Sustainable Scheduling Analysis

Sanjoy Baruah*
The University of North Carolina
at Chapel Hill

Alan Burns
Department of Computer Science
University of York, UK

Abstract

A schedulability test is defined to be sustainable *if any task system deemed schedulable by the test 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 good engineering practice to use sustainable tests if possible, and classify common uniprocessor schedulability tests according to whether they are sustainable or not.*

1. Introduction

The notion of *schedulability* is well understood within the real time systems community. A system is schedulable with respect to a specified scheduling policy if it will meet all its timing requirements when executed on its target platform with that scheduling policy. The system is modelled as being comprised of a number of concurrent tasks (or threads) that are characterised by a small number of parameters. 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.

Although the notion of schedulability is clear the interpretation of the system's parameters is not. Specifically, are these parameters assumed to be exact or upper/lower bounds on the values that can be taken at run-time? In this paper we develop the notion of **sustainability** to formalize the expectation that a system determined to be schedulable should remain schedulable when its real behaviour is 'better' than worst-case.

In order to formally explain this concept, we need to first define what we mean by a real-time system. Let us 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, and sporadic models mandate a minimum temporal separation between the arrivals of successive jobs of a task.
We are now ready to define the concept of sustainability.

Definition 1 *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 job[s] are changed in any, some, or all of the following ways:* (i) *decreased execution requirements;* (ii) *later arrival times;* (iii) *smaller jitter; and* (iv) *larger relative deadlines.*

Intuitively, sustainability requires that schedulability be preserved in situations in which it should be "easier" to ensure schedulability[1]. This is the opposite property to that of *robustness*: a robust system retains schedulability even when it operates beyond the worst-case assumptions used in its schedulability test, e.g., when jobs arrive earlier than expected, or have greater execution requirement than permitted. Clearly a system can never be fully robust - at some point the system will become so 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.

*Supported in part by the National Science Foundation (Grant Nos. CCR-0408996 and CCR-0541056).

[1]Mok and Poon [16] use the term *robustness* to denote 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.

Summary of results. At first glance, it may appear that the concept of sustainability is well-understood and is incorporated into current schedulability tests. However, we demonstrate in this paper that this is a misconception: there are many issues concerning sustainability that have not been explored. 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. Most preemptive schedulability tests turn out to indeed be sustainable with respect to the execution-requirement parameter, but many common schedulability tests are not sustainable with respect to the remaining parameters. Indeed, we demonstrate that several well-known schedulability tests are not sustainable (see, e.g., Theorem 11). 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. At a higher level, we attempt to draw larger lessons concerning which properties of schedulability tests tend to render them sustainable or not.

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 usually more important than sufficient and necessary. The analysis of fixed priority systems is covered in Section 4, and dynamic-priority analysis in Section 5. Factors that tend to make it difficult to achieve sustainable schedulability analysis are identified in Section 6; in particular, it is noted that task offsets, and best-case execution-time estimates, are rather brittle with regard to sustainability.

2. System Model

We focus our attention in this paper to the uniprocessor scheduling of periodic and sporadic task systems. A real-time system, $\mathcal{A}$, is assumed to consist of N tasks each of which gives rise to a series of jobs that are to be executed on a single processor. 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 τ_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.
- 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.

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 most hard-real-time systems. A test can also be labelled 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 intractable. From an engineering point of view, a tractable test with low pessimism is ideal.

Observe that declaring a schedulability test to be sustainable represents a stronger claim than simply that a task system deemed schedulable by the test would remain schedulable with "better" parameters (e.g., with larger periods or relative deadlines, or with smaller execution-requirements or jitters). ***A sustainable system must continue to meet all deadlines even if the parameter changes occur "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. (We are not requiring that this exact execution requirement be known beforehand – it may only become known by actually executing the job to completion.)

A schedulability test 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 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.)
- Minimum 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 BCET's 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. The presence of such shared resources gives rise to *blocking*, and schedulability tests for such systems must take blocking into account. The sustainability of such schedulability tests is considered in Sections 4.4 and 5.1.

3. Motivation

Concentrating first on execution time, it is clear from all forms of WCET analysis that the C_i parameters are upper bounds. These may in itself overestimate the worst-case situations (due to difficulties in modelling 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 sustainable tests are 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].

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. 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 behaviour 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 behaviour is 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 since this does not occur when they arrive with maximum frequency [6]. 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.

Sustainability with respect to the deadline parameter is a significant issue only with respect to those scheduling policies, such as EDF (and associated resource-sharing protocols such as the Stack Resource Policy) 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 job (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 [7]. 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 FP Scheduling

In this section we focus on fixed priority scheduling. First, we show that not all fixed-priority schedulability tests are sustainable: in particular, we prove that the exact fixed-priority schedulability test of Leung and Whitehead [14] is

[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 behaviour. 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).

not sustainable with respect to the jitter or offset parameters. Then, we focus on two common forms of analysis: *utilization-based analysis* [15, 12], and *Response Time Analysis* (RTA) [11, 20]. 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 behaviour 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, by demonstrating 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 deemed fixed-priority scheduled by the Leung and Whitehead test if τ_1 is assigned the higher priority; 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 the Leung and Whitehead test is unsustainable with respect to the $\underline{\text{period}}$ 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 next: 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 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 Expression (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 smallest value of R_i that satisfies the following equation:

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

The RTA schedulability-analysis test 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.

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 "$\leq$"

[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 [7]. Incorporating jitter into RTA is discussed in Section 4.5.

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 [14] 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 smaller the deadline the higher the priority of the task).

Although the above theorem demonstrates that an increase in any D_i parameter cannot jeopardise 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 for the system.

4.3. Utilization-based analysis

For systems comprised of independent periodic or sporadic tasks that all have their deadline parameters at least as large as their deadlines (i.e., $D_i \geq T_i \; \forall i$), it has been shown [15] 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 [15] 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 so, then the system is guaranteed to be RM-schedulable.

Kuo and Mok [12] provide a potentially superior utilization bound for task systems in which the task period parameters tend to be harmonically 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 that

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

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 an improvement on 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.

However, it turns out that both utilization-based tests described above – the original one from [15] and the one considering harmonic chains [12] – 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 [15, 12].

Observation 1 *Any system deemed schedulable by the utilization-based tests in [15, 12] 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 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 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[19, 18]. 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 priority less than that of τ_i's, and some task with priority greater than or equal to that of τ_i's. Otherwise it gives the result 0; $c(k)$ is the worst-case execution time of the k'th 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.,

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

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. From this 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. We shall illustrate in section 4.6 that attempts to provide exact analysis (sufficient and necessary) can lead to sacrificing sustainability.

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 $\lceil (t+J_j)/T_j \rceil)$ jobs of τ_j may get to execute in the interval $[0, t)$ — see [7]. 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.,

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

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 so 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 [8, 21]. 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 sustainable.

The first example is schedulability tests based on model checking (eg [5, 2]). 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 experimental 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. 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.

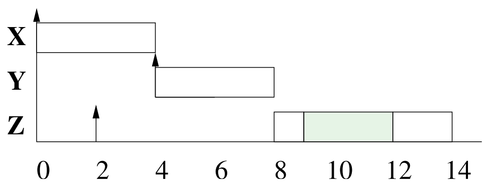

Figure 1. Example Execution - 1

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 in-

[4]An alternative simplification is to allow a short range for C, but to assume non-preemptive dispatching[1].

creases. With RTA, Y is always assumed to have this maximum blocking and hance a reduction in C_X does not impact on schedulability.

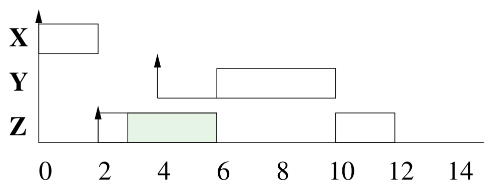

Figure 2. Example Execution - 2

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 [16] 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 a test exists [7] and this is sustainable.

5. Analysis for EDF Scheduling

The Earliest Deadline First scheduling algorithm (EDF) prioritizes active jobs according to their deadlines: the earlier the deadline, the higher the priority, with ties broken arbitrarily. Since the run-time behaviour 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. Hence it is not immediately obvious whether EDF-schedulability analysis will prove sustainable or not.

Fortunately, EDF-schedulability analysis 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 preemptive EDF schedulability tests.

Theorem 4 *Let $I = \{(r_i, c_i, d_i)\}_{i \geq 1}$ denote a collection of independent jobs, each characterized by a release 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 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 feasibility of I'. Since EDF is optimal on preemptive uniprocessors, it is therefore guaranteed to successfully schedule I' to meet all deadlines $\square$

Theorem 4 above directly leads to a sustainability result with respect to the *jitter, execution requirement*, and *relative-deadline* parameters for EDF-scheduling:

Corollary 1 *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 5 *Any sufficient test for the EDF-scheduling of sporadic task systems is trivially 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. $\square$

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 6 (from [4])

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 7 *Any sufficient test for the EDF-scheduling of zero-offset periodic task systems is sustainable with respect to the period parameter.*

Proof: Follows from Theorem 6 (1), and Theorem 5, 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 8 *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 6 (2) and Theorem 5. 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 flavours: *utilization-based*, and one based upon *processor demand criteria.*

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

$$\sum_{i=1}^{N} \left(\frac{C_i}{T_i}\right) \leq 1 \,. \tag{8}$$

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* $\text{DBF}_i(t)$ to be the largest possible cumulative execution requirement by jobs of task τ_i, that have both their ready times and their deadlines over any time interval of length t.

The following observation will be used later in this section:

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

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

$$\forall\, t : \, t \geq 0 \, : \left[\left(\sum_{i=1}^{N} \text{DBF}_i(t) \right) \geq t \right] \,. \tag{9}$$

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

Proof. Follows from Corollary 1 and Theorem 8. □

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) [3] is typically used in EDF-scheduled systems to arbitrate access to such resources. In [3, 17], sufficient conditions for a task system to be schedulable under the EDF+SRP scheduling framework are derived. For any two tasks τ_j and τ_k with $D_j > D_k$, let C_{jk} denote the maximum length of time for which τ_j may hold some non-preemptable resource that is also used by τ_k. Define a blocking function $B(t)$ as follows:

$$B(t) \stackrel{\text{def}}{=} \max\{C_{jk} \mid D_j > t \text{ and } D_k \leq t\} \,.$$

From results in [17], it can be concluded that

$$\forall t \geq 0 \, : \, \left(B(t) + \sum_{\ell=1}^{n} \text{DBF}_\ell(t) \right) \leq t \,. \tag{10}$$

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

Proof Sketch: From Observation 2, it follows that that the summation term in Inequality (10) 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 (10) 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 only 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 (10). Specifically, *every* task's execution was accounted for in the LHS of Inequality (10), 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 (10) for any L. (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 (10).) □

6. Other task parameters: *offsets* and *BCET's*

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: **(i)** they represent features of the system being modelled, for example precedence relations, and **(ii)** 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 [14] that we briefly discussed in Section 4, and the exact EDF-schedulability test proposed by Leung and Merrill [13]. Both these tests simulate the system 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).

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 [14] and the EDF test [13]: 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 [13, 14] is formalized in the following theorem:

Theorem 11 *The exact fixed-priority and EDF- schedulability tests for periodic task systems with offsets presented in [13, 14] 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 <u>even</u> 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 attainable 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 [17] 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 infeasible 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. Conclusions and related work

Schedulability tests are currently characterized as being *necessary* and/ or *sufficient*. This paper has introduced an additional characteristic: *sustainablility*. 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. 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.

In related work, Mok and Poon [16] have studied similar issues in nonpreemptive scheduling of periodic task systems in which all tasks have their relative deadline and period parameters equal – they refer to the concept of sustainability as "robustness." Recently, Buttazzo [9] has identified sources of potentially anomolous behavior in real-time systems executing upon variable-speed processors. On multiprocessor systems, Ha and Liu [10] 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 to meet all deadlines will continue to meet all deadlines if some jobs arrive earlier, or have later deadlines.

References

[1] K. Altisen, G. Gößler, A. Pnueli, J. Sifakis, S. Tripakis, and S. Yovine. A framework for scheduler synthesis. In *IEEE Real-Time Systems Symposium*, pages 154–163, 1999.

[2] K. Altisen, G. Gößler, and J. Sifakis. Scheduler modeling based on the controller synthesis paradigm. *Real-Time Systems*, 23(1-2):55–84, 2002.

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

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

[5] H. Ben-Abdallah, J.-Y. Choi, D. Clarke, Y.-S. Kim, I. Lee, and H. Xie. A process algebraic approach to the schedulability analysis of real-time systems. *Real-Time Systems*, 15(3):189–219, 1998.

[6] G. Bernat and A. Burns. Combining (n, m)-hard deadlines and dual priority scheduling. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 46–57, 1997.

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

[8] A. Burns, K. Tindell, and A. J. Wellings. Effective analysis for engineering real-time fixed priority schedulers. *IEEE Transactions On Software Engineering*, 21(5):475–480, 1995.

[9] G. Buttazzo. Achieving scalability in real-time systems. *IEEE Computer*, pages 54–59, May 2006.

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

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

[12] T.-W. Kuo and A. K. Mok. Load adjustment in adaptive real-time systems. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 160–171, 1991.

[13] J. Leung and M. Merrill. A note on the preemptive scheduling of periodic, real-time tasks. *Information Processing Letters*, 11:115–118, 1980.

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

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

[16] A. K. Mok and W.-C. Poon. Non-preemptive robustness under reduced system load. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 200–209, 2005.

[17] R. Pellizzoni and G. Lipari. Feasibility analysis of real-time periodic tasks with offsets. *Real-Time Systems: The International Journal of Time-Critical Computing*, 30(1–2):105–128, May 2005.

[18] R. Rajkumar. *Synchronization In Real-Time Systems – A Priority Inheritance Approach*. Kluwer Academic Publishers, Boston, 1991.

[19] L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Transactions on Computers*, 39(9):1175–1185, 1990.

[20] A. Wellings, M. Richardson, A. Burns, N. Audsley, and K. Tindell. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8:284–292, 1993.

[21] A. Zuhily. Optimality of (D-J)-monotonic priority assignment. Technical Report YCS404, Department of Computer Science, University of York, 2006.