf{load}(I) = \frac{2m}{3} + \epsilon$ for any positive ϵ, that are not feasible on m unit-capacity processors. Consider the following real-time instance I consisting of four jobs (each job is represented by the three-tuple $(J_i.A, J_i.E, J_i.D)$):

$$I = \{J_1 = (0, \frac{4}{3}, 2); J_2 = J_3 = (1, \frac{2}{3}, 1); J_4 = (1, \frac{4}{3}, 2)\} .$$

It may be verified that $\mathsf{density}(I) = \frac{2}{3}$ and $\mathsf{load}(I) = \frac{4}{3}$. This instance is feasible upon two processors; however, increasing the execution requirement of any of the four jobs by any amount at all would render it infeasible. For any (even) number of processors m, similar instances can be constructed with density two-thirds and load equal to $\frac{2m}{3}$. Since no algorithm, not even an optimal one, is able to schedule such an instance it therefore follows that *our algorithm is worse than optimal by a factor of no more than* $\frac{2/3}{1/4} = \frac{8}{3} = 2\frac{2}{3}$.

4 Determining density and load

The feasibility-approach algorithm in Section 3 above requires that load parameter $\mathsf{load}(I)$ and the density parameter $\mathsf{density}(I)$ of the real-time instance I being analyzed be known. As we had argued in the introduction, many real-time systems are comprised of collections of independent recurrent real-time tasks, each of which generates a potentially infinite sequence of jobs. Given the specifications of such a real-time system, we now discuss the issue of determining bounds on the load and density parameters of any real-time instance that could be generated by this system during run-time. Our approach is based upon the concept of the *demand bound function* (DBF) of recurrent real-time tasks; in Section 4.1 below, we describe this concept and explain how it relates to determining load and density. In Section 4.2, we briefly discuss the computational complexity of this approach towards multiprocessor feasibility.

4.1 The DBF abstraction

We start out with a definition of the demand bound function:

Definition 1 (Demand Bound Function) *Let τ_i denote a recurrent real-time task, and t a non-negative real-number. The demand bound function $\mathrm{DBF}(\tau_i, t)$ denotes the maximum cumulative execution requirement that could be generated by jobs of τ_i that have both ready times and deadlines within any time interval of duration t.*

The demand bound function is efficiently determined for all the recurring real-time task models mentioned in this paper; algorithms for doing so for the Liu and Layland and sporadic task models are to be found in [6], for the multiframe and generalized multiframe models in [4], and for the DAG-based model in [3]. As an illustrative example, we present without proof the formula for computing DBF for a task specified according to the sporadic task model – a proof may be found in [6]. Let a sporadic task τ_i be represented by the 3-tuple (e_i, d_i, p_i), with the interpretation that this task generates an infinite sequence of jobs each with execution requirement e_i and relative deadline d_i, and with the arrival times of successive jobs separated by at least p_i time units. For such a task,

$$\mathrm{DBF}(\tau_i, t) \stackrel{\text{def}}{=} \max\left(0, \left(\left\lfloor \frac{t - d_i}{p_i} \right\rfloor + 1\right) \times e_i\right)$$

Given that we know how to determine the DBF for many of the important recurrent real-time task models, we now discuss how to compute bounds on the value of $\mathsf{load}(I)$ for any real-time instance I that is generated by a collection of recurrent real-time tasks $\tau = \{\tau_1, \tau_2, \ldots\}$.

Let I denote any real-time instance that is generated during run-time by a collection of recurrent real-time tasks τ. The maximum cumulative execution requirement by jobs in I over any time interval $[t_1, t_2)$

is bounded from above by the sum of the maximum execution requirements of the individual tasks in τ:

$$\mathsf{demand}(I, t_1, t_2) \leq (\sum_{\tau_i \in \tau} \mathrm{DBF}(\tau_i, t_2 - t_1)) \ .$$

From the definition of load (Equation 3), it follows that

$$\mathsf{load}(I) \leq \max_{t \geq 0} \left\{ \frac{\sum_{\tau_i \in \tau} \mathrm{DBF}(\tau_i, t)}{t} \right\} \qquad (13)$$

How tight is the bound of Inequality 13? Clearly, it cannot in general be tight for all instances I generated by a recurrent task system τ, since τ may generate different instances that have different loads, while the bound of Inequality 13 is unable to distinguish between such different instances. However, we do not in general know beforehand which specific instance τ may generate during runtime; therefore, the bound of Inequality 13 must hold for all instances that τ might legally generate. So a more reasonable formulation of the "tightness" question for Inequality 13 would be: Is there *some* instance I that could be generated by τ, for which the load bound of Inequality 13 is tight?

The answer to this question depends upon the characteristics of the collection of recurrent tasks comprising τ. Informally, the requirement for Inequality 13 to represent a tight bound (in the sense discussed above) is that the different tasks comprising τ be completely independent of one another. This requirement is formalized in the *task independence assumptions* [4]. We briefly review these independence assumptions below; a more complete discussion may be found in [4].

There are two requirements that are satisfied by systems satisfying the task independence assumptions:

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 τ_i 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 the so-called *asynchronous periodic* task systems violate the task independence assumption since the start-times of each task's first job are defined in terms of an absolute time scale.)

These task independence assumptions are extremely general and are satisfied by a wide variety of the kinds of task systems one may encounter in practice. Most common task models, including the Liu and Layland [11], multiframe [13, 14] generalized multiframe [4], and the DAG-based model [3], satisfy these assumptions. So do more elaborately-specified systems, such as the following.

Example 2 (from [4]) Consider a system τ of two tasks τ_1 and τ_2 that share a resource (Figure 4). Task τ_1 may begin execution at any time, and generates 3 jobs — J_{11} arrives at the shared resource immediately when τ_1 begins execution, J_{12} arrives between 1 and 10 time units after τ_1 begins execution, and J_{13} arrives exactly 6 time units after τ_1 begins execution. Task τ_2, too, may begin execution at any time, and generates 2 jobs with J_{21} arriving no earlier than 3 time units after τ_2 begins execution, and J_{22} arriving between 2 and 8 time units after J_{21}. ■

It is noteworthy that determining feasibility for many interesting 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 in the strong sense) even on preemptive uniprocessor platforms, and

	τ_1		τ_2
J_{11}	request$(1, 5)$;		idle$(3, \infty)$;
J_{12}	(idle$(1, 10)$; request$(1, 2)$)	J_{21}	request$(2, 4)$;
	$\|\|$	J_{22}	(idle$(2, 8)$; request$(3, 7)$)
J_{13}	(idle$(6, 6)$; request$(2, 3)$)		

Example tasks. 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.

hence of limited interest from the perspective of efficient determination of feasibility.

The DBF abstraction is a very general formalism for representing real-time workloads, recurrent or not: in addition to all the recurrent formal models mentioned eariler, we can compute the DBF for other workload models such as, for example, network traffic that is characterized by the well-known (σ, ρ) model (see, e.g, [15, 16]) and subject to real-time bounds. To illustrate the process of determining DBF for other models, we present, in Example 3 below, the DBF functions for the two tasks in the system of Example 2.

Example 3 Consider again the example task system from Example 2. We plot the demand bound functions for tasks τ_1 and τ_2 in Figure 5, for the duration $0 \leq t \leq 10$. These functions have been determined by careful examination of the structures of the tasks; we illustrate the process by means of a few examples. In general, for any τ_i and any t, computing $\text{DBF}(\tau_i, t)$ may require exhaustive-search to determine the maximum cumulative execution requirement by jobs of τ_i with both arrival-times and deadlines within an interval of length t. Let t_1 denote the time at which task τ_1 begins execution:

$\text{DBF}(\tau_1, 3) = 3$**:** 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 $[t_1 + 6, t_1 + 9)$.

$\text{DBF}(\tau_1, 9) = 4$**:** 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 τ_2 begins execution, and let t' denote the arrival time of J_{21} ($t' \geq t_2 + 3$):

$\text{DBF}(\tau_2, 4) = 2$**:** This corresponds to the interval between J_{21}'s arrival time and deadline.

$\text{DBF}(\tau_2, 7) = 3$**:** This corresponds to the interval between J_{22}'s arrival time and deadline.

$\text{DBF}(\tau_2, 9) = 5$**:** 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 $[t', t' + 9)$.

■

We now return to the issue of the tightness of the bound of Inequality 13. Let τ denote a real-time system. If τ satisfies the task independence assumptions, then the bound of Inequality 13 is tight in the sense that there is some real-time instance I that could legally be generated by τ, for which

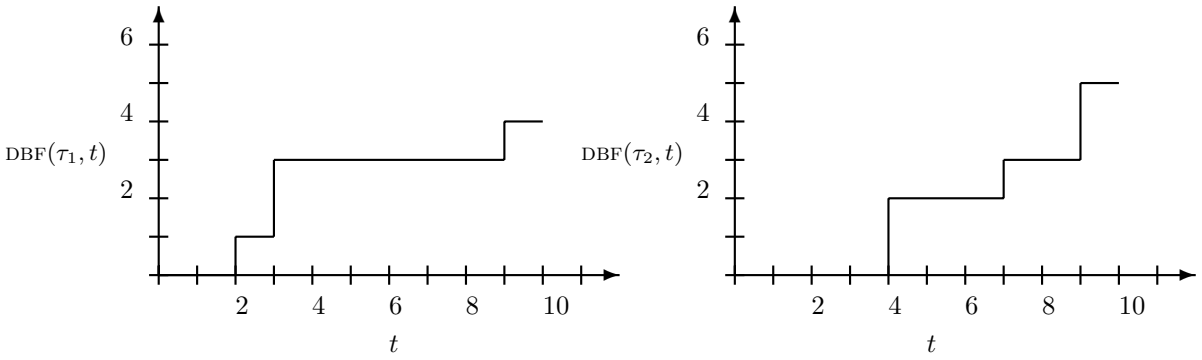
$$\mathsf{load}(I) = \max_{t \geq 0} \left\{ \frac{\sum_{\tau_i \in \tau} \text{DBF}(\tau_i, t)}{t} \right\} .$$

4.2 Run-time complexity

We now discuss the computational complexity of determining bounds on $\mathsf{density}(I)$ and $\mathsf{load}(I)$, when we are given that I is generated by a given system τ of recurrent real-time tasks. From the definition of $\mathsf{density}$ (Equation 2), it follows that $\mathsf{density}(I)$ is easily bound for any system in which all possible jobs that could be generated by each task can be enumerated; the computational complexity of doing so is directly proportional to the computational complexity of the enumeration[3].

The determination of $\mathsf{load}(I)$ is somewhat more complex. From Inequality 13, it can be seen that $\mathsf{load}(I)$ is defined in terms of the DBF functions of

[3]When the jobs are characterized by *upper bounds* on their execution requirements, the value of $\delta_{\max}(\tau)$ so computed also becomes an upper bound.



DBF(τ_1, t) DBF(τ_2, t) $[0, 10)$

all the tasks comprising τ. It has been shown [8, 7] that DBF can be efficiently computed for all the formal models of recurrent tasks (the Liu and Layland [11], the sporadic [12], the multiframe [13, 14], the generalized multiframe [4], and the DAG-based model [3]) discussed in this paper. This is achieved by doing a pseudo-polynomial amount of pre-processing per task, after which DBF(τ_i, t) for any t can be done in polynomial time.

To implement the (sufficient) multiprocessor feasibility test presented in this paper upon a task system τ, we would therefore do the following

1. Perform the DBF-preprocessing on each task in the task system τ.

2. Compute the bound on $\mathsf{density}(I)$.

3. Based upon the computed bound on $\mathsf{density}(I)$ and the available number of processors m, determine the bound B on $\mathsf{load}(I)$ that is implied by Equation 4 of Theorem 1.

4. The question now becomes: Is there a $t \geq 0$ such that
$$\sum_{\tau_i \in \tau} \text{DBF}(\tau_i, t) > (B \times m \times t) \ ? \qquad (14)$$
If the answer is "no," then τ is guaranteed to be feasible on the m unit-capacity processors. On the other hand, if the answer is "yes" then our test is not able to conclude the feasibility or otherwise of τ on the m processors.

For all the formal models of recurrent tasks considered in this paper, it can be shown that if Inequality 14 is to be satisfied at all, it will be satisfied for some "reasonably small" value of t — specifically, for some t with value no more than pseudo-polynomial in the parameters of the task system (a proof of this fact is beyond the scope of this paper – see [3] for ideas regarding how one would construct a proof). Consequently, we can simply check Inequality 14 for all values of t, up to this pseudo-polynomial bound, at which DBF(τ_i, t) changes value for some $\tau_i \in \tau$; if Inequality 14 is not satisfied for all of these values of t, we can conclude that it will not be satisfied for any value of t, and that τ is consequently feasible.

A final observation concerning computational complexity: recent work [2, 9, 5, 10] on *approximation algorithms* for uniprocessor scheduling has introduced many techniques for obtaining polynomial-time feasibility tests for systems of recurrent real-time tasks by "sacrificing" a (quantifiable) fraction of the computing capacity of the available computing capacity. In essence, these techniques approximate the values of DBF(τ_i, t) beyond a certain (small) value of t. We observe that these techniques all apply to our multiprocessor feasibility test as well; hence, a less accurate variant of our test can be devised, with a runtime that is polynomial in the representation of the task system.

5 Conclusions

Feasibility analysis on preemptive uniprocessors is well understood; in the notation of this paper, a necessary and sufficient condition for any real-time instance I to be feasible upon a unit-capacity uniprocessor is that
$$\mathsf{load}(I) \leq 1 \text{ and } \mathsf{density}(I) \leq 1 \ .$$

In this paper, we have reported on our attempts at obtaining similar results for multiprocessor scheduling. We have shown (Lemma 1) that while an analog of this uniprocessor condition, viz.

$$\mathsf{load}(I) \leq m \text{ and } \mathsf{density}(I) \leq 1 \ .$$

is necessary for instance I to be feasible upon a platform comprised of m unit-capacity processors, this condition is not sufficient for ensuring feasibility (see Example 1). We have obtained (Theorem 1) a sufficient condition for a real-time instance I, characterized only by its load and density parameters, to be feasible on a multiprocessor platform. We have explored how the result of Theorem 1 can be extended to the feasibility analysis of real-time systems that are comprised of collections of recurrent real-time tasks. We have considered an extremely general task model, characterized by the task independence assumptions — to our knowledge, there are no prior non-trivial results in scheduling theory concerning the analysis of such multiprocessor real-time systems.

References

[1] Abdelzaher, T., and Lu, C. A utilization bound for aperiodic tasks and priority driven scheduling. *IEEE Transactions on Computers 53*, 3 (2004).

[2] 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.

[3] Baruah, S. Dynamic- and static-priority scheduling of recurring real-time tasks. *Real-Time Systems: The International Journal of Time-Critical Computing 24*, 1 (2003), 99–128.

[4] Baruah, S., Chen, D., Gorinsky, S., and Mok, A. Generalized multiframe tasks. *Real-Time Systems: The International Journal of Time-Critical Computing 17*, 1 (July 1999), 5–22.

[5] 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).

[6] 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.

[7] Chakraborty, S. *System-Level Timing Analysis and Scheduling for Embedded Packet Processors*. PhD thesis, Swiss Federal Institute of Technology (ETH), Zurich, 2003. Available as Diss. ETH No. 15093.

[8] Chakraborty, S., Erlebach, T., and Thiele, L. On the complexity of scheduling conditional real-time code. In *Proceedings of the 7th Workshop on Algorithms and Data Structures* (Providence, RI, 2001), Springer Verlag, pp. 38–49.

[9] 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.

[10] Fisher, N., and Baruah, S. A polynomial-time approximation scheme for feasibility analysis in static-priority systems with bounded relative deadlines. In *Proceedings of the 13th International Conference on Real-Time Systems* (Paris, France, April 2005).

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

[12] Mok, A. K. *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.

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

[14] Mok, A. K., and Chen, D. A multiframe model for real-time tasks. *IEEE Transactions on Software Engineering 23*, 10 (October 1997), 635–645.

[15] Parekh, A. K., and Gallager, R. G. A generalized processor sharing approach to flow control in integrated services networks: the single node case. *IEEE/ACM Transactions on Networking 1*, 3 (June 1993), 344–357.

[16] Parekh, A. K., and Gallager, R. G. A generalized processor sharing approach to flow control in integrated services networks: the multiple node case. *IEEE/ACM Transactions on Networking 2*, 2 (April 1994), 137–150.

[17] Phillips, C. A., Stein, C., Torng, E., and Wein, J. Optimal time-critical scheduling via resource augmentation. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing* (El Paso, Texas, 4–6 May 1997), pp. 140–149.