Schedulability Analysis of EDF-Scheduled Embedded Real-Time Systems with Resource Sharing

FENGXIANG ZHANG, Southwest University
ALAN BURNS, University of York

Earliest Deadline First (EDF) is the most widely studied optimal dynamic scheduling algorithm for uniprocessor real-time systems. In the existing literature, however, there is no complete exact analysis for EDF scheduling when both resource sharing and release jitter are considered. Since resource sharing and release jitter are important characteristics of embedded real-time systems, a solid theoretical foundation should be provided for EDF scheduled systems. In this paper, we extend traditional processor demand analysis to let arbitrary deadline real-time tasks share non-preemptable resources and suffer release jitter. A complete and exact schedulability analysis for EDF scheduled systems is provided. This analysis is incorporated into QPA (Quick Processor-demand Analysis) which provides an efficient implementation of the exact test.

Categories and Subject Descriptors: C.3 [**Computer Systems Organization**]: Special-Purpose and Application-Based Systems—*Real-time and embedded systems*; B.8.1 [**Hardware**]: Reliability, Testing, and Fault-Tolerance; J.7 [**Computer Applications**]: Computers in Other Systems—*Real time*; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Scheduling*

General Terms: Algorithms, Design, Reliability

Additional Key Words and Phrases: Embedded and real-time systems, resource sharing, schedulability analysis, scheduling, algorithms, earliest deadline first, control and reliability.

ACM Reference Format:
Zhang, F. and Burns, A. 2013. Schedulability analysis of EDF-scheduled embedded real-time systems with resource sharing. ACM Trans. Embedd. Comput. Syst. 12, 3, Article 67 (March 2013), 19 pages.
DOI: http://dx.doi.org/10.1145/2442116.2442117

1. INTRODUCTION

The correctness of an embedded real-time system depends not only on the running results but also on the time at which results are produced. The completion of a request after its timing deadline is considered to be of no value, and could even lead to a failure of the whole system. Therefore, the most important characteristic of real-time systems is that they have strict timing requirements that must be guaranteed and satisfied. Schedulability analysis plays a crucial role in enabling these guarantees to be provided.

A real-time system comprises a set of real-time tasks; each task consists of a potentially unbounded stream of jobs. The task set can be scheduled by a number of policies including dynamic priority or fixed priority algorithms. The success of a real-time system depends on whether all jobs of all the tasks can be guaranteed to

This work is supported by the National Natural Science Foundation of China under grant 61202042 and the EPSRC funded Tempo project in the U.K.
Authors' addresses: F. Zhang, School of Computer and Information Science, Southwest University, Beibei, Chongqing, China; email: zhangfx@ieee.org; A. Burns, Department of Computer Science, University of York, York, YO10 5DD, U.K; email: alan.burns@york.ac.uk.

DOI: http://dx.doi.org/10.1145/2442116.2442117

complete their executions before their timing deadlines. If they can then we say the task set is *schedulable*.

The Earliest Deadline First (EDF) algorithm is the most widely studied dynamic priority scheduling policy for real-time systems. It has been proved [Liu and Layland 1973] to be optimal among all scheduling algorithm on a uniprocessor, in the sense that if a real-time task set cannot be scheduled by EDF, then it cannot be scheduled by any other algorithm.

Although many forms of analysis (including that reported in the above citation) assume tasks are independent of each other, in realistic systems the tasks need to access shared non-preemptable resources. These resources are typically protected by semaphores or mutexes provided by an RTOS (real-time operating systems). If a high-priority task is suspended waiting for a lower-priority task to complete its use of a non-preemptable resource, then priority inversion occurs [Sha et al. 1991]. The task is *blocked* by the lower priority task.

As well as blocking, in many real applicatios, each task can also suffer *release jitter*. This could happen when an arrived task is waiting for a signal from another task in a distributed system, or if task and clock periods are not compatible, or in hierarchical systems when a task arrives at a time when the capacity of its server has been exhausted [Saewong et al. 2002; Davis and Burns 2005, 2006; Zhang and Burns 2007; Burns and Wellings 2009]. For fixed priority scheduling, release jitter is a normal part of the system model [Burns et al. 1995; Zuhily and Burns 2007], and analysis that deals with both jitter and blocking is available in standard textbooks [Burns and Wellings 2009; Klein et al. 1993; Buttazzo 2005].

For EDF-scheduled systems, there are similar requirements. When tasks share non-preemptive resources, particular care must be taken in accessing shared data. The blocking time must be measurable and the affect of blocking has to be bounded. Since blocking and release jitter are important characteristics in realistic systems, schedulability analysis should be provided with these characteristics included. However, to the best of our knowledge, there is no complete exact analysis for EDF scheduling when both resource sharing and release jitter are considered. The Stack Resource Policy (SRP) [Baker 1991, 1990] is the most commonly used resource sharing protocol in the EDF scheduling environment. In this paper, under the framework of EDF+SRP, we extend traditional processor demand analysis for EDF scheduled real-time systems. We provide exact schedulability analysis for real-time systems when resource sharing and release jitter are both considered.

In the remainder of this paper we introduce a system model in Section 2, Section 3 considers the resource sharing policies, and Section 4 reviews the forms of analysis used with EDF scheduled systems. The analysis for release jitter is presented in Section 5; it is extended to include resource sharing in Section 6. The integration of the analysis into the QPA framework in given in Section 7. Finally, conclusions are provided in Section 8.

2. SYSTEM MODEL

A hard real-time system comprises a set of n real-time tasks $\{\tau_i, \tau_2, \ldots, \tau_n\}$ executing on a uniprocessor; each task consists of a potentially unbounded stream of jobs which must be completed before their deadlines. Let τ_i indicate any given task of the system, and let r_i indicate any given job of τ_i. Each task can be periodic or sporadic.

All jobs of a periodic task have a regular inter-arrival time T_i; we call T_i the period of τ_i. If a job for a periodic task arrives at time t, then the next job of τ_i must arrive at $t + T_i$. It is assumed that the first job of each periodic task arrives at the same time; without loss of generality, this time is taken to be $t = 0$.

The jobs of a sporadic task arrive irregularly, but they have a minimum inter-arrival time also denoted as T_i; we again call T_i the period of τ_i. If a job of the sporadic task τ_i arrives at time t, then the next job of τ_i can arrive at any time at or after $t + T_i$.

Each job of task τ_i requires up to the same worst-case execution time which equals the task's worst-case execution time C_i. Each job of τ_i has the same relative deadline which equals the task's relative deadline D_i; each D_i could be less than, equal to, or greater than T_i. These three cases being referred to as *constrained* deadlines, *implicit* deadlines, and *unconstrained* deadlines. The smallest relative deadline in the system is denoted by D_{min}; the largest by D_{max}. If a job of τ_i arrives at time t, the required worst-case execution time C_i must be completed within D_i time units, and the absolute deadline of this job (referred to by lower case d) is $t + D_i$.

Each task in the system can experience release jitter. When a job of τ_i arrives at time t, it will be released for execution at the latest time $t + J_i$ (the actual release time can be anywhere between t and $t + J_i$). Its absolute deadline remains at $t + D_i$.

Let U_i denote the utilization of τ_i (i.e., $U_i = C_i/T_i$), and define U to be the total utilization of the task set, computed by $U = \sum_{i=1}^{n} U_i$.

Tasks may access (under mutual exclusion) shared resources (critical sections), but we make no assumption as to when each job accesses its shared resources during its execution. The inclusion of shared resources in the system model implies that tasks may suffer *blocking*, which must be taken into account in the scheduling analysis. In Section 6.1, we will show that the worst-case blocking occurs if these resource accesses occur at the very beginning of the jobs' executions, and the analysis developed is therefore based on this assumption.

According to the EDF scheduling algorithm, in the absence of blocking, the job with the earliest absolute deadline has the highest priority and will be executed on the processor. At any time, a released job with an earlier absolute deadline will preempt the execution of a job with a later absolute deadline. When a job completes its execution the system chooses, for execution, the pending (released) job with the earliest absolute deadline.

3. RESOURCE SHARING POLICIES

There are a number of protocols existing for accessing shared resources under the EDF scheduling environment, for example, Stack Resource Policy (SRP) [Baker 1991, 1990], Dynamic Priority Ceiling [Chen and Lin 1990], Dynamic Priority Inheritance (DPI) [Stankovic et al. 1998], and Dynamic Deadline Modification (DDM) [Jeffay 1992]. The SRP was proposed for accessing shared resources as a generalization of the Priority Inheritance Protocol (PIP) and the Priority Ceiling Protocol (PCP) [Sha et al. 1990]. It has the advantage that it is very easily integrated into the EDF scheduling framework. Under PIP a task is blocked at the time when it attempts to enter a critical section, while under the SRP a task is blocked at the time when it is released and attempts to preempt a lower priority task. This property of SRP reduces context switches and simplifies the implementation [Baker 1991, 1990]. SRP ensures the blocking time of a job/task its bounded by the execution time of the longest outermost critical section of a lower-priority task, where the critical section of this lower-priority task accesses a shared resource that is also needed by a task with equal or higher priority to the blocked task. We now describe in detail SRP and give an example of its use.

3.1. The SRP Algorithm

Each job r_i of task τ_i is assigned a preemption level $\pi(\tau_i)$. Under EDF scheduling, the preemption level of a job correlates inversely to its relative deadline, i.e., $\pi(\tau_i) < \pi(\tau_j) \Leftrightarrow D_i > D_j$.

Table I

Task	C	D	T	Access Time
τ_1	3	12	16	1
τ_2	9	19	22	3
τ_3	16	40	53	4

Define $R_1, R_2, \ldots, R_m$ to be the non-preemptable shared resources in the system. Each resource, R_j, is assigned a ceiling preemption level denoted as $\Pi(R_j)$ which is set equal to the maximum preemption level of any job that may access it. Let $\hat{\pi}$ denote the highest ceiling of all the resources which are held/locked by some job at any time t, that is,

$$\hat{\pi} = max\{\Pi(R_j) \parallel R_j \textit{ is held at time } t\}.$$

Baker [1991, 1990] showed that the Stack Resource Policy (SRP) has the following properties (expressed as a theorem).

THEOREM 1 ([BAKER 1991, 1990]). *If no job r_i is permitted to start execution until $\pi(\tau_i) > \hat{\pi}$, then the following hold.*

(*1*) *No job can be blocked after it starts.*
(*2*) *There can be no transitive blocking or deadlock.*
(*3*) *No job can be blocked for longer than the execution time of one outermost critical section of a lower priority job.*
(*4*) *If the oldest highest-priority (i.e., shortest deadline) job is blocked, it will become unblocked no later than the first instance when the currently executing job is not holding any non-preemptable resource.*

As a result of these properties, a job r_i released at time t can start execution only if the following hold.

—The absolute deadline of this job $(t + D_i)$ is the earliest deadline of the active requests in the task set.
—The preemption level of r_i is higher than the ceiling of any resource that is held at the current time (i.e., $\pi(\tau_i) > \hat{\pi}$).

3.2. Example Usage of SRP

Consider a three-task system, defined in Table I, where all three tasks access a single shared resource R. Note the 'Access Time' in the table refers to the time each task takes to accessing R (it is the duration of its critical section).

As $D_1 < D_2 < D_3$, the preemption levels are related as follows: $\pi(\tau_1) > \pi(\tau_2) > \pi(\tau_3)$. Since every task needs to access R, the ceiling preemption level of this resource is given by $\Pi(R) = \pi(\tau_1)$.

Let a job of τ_3 arrive at $t = 0$ and lock the non-preemptive resource R. The highest ceiling of a locked resource at this time is given by $\hat{\pi} = \Pi(R) = \pi(\tau_1)$. Let the jobs of τ_1 and τ_2 be released at time $t = 2$ (while the resource is still locked). They are not allowed to start execution until τ_3 completes its access to R at time $t = 4$. At this point, τ_1 and τ_2 become unblocked. Since the job of τ_1 has the earliest absolute deadline, it will preempt the execution of τ_3. The job from τ_2 will execute when τ_1 has completed; τ_3 will execute again only when τ_2 is finished.

In this example execution, τ_1 and τ_2 both suffer blocking of duration 2 (i.e., this is the interval during which a task with a later deadline is executing). The worst-case occurs

when these two tasks are released just after τ_3 locks the resource. In this situation, the blocking time would be 4.

4. REVIEW OF SCHEDULABILITY ANALYSIS FOR EDF SCHEDULING

This section describes the previous research results on exact schedulability analysis for EDF scheduling with arbitrary relative deadlines (i.e., D unrelated to T). Leung and Merrill [1980] noted that a set of periodic tasks is schedulable if and only if all absolute deadlines in the interval $[0, \max\{s_i\} + 2H]$ are met, where s_i is the start time of task τ_i, $\min\{s_i\} = 0$, and H is the least common multiple of the task periods. Baruah et al. [1990] extended this condition for sporadic task systems, and showed that the task set is schedulable if and only if $\forall t > 0, h(t) \leq t$, where $h(t)$ is the processor demand function given by

$$h(t) = \sum_{i=1}^{n} \max\left\{0, 1 + \left\lfloor \frac{t - D_i}{T_i} \right\rfloor\right\} C_i. \tag{1}$$

Using the above necessary and sufficient schedulability test, the value of t can be bounded by a certain value. We refer to this value as the *upper bound* for task schedulability. The following Theorem introduces one of these upper bounds (note the total utilization of the task set has to be strictly less than 1).

THEOREM 2 ([ZHANG AND BURNS 2008A]). *An arbitrary deadline task set with $U < 1$ is schedulable if and only if*

$$\forall t < L_a^*, \quad h(t) \leq t,$$

where

$$L_a^* = \max\left\{(D_1 - T_1), \ldots, (D_n - T_n), \frac{\sum_{i=1}^{n}(T_i - D_i)U_i}{1 - U}\right\}. \tag{2}$$

As the processor demand function can only change at the absolute deadlines of the tasks, only the absolute deadlines require checking in the upper bound interval.

Spuri [1996] and Crespo and Mok [1996] derived another upper bound (L_b) for the time interval which guarantees we can find an overflow (i.e., deadline miss) if the task set is not schedulable. This interval is called the *synchronous busy period* (the length of the first processor busy period when all tasks are released simultaneously at the beginning of the period). However, Crespo and Mok [1996] only considered the situation where $D_i \leq T_i$. The length of the synchronous busy period can be computed by the following process [Spuri 1996; Crespo and Mok 1996].

$$w^0 = \sum_{i=1}^{n} C_i, \tag{3}$$

$$w^{m+1} = \sum_{i=1}^{n} \left\lceil \frac{w^m}{T_i} \right\rceil C_i, \tag{4}$$

the recurrence stops when $w^{m+1} = w^m$, and then $L_b = w^{m+1}$.

Since the calculation of L_b has an iterative form, compared with the low complexity ($O(n)$) of the calculation of L_a^*, we should avoid using L_b whenever $U \neq 1$. Moreover, in extensive simulation studies [Zhang and Burns 2007, 2008b] it was nearly always the case that $L_a^* < L_b$.

4.1. Processor Demand Analysis for EDF+SRP

Baker [1991, 1990] provided a sufficient schedulability condition for EDF+SRP; a system is schedulable if

$$\forall_{k=1,\ldots,n} \quad \left(\sum_{i=1}^{k} \frac{C_i}{D_i} + \frac{B_k}{D_k} \right) \leq 1,$$

where B_k is the maximum blocking time of τ_k. This sufficient test requires that $D_i \leq T_i$ for all tasks, and it is utilization based; a set of experiments [Zhang and Burns 2008b] showed that nearly all task sets which are randomly generated cannot be evaluated (or not deemed schedulable) by such a test. Hence, an exact schedulability analysis which is based on the processor demand analysis is needed by the EDF+SRP scheduling framework.

Let $b(t)$ be a function representing the maximum time a job r_k with relative deadline $D_k \leq t$ possibly being blocked by job r_α with relative deadline $D_\alpha > t$ in any given time interval $[0,t]$.

Spuri [1996] showed that a condition for the schedulability of a task set is that for any absolute deadline d_i in a synchronous busy period,

$$h(d_i) + b(d_i) \leq d_i.$$

Although the definition of $b(t)$ given by Spuri [1996] implies that release jitter is considered; in the proof of the schedulability condition, they did not take account of release jitter, only the blocking time is incorporated.

The definition of $b(t)$ given by Baruah [2006] is more intuitive. Let $C_{\alpha,k}$ denote the maximum length of time for which task τ_α needs to hold some resource that may also be needed by task τ_k. Then $b(t)$ can be defined and calculated by

$$b(t) = \max \{C_{\alpha,k} \| D_\alpha > t, D_k \leq t\},$$

and the exact schedulability condition becomes the following [Baruah 2006].

$$\forall d_i < L, \quad d_i - h(d_i) \geq b(d_i),$$

where L is an upper bound for schedulability analysis, and d_i is any absolute deadline of the tasks in the time interval $[0,t]$ when all tasks arrive initially at time 0.

Note that if t is greater than the maximum relative deadline (i.e., $t \geq D_{max}$) then the blocking term, $b(t)$, is zero. This property can be exploited to improve the efficiency of the analysis's implementation.

4.2. Quick Processor-Demand Analysis

In a given interval (e.g., between 0 and L_a^*), there can be a very large number of absolute deadlines that need to be checked. This level of computation has been a serious disincentive to the adoption of EDF scheduling in practice. Fortunately a much less computation-intensive test known as Quick convergence Processor-demand Analysis (QPA) [Zhang and Burns 2008b] has recently been proposed. Extensive experiments [Zhang and Burns 2008b] reported that the required volume of calculations needed to perform an exact schedulability analysis can be exponentially decreased by QPA, and that the required computation load for QPA is similar for all kinds of task sets.

Let L be an upper bound for schedulability. Considering that the upper bound L_a^* is not well defined (divide by 0) when the utilization of the task set U is equal to 1, let L be defined as

$$L = \begin{cases} L_a^* & : \quad U < 1, \\ L_b & : \quad U = 1. \end{cases} \tag{5}$$

Define a *failure point* to be any time t satisfying $h(t) > t$. QPA works by starting with a value of t close to L and then iterates back through a simple expression toward 0 or the largest failure point in (0,L). It jumps over absolute deadlines that can safely be ignored and hence only a small number of points are checked.

Let d_i be an absolute deadline of a job for task τ_i, then $d_i = kT_i + D_i$ for some k, $k \in N$. The QPA test is given by the following algorithm and theorem.

ALGORITHM 1: QPA

$t \leftarrow \max\{d_i \| d_i < L\}$
while $(t > D_{min})$ **loop**
  if $(h(t) < t)$ **then** $t \leftarrow h(t)$;
  else if $(h(t) = t)$ **then** $t \leftarrow \max\{d_i \| d_i < t\}$;
  else exit; // failure deadline is found
end loop;
if $(t > D_{min})$ **then** the task set is unschedulable;
else the task set is schedulable;

THEOREM 3 ([ZHANG AND BURNS 2008B]). *An arbitrary deadline task system is schedulable if and only if the iterative result of Algorithm 1 is $t < D_{min}$, where $D_{min} = \min_{1 \le i \le n}\{D_i\}$.*

Brief overview of the proof of Theorem 3. Let d_0 be the largest failure deadline in the interval [0,L] for an unschedulable system. At the beginning of the QPA algorithm, $t = \max\{d_i \| d_i < L\}$, hence $t \ge d_0$. Since $h(t)$ is a non-decreasing function of t, when $t \ge d_0$, $h(t) \ge h(d_0)$, and there is no failure in the interval $[h(d_0), t]$. Let t take the value of $h(d_0)$, hence the deadlines in $[h(d_0), t]$ are safely ignored. During this process, t is always greater than or equal to $h(d_0)$. The algorithm continues until $t = h(d_0)$, then a failure deadline must be found when $t \leftarrow \max\{d_i \| d_i < h(d_0)\}$. If this algorithm cannot find any failure deadline then the process ends with $t \le D_{min}$ in which case the task set is schedulable.

5. SCHEDULABILITY ANALYSIS WITH RELEASE JITTER

This section addresses exact schedulability analysis for task sets that experience release jitter. In previous literature, Spuri [1996] mentioned (but did not prove) that if we remove the assumption of null release jitter then the worst-case arrival pattern for testing the task set's schedulability by processor demand analysis is obtained by releasing the first job of each task at time $t = 0$, and all other jobs at $t = \max\{kT_i - J_i, 0\}$. Under this worst-case arrival pattern, the processor demand function is changed to the following [Spuri 1996].

$$h_J(t) = \sum_{i=1}^{n} \max\left\{0, 1 + \left\lfloor \frac{t + J_i - D_i}{T_i} \right\rfloor\right\} C_i, \tag{6}$$

and the length of the synchronous busy period, L_b^J, is calculated by

$$w^{m+1} = \sum_{i=1}^{n} \left\lceil \frac{w^m + J_i}{T_i} \right\rceil C_i. \tag{7}$$

The recurrence of Equation (7) starts with the value provided by Equation (3), and again ends when $w^{m+1} = w^m (= L_b^J)$.

In the following we give the above conclusion a formal proof, and provide a complete schedulability test which incorporates jitter.

LEMMA 1 ([LIU AND LAYLAND 1973]). *When the EDF algorithm is used to schedule a set of tasks on a processor, there is no processor idle time prior to an overflow (deadline miss).*

Definition 1. The *task load* in the time interval $[t_1,t_2]$ is the task execution time that must be completed at or before t_2 (i.e., the total execution time of jobs which arrive before t_2, with absolute deadlines less than or equal to t_2).

LEMMA 2. *The worst-case arrival pattern for task schedulability happens when all tasks are released simultaneously at time $t = 0$ after having experienced their maximum release jitter, and the subsequent jobs of each task are released at their maximum rate (without jitter). This task arrival pattern gives the longest length of processor busy period, and a failure must be found in this busy period for an unschedulable task set.*

PROOF. At first, we remove the release jitter assumption for each task. Let t_1 be a failure time point in any task's arrival pattern, and let t_0 be the last time before t_1 such that there are no pending jobs with absolute deadline $d_i \leq t_1$. Then we change the arrival pattern, and move "left" all tasks first jobs which arrive after t_0, to let all tasks be released simultaneously at t_0, then the task load which must be completed before t_1 in the interval could only be increased, hence, there is still an overflow at or before t_1. Define t_2 to be the new overflow time after the arrival pattern change, we have $t_2 \leq t_1$.

Now we add the release jitter assumption. If we let the first job of each task τ_i be released at time t_0, and move "left" all subsequent jobs of τ_i which arrive after t_0, making them closer to the first job, then the task load in the period $[t_0,t_2]$ becomes larger, and the maximum distance we can move forward for each τ_i is J_i. Therefore, the maximum task load in the interval $[t_0,t_2]$ is obtained when each task arrives simultaneously at time t_0 after having experienced its maximum release jitter, and the subsequent jobs arrive at their maximum rate without jitter. Hence, this is the worst-case arrival pattern for schedulability, and any overflow still occurs at or before t_2. Denote t_3 to be the new overflow time, then we have $t_3 \leq t_2 \leq t_1$.

Similarly, after changing the tasks' arrival patterns, the length of any processor busy period could only be increased, therefore, this arrival pattern gives the longest busy period. Denote L_b^J to be the length of such a busy period; we have $L_b \leq L_b^J$. From Lemma 1 and the definition of t_0, there is no processor idle time during $[t_0,t_1]$. As L_b is the longest busy period [Spuri 1996] without considering release jitter, $t_1 - t_0 < L_b$. Since $t_3 \leq t_2 \leq t_1$, we always have $t_3 - t_0 \leq t_1 - t_0 < L_b \leq L_b^J$. Hence a deadline failure must be found in the period $(0,L_b^J)$ for any unschedulable system. □

THEOREM 4. *A general task set with $U \leq 1$ is schedulable by EDF if and only if*

$$\forall t \in P, \quad h_J(t) \leq t,$$

where P is the set of absolute deadlines in the time interval (0, L), that is,

$$P = \{d_i \| d_i = kT_i + D_i - J_i \wedge d_i < L, k \in N\},$$

with

$$L = \begin{cases} \min(L_a^J, L_b^J) & : \quad U < 1, \\ L_b^J & : \quad U = 1, \end{cases}$$

and (for $U < 1$),

$$L_a^J = \max\left\{(D_1 - T_1 - J_1), \ldots, (D_n - T_n - J_n), \frac{\sum_{i=1}^{n}(T_i + J_i - D_i)U_i}{1 - U}\right\},$$

and L_b^J is the synchronous busy period with the arrival pattern of Lemma 2 (i.e., the solution of Equation (7)).

PROOF. As the task arrival pattern of Lemma 2 is the worst-case for the schedulability analysis, the task set is schedulable if and only if all tasks absolute deadlines are satisfied under this arrival pattern. From Lemma 2, we can check the schedulability in the time interval $(0,L_b^J)$. The remainder of the proof concerns the other upper bound for the schedulability test, L_a^J. When $t \geq \max_{1\leq i\leq n}\{D_i - T_i - J_i\}$,

$$t \geq D_i - T_i - J_i \Leftrightarrow t + J_i - D_i \geq -T_i$$

$$\Leftrightarrow \left\lfloor \frac{t + J_i - D_i}{T_i} \right\rfloor \geq -1 \Leftrightarrow 1 + \left\lfloor \frac{t + J_i - D_i}{T_i} \right\rfloor \geq 0,$$

then we have

$$h_J(t) = \sum_{i=1}^{n} \max\left\{0, 1 + \left\lfloor \frac{t + J_i - D_i}{T_i} \right\rfloor\right\} C_i = \sum_{i=1}^{n} \left\{1 + \left\lfloor \frac{t + J_i - D_i}{T_i} \right\rfloor\right\} C_i$$

$$\leq t \sum_{i=1}^{n} \frac{C_i}{T_i} + \sum_{i=1}^{n} \frac{C_i}{T_i}(T_i + J_i - D_i).$$

If the system is unschedulable, when $t \geq \max_{1\leq i\leq n}\{D_i - T_i - J_i\}$ and $h_J(t) > t$

$$\Rightarrow t < t \sum_{i=1}^{n} \frac{C_i}{T_i} + \sum_{i=1}^{n} \frac{C_i}{T_i}(T_i + J_i - D_i)$$

$$\Leftrightarrow t\left(1 - \sum_{i=1}^{n} \frac{C_i}{T_i}\right) < \sum_{i=1}^{n} \frac{C_i}{T_i}(T_i + J_i - D_i)$$

$$\Leftrightarrow t < \frac{\sum_{i=1}^{n}(T_i + J_i - D_i)U_i}{1 - U}. \quad \square$$

6. SCHEDULABILITY ANALYSIS WITH RESOURCE SHARING

In this section, under the framework of EDF+SRP, we show that blocking and release jitter can be considered at the same time within the traditional processor demand analysis. The treatment is not, however, straightforward as the amount of jitter affects the amount of blocking. The worst-case position, therefore, needs to be determined.

As the jobs of each task may get to their critical sections by more than one route, we assume that the jobs access the critical sections at the very beginning of their executions, therefore, we do not need to know the actual location of the critical sections. In Section 6.1, we show that this assumption is the worst-case for schedulability when blocking and release jitter are both considered.

When each task experiences its maximum release jitter, with the arrival pattern described by Lemma 2, then the first absolute deadline of each task τ_i (assuming an absolute start time of 0) is $D_i - J_i$. Hence, in any given time interval $[0,t]$, only a task with $D_i - J_i > t$ can block other tasks, and only the tasks with $D_i - J_i \leq t$ are required to complete their executions before t and thus can be blocked.

In the following sections, we suppose for each task $J_i < T_i$, this ensures a task will not be blocked by itself. To incorporate release jitter, we define $b_J(t)$ to be a function representing the maximum time any job r_k with $D_k - J_k \leq t$ may be blocked by any other

job r_a with $D_a - J_a > t$ in any given time interval $[0,t]$. Let $C_{a,k}$ denote the maximum length of time for which task τ_a needs to hold some resource that may also be needed by τ_k.

The blocking function $b_J(t)$ can be defined as

$$b_J(t) = \max\left\{C_{a,k} \| D_a - J_a > t, \quad D_k - J_k \le t\right\}; \tag{8}$$

note that $b_J(t)$ can also be defined as

$$b_J(t) = \max\left\{C_{a,k} \| D_a > t + J_a, \quad D_k \le t + J_k\right\}. \tag{9}$$

Also note that this blocking term is defined under the assumption that all tasks experience their maximum release jitter. Again the blocking term becomes zero when t is greater than the maximum D-J value in the task set.

If each task does not experience its maximum release jitter but only experiences a jitter equals to J_i^* where $0 \le J_i^* \le J_i$, then the maximum blocking time should be defined and calculated as

$$b_{J^*}(t) = \max\left\{C_{a,k} \| D_a > t + J_a^*, \quad D_k \le t + J_k^*\right\}.$$

It follows that the definitions of $b_J(t)$ and $b(t)$ are special cases of $b_{J^*}(t)$.

Under the assumption that each task τ_i only experiences a jitter equal to J_i^*, we can apply the same argument of Theorem 4 to show that the worst-case arrival pattern for schedulability occurs when each task τ_i is released at time 0, after having experienced a jitter J_i^*, and then it is released at its maximum rate. In this task arrival pattern, the processor demand in a given time interval $[0,t]$ becomes

$$h_{J^*}(t) = \sum_{i=1}^{n} \max\left\{0, 1 + \left\lfloor \frac{t + J_i^* - D_i}{T_i} \right\rfloor\right\} C_i. \tag{10}$$

Thus, the processor demand plus the blocking time in $[0,t]$ is calculated by $h_{J^*}(t)+b_{J^*}(t)$.

The maximum blocking time depends on how many jobs can be blocked and how many jobs can block other jobs. Comparing the definitions of $b_J(t)$ and $b_{J^*}(t)$, we can see that in the definition of $b_J(t)$, although the number of jobs which could be blocked is increased (due to $D_a \le t + J_a$), the number of jobs which could block other jobs is decreased (due to $D_a > t + J_a$). So at a given t, the value of $b_J(t)$ can be less than $b_{J^*}(t)$. In other words, when all tasks experience their maximum jitter, the maximum blocking time could be decreased. In order to show that Lemma 2 is still the worst-case arrival pattern for schedulability when incorporating blocking, we have to prove the following theorem.

THEOREM 5.

$$\forall t > 0, \ h_{J^*}(t) + b_{J^*}(t) \le h_J(t) + b_J(t).$$

PROOF. If $b_{J^*}(t) \le b_J(t)$, since $h_{J^*}(t) \le h_J(t)$, we have

$$h_{J^*}(t) + b_{J^*}(t) \le h_J(t) + b_J(t).$$

Therefore, we only need to prove the situation when $b_J(t) < b_{J^*}(t)$. From the definition of $b_J(t)$ and $b_{J^*}(t)$, when $b_J(t) < b_{J^*}(t) \Rightarrow$

$$\begin{aligned} &\max\left\{C_{\alpha,k} \| D_a > t + J_\alpha, \ D_k \le t + J_k\right\} \\ &< \max\left\{C_{\alpha,k} \| D_a > t + J_\alpha^*, \ D_k \le t + J_k^*\right\}. \end{aligned} \tag{11}$$

At any given time t, we always have

$$\begin{aligned} &\max\left\{C_{\alpha,k} \| D_a > t + J_\alpha^*, \ D_k \le t + J_k^*\right\} \\ &\le \max\left\{C_{\alpha,k} \| D_a > t + J_\alpha^*, \ D_k \le t + J_k\right\}. \end{aligned} \tag{12}$$

From Equalitions (11) and (12),

$$\begin{aligned} &\max\left\{C_{\alpha,k} \| D_\alpha > t + J_\alpha,\ D_k \le t + J_k\right\} \\ &< \max\left\{C_{\alpha,k} \| D_\alpha > t + J^*_\alpha,\ D_k \le t + J_k\right\}. \end{aligned} \tag{13}$$

And from this inequality, there exists a task τ_y which satisfies all the following conditions.

(1) $t + J^*_y < D_y \le t + J_y$.
(2) Task τ_y needs to hold some shared non-preemptive resource that is also needed by a task τ_k (with $D_k \le t + J_k$), $y \ne k$ – denote by $C_{y,k}$ the maximum critical length of this resource.
(3) $C_{y,k}$ is no shorter than any other critical section in $\left\{C_{\alpha,k} |\ D_\alpha > t + J^*_\alpha,\ D_k \le t + J_k\right\}$.

From condition (1),

$$D_y \le t + J_y \Rightarrow t + J_y - D_y \ge 0 \Rightarrow \left\{1 + \left\lfloor \frac{t + J_y - D_y}{T_y} \right\rfloor\right\} C_y \ge C_y. \tag{14}$$

And

$$t + J^*_y < D_y \Rightarrow t + J^*_y - D_y < 0 \Rightarrow \left\{1 + \left\lfloor \frac{t + J^*_y - D_y}{T_y} \right\rfloor\right\} C_y \le 0. \tag{15}$$

Hence,

$$h_J(t) = \sum_{i=1(i\ne y)}^{n} \max\left\{0,\ 1 + \left\lfloor \frac{t + J_i - D_i}{T_i} \right\rfloor\right\} C_i + \max\left\{0,\ 1 + \left\lfloor \frac{t + J_y - D_y}{T_y} \right\rfloor\right\} C_y$$

$\ge$ (*as* $J_i \ge J^*_i$ *and using Equation (14)*)

$$\sum_{i=1(i\ne y)}^{n} \max\left\{0,\ 1 + \left\lfloor \frac{t + J^*_i - D_i}{T_i} \right\rfloor\right\} C_i + C_y$$

$=$ (*using Equation (15)*) (16)

$$\sum_{i=1(i\ne y)}^{n} \max\left\{0,\ 1 + \left\lfloor \frac{t + J^*_i - D_i}{T_i} \right\rfloor\right\} C_i + \max\left\{0,\ 1 + \left\lfloor \frac{t + J^*_y - D_y}{T_y} \right\rfloor\right\} C_y + C_y$$
$$= h_{J^*}(t) + C_y$$

From condition (3), $C_{y,k} = \max\left\{C_{\alpha,k} |\ D_\alpha > t + J^*_\alpha,\ D_k \le t + J_k\right\}$, therefore,

$$C_{y,k} \ge \max\left\{C_{\alpha,k} |\ D_\alpha > t + J^*_\alpha,\ D_k \le t + J^*_k\right\} = b_{J^*}(t). \tag{17}$$

From the definition of $C_{y,k}$ and Equation (17),

$$C_y \ge C_{y,k} \ge b_{J^*}(t). \tag{18}$$

From Equalitions (16) and (18),

$$h_J(t) + b_J(t) \ge h_J(t) \ge h_{J^*}(t) + C_y \ge h_{J^*}(t) + b_{J^*}(t). \quad \square$$

Theorem 5 has proved that Lemma 2 is the worst-case arrival pattern for task schedulability. We can therefore derive the schedulability analysis which considers both blocking and release jitter in the following theorem.

THEOREM 6. *A general task set with $U \leq 1$ is schedulable by EDF+SRP if and only if*

$$\forall t \in P, h_J(t) + b_J(t) \leq t,$$

where P is the set of absolute deadlines in the time interval (0, L), that is,

$$P = \{d_i \| d_i = kT_i + D_i - J_i \wedge d_i < L, k \in N\},$$

with

$$L = \begin{cases} \min(L_a^B, L_b^J) & : \quad U < 1, \\ L_b^J & : \quad U = 1, \end{cases}$$

and (for $U < 1$)

$$L_a^B = \max\left\{(D_1 - T_1 - J_1), \ldots, (D_n - T_n - J_n), \frac{\max_{d_i < D_{max}}\{b_J(d_i)\} + \sum_{i=1}^{n}(T_i + J_i - D_i)U_i}{1 - U}\right\}, \tag{19}$$

and L_b^J is the synchronous busy period with the arrival pattern of Lemma 2, $h_J(t)$ is given by Equation (6), and $b_J(t)$ is given by Equation (8).

PROOF. Using a similar approach to that employed in the proof of Theorem 4 we first assume there is no release jitter. Let t be a time when there is a job missing its deadline, and let t^* be the last time before t such that there are no pending jobs with absolute deadlines less than or equal to t. At time $t^* = 0$, if there is an arrived job r_α with relative deadline $D_\alpha > t$ holding some shared resources which are also needed by a later arrived job r_k with $D_k \leq t$, then the lower priority job r_α may block the higher priority (earlier deadline) job r_k. Let t_u be the first time when r_α complete its access to a shared resources and does not hold any other shared resources. From Theorem 1, the oldest highest-priority blocked job will become unblocked immediately at time t_u, then there are always pending jobs with absolute deadlines less than or equal to t in the period $[t_u,t]$. In addition, no earlier deadline jobs can be blocked again after t_u, and r_α is the only job which could be executing in $[0,t]$ with absolute deadline larger than t. The maximum time that job r_α could be executing in the time interval $[0,t]$ is $C_{\alpha,k}$ and hence the maximum blocking time in $[0,t]$ is

$$b(t) = \max\left\{C_{\alpha,k} \| D_\alpha > t, \ D_k \leq t\right\}.$$

Now we change the task arrival pattern; let each task with $D_k \leq t + J_k$ be released simultaneously at time $t = 0$, after having experienced its maximum release jitter. It is then released at its maximum rate. Let each task with $D_\alpha > t + J_\alpha$ be released at time $-\epsilon$, after experiencing the release jitter equal to $J_\alpha - \epsilon$, and then at its maximum rate. The constant ϵ is a infinite small positive number. For each task with $D_\alpha > t + J_\alpha$, each absolute deadline of its job is given by

$$kT_\alpha + D_\alpha + (-\epsilon) - (J_\alpha - \epsilon) = kT_\alpha + D_\alpha = J_\alpha, \quad k \in N.$$

We now have the arrival pattern of Lemma 2; this is the worst-case for task schedulability without considering the blocking. Since each task with $D_\alpha > t + J_\alpha$ is released ϵ time units before the release time of each task with $D_k \leq t + J_k$, any task with $D_k \leq t + J_k$ can be blocked by any task with $D_\alpha > t + J_\alpha$ if they need to access the same non-preemptable resource, and the maximum blocking time in $[0,t]$ becomes

$$b_J(t) = \max\left\{C_{\alpha,k} \| D_\alpha - J_\alpha > t, \ D_k - J_k \leq t\right\}.$$

After adding this blocking, although the blocking time could be decreased when the task arrival pattern is changed to that of Lemma 2, from Theorem 5, the total task load in [0,t] calculated by $h_J(t) + b_J(t)$ could only be increased. Therefore there is still an overflow at or before time t, and the arrival pattern of Lemma 2 is still the worst-case for the schedulability analysis under EDF+SRP.

The following considers the upper bounds we need to derive. As blocking only changes the execution order of the jobs in a busy period, the length of the longest busy period is unchanged. Therefore, L_b^J can still be used as an upper bound for the schedulability test. The upper bound L_a^B is obtained by the following derivation.

Following the arguments used in the proof of Theorem 4, when

$$t \geq \max_{1 \leq i \leq n} \{D_i - T_i - J_i\},$$

$$\Leftrightarrow 1 + \left\lfloor \frac{t + J_i - D_i}{T_i} \right\rfloor \geq 0,$$

therefore,

$$h_J(t) + b_J(t) = b_J(t) + \sum_{i=1}^{n} \max \left\{ 0, 1 + \left\lfloor \frac{t + J_i - D_i}{T_i} \right\rfloor \right\} C_i$$

$$= b_J(t) + \sum_{i=1}^{n} 1 + \left\lfloor \frac{t + J_i - D_i}{T_i} \right\rfloor C_i$$

$$\leq b_J(t) + t \sum_{i=1}^{n} \frac{C_i}{T_i} + \sum_{i=1}^{n} \frac{C_i}{T_i}(T_i + J_i - D_i)$$

$$\leq \max_{d_i < D_{max}} \{b_J(d_i)\} + \sum_{i=1}^{n} \frac{C_i}{T_i} + \sum_{i=1}^{n} \frac{C_i}{T_i}(T_i + J_i - D_i).$$

If the system is unschedulable, when $t \geq \max_{1 \leq i \leq n} \{D_i - T_i - J_i\}$ and $h_J(t) + b_J(t) > t$

$$\Rightarrow t < \max_{d_i < D_{max}} \{b_J(d_i)\} + t \sum_{i=1}^{n} \frac{C_i}{T_i} + \sum_{i=1}^{n} \frac{C_i}{T_i}(T_i + J_i - D_i)$$

$$\Leftrightarrow t \left(1 - \sum_{i=1}^{n} \frac{C_i}{T_i}\right) < \max_{d_i < D_{max}} \{b_J(d_i)\} + + \sum_{i=1}^{n} \frac{C_i}{T_i}(T_i + J_i - D_i)$$

$$\Leftrightarrow t < \frac{\max_{d_i < D_{max}} \{b_J(d_i)\} + \sum_{i=1}^{n} (T_i + J_i - D_i)U_i}{1 - U}. \quad \square$$

In Equation (19) of Theorem 6, $\max_{d_i < D_{max}} \left\{b_J(d_i)\right\}$ can be obtained by calculating the value of $b(t)$ at every $d_i \in [0, D_{max}]$.

6.1. Locations of Critical Sections

If we remove the assumption that the jobs access the share resources at the beginning of their executions, then we can potentially exploit knowledge of the actual locations of each job's critical section. Let w_i be the minimum offset of a critical section from the

start of task τ_i's execution, then τ_i has to be released w_i time units earlier than other tasks to hold the shared resource. Therefore, in any time interval [0,t], only a task with $D_i - J_i - w_i > t$ can block other tasks. The blocking function $b_J(t)$ is redefined as

$$b_J(t) = \max\left\{C_{\alpha,k} \| D_\alpha - J_\alpha - w_\alpha > t,\ D_k - J_k \le t\right\}, \tag{20}$$

where $C_{\alpha,k}$ again represents the maximum length of time τ_α needs to hold a resource that may also be needed by τ_k.

If there is more than one critical section, let the offsets of each critical section be $w_\alpha^1, w_\alpha^2, \ldots$. All these critical sections then need to be evaluated by Equation (20).

In any given time interval, the processor demand function $h_J(t)$ is not changed, therefore, after the redefinition of $b_J(t)$, we can still use Theorem 6 to test the schedulability of the system. However, the worst-case arrival pattern of Lemma 2 becomes the following.

The tasks with $D_k - J_k \le t$ are released simultaneously at time $t = 0$ after having experienced their maximum release jitter; each task with $D_\alpha - J_\alpha - w_\alpha > t$ is released w_α time units earlier than $t = 0$ after having experienced its maximum release jitter; then all tasks' subsequent jobs are released at their maximum rate.

From Equation (20), a smaller value of w_α implies that more tasks could block other tasks, and the value of $b_J(t)$ is greater when for each task, w_i is zero. The factor $b_J(t)$ then attains its maximum value. Therefore, the assumption that the jobs access their critical sections at the beginning of their executions is the worst-case for task schedulability.

Although we provide analysis for task resource sharing when the location of each critical section is available, we suggest that careful use is made of this information in practice. As w_i is determined by the speed of processor, a system which is guaranteed to be schedulable could subsequently miss a deadline if the processors speed is increased. When the processor is speeded up, w_i is decreased and more tasks could block other tasks thus leading to increased blocking and an unschedulable system. This property (increasing processor speed leading to deadlines being missed) is an example of analysis that is not sustainable [Baruah and Burns 2006]. Such analysis is not recommended.

7. SCHEDULING ANALYSIS WITH QPA

The analysis presented above provides exact schedulability analysis for task sets that suffer blocking and release jitter. The analysis is built upon the traditional processor demand approach, hence, it could require a very large number of absolute deadlines to be calculated and checked. This level of computation can be decreased by employing QPA (see Section 4.2).

To integrate QPA with blocking and jitter, first we have to prove that $h_J(t) + b_J(t)$ is non-decreasing with t.

THEOREM 7. *$\forall t > 0,\ h_J(t) + b_J(t)$ is a non-decreasing function of t.*

PROOF. To prove this property, we only need to show that $\forall t_1, t_2,\ 0 < t_1 < t_2$, when $b_J(t_1) > b_J(t_2)$ the following holds: $h_J(t_1) + b_J(t_1) \le h_J(t_2) + b_J(t_2)$.

From the definition of $b_J(t)$, $b_J(t_1) > b_J(t_2) \Rightarrow$

$$\begin{aligned} &\max\left\{C_{\alpha,k} \| D_a - J_a > t_1,\ D_k - J_k \le t_1\right\} \\ &> \max\left\{C_{\alpha,k} \| D_a - J_a > t_2,\ D_k - J_k \le t_2\right\}. \end{aligned} \tag{21}$$

For any $t_1 < t_2$, we have

$$\begin{aligned} &\max\left\{C_{\alpha,k} \| D_a - J_a > t_2,\ D_k - J_k \le t_2\right\} \\ \ge &\max\left\{C_{\alpha,k} \| D_a - J_a > t_2,\ D_k - J_k \le t_1\right\}. \end{aligned} \tag{22}$$

Therefore,

$$\begin{aligned} &\max\left\{C_{\alpha,k} \| D_a - J_a > t_1,\ D_k - J_k \le t_1\right\} \\ > &\max\left\{C_{\alpha,k} \| D_a - J_a > t_2,\ D_k - J_k \le t_1\right\}. \end{aligned} \tag{23}$$

From Equation (23), there exists a task τ_f which satisfies all of the following conditions.

(1) $t_1 < D_f - J_f \le t_2$.
(2) Task τ_f needs to hold some non-preemptable resource that is also needed by a task τ_k with $D_k - J_k \le t_1$, and denote by $C_{f,k}$ the maximum length of such a resource access.
(3) $C_{f,k}$ is no shorter than any other critical section: $\left\{C_{\alpha,k} \| D_a - J_a > t_1,\ D_k - J_k \le t_1\right\}$, which means that $C_{f,k} = b_J(t_1)$.

Let the execution time of τ_f be C_f, we have $b_J(t_1) = C_{f,k} \le C_f$. As $t_1 < D_f - J_f$,

$$\begin{aligned} t_1 + J_f - D_f < 0 &\Rightarrow \left(1 + \left\lfloor \frac{t_1 + J_f - D_f}{T_f} \right\rfloor\right) C_f \le 0, \\ &\Rightarrow \max\left\{0,\ 1 + \left\lfloor \frac{t_1 + J_f - D_f}{T_f} \right\rfloor\right\} C_f = 0. \end{aligned} \tag{24}$$

As $D_f - J_f \le t_2$,

$$t_2 + J_f - D_f \ge 0 \Rightarrow \left(1 + \left\lfloor \frac{t_2 + J_f - D_f}{T_f} \right\rfloor\right) C_f \ge C_f. \tag{25}$$

From Equalities (24) and (25), and $t_2 > t_1$,

$$h(t_2) = \sum_{i=1(i\neq f)}^{n} \max\left\{0,\ 1 + \left\lfloor \frac{t_2 + J_i - D_i}{T_i} \right\rfloor\right\} C_i + \max\left\{0,\ 1 + \left\lfloor \frac{t_2 + J_f - D_f}{T_f} \right\rfloor\right\} C_f$$

$$\ge \sum_{i=1(i\neq f)}^{n} \max\left\{0,\ 1 + \left\lfloor \frac{t_2 + J_i - D_i}{T_i} \right\rfloor\right\} C_i + \max\left\{0,\ 1 + \left\lfloor \frac{t_2 + J_f - D_f}{T_f} \right\rfloor\right\} C_f + C_f$$

$$= h(t_1) + C_f.$$

Therefore, we have

$$h_J(t_1) + b_J(t_1) = h_J(t_1) + C_{f,k} \le h_J(t_1) + C_f \le h_J(t_2) \le h_(t_2) + b_J(t_2). \quad \square$$

In the remainder of this section, we define L as:

$$L = \begin{cases} \min(L_a^B, L_b^J) & : \ U < 1 \\ L_b^J & : \ U = 1 \end{cases}$$

where L_a^B is given by Equation (19), and L_b^J is calculated by Equation (7).

Table II

Task	C	D	T	J	R1	R2
τ_1	7	37	60	6	2	0
τ_2	19	70	160	18	6	9
τ_3	60	280	380	30	12	18
τ_4	47	590	460	40	0	14
τ_5	53	320	510	37	16	12
τ_6	70	360	490	46	13	11

The revised QPA test is given by the following algorithm and theorem.

ALGORITHM 2: QPA with Resource Sharing

$t \leftarrow \max\{d_i \| d_i < L\}$
while ($h_J(t) + b_J(t) \leq t \wedge h_J(t) + b_J(t) > D_{min}$) **loop**
  if $(h_J(t) + b_J(t) < t)$ **then** $t \leftarrow h_J(t) + b_J(t)$;
  else $t \leftarrow \max\{d_i \| d_i < t\}$;
end loop;
if $(h_J(t) + b_J(t) > D_{min})$ **then** the task set is unschedulable;
else the task set is schedulable;

THEOREM 8. *For a general task set scheduled by EDF+SRP with release jitter, it is schedulable if and only if the iterative result of Algorithm 2 is* $h_J(t) + b_J(t) \leq D_{min}$. *Where* $d_i = kT_i + D_i - J_i$, $k \in N$ *and* $D_{min} = \min_{1 \leq i \leq n}\{D_i - J_i\}$.

PROOF. From Theorem 7, $h_J(t) + b_J(t)$ is non-decreasing with t, therefore $h_J(t) + b_J(t)$ is equivalent to $h(t)$ in Algorithm 1. Based on the result of Theorem 6, the current theorem can be proved by the same process employed for Theorem 3. □

The QPA algorithm which considers the critical sections locations within each task's execution can be obtained directly from the discussions within Section 6.1 and Algorithm 2.

7.1. Efficient Implementation of QPA

For an unschedulable system there must be an absolute deadline, d_i, between time D_{min} and L at which the total load on the system is greater then d_i. For systems without resource sharing or jitter is was found that the largest failure deadline is often much closer to 0 than to L. This observation was used to define a variation on QPA (called QPA* [Zhang and Burns 2009, 2010]) that used a point P ($0 < P < L$ - but P closer to 0 than L) to define an algorithm that started at P and progressed to D_{min}. If no failure was found then the algorithm starts again at L and iterates back to P.

As a result of this modification [Zhang and Burns 2009], an unschedulable system is so identified with far fewer iterations. And a schedulable system, which required QPA to go from P to D_{min}, and then from L to P, takes at most one more iteration than before.

When resource sharing is included in the task model then the tendency for a task to miss its deadline early in the $0..L$ interval is exasperated by the inclusion of the blocking term only when $t < D_{max}$; after that time all $b(t)$ terms are zero. This observation gives rise to a QPA* version of Algorithm 2—in the following it is assumed that $P = D_{max} < L$, if $D_{max} \geq L$ then Algorithm 2 should be used.

7.2. Illustrative Example

This section gives an example to illustrate how to use the result of Theorem 8 to test for schedulability. This example contains 6 tasks (see Table II), and there are two distinct

ALGORITHM 3: QPA* with Resource Sharing

$t \leftarrow \max\{d_i \| d_i < P\}$
while $(h_J(t) + b_J(t) \leq t \wedge h_J(t) + b_J(t) > D_{min})$ **loop**
  if $(h_J(t) + b_J(t) < t)$ **then** $t \leftarrow h_J(t) + b_J(t)$;
  else $t \leftarrow \max\{d_i \| d_i < t\}$;
end loop;
if $(h_J(t) + b_J(t) > D_{min})$ **then** the task set is unschedulable; **exit**;
else
$t \leftarrow \max\{d_i \| d_i < L\}$
while $(h_J(t) \leq t \wedge h_J(t) > P)$ **loop**
  if $(h_J(t) < t)$ **then** $t \leftarrow h_J(t)$;
  else $t \leftarrow \max\{d_i \| d_i < t\}$;
end loop;
if $(h_J(t) > P)$ **then** the task set is unschedulable;
else the task set is schedulable;

non-preemptable shared resources R1 and R2 in the system. The task parameters and the non-preemptable resource access times are provided in Table II. Note a zero resources access time means that the task does not use that resource; so for example, task τ_4 has an execution time of 47, a relative deadline of 590, a period of 460 (which is less than its deadline), a maximum release jitter of 46, and a maximum execution time whilst accessing R2 of 14 (it does not use R1).

The schedulability of the task set is tested by the following steps.

—*Step 1.* Calculate the utilization of the task set, $U = 0.7423 \leq 1$.
—*Step 2.* Calculate the upper bound L_α^B by Equation (19), $L_\alpha^B = 365$; calculate upper bound L_b^J by Equation (6), $L_b^J = 329$.
—*Step 3.* As $L_b^J < L_\alpha^B$, $L = L_b^J = 329$; $\max\{d_i \| d_i < L\} = 314$.
—*Step 4.* Verify the schedulability by Algorithm 2.
  (1) $t = 314$, $h_J(t) = 256$, $b_J(t) = 14$, $h_J(t) + b_J(t) = 270$.
  (2) $t = 270$, $h_J(t) = 126$, $b_J(t) = 16$, $h_J(t) + b_J(t) = 142$.
  (3) $t = 142$, $h_J(t) = 33$, $b_J(t) = 18$, $h_J(t) + b_J(t) = 51$.
  (4) $t = 51$, $h_J(t) = 7$, $b_J(t) = 16$, $h_J(t) + b_J(t) = 23$.

Since 23 is less than the minimum deadline-jitter (i.e., $37 - 6 = 31$) the task set is schedulable. Note only four $h_J(t) + b_J(t)$ calculations are required by QPA. In Step 2 we used the smaller value of 314 computed by the iterative method; however as $U \neq 1$ we could have started with the value given by the simpler formula, L_α^B.

8. CONCLUSION

In this paper we had presented an exact and complete schedulability analysis for EDF+SRP scheduling. The interactions between release jitter and blocking can affect the schedulability of a system. Therefore, incorporating blocking and release jitter within the same analysis framework is not straightforward. We have identified what is the worst-case arrival pattern for task schedulability, and we have extended the traditional processor demand analysis to the situation where both blocking and release jitter are experienced. A complete schedulability analysis for EDF+SRP has been provided. We also discussed the situation when the location of the tasks' critical sections are available; this information can be used to perform more accurate analysis that takes this knowledge into account.

The proposed schedulability analysis is based on the standard processor demand approach, thus it could require a large number of absolute deadlines to be checked for

a single test. We have demonstrated that the efficient QPA scheme can be extended to include both blocking and release jitter. As a result, the presented analysis is an effective means of determining the schedulability of EDF+SRP scheduled arbitrary deadline real-time systems.

REFERENCES

BAKER, T. 1990. A stack-based resource allocation policy for realtime processes. In *Proceedings IEEE Real-Time Systems Symposium (RTSS)*. 191–200.

BAKER, T. March 1991. Stack-based scheduling of realtime processes. *Journal of Real-Time Systems 3,* 1.

BARUAH, S. 2006. Resource sharing in EDF-scheduled systems: A closer look. In *Proceedings of IEEE Real-Time Systems Symposium (RTSS)*. 379–387.

BARUAH, S. AND BURNS, A. 2006. Sustainable schedulability analysis. In *Proceedings of IEEE Real-Time Systems Symposium (RTSS)*. 159–168.

BARUAH, S., MOK, A., AND ROSIER, L. 1990. Preemptive scheduling of hard real-time sporadic tasks on one processor. In *Proceedings of IEEE Real-Time Systems Symposium (RTSS)*. 182–190.

BURNS, A., TINDELL, K., AND WELLINGS, A. J. 1995. Effective analysis for engineering real-time fixed priority schedulers. *IEEE Trans. Software Enginering 21,* 5, 475–480.

BURNS, A. AND WELLINGS, A. J. 2009. *Real-Time Systems and Programming Languages* 4th Ed. Addison Wesley Longman.

BUTTAZZO, G. 2005. *Hard Real-Time Computing Systems*. Springer.

CHEN, M. AND LIN, K. 1990. Dynamic priority ceilings: A concurrency control protocol for real-time systems. *Journal of Real Time Systems 2,* 4, 325–346.

CRESPO, I. R. A. AND MOK, A. 1996. Improvement in feasibilty testing for real-time tasks. *Journal of Real-Time Systems 11,* 1, 19–39.

DAVIS, R. AND BURNS, A. 2005. Hierarchical fixed priority preemptive scheduling. In *Proceedings of IEEE Real-Time Systems Symposium (RTSS)*. 389–398.

DAVIS, R. AND BURNS, A. 2006. Resource sharing in hierarchical fixed priority preemptive systems. In *Proceeding IEEE Real-Time Systems Symposium (RTSS)*.

JEFFAY, K. 1992. Scheduling sporadic tasks with shared resources in hard-real-time systems. In *Proceedings of IEEE Real-Time Systems Symposium (RTSS)*. 89–99.

KLEIN, M. H., RALYA, T. A., POLLAK, B., OBENZA, R., AND HARBOUR, M. G. 1993. *A Practitioner's Handbook for Real-Time Analysis: A Guide to Rate Monotonic Analysis for Real-Time Systems*. Kluwer Academic Publishers.

LEUNG, J. AND MERRILL, M. 1980. A note on preemptive scheduling of periodic real-time tasks. *Information Processing Letters 11,* 3, 115–118.

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

SAEWONG, S., RAJKUMAR, R., LEHOCZKY, J., AND KLEIN, M. 2002. Analysis of hierarchical fixed- priority scheduling. In *Proceedings of the 14th Euromicro Conference on Real-Time Systems (ECRTS)*. 173–181.

SHA, L., R., R., S., S., AND C-H, C. 1991. A real-time locking protocol. *IEEE Transactions on Computers 40,* 7, 793–800.

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

SPURI, M. 1996. Analysis of deadline schedule real-time systems. Tech. Rep. 2772, INRIA.

STANKOVIC, J., RAMAMRITHAM, K., SPURI, M., AND BUTTAZZO, G. 1998. *Deadline Scheduling for Real-Time Systems*. Kluwer Academic Publishers.

ZHANG, F. AND BURNS, A. 2007. Analysis of hierarchical EDF preemptive scheduling. In *Proceedings of IEEE Real-Time Systems Symposium (RTSS)*. 423–435.

ZHANG, F. AND BURNS, A. 2008a. Schedulability analysis for real-time systems with EDF scheduling. *IEEE Transaction on Computers 58,* 9, 1250–1258.

ZHANG, F. AND BURNS, A. 2008b. Schedulability analysis for real-time systems with EDF scheduling. Tech. Rep. YCS 426, University of York.

ZHANG, F. AND BURNS, A. 2009. Improvement to quick processor-demand analysis for EDF-scheduled real-time systems. In *Proceedings of the 21st Euromicro Conference on Real-Time Systems (ECRTS)*. 76–86.

ZHANG, F. AND BURNS, A. 2010. Dividing point value selections for improved quick processor-demand analysis. In *Proceedings of the 2nd International Conference on Software Technology and Engineering (ICSTE)*. Vol. 1. 170–175.

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

Received July 2010; revised November 2010, October 2011; accepted May 2011