The stack resource protocol based on real-time transactions

P.G. Jansen and R. Laan

Abstract: Current hard real-time (HRT) kernels have their timely behaviour guaranteed at the cost of a rather restrictive use of the available resources. This makes current HRT scheduling techniques inadequate for use in a multimedia environment where one can profit by a better and more flexible use of the resources. It is shown that one can improve the flexibility and efficiency of real-time kernels and a method is proposed for precise quality of service schedulability analysis of the stack resource protocol. This protocol is generalised by introducing real-time transactions, which makes its use straightforward and efficient. Transactions can be refined to nested critical sections if the smallest estimation of blocking is desired. The method can be used for hard real-time systems in general and for multimedia systems in particular.*

1 Introduction

Dedicated real-time (RT) kernels have the capability to perform hard real-time (HRT) tasks, such as reservation of shared resources and schedulability analysis. Reservations can be done offline, at run-time, or both. Offline handling offers speed but is inflexible. Handling at run-time might be complex and time consuming due to a complex administration if not adequately organised.

This paper investigates a simple but powerful, flexible scheduling strategy with low administration overhead in such a way that a straightforward analysis of RT behaviour is attainable. Flexibility can be provided when desired or timely precision can be guaranteed when needed. We have good reasons to believe that this strategy can be used for targets ranging from dedicated RT kernels to 'general purpose operating systems' with RT support. In particular our technique allows for *dynamic* admission of new tasks and for Quality of Service (QoS) variation of running tasks.

The RT task scheduling techniques that we will use are based on the principle of real-time transactions. These are scheduled by the 'earliest deadline first' rule and extended with selected *inheritance* strategies to limit blocking. From these ingredients we have constructed RT scheduling variants of the 'ceiling protocol' and the 'stack resource protocol' and have evaluated their properties. Real-time transactions make these protocols simple and transparent, which gives them an educational advantage; they are very easy to explain and easy to reason about. A second advantage is that transactions make the implementations of these protocols straightforward and consequently the administration overhead is limited to a minimum. Also, feasibility analysis is made comprehensible, as we show in Section 5.5 where we present an improved algorithm. A drawback of transactions is, however, that they do not limit blocking overhead to a minimum. In Section 5.6 we present measures to overcome this.

IEE Proceedings online no. 19990406

Paper first received 3rd September 1998, and in revised form 16th February 1999

P.G. Jansen is with the Deptartment of Computer Science, University of Twente, PB 217 7500 AE, Enschede, Netherlands
E-mail: jansen@cs.utwente.nl
R. Laan is with Research & Development, Océ Technologies B.V., PB 101 5900 MA, Venlo, Netherlands
E-mail: rla@oce.nl

*This research is being undertaken at the University of Twente in the multimedia *Tukker* project.

2 Existing pre-emptive scheduling methods

In pre-emptive scheduling, tasks are scheduled according to a priority. A task may pre-empt another task if it has a higher priority. The priorities of the following well known scheduling methods are determined as follows:

- Earliest deadline first (EDF). Priority increases dynamically when the deadline comes closer.
- Rate monotonic (RM). Priority is static and is inversely proportional to the period time: short periods are mapped on high priorities. The deadline is equal to the end of the period. (Note that in a dynamic scheduling algorithm static as well as dynamic priorities can be used.)
- Deadline monotonic (DM). Priority is static and inversely proportional to the deadline interval. The deadline is before the end of the period.

Without further precautions scheduling methods may lead to phenomena like blocking, priority inversion, or transitive waiting. Blocking may happen when shared resources are used. In this context we mean by shared resources those resources for which a task has to enter a mutual exclusive (mutex) critical section. A waiting task cannot pre-empt a running one that is in a critical section. Blocking happens when a high priority task must wait for the release of a resource by a low priority task. Priority inversion is a special form of blocking. It occurs when a high priority task is blocked, waiting for a resource that is held by a low priority task that is pre-empted by a medium priority task. Transitive waiting occurs in a chain of tasks which are all waiting for the release of resources by their predecessors. This may cause large (indirect) blocking values.

The priority of a task can be *static* or *dynamic*. A static priority does not vary in time while a dynamic priority does. Note that in EDF a deadline can be expressed as a static or a dynamic priority. A deadline interval (from release time to deadline) is mostly associated with a static priority while, in the context of this paper, an absolute deadline is associated with a dynamic priority.

A scheduler orders tasks in terms of priority and a dispatcher assigns these tasks to the processor(s) in the resulting order. In dynamic RT systems the dispatcher and scheduler are generally combined in one entity and referred to as 'the scheduler'. A scheduler executes protocols such as:

1. basic protocols,
2. ceiling protocols,
3. stack resource (SR) protocol,
4. transaction protocols.

Basic protocols are, for instance, the fixed priority (FP) protocol and the basic inheritance (BI) protocol. All but the FP-protocol provide methods to bound blocking. BI realises this by inheriting either static or dynamic priority. A low priority task τ_l, owning shared resources that are also requested by high priority tasks τ_h, inherits the high priority from τ_h. BI limits blocking; however, it cannot avoid transitive waiting.

The ceiling protocols have RM or DM as their ancestor. The basic idea is to make way for a high priority task, say τ_h, by *not* allowing pre-emption of a low priority task, say τ_l by any medium priority task , say τ_m, if τ_l uses resources also claimed by τ_h. This strategy limits blocking to one single task only, or more precisely, to one critical section only. This implies that transitive waiting is not possible and consequently *deadlock* is impossible. Among these is the original priority ceiling (PC) protocol[1]. Variations of it, such as the ceiling abort (CA) protocol [2] and the conditional abortable priority ceiling (CAPC) protocol [3], have critical sections, segmented in an abortable and a nonabortable part, to further limit the effects of blocking. An in-depth overview of inheritance protocols is given by Rajkumar in [4]. Sha *et al.* give an overview in [5] of how to generalise PC for DM under blocking and they discuss how to use this protocol for practical system implementation.

SR [6] has its roots in EDF and PC. It executes a similar *ceiling* mechanism to PC to limit blocking but uses on top of this a scheduling algorithm, generally EDF. We explain in Section 4.3 why this protocol is favoured and why prime attention is paid to SR/EDF. Buttazzo *et al.* also found the SR/EDF protocol attractive: in [7] they explained how to use it in a hybrid environment of soft and hard RT tasks. They also introduced a schedulability analysis for SR/EDF. We refine their results and present a feasibility algorithm.

Transactions are used for their conceptual simplicity. They make it possible to give a comprehensive overview of relations between PC and SR and give a clear view of their important properties for feasibility analyses. If the smallest amount of blocking is an issue, then transactions are not the best way to go. In such case we can easily refine transactions to nested critical sections. This gives the advantage of developing a general framework in which the burden of detailed requirements can be introduced at a later stage. The refinement of transactions is described in Section 5.6.

We now introduce a task model suited for flexible scheduling of transactions. Based on this model we introduce our variant of SR/EDF and analyse the pros and cons in Section 4.4.

3 Transactions

The real-time transaction protocols also avoid priority inversion and transitive waiting. When a transaction starts, it simultaneously acquires all resources it needs to complete the transaction. During the transaction, resources can only be released. A transaction completes when it has released all resources. Priority inheritance is applied dynamically when a high priority transaction must wait for resources in use by a low priority transaction. This avoids pre-emption of low priority transactions and advances the release of resources.

Tasks are based on transactions. This simplifies the use of critical sections for mutual exclusive resources considerably. It makes our transaction model straightforward, which has the positive consequences of a low administration overhead and clear schedulability analysis. If shared resources are to be used there are also disadvantages, such as larger blocking values. In order to avoid these, some scheduling strategies can straightforwardly substitute transactions by *nested critical sections*, as we will show in Section 5.6. Until this section our emphasis is on transactions because of their conceptual clearness.

In this paper we refer only to periodic tasks, which we model as a periodic transaction. Whether transactions are periodic is not relevant for the proposed scheduling algorithm, however, the QoS schedulability analysis in Section 5 needs information such as length of the period and maximum run-time per invocation.

In practice a transaction might suspend itself, for example to await the ready event of an I/O-action. Of course there must be an upper bound on the time a transaction has to wait. Moreover, the transaction model depends on resource-holding during waiting. If no resources are held we can split up the transaction, however, a precedence relation has to be introduced then: the first part should precede the second. It should be discouraged from holding resources during waiting. However, if necessary, this complicates the QoS schedulability analysis since resource usage does not imply the active use of the processor anymore. In this paper we will not go into further detail on these complications.

3.1 Transaction state

A transaction may be in the *sleeping* or in the *ready* state. The ready state is split up in *released, running* or *pre-empted*. A transaction is put into the administration after it is admitted to the system. It is then put into the sleeping state where it waits for its release, at which time it enters the ready state. In the ready state a transaction can be released when it is waiting for the processor, running when it has the processor or pre-empted when it has to leave the processor to a transaction with a higher priority. When a transaction is done, it is put into the sleeping state, waiting for the following release event. When a transaction is completely finished it is withdrawn from the administration.

3.2 Transaction model

A transaction is a member of the set of all transactions $T = \{\tau_1, \ldots, \tau_n\}$.

Definition 1 (Transaction): Transaction τ_i is defined by a tuple of static parameters (D_i, T_i, C_i, R_i) where D_i is the deadline interval, T_i is the time interval between two successive invocations (the period). C_i is the maximum run-time interval to complete. (Run-times are mainly used

in conjunction with QoS schedulability analysis for periodic processes. The scheduling algorithm does not need them.) R_i is the set of mutual exclusive resources used by τ_i. If two transactions τ_i and τ_k require the same mutex resource ($R_i \cap R_k \neq \emptyset$), then they are not allowed to pre-empt each other.

Definition 2 (Invocation): Invocation τ_i^j is defined by the tuple $(\tau_i, j, r_i^j, d_i^j)$ where τ_i^j is the jth invocation of τ_i. The first invocation of τ_i is denoted by τ_i^o, τ_i^j is associated with the parameters (r_i^j, d_i^j), where r_i^j is the absolute release time from which an invocation j may run, and d_i^j is the absolute deadline at which an invocation j has to be completed. Note that $D_i = d_i^j - r_i^j$, $d_i^j \leq r_i^{j+1}$ and $r_i^{j+1} - r_i^j \geq T_i$ for any $i \geq 1, j \geq 0$.

A transaction or invocation with a priority smaller than or equal to a running invocation may not pre-empt that running invocation. We shall derive the priority from the deadline, either absolute or relative, depending on the choice of protocol.

If a transaction τ_a must be executed before τ_b then there exists a precedence relation between them denoted by $\tau_a \prec \tau_b$. Precedence relations are beyond the scope of this paper. They do not present a problem for a scheduler, however, they complicate the analysis of schedulability. Note that $\tau_a \prec \tau_b$ can be enforced by signalling τ_b at the end of τ_a.

4 The inheritance protocols

This section discusses two protocols for scheduling tasks: the priority ceiling (PC) protocol [1] and the stack resource (SR) protocol [6]. In fact we use PC as an introduction to SR, for which we also derive QoS feasibility analysis in Section 5. We have based both protocols on real-time transactions and both maintain an inherited *pre-emption level*. This pre-emption level determines which transactions may pre-empt a running one. Pre-emption levels can be based on absolute deadlines or on deadline intervals. PC as well as SR has a pre-emption level that is statically derived from deadline intervals; SR has a dynamic refinement.

4.1 Ceiling protocols

Ceilings are used in PC and SR. We shall introduce variants of these protocols and evaluate their advantages and disadvantages. For clarity we have chosen not to introduce these protocols in their full glory but only in their essentials. We shall use transactions instead of nested critical sections and single-unit resources instead of multiple-unit resources.

In the following we will introduce the notion of *floor*—the inverse of ceiling—and pre-emption level. Then we introduce a simple variant of PC and successively extend this protocol to an interesting variant of it: SR. PC and SR have been defined originally in terms of priority. To prevent confusion with the notions used in this paper, we prefer to use deadlines instead of priorities.

4.2 Priority ceiling protocol

In PC the ceiling of a resource refers to the highest static priority of all tasks that may ever require that resource. We will use deadline interval instead of priority and consequently floor instead of ceiling. The floor D_R of a resource R is defined as the size of the shortest deadline interval D_i of any transaction τ_i that requires R.

$$D_R = \min\{D_i | R \in R_i\} \tag{1}$$

The minimum of all floors of a transaction τ_i is defined as the pre-emption deadline Δ_i of a transaction τ_i. It is defined as follows:

$$\Delta_i = \min\{D_i, D_R | R \in R_i\} \tag{2}$$

Δ_i is a static property of τ_i and can be computed offline for a given set T. The smallest pre-emption deadline of all currently running or pre-empted transactions is the running one τ_r^l and is denoted Δ_r.

Definition 3 (PC): PC is defined by the following rules:

(1) Released but not yet running or pre-empted invocations are ordered to their deadline intervals D_i.
(2) The invocation τ_i^j with the shortest deadline (say D_i) is selected for processor competition.
(3) τ_i^j will pre-empt the running invocation iff $D_i < \Delta_r$.

All static information can be computed offline or in the background. This variant is very easy to implement. This is also due to the use of transactions, which is stricter than the original PC. A task that is free running and does not (yet) use resources may pre-empt in the original PC when it has a higher priority. It may not pre-empt in our transaction variant. Note that our PC shows a last-in first-out behaviour of running transactions. This opens the possibility for using a single shared stack for all transactions. This is not possible in the original PC. A shared stack for all transactions would considerably limit the amount of memory needed. We now introduce a refined variant of the PC protocol, the SR protocol.

4.3 Stack resource protocol

Under SR an invocation τ_i does not only use a static deadline interval D_i but also d_i^j, the dynamic absolute deadline. Its pre-emption level is determined by a pair (Δ_i, d_i^j) where Δ_i is defined as in (2) under PC.

Definition 4 (SR): SR is defined by the following rules:

(1) Released but not yet running and pre-empted invocations are ordered to their absolute deadlines d_i^j.
(2) The invocation τ_i^j with the shortest dynamic deadline (say d_i^j) is selected for processor competition.
(3) τ_i^j pre-empts the running invocation τ_r^l iff $(D_i < \Delta_r) \wedge (d_i^j < d_r^l)$.

Due to the last-in first-out property of SR we may conclude that the running invocation is on top of a stack of pre-empted invocations. The original SR has been used in practice for some time. Baker [6] proposed several refinements such as multiple-unit resources, nested critical sections and a schedulability analysis. For more details see [6].

4.4 Evaluation of PC and SR with transactions

Both protocols do not need explicit use of synchronisation primitives such as semaphores if they are built on top of transactions. Due to inheritance and order, synchronisation is implicitly accomplished. This obliterates the explicit request for mutual exclusion; no additional synchronisation primitives are needed from a kernel. Not using transactions would require explicit synchronisation, generally offered by the kernel. This makes the development of the kernel and applications straightforward, clear, and easy

to implement. Furthermore, transactions make it possible to view SR as an orthogonal extension of PC by just adding EDF to the priority rule of PC.

Transactions do not only bring advantages; they may also limit pre-emption since they claim the mutual exclusive use of resources during the run-time of the transaction. Refinements are possible and are treated in Section 5.6.

PC is very straightforward; it has a small overhead and our variant can run on a stack. Ceilings and pre-emption levels can be computed offline or in the background. Blocking is limited to only one invocation. PC is a good and powerful candidate for use in any RT environment. It has the small disadvantage that all scheduling information used is static. This might make its dynamic behaviour somewhat inflexible.

SR brings dynamic behaviour into play again by adding dynamic priority to static priority when scheduling decisions have to be made. SR inherits all the good static properties from PC: small overhead, possibility of offline computation of ceilings, a maximum of one blocking invocation and, in the case of transactions, the possibility of a shared stack.

SR uses pre-emption levels and a scheduling mechanism of any choice. Pre-emption levels avoid multiple blocking. As a scheduling mechanism we may take EDF, rate monotonic (RM), deadline monotonic (DM) or anything else. In fact the given definition of SR in the previous section is SR with EDF: SR/EDF. Choosing RM would lead to a resource-using variant of the original RM, but with the possibility of using a stack and with limited blocking. Choosing DM as the dynamic part of DM, in our opinion, does not make much sense.

If we compare the utilisation of SR/EDF to PC, we see that SR/EDF is part of the EDF family with an upper bound of 1 while PC is part of the rate monotonic family with an upper bound between ln(2) and 1.

Our conclusion is that SR is an attractive protocol because (a) it is general in the sense that it executes any scheduling algorithm, (b) it limits blocking to one blocker only, (c) it limits context switches by letting current tasks run to completion unless pre-empted by a task of higher priority, (d) it can share one stack for all processes, and (e) it is easy to implement.

In the following sections we take a closer look at SR/EDF. In Section 5.3 we show that it has a straightforward feasibility analysis, and in Section 6 we show that it has a low administration overhead.

5 QoS schedulability analysis

We present a schedulability analysis for SR/EDF. Schedulability has been extensively studied. The following references introduce important milestones of scheduling theory. Liu and Layland [8] stated that EDF is optimal in the following sense: if any algorithm can schedule a set of tasks, then EDF can also schedule these tasks (note that EDF is optimal only under restrictive conditions in Liu and Layland's paper). Shared resources were not taken into account. Mok [9] has shown that the problem of deciding schedulability of a set of periodic tasks with shared resources (mutex constraints) is NP-hard. Jeffay [10] stated that *nonpre-emptable* EDF can schedule any schedulable task set with shared resources. Audsley *et al.* [11] presented a necessary and sufficient schedulability test for DM based on interferences, which is related to available computation time. Their technique does not work for EDF. However Ripoll *et al.* [12] proposed a sufficient and necessary schedulability test for EDF without shared resources. They also limited the duration of the test by the introduction of an upper bound. Our proposal is based on their results. We have added the possibility of using shared resources under the SR/EDF protocol and we propose an extension on Ripoll's test algorithm for feasibility.

The utilisation U_T of a task set T is given by:

$$U_T = \sum_{i=1}^{n} \frac{C_i}{T_i} \tag{3}$$

where C_i is the maximum run-time per period and T_i the length of the periodic interval. Furthermore we assume that overhead due to context switching is included in the execution times of the transactions and transactions are independent of each other.

Our analysis is based on the computation of the maximum *required load resolution* (RLR) for a set that must satisfy the deadline requirements of all its transactions. If for any interval $[t_1, t_2]$ with length $L = t_2 - t_1$ the RLR does not exceed the capacity of the processor, then the task set is called *feasible* else it is not. Baruah *et al.* [13] proved that, if for any L, all load offered during $[0, L]$ can be resolved before or at L, then the same can be concluded for any interval $[t, t+L]$. We will use this result and start our investigation at $t=0$. It will end at some suitable upper bound L, which will be determined in the following sections.

Our reasoning for the schedulability analysis of transactions under SR/EDF is based on similar arguments. We investigate the RLR during $[0, L]$, however under the assumption that blocking may occur.

In the following Section we present a method to bind the investigation length of the interval $[0, L]$ and we present an algorithm for feasibility analysis.

5.1 Schedulability of EDF without resources

Before we consider SR/EDF in more detail we first introduce an EDF schedulability test without blocking similar to the one presented by Ripoll *et al.* [12]. A function $H(L)$ is the maximum RLR function. It computes the amount of load, which must have been executed in the interval L such that all transactions from the set T are ready before their deadlines. All transactions are released periodically. The function $H(L)$ for a transaction set T of n transactions is defined as follows:

$$H(L) = \sum_{i=1}^{n} C_i \left\lfloor \frac{L + T_i - D_i}{T_i} \right\rfloor \tag{4}$$

where the $\lfloor (L + T_i - D_i)/T_i \rfloor$ denotes the maximum number of interferences of τ_i in L.

Fig. 1 shows $H(t)$ for four transactions $\{\tau_1, \ldots, \tau_n\}$. The up-arrows (↑) denote releases and the down-arrow (↓) deadlines. The small number above ↑ reflects the maximal load requirement. This number is equal to the length of the shaded portion, the run-time. Note that the deadlines are the only events at which $H(t)$ changes. The following theorem is from Liu and Layland [8].

Theorem 1: A set T scheduled by EDF is feasible if

$$\forall L > 0 : H(L) \leq L \tag{5}$$

In other words, $H(L)$ should not exceed the possibilities of the processor at any point in time. This condition is hard to check without further refinement since t may run to infinity.

A consequence of relation (5) is that a feasible schedule cannot exist if

$$L > H(L) \tag{6}$$

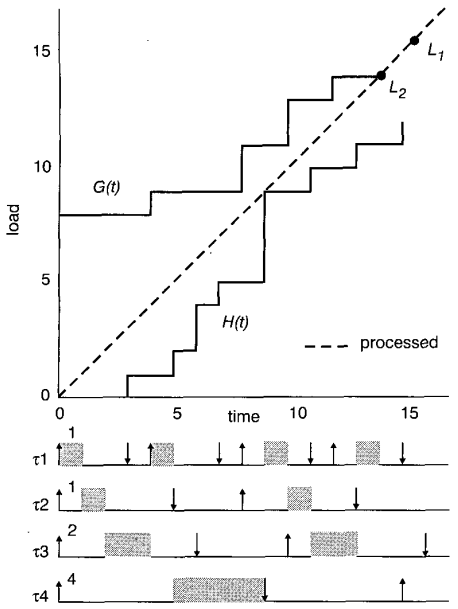

Fig. 1 *G(t), H(t), L_1 and L_2*

Using eqns. 3 and 4 Baruah *et al.* [13] found that condition (6) implies:

$$L \leq \frac{\sum_{i=1}^{n} i\left(1-\frac{D_i}{T_i}\right)}{(1-U_T)} = L_1 \tag{7}$$

and from eqn. 7 it follows that it makes no sense to search for infeasibility beyond an interval longer than L_1. Consequently we can refine condition (5) to

$$\forall L : 0 < L \leq L_1 : H(L) \leq L \tag{8}$$

A second approach is from Ripoll *et al.* [12]. They introduced the initial critical interval (ICI), also termed by others the longest non-idle interval (LNI). The length of this interval is determined by searching from $t=0$ for the earliest idle interval $t=L_2$. During the interval a maximum amount of load is offered. Interval [0, L_2] is finite if the total amount of offered load is equal to the capacity of the processor. It is not necessary to examine $H(t)$ beyond L_2. Denote the incremental amount of offered load by the function $G(t)$. Let

$$G(t) = \sum_{i=1}^{n} \left\lceil \frac{t}{T_i} \right\rceil C_i \tag{9}$$

and let L_2 be the first solution for $G(t)=t$. Our transaction set T becomes feasible iff

$$\forall L > 0, L \leq \min(L_1, L_2) : H(L) \leq L \tag{10}$$

Fig. 1 shows the function $G(t)$ and the point L_2.

5.2 Feasibility test algorithm for EDF

The following algorithm is used as an introduction for the SR/EDF test algorithm in Section 5.3. It determines whether statement (10) is true under EDF without shared resources. The algorithm uses an ordered list of events. An event is a tuple (*t, i, flag*) of time *t*, transaction-index *i*, and event-type *flag*. The event-type is either a release or deadline. Times are relative to $t=0$. The algorithm below evaluates the events ordered to time starting from $t=0$.

```
H=0; G=0; schedulable=unknown;
compute L;
while schedulable=unknown do
  (t,i,flag)=GetNextEvent;
  case flag of
  deadline:
    H=H+Ci;
    if H>t then schedulable=no fi;
  release:
    G=G+Ci;
    if G≤t or L≤t then schedulable=yes fi;
  esac;
od;
```

Algorithm 1 Feasability analysis for EDF

Algorithm 1 will be extended for the feasibility analysis of SR/EDF in the following Section. Note that in the algorithm above all release times r_i^j and deadlines d_i^j are 'points of interest'. This is also the case in SR/EDF. However, in SR/EDF blocking can influence RLR and the interval investigation length. Our algorithm in the following section will correct for this.

5.3 Schedulability of SR/EDF

As already explained in Section 4.3 SR allows the use of resources, which may cause blocking. Fortunately blocking is limited to only one transaction. In [6] Baker derived a schedulability condition for task sets with a correction for blocking. The task sets were ordered to their deadline intervals and a sufficient schedulability guarantee could be given if

$$\forall k = 1, \ldots, n : \left(\sum_{i=1}^{k} \frac{C_i}{D_i}\right) + \frac{B_k}{D_k} \leq 1 \tag{11}$$

where B_k is the maximum blocking experienced by the set of ordered tasks $\{1, \ldots, k\}$. However in condition (11) processor utilisation is overestimated because $D_i \leq T_i$ is substituted for the periods. Baruah *et al.* [13] corrected for $D_i \leq T_i$, and algorithm 1 in the previous section can perform the feasibility analysis without any change by offering the deadline events at D_i instead of at T_i. However, they did not take shared resources and blocking into account. Our SR/EDF algorithm will be extended to do so.

Before presenting our schedulability analysis we take a closer look at what happens under blocking in an example. Fig. 2 shows four transactions $\tau_1, \ldots, \tau_4$ ordered to deadline intervals D_i. All transactions start at $t=0$. From the specification we can generate release times $r_i^j(\uparrow)$ and deadlines d_i^j ($\downarrow$) for a transaction τ_i.

For the computation of the amount of blocking that τ_i^j experiences by τ_k^l we consider the conditions under which blocking may occur. τ_i^j experiences blocking by τ_k^l if

$$(\Delta_k \leq D_i) \wedge (r_k^l \leq r_i^j < d_i^j < d_k^l) \tag{12}$$

Note that the right-hand condition implies that $D_i < D_k$.

In Fig. 2 transactions are ordered to deadline interval. Transaction τ_1 does not share resources with any other transaction, so no blocking has to be taken into account at d_1^0. τ_2 may experience blocking from τ_4 over a shared resource if the blocking conditions (12) are met. To account for this potential blocking the RLR has to be corrected by C_4 at d_2^0. Also at d_3^0 this correction is needed since C_4 might have been executed before τ_2 and hence before τ_3. The last correction is at d_1^1. At d_4^0 only the load C_4 has to be added to RLR; no transaction can block τ_4.

In general a blocker τ_k can be active until its own deadline. If τ_i is the first transaction of $\{\tau_i, \ldots, \tau_{k-1}\}$, ordered to increasing deadline, then, at their deadlines, τ_k impose blocking corrections. In general there can be more potential blockers of which (due to SR/EDF) only one can be active. In such a case we have to account for the possible worst-case blocking. In the following Section we will consider blocking in more detail.

5.4 Interval blockers

Under SR/EDF blockers might add processor time to the RLR. For feasibility analysis we have to account for this. Let us denote $C_B(L)$ as the blocking component in interval L. The total amount of load in interval L is then given by $H(L) + C_B(L)$ where $H(L)$ is defined as in eqn. 4 and $C_B(L)$ is defined in this Section.

Theorem 2: For a feasibility analysis under SR/EDF the maximum number of blockers in L that has to be accounted for is *one* and for such a blocker τ_k it holds that $D_k > L$. The proof is given in the Appendix, Section 10.1.

Let $A(L) \subseteq \{\tau_1, \ldots, \tau_n\}$ be the set of active transactions in L. Then the set of potential blockers for τ_i is given by:

$$B_{A(L)} =$$

$$\{\tau_k | (\tau_i \in A(L)) \wedge (D_i < D_k) \wedge (D_i \geq \Delta_k) \wedge (D_k > L)\} \quad (13)$$

where $(D_i < D_k)$ denotes that τ_i has a higher priority than τ_k, $(D_i \geq \Delta_k)$ that τ_k can block τ_i because of the pre-emption level of τ_k and $(D_k > L)$ is a direct consequence of theorem 2. Since there can be only one blocker in L the maximum load is given by:

$$C_B(L) = \max\{C_k | \tau_k \in B_{A(L)}\} \quad (14)$$

With this result we can determine feasibility by the following theorem.

Theorem 3: Given a set of transactions T under SR/EDF, ordered to deadline interval, an interval [0, L], its maximal interference $H(L)$ and the maximum blocking load $C_B(L)$ of any possible blocker $\tau_k \in$ T with $D_k > L$. T is feasible if

$$\forall L \geq 0 : H(L) + C_B(L) \leq L$$

$$\text{with } H(L)$$

$$= \sum_{i=1}^{n} C_i \left\lfloor \frac{L + T_i - D_i}{T_i} \right\rfloor \quad (15)$$

The proof is given in the Appendix, Section 10.2.

Let $B_{\max}$ be the longest execution time of any possible blocker. The upper bound L_2 is not affected by blocking because its depends on the *offered* load. L_1 may have to be recomputed by adding $B_{\max}$ to the dividend in formula (7) (this is only needed for values of L_2 below the longest deadline of any blocker because, due to theorem 2, above this value blocking cannot influence feasibility analyses). From these arguments and from condition (13) our transaction set T is also feasible if

$$\forall L \leq \min(L_1, L_2) : H(L) + C_B(L) \leq L \quad (16)$$

5.5 Feasibility test algorithm for SR/EDF

The following algorithm tests the feasibility of a set of transactions T. It has the same structure as the algorithm for plain EDF in Section 5.1. The algorithm is extended with computation for the maximum value of any possible blocker in time C_B. A is the set of transaction active in L and $B = \{\tau_k | D_k > L\}$. C_B is straightforwardly derived from conditions (13) and (14) with $\tau_i \in A$, $\tau_k \in$ T\A and

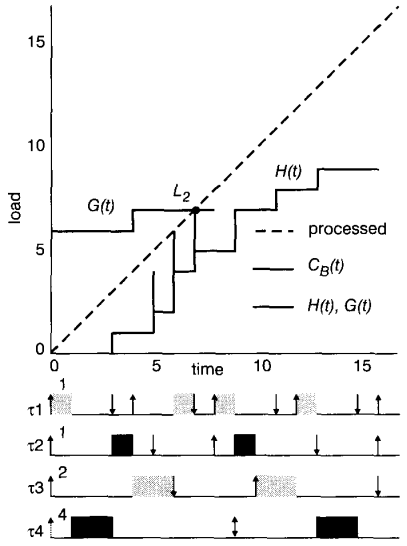

Fig. 2 $H_t + C_B(t)$

$C_B = \max\{C_k | (D_i < D_k) \wedge (D_i \geq \Delta_k)\}$. C_B depends on A and on static values only and can be computed offline for the n different values of A.

```
H=0; G=0; A=Ø; schedulable=unknown;
compute L1;
while schedulable=unknown do
  (t, i, flag)=GetNextEvent;
  case flag of
  deadline:
    H=H+Ci;
    A=A∪τi;
    CB=MaxBlocker(A);
    if H+CB>t then schedulable=no fi;
  release:
    G=G+Ci;
    if G≤t or L1≤t then schedulable=yes fi;
  esac;
od;
```

Algorithm 2 Feasability analysis for SR/EDF with use of shared rescources

The number of events that must be processed is hard to predict in advance and can vary considerably. The most complex operation is the computation of C_B. Its complexity is of $O(n)$ where n is the number of transactions.

If minimal blocking is an issue then transaction are too pessimistic. In the following section we consider some refinements and their consequences.

Our main contributions to feasibility analyses can now be evaluated. They are: (a) a new way of accounting for blocking under SR/EDF, based on formula (13) and (b) to relate the work of Ripoll *et al.* [12] in which neither SR nor blocking is addressed, and (c) to map formula (13) and (16) to the straightforward algorithm 2.

5.6 Further refinements

As stated in Section 4.3, refinements such as multiple-unit resources or nested critical sections are possible. Nested critical sections (NCSs) can be used instead of transactions. Resource usage is then limited to the length of the longest critical section. Denote this length as S_i with

$S_i \leq C_i$. Using S_i instead of C_i may limit blocking considerably. Furthermore under NCSs the behaviour of the pre-emption level can be made more dynamic than under transactions: the original pre-emption level of a task invocation τ_i^j can be dropped when resources are obtained and raised when they are released again. If the original pre-emption level is D_i and the first resource required is R then the level may drop to min (D_i, D_R). Here D_R is the floor as defined in Section 4.2 in eqn. 1. The pre-emption level may further drop to ultimately $\Delta_i = \min\{D_i, D_R | R \in R_i\}$, which is exactly the value used in the ceiling protocol in Section 4.2 in eqn. 2. By releasing resources the pre-emption level may be raised to previous levels again. In fact we have reconstructed Baker's original SR [6] in which tasks with NCSs have a more dynamic behaviour of their pre-emption level then transactions.

For feasibility analysis this dynamic behaviour does not change the functions $G(t)$ and $H(t)$ since the offered load and the required load resolution cannot change due to nested resource usage without blocking. However substituting S_i for C_i in the computation of $C_B(L)$ limits the value of this function. In our test algorithm for SR/EDF this can influence only the function *MaxBlocker* because blockers are associated with S_i instead of C_i.

Note that for further refinements, blocking takes not necessarily as long as the longest critical section S_i, because the longest critical section could be associated with a resource with a low ceiling. It is, however, beyond the scope of this paper to consider these refinements in detail.

A disadvantage of the refinements is that they introduce some run-time overhead. Resources must be claimed and released explicitly and the administration must be updated more often.

In conclusion we can state that changing from transactions to nested critical sections hardly affects the algorithm for feasibility analyses. Scheduling itself, however, has to be adapted. This is not really a serious point because we can use Baker's original algorithm [6] or a direct variant of it.

6 Implementation and tests

Scheduling strategies for transactions under EDF and CP have been simulated in Python [14]. Variants of both have been added to several operating system kernels such as Nemo [15] and Nemesis [16] where the protocols had to be integrated with their domain scheduler. Domain scheduling combined with EDF and CP is problematic and is certainly not a good idea for HRT. However, we have experimented with a multimedia application and measured an administration overhead below 5%. A framework for scheduling experiments with CP and SR/EDF has been added to Inferno and is described in [17]. A similar activity is going on under RT-Linux [18] in which we implemented a prototype of SR/EDF. Our test application implements six transactions, each handling a separate video stream. There is one 50 Hz stream, two 25 Hz streams and three 20 Hz streams. The processor utilisation is about 40% on a Pentium II (233 MHz). The measured overhead per task switch is 13 μs. The administration overhead for the protocol increases linearly with about 0.5 μs per transaction. For a fully utilised processor the average scheduling overhead would be about 1% and, if using fine-tuned memory block transfers, 20 high quality 50 Hz streams would require a scheduling overhead below 3%. On a Pentium Pro (155 MHz) and on an Intel 486 (66 MHz) we ran tests with shared resources and found for the (preemption time, protocol overhead per transaction) (35 μs, 1 μs) and (105 μs, 3 μs) respectively. We conclude that SR/EDF under transactions has a low overhead.

From these test we conclude that typical multimedia application can be scheduled with the proposed algorithms. Algorithm 2 can immediately be used for quality of service analyses for flexible admission of multimedia tasks. Blocking can occur if shared communication buffers are used. In general blocking is not a critical issue in multimedia. Because of these arguments and also because of the arguments in Section 4.4 we believe that SR/EDF is a perfect candidate for scheduling multimedia. Test results and statistics for schedulability analysis are not yet available, but are planned in the near future.

7 Conclusions

We have investigated two real-time scheduling policies, both adapted to multimedia requirements but also suited to hard real-time purposes. These policies are based on the principle of (a) real-time transactions, which are scheduled by (b) the earliest deadline first rule, and (c) enriched by selected inheritance strategies. From these ingredients we have constructed dynamic transaction scheduling variants of the ceiling protocol and the stack resource protocol and have evaluated their properties. These protocols can be constructed with a low administration overhead. This is due to the orthogonality of the ingredients, which enable a systematic implementation. Among others, mutual exclusive usage of shared resources is guaranteed by the aforementioned ingredients (a) to (c); no additional synchronisation primitives are needed. Consequently the scheduling is easy to use and without much effort quickly adapted to the requirements of the applications. On a typical state of the art processor (running under RT-Linux) with scheduling adapted to SR/EDF, we measured a scheduling overhead of $16 + 0.5n$ μs for n transactions, without special effort at optimisation. With these results it is possible to schedule 20 typical video streams with a scheduling overhead of about 2.5%. It is not likely that SR/EDF scheduling will become a bottleneck in multimedia systems.

An improved algorithm for feasibility analysis of SR/EDF is presented. The algorithm accounts for blocking and is simple to implement. It can be used for static and dynamic hard real-time systems. In particular it is useful for online quality of service analysis in multimedia systems. These properties make the stack resource protocol favourite for further analysis

7 Acknowledgements

Thanks are due to Ties Bos and Ferdy Hanssen for their contributions during the Wednesday afternoon sessions and for comments on an early version of this manuscript.

8 References

1 SHA, L., RAJKUMAR, R., and LEHOCZKY, J.P.: 'Priority inheritance protocols: an approach to real-time synchronization', *IEEE Trans. Comput.*, 1990, **39**, (9), pp. 1175–1185

2 TAKADA, H., and SAKAMURA, K.: 'Real-time synchronization protocols with abortable critical sections', Proc of 1st Intl Workshop on Real-Time Computing Systems and Applications (RTCSA), Dec 1994, pp. 48–52

3 LAM, K.Y., and NG, J.K.: 'A conditional abortable priority ceiling protocol for scheduling mixed real-time tasks', *to appear in J. Syst. Architecture*, 1999

4 RAJKUMAR, R.: 'Synchronization in real-time systems, a priority inheritance approach', Kluwer Academic Press, 1991
5 SHA, L., RAJKUMAR, R., and SATHAYE, S.: 'Generalised rate-monotonic scheduling theory: a framework for developing real-time systems', *Proc. IEEE*, **82**, (1), 1994, pp. 68–82
6 BAKER, T.P.: 'Stackbased scheduling of real-time processes', *J. Real-time Syst.*,, 1991, **2**, pp. 67–99
7 BUTTAZZO, G.C., and LIPARI, G.: 'Scheduling analysis of hybrid real-time task sets', Proc. of the 9th IEEE Euromicro Workshop on Real-Time Systems, 11–13, June 1997 pp. 200–206
8 LIU, C.L., and LAYLAND, J.W.: 'Scheduling algorithms for multiprogramming in a hard real-time environment', *J. ACM*, **20**, (1), Jan. 1973, pp. 46–61
9 MOK, A.K.: 'Fundamental design problems of distributed systems for the hard real-time environment', 1983 PhD Thesis, MIT
10 JEFFAY, K., STANAT, D.F., and MARTEL, C.U.: 'On non-preemptive scheduling of periodic and sporadic tasks', Proc. of the 12th IEEE Real-Time Sys. Symp, 1991, pp. 129–139
11 AUDSLEY, N.C., BURNS, A., RICHARDSON, M.F., and WELLINGS, A.J.: 'Hard real-time scheduling: the deadline monotonic approach', Proc. of IEEE Workshop on Real-time Operating Systems and Software, May 1991, pp. 133–137
12 RIPOLL, I., CRESPO, A., and MOK, A.K.: 'Improvement in feasibility testing for real-time tasks', *J. Real-time Syst.*, 1996, **11**, (1), pp. 19–39
13 BARUAH, S., ROSIER, L., and MOK, A.: 'The preemptive scheduling of sporadic, real-time tasks on one processor', Proceedings of the 11th Real-Time Systems Symposium, 1990, pp. 182–190
14 VAN ROSSUM, G.: 'Python reference manual', Dept. CST, CWI, Amsterdam 1994
15 JANSEN, P. G., and WIJGERINK, E.: 'Flexible real-time scheduling of continuous media streams: a multimedia experiment', Multimedia 96 Conference, Yoko-hama, Japan 1996, pp. 323–330
16 LESLIE, I., MCAULEY, D., BLACK, R., ROSCOE, T., BARHAM, P., EVERS, D., FAIRBAIRNS, R., and HYDEN, E.: 'The design and implementation of an operating system to support distributed multimedia applications', J. SAC paper, updated June 1997
17 BOS, M.: 'Real-time scheduling in Inferno', Oct. 1997 Master's thesis, Univ. of Twente
18 YODAIKEN, V.: 'Real time Linux', 1998 http://luz.cs.nmt.edu/rtlinux/

10 Appendix

10.1 Proof of theorem 2

Consider the interval $[t_1, t_2]$ with $L = t_2 - t_1$ such that t_2 is the earliest time a deadline is missed and t_1 is the last time before t_2 such that there are no pending transactions τ_k^l with $r_k^l \leq t_1 < d_k^l$.

A potential blocker τ_k^l for this interval has the following properties:

$$r_k^l \leq t_1 < t_2 < d_k^l \tag{17}$$

This is due to the fact that (a) if $t_1 < r_k^l < t_2 < d_k^l$ then t_1 must be chosen to be equal to r_l^k and (b) $r_k^l < t_1 < d_k^l < t_2$ contradicts with the requirements of the interval.

During the interval $[t_1, t_2]$ there can be only one blocker that contributed $C_B(L)$ to the RLR at t_2 or no blocker at all. This is due to the fact that two blockers τ_1 and τ_2 lead to the following contradiction: if both τ_1 and τ_2 are blockers than both must have pre-emption deadlines $\Delta_1 \leq L < D_1$ and $\Delta_2 \leq L < D_2$ since they must have inherited their Δ_1 and Δ_2 from transactions within the interval L. From this it follows that $(\Delta_1 < D_2) \wedge (\Delta_2 < D_1)$. If both are active then one must have pre-empted the other, which implies $(\Delta_1 > D_2) \vee (\Delta_2 > D_1)$, a contradiction. Consequently they may not pre-empt each other and therefore there can be one blocker only.

To compute the maximum value of $C_B(L)$ we have to consider any possible blocker of any active transactions in $[t_1, t_2]$, however, with the restriction that the blocking deadline interval may not be smaller than L. A consequence is that there are no blockers beyond $L > D_{\max}$ where $D_{\max}$ is the longest deadline interval of all transactions.

10.2 Proof of theorem 3

Let $H(t, L)$ be RLR by maximum interferences $H(L)$ starting from t and let $C_B(t, L)$ be the maximum blocking load $C_B(L)$ starting from t. Baruah *et al.* [13] proved that for EDF with deadlines shorter than periods but without resources:

$$\forall t \geq 0, L \geq 0 : \{H(0, L) \leq L\} \Rightarrow \{H(t, L) \leq L\} \tag{18}$$

The argument they used is that there can never be less interference in the interval $[0, L]$ than any interval $[t, t+L]$ for $t > 0$. We can use these results for SR/EDF and extend their arguments, accounting for blocking also. There is never less blocking in $[0, L]$ than in $[t, t+L]$ since by condition (13) the amount of blocking cannot decrease if the maximum number of interferences increases:

$$\forall t \geq 0, L \geq 0 : \{H(0, L) \geq H(t, L)\} \Rightarrow \{C_B(0, L) \geq C_B(t, L)\} \tag{19}$$

From conditions (18) and (19) we may conclude

$$\forall t \geq 0, L \geq 0 : \{H(0, L) + C_B(0, L) \leq L\} \Rightarrow \{H(t, L) + C_B(t, L) \leq L\} \tag{20}$$

Therefore a set T is feasible if

$$\forall L \geq 0 : H(0, L) + C_B(0, L) \leq L \tag{21}$$