Sustainability in Real-time Scheduling

Alan Burns
The University of York
burns@cs.york.ac.uk

Sanjoy Baruah
The University of North Carolina
baruah@cs.unc.edu

A scheduling policy or a schedulability test is defined to be sustainable if any task system determined to be schedulable remains so if it behaves "better" than mandated by its system specifications. We provide a formal definition of sustainability, and subject the concept to systematic analysis in the context of the uniprocessor scheduling of periodic and sporadic task systems. We argue that it is, in general, preferable engineering practice to use sustainable tests if possible, and classify common uniprocessor schedulability tests according to whether they are sustainable or not.

Categories and Subject Descriptors: Systems and Architecture

General Terms:
Additional Key Words and Phrases: Real-time scheduling, Uniprocessors, Schedulability tests, Fixed-priority, Earliest Deadline First

1. INTRODUCTION

The notion of **schedulability** is well understood within the real time systems community. A real-time system is specified according to some formal notation, and is determined to be schedulable with respect to a particular scheduling policy if it will meet all its timing requirements when executed on its target platform with that scheduling policy. The timing requirements are usually expressed as deadlines. So schedulability implies that all deadlines are satisfied if the system behaves according to its parameterized specification.

1.1 Sustainability

Although there is a general consensus on the notion of schedulability, there is some ambiguity regarding the interpretation of a given system specification. Specifically, do the specifications represent exact values, or worst-case bounds, of the parameters that are experienced by the system during run-time? There seems to be general agreement about the expected interpretation for some of the parameters; for example, the "worst-case execution time" (WCET) parameter is, as its name makes clear, expected to represent an upper bound on execution requirement (rather than the exact execution requirement during every run of the system). However, there is to our knowledge no such agreement regarding the interpretation of many other parameters that are commonly used in real-time system specification.

We recently [Baruah and Burns 2006] introduced the notion of **sustainability** to formalize the expectation that a system that is schedulable under its worst-case specifications should remain schedulable when its real behavior is "better" than worst-case. In this paper, we further generalize, extend, and elaborate upon the ideas introduced in [Baruah and Burns 2006].

Sustainability is necessarily defined with respect to a given intended interpretation of the parameters used in specifying a real-time system. With respect to a given such interpretation:

- A given *scheduling policy* is sustainable if any system that is schedulable under its worst-case specification remains so when its behavior is better than worst-case. We will see that many well-known and widely-used scheduling policies, including the uniprocessor earliest deadline first (EDF) [Liu and Layland 1973; Dertouzos 1974] and Deadline Monotonic (DM) [Leung and Whitehead 1982] scheduling of periodic task systems, are in fact not sustainable under commonly (if implicitly) understood interpretations of system parameters.
- For scheduling policies that are not sustainable, we introduce the concept of a sustainable *schedulability test*. We will see that in designing schedulability tests generally there is often a tradeoff between accuracy and sustainability: exact schedulability tests are usually not sustainable, but sustainable sufficient schedulability tests can often be designed. It turns out that this fact has been implicitly taken into account by the real-time systems community: we will see that many of the more widely-used schedulability tests are sufficient and sustainable, rather than exact but non-sustainable.

1.2 Sustainability versus robustness

We distinguish between sustainability and the related, but distinct, concept of *robustness*. A robust system retains schedulability even when it operates beyond the worst-case assumptions as permitted by the interpretation of its specification. Clearly a system can never be fully robust – at some point the system will become so

[1][Mok and Poon 2005] use the term robustness to imply what is called sustainability in this paper, we feel the use of 'robustness' confuses two distinct properties. The term 'stability' is also used in this context, but again there is confusion with the more normal use of this term in the control literature.

overloaded that it will fail. However it is not ruled out that a system could in principle be fully sustainable, since no amount of 'underload' need force failure[1].

Intuitively, sustainability requires that schedulability be preserved in situations in which it should be "easier" to ensure schedulability. Robustness, on the other hand, is an examination of system behavior when job parameters are outside the defined 'limits' of the system: e.g., when jobs arrive earlier than expected, or have greater execution requirement than permitted. Sensitivity analysis can be applied to explore the parameter space for schedulability in order to determine the robustness of schedulability tests [Davis and Burns 2007].

1.3 Summary of results

At first glance, it may appear that the concept of sustainability is well-understood by designers of real-time systems, and that this concept is already incorporated into the many schedulability tests in use. However, we demonstrate in this paper that this is not true: there are many issues concerning sustainability that have not been explored before. This extremely important schedulability property deserves a methodical and systematic study; in this paper, we have attempted to perform such a study in the context of preemptive uniprocessor scheduling. Although it is clear that most preemptive schedulability tests are indeed sustainable with respect to the execution-requirement parameter, we demonstrate that it is a misconception to suppose that all common schedulability tests are sustainable with respect to the remaining parameters. Indeed we present proofs here to demonstrate that several well-known schedulability tests are not sustainable (see, e.g., Theorem 13). We also prove that both the *response time analysis* and the *processor demand criteria* tests are sustainable under reasonable assumptions on their use; perhaps this property contributes to the popularity of these particular schedulability tests. We attempt to draw larger lessons concerning which properties of schedulability tests tend to render them sustainable or not.

1.4 Organization

The remainder of this paper is organized as follows. Section 2 describes the terminology, notations and system model. Section 3 further motivates the use of sustainable schedulability analysis and argues that sufficient and sustainable schedulability is more important than sufficient and necessary. The analysis of fixed priority systems is covered in Section 4, and of dynamic-priority systems in Section 5. In Section 6, we identify factors that tend to make it difficult to achieve sustainable schedulability analysis; in particular, we note that task offsets, and best-case execution-time estimates, are rather brittle with regard to sustainability. In Section 7, we briefly comment on sustainability issues concerning the server abstraction.

2. SYSTEM MODEL

In order to formalize the concept of sustainability, we need to define what we mean by a real-time system. Let us therefore model a real-time system as being comprised of several **tasks**, each of which gives rise to a series of **jobs** that are to be executed on

a single processor. Each job is characterized by: an *arrival time*, denoting the time-instant at which the job is said to arrive at the processor; a *ready time* (which is zero or more time units after its arrival time), denoting the earliest time-instant at which the job may begin executing – the length of time that elapses between the job's arrival time and its ready time is called its *release jitter*; an *execution requirement*; and a *relative deadline*, denoting the length of time that may elapse after a job's arrival before it must complete execution. Different task models place different restrictions on the parameters of the sequences of jobs that may be generated by each task; for instance, periodic task models specify that successive jobs of a task arrive an exact pre-specified time apart, sporadic models mandate a minimum temporal separation between the arrivals of successive jobs of a task, etc.

We focus our attention in this paper to the uniprocessor scheduling of periodic and sporadic task systems. A real-time system, Γ, is assumed to consist of N tasks (τ_1, ..., τ_N) each of which gives rise to a, potentially infinite, series of jobs that are to be executed on a single processor.

For the most part, we assume that this shared processor is preemptive, but allow (Section 2.2) for the presence of additional non-preemptive serially reusable resources. In Sections 4.4 and 5.2, we discuss how our results may be extended to apply to non-preemptive processors as well.

Each task τ_i is characterized by several parameters:

• A *worst-case execution time* C_i, representing the maximum amount of time for which each job generated by t_i may need to execute.

• A *relative deadline* parameter D_i, with the interpretation that each job of τ_i needs to complete execution within D_i time units of its arrival. Without loss of generality, we will assume that *tasks are indexed according to non-decreasing order of their relative deadline parameters*: $D_i \leq D_{i+1} \forall i$.

• A *release jitter* J_i, with the interpretation that each job of τ_i is not eligible to execute until J_i time units after its arrival.

• A *period* or *minimum inter-arrival time* T_i; for *periodic* tasks, this defines the exact temporal separation between successive job arrivals, while for *sporadic* tasks this defines the minimum temporal separation between successive job arrivals. (Thus while a periodic task system defines by a unique pattern of job arrivals, a sporadic task system may legally have infinitely many different patterns of job arrivals, with the arrival patterns differing in the inter-arrival separations of different jobs of the same task.)

Some terminology: the least common multiple of the period parameters of all the tasks is commonly referred to as the *hyperperiod* of the task system., and is often denoted by H.

Each job is also characterized by a series of parameters: ready time r_i, execution time e_i and deadline d_i. The set of all jobs is denoted by $\mathcal{J}$.

Let Π denote a dispatching policy such as EDF or fixed-priority. A periodic task system is said to be Π-*schedulable* if all jobs of all tasks meet their deadlines when the task system is scheduled using dispatching policy Π. A sporadic task system is said to be Π-schedulable if all jobs of all tasks meet their deadlines for all patterns of jobs arrivals that satisfy the specified inter-arrival constraints. A task system is said to be

feasible if it is Π-schedulable for some dispatching policy Π.

For a system that must be guaranteed, a *schedulability test* is applied that is appropriate for the dispatching policy of the execution platform. A schedulability test is defined to be *sufficient* if a positive outcome guarantees that all deadlines are always met. Clearly sufficiency is critically important for almost all real-time systems. A test can also be labeled as *necessary* if failure of the test will indeed lead to a deadline miss at some point during the execution of the system. A *sufficient and necessary* test is *exact* and hence is in some sense optimal; a sufficient but not necessary test is pessimistic, but for many situations an exact test is computationally intractable. From an engineering point of view, a tractable test with low pessimism is ideal.

Above, we have characterized each recurring task by the four parameters (C_i, D_i, T_i, J_i). Sustainability for such periodic and sporadic task systems may be defined with respect to any of the $2^4 = 16$ subsets of this set of 4 parameters. For example, a scheduling policy that guarantees to retain schedulability if actual execution requirements during run-time are smaller than specified WCET's, and if actual jitter is smaller than the specified maximum jitters, would be said to be *sustainable with respect to WCET's and jitter*. A scheduling policy or a schedulability test for periodic and sporadic task systems is said to be sustainable if it is sustainable with respect to all four parameters:

Definition 1 *A scheduling policy and/or a schedulability test for a scheduling policy is* sustainable *if any system deemed schedulable by the schedulability test remains schedulable when the parameters of one or more individual tasks[s] are changed in any, some, or all of the following ways*: (i) *decreased execution requirements*; (ii) *larger periods*; (iii) *smaller jitter*; and (iv) *larger relative deadlines.*

We would like to stress that declaring a scheduling policy or a schedulability test for periodic/sporadic task systems to be sustainable represents a stronger claim than simply that a task system deemed schedulable would remain schedulable with "better" parameters (e.g., with a larger period or relative deadline, or a smaller execution-requirement or jitter). In addition, we require that such a system continues to meet all deadlines even if the parameter changes are occurring "on line" during run-time. We also permit that the parameters change back and forth arbitrarily many times. The only restriction we will place on such parameter-changing is that *each generated job have exactly one arrival time, ready-time, execution requirement, and deadline during its lifetime.*

A schedulability test for periodic/sporadic task systems may be sustainable with respect to some, but not all, task parameters. For example, it is easy to show that all sufficient schedulability tests for fixed-priority preemptive scheduling are sustainable with respect to execution-requirement; however, Example 1 illustrates that no exact schedulability test for the fixed-priority preemptive scheduling of periodic task systems can be sustainable with respect to jitter. One of the objectives of our research is to identify the parameters, with respect to which, some of the more commonly-used schedulability tests are sustainable.

2.1 Other task parameters

Some additional parameters are sometimes used in representing periodic and sporadic

task systems:

• An **offset** parameter O_i for periodic tasks, denoting the arrival time of the first job of task τ_i. (Periodic task systems in which all tasks have the same value for the offset parameter are referred to as *synchronous* or *zero-offset* periodic task systems.)

• Mimimum or **best-case** execution times (BCET's) of jobs of a periodic or sporadic task.

Offsets and BCET's may allow for a more accurate representation of actual system characteristics; unfortunately, they both seem inherently incompatible with sustainability. That is, schedulability tests that do not "ignore" these parameters are generally not sustainable in that task systems deemed schedulable cease to be so if not just these parameters change, but also if other parameters such as the deadline or period change "for the better" (see Section 6). It seems that the safe way to analyse systems with offsets and best-case execution times is to ignore them (i.e., assume that all these parameters are equal to zero).

2.2 Sharing non-preemptable resources

Otherwise independent tasks may interact through the sharing of additional serially reusable non-preemptable resources, access to which are controlled through the use of mechanisms such monitors, semaphores or critical sections. The presence of such shared resources gives rise to *blocking*, and schedulability tests for such systems take such blocking into account. The sustainability of such schedulability tests is considered in Sections 4.4 and 5.1

3. MOTIVATION FOR SUSTAINABLE ANALYSIS

Concentrating first on execution time, it is clear from all forms of WCET analysis that the parameter C_i is an upper bound. It may in itself overestimate the worst-case situation (due to difficulties in modeling a processor with caches, pipeline, out-of-order execution, branch prediction etc). Additionally, any particular job may not take its worst-case path and hence will require a computational resource significantly less than even a tight upper bound. For all realistic systems considerable variability in execution times are to be expected and hence sustainability is required. This may seem obvious but a number of forms of schedulability analysis fail to have this property. Two of these will be reviewed later in Section 4.6.

As well as the natural variability in computation time, a significant change in the C_i values can arise during a system upgrade. Typically a faster CPU is introduced and one would not expect time failures to then appear in what was a timely system. There is however practical evidence of such failures occurring in real systems due to the use of unsustainable analysis[2].

[2]It is difficult to provide reference material to substantiate this observation, nevertheless the first author is aware of a number of real projects that have failed or underachieve due to this type of timing problem. For example a control system that could not generate output in the same minor cycle as the associated input. The time constant in the control loop was therefore chosen to reflect this behavior. During system upgrade a faster processor was used with the result that output was now produced in the same minor cycle leading to a degrade in control (as the time constant was not updated - the control engineers were not involved in the system upgrade).

Although variability in computation time is the more common, changes in the other parameters are also possible. Polling rates may change with a decrease in frequency usually assumed to be safe. But for the T_i parameter it is sometimes the simplifying assumptions made in the schedulability test itself that can give rise to unsustainable analysis. Consider the following example.

Sporadic tasks can arrive at any time as long as there is a least T_i between any two arrivals. For simple system models it is easy to prove that the worst-case behavior of a sporadic task is when it arrives at its maximum rate (i.e. performs like a periodic task). But for other models the worst-case critical behavior its not as easy to identify. For example in the analysis of (n,m) hard deadlines a sufficient and necessary test would need to first model the worst-case arrival patterns of high priority sporadic jobs as this does not occur when they arrive with maximum frequency [Bernat and Burns 1997]. The resulting analysis is complex (indeed intractable for large systems) and unsustainable. Better to use a simple form of sufficient but not necessary test that is sustainable, and in which the worst-case occurs (for the analysis) when sporadic tasks behave as if they are periodic.

Sustainability with respect to the deadline parameter is a significant issue only with respect to those scheduling policies, such as the earliest deadline first (EDF) policy (and associated resource-sharing protocols such as the Stack Resource Policy [Baker 1991]) that incorporate the deadline parameter into their specification. Otherwise since all models of time are transitive, if a job completes before its deadline d_i and a new deadline d_i' is introduced with $d_i' > d_i$ then it follows that the job completes before d_i' as well.

The final parameter, release jitter, J_i, can decrease due to improvements in the system timer or because of reduced variability in other parts of the system. For example in a distributed system where a task (τ_j) is on another node but causes the release of τ_i, the jitter for τ_i is equal to the response time of τ_j [Burns 1994]. If τ_j has its response time reduced via an increase in processor speed then τ_i's release jitter will also reduce. We require sustainability over the jitter parameter to prevent such circumstances becoming a problem.

4. ANALYSIS FOR FIXED PRIORITY SCHEDULING

In this section we focus on fixed priority scheduling. In Table I, we summarize the sustainability of preemptive fixed-priority scheduling. The explanation for these results is as follows. With respect to the D_i parameter, sustainability follows from the transitivity of time: if a job would complete before its specified deadline and a new deadline greater than its specified one is assigned to it, then it follows that the job would complete before this new deadline as well. Sustainability of preemptive fixed-priority scheduling with respect to the C_i parameter is obvious; the two non-sustainability results are illustrated via examples.

We now focus on schedulability *tests* for fixed-priority preemptive scheduling – this is the most interesting case since such schedulability tests are very widely used in real-time systems design, implementation, and analysis. First, we show that not all fixed-priority schedulability tests that have been proposed are sustainable: in

Table I. Sustainability results concerning uniprocessor fixed-priority scheduling.

parameter	sustainable?
C_i	✓
D_i	✓
T_i	× (Example 2)
J_i	× (Example 2)

particular, we prove that the exact fixed-priority schedulability test of [Leung and Whitehead 1982] is not sustainable with respect to the jitter or offset parameters. Then, we focus on two common forms of analysis: *utilization-based analysis* [Liu and Layland 1973; Kuo and Mok 1991], and *Response Time Analysis* (RTA) [Joseph and Pandya 1986; Wellings et al. 1993]. We prove that both are sustainable with respect to all parameters, and that RTA remains sustainable even when additional non-preemptable resources must be shared among jobs of different tasks.

4.1 The Leung and Whitehead test

The exact fixed-priority schedulability test for periodic task systems proposed by Leung and Whitehead essentially consists of simulating system behavior over the time interval $[0,\ 2H + \max\{O_i\})$ (here, H denotes the *hyper-period* – the least common multiple of the periods – of the task system), and declaring the system schedulable if and only if no deadlines are missed.

Example 1 below illustrates that this test is not sustainable with respect to the jitter parameter. Specifically, this example demonstrates that a system that is deemed fixed-priority schedulable by the Leung and Whitehead test in the presence of release jitter may cease to be so when the jitter is reduced.

Example 1 Consider the periodic task system comprised of the two tasks τ_1 and τ_2. Both tasks have zero offset; task τ_1's parameters are $C_1 = 1$, $D_1 = T_1 = 2$, and $J_1 = 0.5$; and task τ_2's parameters are $C_2 = 1.5$, $D_2 = T_2 = 3$, and $J_2 = 0$. It may be verified that this system is schedulable if τ_1 is assigned the higher priority, and is correctly deemed to be so by the Leung and Whitehead test. However, if J_1 is reduced to zero then τ_2's first job misses its deadline.

We will see in Section 6 that the Leung and Whitehead test is not sustainable with respect to offset parameters, either. In fact if some of the tasks have non-zero offset parameters, Example 2 in Section 6 illustrates that fixed-priority scheduling (and consequently, the Leung and Whitehead test) is unsustainable with respect to the period parameter *even if the values of the offset parameters are maintained* – neither increased nor decreased.

Thus, it is evident that the Leung and Whitehead fixed-priority schedulability test, despite being an exact test, is not sustainable along many dimensions. This is one of the reasons (the computational complexity of the test – exponential in the representation of the task system – is another) that this test is not nearly as commonly used as the schedulability test we study in the remainder of this section: the *response time analysis* test.

4.2 Response Time Analysis (RTA)

We shall start by assuming independent tasks, no jitter, and deadline parameters no larger than periods (i.e., $D_i \leq T_i \forall i$). Suppose that we are given a specific priority assignment for such a task system. For all i, let $\mathbf{hp}(i)$ denote the set of tasks with higher priority tasks than τ_i. Suppose that all tasks generate a job at the same instant in time – such a time-instant is called a *critical instant* for the task system – and each τ_j generates subsequent jobs exactly T_j time units apart. (Such a job arrival sequence, starting with a critical instant, is sometimes referred to as a *critical arrival sequence*). It has been proved that if all deadlines are met for the critical arrival sequence, then all deadlines are also met for any other job arrival sequence that the task system may legally generate[3].

Let $A_i(t)$ denote the amount of execution that task τ_i is guaranteed over the time-interval $[0,t)$, if the critical arrival sequence of jobs occurs. The RTA methodology is based on the observation that

$$A_i(t) \geq t - \sum_{j \in \mathbf{hp}(i)} \left\lceil \frac{t}{T_j} \right\rceil C_j \,, \tag{1}$$

since the second term (the summation) in equation (1) represents the total amount of time over $[0,t)$ that tasks in $\mathbf{hp}(i)$ are executing, thereby denying execution to τ_i.

If $A_i(t) \geq C_i$, then it is guaranteed that τ_i's first job has completed by time-instant t. Hence, the worst-case response time for τ_i is given by the following smallest value of R_i that satisfies the following equation:

$$A_i(R_i) \geq C_i$$

$$\text{i.e., } \left(R_i \geq C_i + \sum_{j \in \mathbf{hp}(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \right). \tag{2}$$

The RTA schedulability-analysis tests consists of validating that the smallest R_i satisfying equation (2) above is $\leq D_i$, $\forall i$. This is an exact (i.e., sufficient and necessary) test if system semantics permit the possibility of a critical arrival sequence; else it is a sufficient test.

Theorem 1 *Response time analysis of fixed priority preemptive systems with independent tasks and zero jitter is sustainable with respect to execution requirements, relative deadlines and periods.*

Proof Suppose that a system is deemed schedulable; i.e., for all tasks τ_i the response-time R_i is no larger than the relative deadline D_i. Observe that $A_i(t)$ – the amount of execution available to task τ_i over $[0,t)$ – can only – *in*crease if job execution requirements decrease, and/or job periods increase. Hence if a value of R_i satisfying equation (2) was obtained for the specified system, it follows that further increasing $A_i(R_i)$ cannot deny τ_i's job C_i units of execution over $[0, R_i)$. Consequently, the RTA methodology is sustainable with respect to execution-requirements and periods.

---

[3]Strictly speaking, this is only true for systems of independent tasks in which all jobs are ready for execution immediately upon arrival. The analog arrival sequence for the case when tasks may exhibit jitter is derived in [Burns 1994]. Incorporating jitter into RTA is discussed in Section 4.5.

Changing deadlines of jobs of tasks other than τ_i's has no effect on τ_i's ability to meet its deadline. Since the "$\leq$" relationship is transitive in all models of time used in real-time scheduling theory, if $R_i \leq D_i$ and a new deadline D_i' is introduced, then it follows that $R_i \leq D_i$ as well. Hence, the RTA methodology is sustainable with respect to relative deadlines as well.

Under the assumptions adopted above – independent tasks, no jitter, and $D_i \leq T_i \forall i$ – it has been shown [Leung and Whitehead 1982] that the *Deadline Monotonic* (DM) priority assignment policy is optimal among all fixed priority assignments. In DM priority assignment, tasks are assigned priorities in inverse relationship to their deadline parameters (the shorter this deadline the higher the priority of the task - all jobs from the same task have the same priority). Although the above theorem demonstrates that an increase in any D_i parameter cannot jeopardies schedulability, after such a parameter change the priority-assignment may no longer correspond to DM. That is, there may be a "more" optimal priority ordering if a deadline is extended such that it is now longer than some other task's deadline. However, this does not effect sustainability since the system was schedulable even before the move to optimal – deadline monotonic – order.

Note that as any C_i parameter can be reduced by an arbitrary amount, sustainability covers the case of where a C_i value can be reduced to zero, i.e. the removal of a task from the system.

4.3 Utilization-based analysis

For systems comprised of independent periodic or sporadic tasks that all have their period parameters at least as large as their deadlines (i.e., $T_i \geq D_i \ \forall i$), it has been shown [Liu and Layland 1973] that the rate-monotonic (RM) priority assignment, which assigns higher priorities to tasks with smaller values of the period parameter, is an optimal fixed-priority assignment scheme. Utilization-based RM-schedulability analysis [Liu and Layland 1973] consists of determining whether the following inequality is satisfied:

$$\sum_{i=1}^{N} \left(\frac{C_i}{T_i}\right) \leq N(2^{1/N}-1)\,; \tag{3}$$

if it is, then the system is guaranteed to be RM-schedulable.

[Kuo and Mok 1991] provide a potentially superior utilization bound for task systems in which the task period parameters tend to be hamonically related. Let $\tilde{N}$ denote the number of *harmonic chains* is the task system; then a sufficient condition for a task system to be RM-schedulable is

$$\sum_{i=1}^{N} \left(\frac{C_i}{T_i}\right) \leq \tilde{N}(2^{1/\tilde{N}}-1)\,. \tag{4}$$

Consider, as an example, a system comprised of N=4 tasks with T_1=2, T_2=4, T_3=8, and T_4=16. By equation (3), the utilization bound for this task system is $4\times(2^{1/4}-1)$, which equals ≈ 0.757. However, since 2, 4, 8, and 16 comprise one harmonic chain, $\tilde{N}=1$ and the utilization bound as given by equation (4) is $1\times(2^1-1)$, which equals 1. Thus, the bound of equation (4) is superior to the bound of equation (3). In general,

since the number of harmonic chains in a system of N tasks is never more than N and may be less, the bound of equation (4) is superior to the bound of equation (3).

Observe, however, that increasing period parameters of tasks in a system may result in an increase or a decrease in the number of harmonic chains. Hence at first glance, it may appear that the utilization bound of equation (4) is not sustainable with respect to period parameters. Returning to our previous 4-task example, suppose that T_3 increases to 10; while 2, 4, and 16 continue to comprise a single harmonic chain, 10 is not a part of this chain and hence $\tilde{N}$ becomes equal to 2. The resulting utilization bound given by equation (4) is $2\times(2^{1/2}-1)$, which equals ≈ 0.83. The utilization bound thus fell from 1.00 to 0.83 when the period parameter of a task *in*creased, which would seem to violate sustainability.

In fact, however, it turns out that both utilization-based tests described above – the original one from [Liu and Layland 1973] and the one considering harmonic chains [Kuo and Mok 1991] – are in fact as sustainable as the RTA-based test. This is a consequence of Observation 1 below, which follows directly from analysis of the proofs of the utilization bounds in [Liu and Layland 1973; Kuo and Mok 1991].

Observation 1 *Any system deemed schedulable by the utilization-based tests in [Liu and Layland 1973; Kuo and Mok 1991] is also deemed schedulable by the RTA test of Section 4.2.*

Theorem 2 below follows from Theorem 1 and this observation.

Theorem 2 *The utilization-based RM-schedulability tests represented by equation (3) and equation (4) are sustainable with respect to execution requirements, deadlines, and periods.*

4.4 RTA: Incorporating task interaction and blocking

When tasks share non-preemptable serially reusable resources, a lower-priority job holding such a resource may delay ("block") the execution of some higher-priority job. Hence there is a need to incorporate a blocking term into the response-time analysis for such systems, and equation (1) can be generalized to account for such blocking as follows:

$$A_i(t) \geq t - B_i - \sum_{j \in \mathbf{hp}(i)} \left\lceil \frac{t}{T_j} \right\rceil C_j \,,$$

The blocking term B_i arises from the task interactions via shared objects implementing some form of concurrency control protocol such as a priority ceiling protocol [Sha et al. 1990; Rajkumar 1991]. The formula for calculating B_i is typically as follows. Let K be the number of critical sections (shared objects).

$$B_i = \max_{k=1}^{K} usage(k, i)c(k), \tag{5}$$

where *usage* is a 0/1 function: $usage(k, i) = 1$ if resource k is used by some task with index greater than i (equivalently, priority less than that of τ_i), and some task with index $\leq i$ (equivalently, priority greater than or equal to that of τ_i). Otherwise it gives the result 0; $c(k)$ is the worst-case execution time of the k critical section.

As in Section 4.2, the worst-case response time for τ_i is once again given by the smallest value of R_i that satisfies $A_i(R_i) \geq C_i$, i.e.,

$$R_i \geq C_i + B_i + \sum_{j \in \mathbf{hp}(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \,. \tag{6}$$

For the scheduling analysis implied by equation (6) to be sustainable it must be the case that, in addition to the full C parameters being bounded by their initially specified value, none of the $c(k)$ section must increase in size.

Observe that the blocking term is not effected by changes to relative deadlines and periods. Decreases in some $c(k)$ values may decrease B_i, but since B_i appears only in the RHS of equation (6) any R_i satisfying equation (6) will continue to do so subsequent to any such decrease in B_i. However, an extension of a deadline may lead to a rearrangement of priorities (to retain the optimal order) and this could lead to a task having a larger blocking term. Nevertheless, this increase could only be generated by a task that is now of lower priority but which was previously of higher priority. It follows that the increase in blocking is more than compensated for by a decrease in interference, and the task set thus remains schedulable.

From these considerations we conclude that *RTA incorporating blocking is sustainable with respect to execution requirements, relative deadlines, and periods.*

Note the blocking term is an upper bound. There is no attempt in the analysis framework to decide if this level of blocking will actually occur at run-time. For well structured preemptive systems, the blocking term is usually very small in comparison with the task's full execution time and hence the pessimism within equation (6) is acceptably small. However, attempts to provide exact analysis (sufficient and necessary) can lead to sacrificing sustainability – see Section 4.6.

4.5 RTA: Incorporating release jitter

The final parameter to be incorporated into the definition of sustainability is release jitter. The effect of jitter is that if each higher-priority task τ_j's first job in the critical arrival sequence executes with maximum jitter, then $\lceil (t+J_i)/T_j \rceil$ jobs of τ_j may get to execute in the interval $[0,t)$ – see [Burns 1994]. Equation (1) can be generalized to account for the effect of jitter as follows:

$$A_i(t) \geq t - \sum_{j \in \mathbf{hp}(i)} \left\lceil \frac{t+J_j}{T_j} \right\rceil C_j \,.$$

Once again, the worst-case response time for τ_i is equal to the smallest value of R_i that satisfies $A_i(R_i) \geq C_i$, i.e.,

$$R_i \geq C_i + B_i + \sum_{j \in \mathbf{hp}(i)} \left\lceil \frac{R_i+J_j}{T_j} \right\rceil C_j \,. \tag{7}$$

Again it is clear that for any given R_i decreasing the jitter for any higher-priority job will either decrease, or leave unchanged, the RHS of equation (7); hence, any R_i satisfying equation (7) will continue to do so subsequent to any such decrease in jitter. Putting this result together with the results in Sections 4.2-4.4, we can conclude

that

Theorem 3 *Response time analysis of fixed priority preemptive systems is sustainable.*

The optimal priority assignment for systems with jitter is (D-J) monotonic [Zuhily and Burns 2007]. A decrease in J, like an increase D, may impact on priority ordering, but again schedulability will be sustained as the optimal scheme must schedule any task set that is already schedulable by a different priority ordering.

4.6 Sufficient, necessary but not sustainable analysis

The above discussion has shown that in general RTA is sufficient and sustainable, but may not be necessary. Here we consider tests that attempt to be exact but which cannot be proved to be sustainable.

The first example is schedulability tests based on model checking. (eg [Guan et al. 2007]). Advocates of this approach argue that model checking can produce exact analysis. Typically they do this by exploring the real blocking that can occur in a system. This may be less than the upper bound given in equation (5). A system may therefore be deemed to be unschedulable by RTA but schedulable by a model checking test. Whilst this is true, the exponential state explosion associated with model checking makes this form of analysis totally impractical for realistic systems unless some form of simplification of the system model is undertaken – indeed, Fersman et al. note [Krcal and Yi 2004; Fersman et al. 2007] that the general problem is in fact undecidable. The most common simplification is to assure that each task's execution is exactly C (not less)[4].

This results in an exact test for these particular parameters but not a sustainable test. A small reduction in one C value may lead to a change in execution order that could significantly increase blocking. Consider the simple example illustrated in Figure 1. Here task X is released at time 0, task Y is released at time 4 and task Z at time 2. As X has the highest priority, then Y and then Z, Z does not start its execution until time 8 and the response time of Y is 4 (its execution time). Note that shaded area of Z represent the time during which Z executes with an inherited priority due to it accessing a shared object.

In Figure 2, task X completes early (smaller C). Now Z starts to execute before Y arrives, at time 3 it accesses the shared object with a priority higher than task Y. As a result Y's execution is postponed and its response time increases. With RTA, Y is

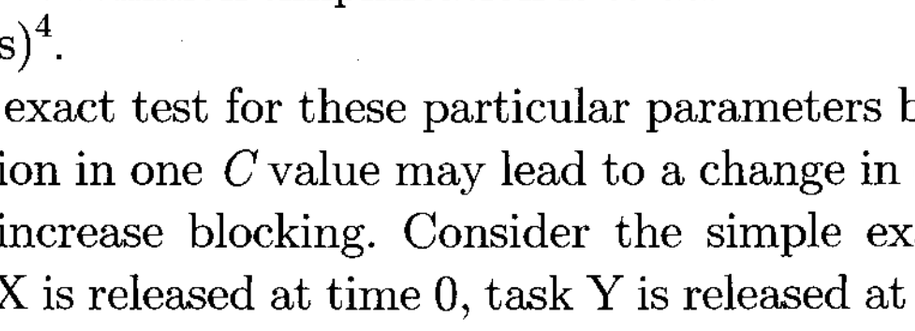


Figure 1. Example Execution - 1.

[4]An alternative simplification is to allow a short range for C, but to assume non-preemptive dispatching.

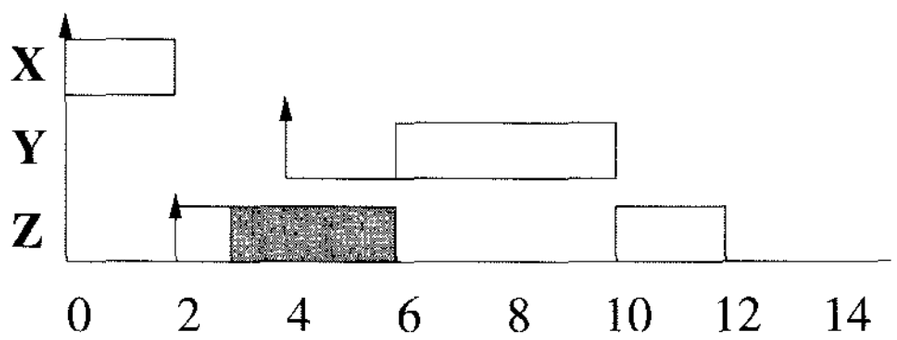


Figure 2. Example Execution - 2.

always assumed to have this maximum blocking and hance a reduction in C_X does not impact on schedulability.

The second example also comes from the problem of blocking but concerns the more sizeable blocking that can occur with non-preemptive systems. [Mok and Poon 2005] observe that exact analysis of such systems can lead to anomalies in that a system that is initially schedulable can become unschedulable when C values are reduced or T values increased. They proceed to analyse in detail the properties of these anomalies, but arguably what is actually required is a sustainable test for non-preemptive systems. Fortunately such tests exists [Bril et al. 2007]. A sufficient and sustainable test based upon response-time analysis is provided by [Davis et al. 2007]. The blocking term, to be used in the standard response time relation – equation (7) is

$$B_i = \max_{l \in \mathbf{lep}(i)} C_l \,, \tag{8}$$

where $\mathbf{lep}(i)$ is the set of lower or equal priority tasks (than τ_i).

As the task cannot be preempted once it has started, it is possible for a non-preemptive scheduler to succeed where a preemptive one would fault. However, in general non-preemption reduces schedulability as the blocking term is much larger. Where non-preemption is used in real systems it is normal practice to insert 'preemption points' into the code of long low priority tasks (making sure critical sections of the code are avoided!). This moves the dispatching policy to be closer to preemptive. It also allows the associated response time analysis to more closely model the actual behavior of this cooperative dispatching:

$$R_i^* \geq C_i - F_i + B_i + \sum_{j \in \mathbf{hp}(i)} \left\lceil \frac{R_i^*}{T_j} \right\rceil C_j \,, \tag{9}$$

$$B_i = \max_{l \in \mathbf{lep}(i)} c(l), \tag{10}$$

where $c(l)$ is the longest non-preemptive segment of code in task τ_l and F_i is the length of the final segment of the code of τ_i. Note $B_i \geq F_i$.

Once the smallest value of R_i^* has been calculated the true response time is obtained by adding back in the length of the final section:

$$R_i = R_i^* + F_i \tag{11}$$

We conclude this section by noting:

Theorem 4 *Response time analysis of fixed priority non-preemptive systems is sustainable.*

Proof is again by inspection of the analysis equations. Note this form of RTA is not sustainable in terms of the inclusion of extra preemption points. If this property is important (i.e. the addition of extra preemption points to a schedulable system should not jeopardize schedulability) then the analysis should not take advantage of the lack of preemption in the final phase of execution.

4.7 Other priority allocation algorithms

Although the deadline monotonic assignment (or D-J monotonic ordering if these is release jitter in the system) is the optimal scheme from a temporal point of view there are other allocation algorithms. For example, setting priority according to computation time (small C_i, higher priority) is a scheme that reduces average response time, and value density (V_i/C_i ordering, where V_i is a parameter denoting the amount of *value* obtained by completing a job of the task before its deadline) maximizes output quality (if deadlines are ignored). For both of these cases a reduction in C will retain schedulability for these schemes as long as the same priority ordering is retained. A reordering due to a decrease in C_i may break schedulability.

5. ANALYSIS FOR EDF SCHEDULING

In this section we focus on dynamic priority scheduling. In dynamic priority scheduling, the priority assigned to different jobs of the same task may be different, and indeed any individual job may change its priority arbitrarily often between its arrival instant and its completion time/deadline. Examples of dynamic priority scheduling policies include Least Laxity [Mok 1983] and Earliest Deadline First (EDF) [Liu and Layland 1973; Dertouzos 1974]; of these, EDF is the most commonly used in real-time system design and implementation. EDF prioritizes active jobs according to their deadlines: the earlier the deadline, the higher the priority, with ties broken arbitrarily. Since the run-time behavior of an EDF-scheduled system is determined by the deadlines of the jobs, an increase in the deadline parameter may change the relative ordering of execution of the jobs; as a consequence, it is not immediately obvious whether the EDF scheduling policy will prove sustainable or not.

Fortunately, preemptive EDF scheduling algorithms turn out to be remarkably sustainable. We can easily prove such sustainability indirectly by exploiting the well-known *optimality* property of preemptive uniprocessor EDF: that EDF is an optimal scheduling algorithm upon preemptive uniprocessors, in the sense that if a collection of jobs can be scheduled by any scheduling algorithm to meet all deadlines then EDF also schedules this collection of jobs to meet all deadlines.

The following theorem is useful in proving the sustainability of the EDF scheduling policy.

Theorem 5 *Let* $I = \{(r_i, c_i, d_i)\}_{i \geq 1}$ *denote a collection of independent jobs, each characterized by a ready time* r_i*, an exact execution requirement* c_i*, and an absolute deadline* d_i*. If* I *is EDF-schedulable on a preemptive uniprocessor, then so is* $I' = \{(r_i', c_i', d_i')\}_{i \geq 1}$*, provided that* $r_i' \leq r_i$*,* $c_i' \leq c_i$*, and* $d_i' \geq d_i$*, for all* $i \geq 1$*.*

Proof Observe that the preemptive EDF schedule of I is also a schedule for I' (although this is not necessarily the schedule that would have been generated by

EDF on I'). Hence, this EDF schedule of I bears witness to the fact that I' can be scheduled to meet all deadlines. Since EDF is optimal on preemptive uniprocessors, it is therefore guaranteed to successfully schedule I' to meet all deadlines.

Theorem 5 above directly leads to a sustainability result with respect to the *jitter*, *execution requirement*, and *relative-deadline* parameters for EDF-scheduling:

Corollary 1 *Any EDF scheduling policy (and consequently, any sufficient EDF-schedulability test) for periodic or sporadic task systems on preemptive uniprocessors is sustainable with respect to jitter, execution requirement, and relative deadlines.*

It remains to consider sustainability issues with respect to the period/inter-arrival separation parameter.

Theorem 6 *Any EDF scheduling policy (and consequently, any sufficient EDF-schedulability test) for sporadic task systems is sustainable with respect to the inter-arrival separation parameter.*

Proof This follows from the observation that any job arrival pattern resulting from increasing the inter-arrival separation parameter[s] are among the legal job arrival patterns of the original system.

For any periodic task system, let us define the ***derived*** *sporadic task system* to be the sporadic task system comprised of tasks with identical jitter, relative-deadline, execution requirement, and period parameters. The following results from prior research relate the EDF-schedulability of periodic task systems to the EDF-schedulability of sporadic task systems (recall that a *zero-offset* periodic task system is one in which the offset parameter O_i of all tasks is the same).

Theorem 7 (from [Baruah et al. 2008])

1. A zero-offset periodic task system is EDF-schedulable if and only if its derived sporadic task system is EDF-schedulable.

2. A periodic task system is EDF-schedulable if its derived sporadic task system is EDF-schedulable.

We now state our results concerning sustainability of EDF-schedulability tests for periodic task systems with respect to the period parameter.

Theorem 8 *Any EDF scheduling policy (and consequently, any sufficient EDF-schedulability test) for zero-offset periodic task systems is sustainable with respect to the period parameter.*

Proof Follows from Theorem 7 (1), and Theorem 6, since the job arrival pattern resulting from increasing period parameter[s] of a zero-offset periodic task system is among the legal job arrival patterns of the derived sporadic task system.

Theorem 9 *Any sufficient EDF-schedulability test applied to the derived sporadic task system is a sustainable EDF-schedulability test for periodic task systems with respect to the period parameter.*

Proof Follows from Theorem 7 (2) and Theorem 6. Observe, however, that the test may prove to be more pessimistic for the periodic task system than it is for the derived sporadic task system; in particular, using an *exact* EDF-schedulability test for the derived sporadic task system would result in a sufficient (but not exact) test for the periodic task system.

We now discuss the sustainability of specific commonly-used EDF-schedulability tests. These tests come in two flavors: *utilization-based*, and based upon *processor demand criteria.*

Utilization based test. The utilization-based test deems task system $\Gamma = \{\tau_1, \tau_2, \cdots, \tau_N\}$ to be EDF-schedulable upon a unit-capacity preemptive processor if the following condition is satisfied:

$$\sum_{i=1}^{N} \left(\frac{C_i}{T_i}\right) \leq 1 \,. \tag{12}$$

The utilization-based test is exact for systems comprised entirely of tasks that each have **(i)** zero jitter, and **(ii)** deadline parameters no smaller than period parameters; if some tasks τ_i have $D_i < T_i$ or $J_i > 0$, the utilization-based test can be shown to not be sufficient for ensuring EDF-schedulability.

The processor demand criteria tests. For any task τ_i and any non-negative number t, let us define the *processor demand* $\mathrm{DBF}_i(t)$ to be the largest possible cumulative execution requirement by jobs of task τ_i, that have both their ready times and their deadlines within any time interval of length t.

The following observation will be used later in this section:

Observation 2 *The processor demand* $\mathrm{DBF}_i(t)$ *is monotonically non-increasing with respect to increasing* D_i *and* T_i*, and decreasing* C_i *and* J_i*.* (These changes may be online.)

By the processor demand criteria EDF-schedulability test, a task system τ is deemed to be EDF-schedulable if

$$\forall t: t \geq 0: \left[\left(\sum_{i=1}^{N} \mathrm{DBF}_i(t)\right) \leq t\right]. \tag{13}$$

The processor demand criterion EDF-schedulability test algorithm is exact if there may be a critical instant in which all tasks in the system generate a job simultaneously; otherwise it is sufficient but not necessary. Hence, it is exact for sporadic and zero-offset periodic task systems, and sufficient for periodic task systems with arbitrary offsets.

Theorem 10 *Both the utilization-based and the processor demand criterion EDF schedulability tests are sustainable.*

Proof Follows from Corollary 1 and Theorem 9.

5.1 Task interaction and blocking

When tasks interact through the sharing of non-preemptable serially reusable resources, *blocking* of higher-priority (i.e., earlier deadline) jobs by lower-priority (later-deadline) jobs may occur. The *Stack Resource Policy* (SRP) [Baker 1991] is typically used in EDF-scheduled systems to arbitrate access to such resources; when used in conjunction with EDF, the rules used by SRP to determine execution rights are summarized in Figure 3.

In [Baker 1991; Pellizzoni and Lipari 2005; Baruah 2006], sufficient conditions for

(Tasks are assumed to be indexed according to non-decreasing relative deadlines: $D_i \leq D_{i+1}$ for all i.)

1. Each task τ_i is defined to have *level* i.
2. Each resource is statically assigned a *ceiling*. The ceiling of a resource is set equal to the level of the minimum-level (lowest-indexed) task that may access it.
3. A *system ceiling* is computed each instant during run-time. This is set equal to the minimum ceiling of any resource that is currently being held by some job.
4. At any instant in time, a job may execute only if it is the earliest-deadline job, and the level of the task that generated it is strictly less than the system ceiling.

Figure 3. EDF + SRP

a task system to be schedulable under the EDF+SRP scheduling framework are derived. Recall the definition of the blocking term parameter - equation (5). Let us define a *blocking function* $B(t)$, that, for $t \geq 0$, equals the largest blocking term parameter of any task with relative deadline no larger than t:

$$B(t) \equiv \max\{B_i \mid D_i \leq t\} . \tag{14}$$

It was shown in [Baruah 2006] that

$$\forall t \geq 0: \left(B(t) + \sum_{i=1}^{N} \mathrm{DBF}_i(t) \right) \leq t . \tag{15}$$

is sufficient for guaranteeing that all deadlines are met under EDF+SRP. We will refer to this schedulability test as the **enhanced processor demand test.**

Theorem 11 *The enhanced processor demand test for EDF+SRP schedulability analysis is sustainable with respect to the deadline, execution-time, period, and jitter parameters.*

Proof From Observation 2, it follows that the summation term in inequality (15) can only *decrease* with increasing deadlines and periods, and decreasing execution-times and jitter. Furthermore, $B(t)$ does not change with changes to period, and can only decrease with decreasing execution-time. It therefore follows that the enhanced processor demand test is sustainable with respect to execution requirement and period.

It remains to determine whether inequality (15) continues to hold if deadlines increase, or jitters decrease. The concern here is with respect to the blocking term ($B(t)$) – in order for one job to block another under EDF+SRP scheduling, it is necessary that the blocking job have a ready-time earlier than, and a deadline later than, the blocked job. The following scenario illustrates this with respect to increasing deadlines:

Suppose that jobs $\mathcal{J}_1$ and $\mathcal{J}_2$ share some non-preemptable resource. Job $\mathcal{J}_1$ has an earlier deadline than $\mathcal{J}_2$, and $\mathcal{J}_2$ has an earlier deadline than $\mathcal{J}_3$, in the original system (i.e., as per system specifications). Since a job may only be blocked by lower-priority (later-deadline) jobs, neither $\mathcal{J}_1$ nor $\mathcal{J}_2$ can possibly block $\mathcal{J}_3$.

However, suppose that $\mathcal{J}_2$'s deadline increases *during run-time to later than $\mathcal{J}_3$*'s. *As a consequence, $\mathcal{J}_3$ can now be blocked by $\mathcal{J}_2$.*

A similar scenario can be envisioned in which blocking is introduced when a task's jitter is decreased, resulting in some job's ready-time being moved forward.

These scenarios lead us to the conclusion that ***increasing deadlines and/or reducing jitter may introduce blocking where earlier there was none.***

However, a moment's reflection should convince us that the introduction of such blocking cannot result in missed deadlines if the original system passed the enhanced processor demand test. This is because all such newly-introduced blocking was already accounted for in the analysis that resulted in inequality (15). Specifically, *every* task's execution was accounted for in the LHS of inequality (15), either as a potential blocker or as a contributor to the summation term. Increasing deadlines can change a task's contribution from the summation term to a role as a potential blocker; however, such change can only decrease (or leave unchanged) the total contribution of the task to the LHS of inequality (15) for any t. (Thus in the scenario described above, job J_2's entire worst-case execution requirement, and not just the length of its blocking critical section, would have been included in the summation term on the LHS of inequality (15.)).

5.2 Non-preemptive scheduling

A non-preemptive system of independent periodic or sporadic tasks can be modelled as a preemptive system of the same tasks, with an additional non-preemptable shared resource that is needed by all tasks' jobs for the entire duration of their executions. In such a preemptive resource-sharing system, it is straightforward to observe that the blocking terms can be computed as follows:

$$B_i = \max_{j=i+1}^{n}\{C_j\}. \tag{16}$$

Theorem 12 follows.

Theorem 12 *The enhanced processor demand test is a sufficient sustainable test for non-preemptive EDF-schedulability analysis of periodic and sporadic task systems, with the B_i's (the blocking terms) defined as in equation (16) above.*

6. ADDITIONAL TASK PARAMETERS: *OFFSETS* AND *BEST-CASE EXECUTION TIMES*

As stated in Section 2, an additional parameter that is sometimes defined for periodic tasks is the **offset** parameter O_i, denoting the arrival time of the first job of τ_i. There are two common reasons for using offsets in the system model:

- they represent features of the system being modeled, for example precedence relations,
- they are introduced to enhance the schedulability of the system.

Examples of schedulability tests that take account of offsets includes the exact fixed-priority-test of [Leung and Whitehead 1982] that we briefly discussed in Section 4, and the exact EDF-schedulability test proposed by [Leung and Merrili 1980]. Both

these tests essentially simulate system behavior over the time interval $[0, 2H + \max\{O_i\})$, and declare the system schedulable if no deadlines are missed (recall that H denotes the *hyper-period* - the least common multiple of the period parameters).

Unfortunately, schedulability tests that use the offset parameter are not sustainable: A system may be schedulable with a particular set of offsets but become unschedulable with either a decrease or increase in any of these values. Furthermore, *periodic task systems with non-zero offsets that are deemed to be schedulable may miss deadlines if task periods are increased.* This is illustrated in the following example.

Example 2 Consider the periodic task system comprised of the two tasks τ_1 and τ_2. Task τ_1's parameters are $O_1 = 0$, $C_1 = D_1 = 1$, $T_1 = 2$, and task τ_2's parameters are $O_2 = 1$, $C_2 = D_2 = 1$, $T_2 = 2$. Both tasks have zero release jitter ($J_1 = J_2 = 0$). It may be verified that this system is deemed schedulable by both the fixed-priority test [Leung and Whitehead 1982] and the EDF test [Leung and Merrill 1980]: tasks τ_1 and τ_2 execute in alternate time-slots.

However, if T_2 is now increased from 2 to 3, both tasks will generate jobs with execution-requirement one each at time-instant 4; one of these jobs necessarily misses its deadline at time-instant 5.

Thus the presence of non-zero offsets may compromise a system's schedulability with respect to not just changing offsets, but increasing periods as well. This effect on the exact schedulability tests of [Leung and Whitehead 1982; Leung and Merrill 1980] is formalized in the following theorem:

Theorem 13 *The exact fixed-priority and EDF-schedulability tests for periodic task systems with offsets presented in [Leung and Whitehead 1982; Leung and Merrill 1980] are not sustainable with respect to the period parameter.*

Thus offsets seem inherently incompatible with sustainability: introducing offsets during system design time in order to enhance feasibility risks rendering the system unsustainable (even if the values of the offset parameters are kept constant). The only safe way to analyse offsets seems to be to ignore them; i.e. assume that they are all equal to zero. (This is of course an unreasonable approach if the whole point of using offsets was to obtain schedulability - if that be the case, then it seems that sustainability is not an obtainable goal.)

An additional parameter that is sometimes defined for periodic and sporadic tasks is the **best-case**[5] **execution time** (BCET) of either the entire job, or a portion thereof. These parameters are used, for example, in [Lipari and Buttazzo 2000] to determine bounds on the maximum amount by which lower-priority jobs may *block* higher-priority ones when non-preemptable resources must be shared between jobs. Best-case execution time parameters, too, seem inherently incompatible with sustainability: feasible systems may become unfeasible if not just the best-case execution time parameters change, but also if other parameters such as the deadline change "for the better." As with offsets, it seems that the only safe way to analyse systems with best-case execution times is to ignore them (i.e., assume that all these parameters are equal to zero).

[5]The term "best" is used in this context as a synonym for "minimum".

7. SERVERS

There is currently much interest in using servers (virtualb resources) and hierarchical scheduling to partition applications on shared processing resources [Kuo and Li 1998; Bernat and Burns 2002; Davis and Burns 2005]. With all the common servers (e.g. Periodic, Deferable, Bandwidth Preserving, Sporadic, CBS, etc [Lehoczky et al. 1987; Sprunt et al. 1989; Bernat and Burns 1999; Abeni and Buttazzo 1998]) there are two useful, if somewhat apposing sustainability properties.

1. An increase in the capacity of any server will sustain the schedulability of any task serviced by that server.

2. A decrease in the capacity of any server will sustain the schedulability of any task serviced by any other server.

Proof of these properties can be obtained by examining the associated scheduling equations. This is straightforward and is not included in this paper. The usefulness of these observations comes from the application of run-time acceptance tests in open systems. The computational overhead of the necessary on-line schedulability tests is significantly reduced by taking into account these properties.

8. CONCLUSIONS

Schedulability tests are currently characterized as being *necessary* and/or *sufficient*. This paper has introduced an additional characteristic: *sustainability*. Sustainable schedulability tests ensure that a system that has been successfully verified will meet all its deadlines at run-time even if its operating parameters change for the better during system run-time. The parameters of interest for a periodic or sporadic task are its execution time and release jitter (that can be reduced) and its deadline and period (that can be increased). It seems difficult to obtain schedulability tests that are necessary, sufficient, and sustainable for non-trivial task models; this paper has argued that from an engineering standpoint, sufficient and sustainable tests are more useful than the classic sufficient and necessary tests (even more so if these exact tests are infeasible for sizeable problems). EDF and Fixed Priority scheduling tests have been investigated for their sustainability: the evidence indicates that although some exact tests are not sustainable, response-time analysis and processor demand criteria tests do indeed have the important property of sustainability. Also, it seems that schedulability tests that do not ignore knowledge of task offsets and BCET estimates tend not to be sustainable, even with respect to the parameters other than offsets and BCET's.

Future work will look to extend the parameters over which the notion of sustainability can be applied, and to consider multiprocessor implementations. Recently, Buttazzo [Buttazzo 2006] has identified sources of potentially anomalous behavior in real-time systems executing upon variable-speed processors. On multiprocessor systems, a number of anomalies have been know for many year [Graham 1969]. [Ha and Liu 1994] have defined and studied a property of scheduling algorithms that they call "predictability"; informally, a scheduling algorithm is predictable if any task system that is scheduled by it will continue to meet all deadlines if some jobs arrive earlier, or have later deadlines.

ACKNOWLEDGEMENTS

Supported in part by EU projects FRESCOR and ARTIST2, and NSF Grant Nos. CNS-0541056, and CCR-0615197, ARO Grant No. W911NF-06-1-0425, and funding from the Intel Corporation.

REFERENCES

ABENI, L. AND BUTTAZZO, G. 1998. Integrating multimedia applications in hard realtime systems. In *Proceedings of the Real-Time Systems Symposium*. IEEE Computer Society Press, Madrid, Spain, 3–13. December.

BAKER, T. P. 1991. Stack-based scheduling of real-time processes. *Real-Time Systems: The International Journal of Time-Critical Computing 3*.

BARUAH, S., MOK, A., AND ROSIER, L. 1990. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Proceedings of the 11th Real-Time Systems Symposium*. IEEE Computer Society Press, Orlando, Florida, 182–190.

BARUAH, S. 2006. Resource sharing in EDF-scheduled systems: A closer look. In *Proceedings of the IEEE Real-time Systems Symposium*. IEEE Computer Society Press, Rio de Janeiro, 379–387. December.

BARUAH, S. AND BURNS, A. 2006. Sustainable scheduling analysis. In *Proceedings of the IEEE Real-time Systems Symposium*. IEEE Computer Society Press, Rio de Janeiro, 159–168. December.

BERNAT, G. AND BURNS, A. 1997. Combining (n;m)-hard deadlines and dual priority scheduling. In *Proceedings of the IEEE Real-Time Systems Symposium*, 46–57.

BERNAT, G. AND BURNS, A. 1999. New results on xed priority aperiodic servers. In *Proceedings of the IEEE Real-Time Systems Symposium*, 68–78.

BERNAT, G. AND BURNS, A. 2002. Multiple servers and capacity sharing for implementing exible scheduling. *Real- Time Systems 22*, 49–75.

BRIL, R. J., LUKKIEN, J. J., AND VERHAEGH, W. F. J. 2007. Worst-case response time analysis of realtime tasks under xed-priority scheduling with deferred preemption revisited. In *Proceedings of the 19th Euromicro Conference on Real-Time Systems (ECRTS 07)*. IEEE Computer Society, 269–279.

BURNS, A. 1994. Preemptive priority based scheduling: An appropriate engineering approach. *Advances in Real-Time Systems*, 225–248. editor: Son, S. H., Prentice-Hall.

BUTTAZZO, G. 2006. Achieving scalability in real-time systems. *IEEE Computer*, 54–59. May.

DAVIS, R. AND BURNS, A. 2005. Hierarchical xed priority preemptive scheduling. In *IEEE Real-Time Systems Symposium*, 389–398.

DAVIS, R. AND BURNS, A. 2007. Robust priority assignment for xed priority real-time systems. In *IEEE Real-Time Systems Symposium*.

DERTOUZOS, M. 1974. Control robotics: the procedural control of physical processors. In *Proceedings of the IFIP Congress*, 807–813.

FERSMAN, E., KRCAL, P., PETTERSSON, P., AND YI, W. 2007. Task automata: Schedulability, decidability and undecidability. *International Journal of Information and Computation 205*, 8, 1149–1172. August.

GRAHAM, R. L. 1969. Bounds on multiprocessing timing anomalies. *SIAM J. Appl. Math. 12*, 2, 416–429.

GUAN, N., GU, Z., DENG, Q., GAO, S., AND YU, G. 2007. Exact schedulability analysis for static priority global multi-processor scheduling using model-checking. In *Software Technologies for Embedded and Ubiquitous Systems, LNCS 4761*, 263–272.

HA, R. AND LIU, J. W. S. 1994. Validating timing constraints in multiprocessor and distributed real-time systems. In *Proceedings of the 14th IEEE International Conference on Distributed Computing Systems*. IEEE Computer Society Press, Los Alamitos. June.

JOSEPH, M. AND PANDYA, P. 1986. Finding response times in a real-time system. *The Computer*

Journal 29, 5, 390–395. October.

KRCAL, P. AND YI, W. 2004. Decidable and undecidable problems in scheduling analysis using timed automata. In *Proceedings of the 10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2004), LNCS 2988*, 236–250. editors: Jenson, K. and Podelski, A., Springer-Verlag.

KUO, T.-W. AND LI, C.-H. 1998. A xed priority driven open environment for real-time applications. In *IEEE Real-Time Systems Symposium.*

KUO, T.-W. AND MOK, A. K. 1991. Load adjustment in adaptive real-time systems. In *Proceedings of the IEEE Real-Time Systems Symposium*, 160–171.

LEHOCZKY, J., SHA, L., AND STRONIDER, J. 1987. Enhanced aperiodic responsiveness in hard realtime environments. In *Proceedings of the Real-Time Systems Symposium.* IEEE, San Jose, CA, 261–270. December.

LEUNG, J. AND MERRILL, M. 1980. A note on the preemptive scheduling of periodic, real-time tasks. *Information Processing Letters 11*, 115–118.

LEUNG, J. AND WHITEHEAD, J. 1982. On the complexity of xed-priority scheduling of periodic, real-time tasks. *Performance Evaluation 2*, 237–250.

LIPARI, G. AND BUTTAZZO, G. 2000. Schedulability analysis of periodic and aperiodic tasks with resource constraints. *Journal Of Systems Architecture 46*, 4, 327–338.

LIU, C. AND LAYLAND, J. 1973. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM 20*, 1, 46–61.

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

MOK, A. K. AND POON, W.-C. 2005. Non-preemptive robustness under reduced system load. In *Proceedings of the IEEE Real-Time Systems Symposium*, 200–209.

PELLIZZONI, R. AND LIPARI, G. 2005. Feasibility analysis of real-time periodic tasks with offsets. *Real-Time Systems: The International Journal of Time-Critical Computing 30*, 1.2 (May), 105–128.

RAJKUMAR, R. 1991. *Synchronization In Real-Time Systems. A Priority Inheritance Approach.* Kluwer Academic Publishers, Boston.

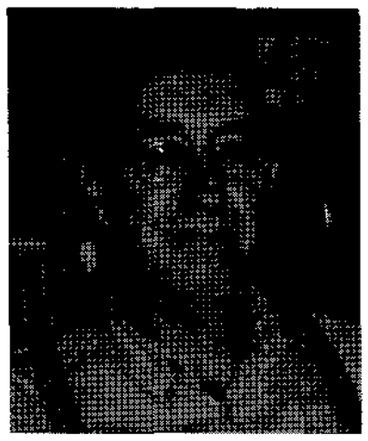
DAVIS, R. I., BURNS, A., BRIL, R. J., AND LUKKIEN, J. J. 2007. Controller area network (CAN) schedulability analysis: Refuted, revisited and revised. *Real-Time Systems 35*, 3, 239–272.

SHA, L., RAJKUMAR, R., AND LEHOCZKY, J. P. 1990. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Transactions on Computers 39*, 9, 1175–1185.

SPRUNT, B., SHA, L., AND LEHOCZKY, J. P. 1989. Aperiodic task scheduling for hard real-time systems. *Real-Time Systems 1*, 27–69.

WELLINGS, A., RICHARDSON, M., BURNS, A., AUDSLEY, N., AND TINDELL, K. 1993. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal 8*, 284–292.

ZUHILY, A. AND BURNS, A. 2007. Optimal (d- j)-monotonic priority assignment. *Information Processing Letters 103*, 6, 247–250.



Alan Burns is a Professor of Real-Time Systems in the Department of Computer Science, University of York, U.K. He graduated in Mathematics from the University of Sheffield in 1974, undertook his PhD at the University of York before taking up a tenured post at the University of Bradford. He joined the University of York again in 1990 and was promoted to a personal chair in 1994. Together with Professor Andy Wellings he heads the Real-Time Systems Research Group at the university; a group that has currently has over 30 members (faculty, Postdocs and PhD students). He has served as Chair of the IEEE Technical Committee on Real-Time and was awarded, in 2006, the IEEE Technical and Leadership Achievement Award. He has published widely (over 350 papers and articles, and 10 books) and works on a number of research areas within the real-time field.

Sanjoy Baruah is a professor in the Department of Computer Science at the University of North Carolina at Chapel Hill. He received his Ph.D. from the University of Texas at Austin in 1993. His research and teaching interests are in scheduling theory, real-time and safety-critical system design, and resource-allocation and sharing in distributed computing environments.