Available online at www.sciencedirect.com
ScienceDirect
ELSEVIER
The Journal of Systems and Software xxx (2007) xxx–xxx
The Journal of Systems and Software
www.elsevier.com/locate/jss

Improving the schedulability of soft real-time open dynamic systems: The inheritor is actually a debtor

Rodrigo Santos [a,*], Giuseppe Lipari [b], Jorge Santos [a]

[a] *Dep. Ing. Eléctrica y Computadoras, Universidad Nacional del Sur, Avda. Alem 1253, 8000 Bahía Blanca, Argentina*
[b] *RETIS Lab, Scuola Superiore Sant'Anna, Piazza Martiri della Liberà 33, 56127 Pisa, Italy*

Received 27 September 2006; received in revised form 11 July 2007; accepted 11 July 2007

Abstract

This paper presents the Clearing Fund Protocol, a three layered protocol designed to schedule soft real-time sets of precedence related tasks with shared resources. These sets are processed in an open dynamic environment. Open because new applications may enter the system at any time and dynamic because the schedulability is tested on-line as tasks request admission. Top-down, the three layers are the Clearing Fund, the Bandwidth Inheritance and two versions of the Constant Bandwidth Server algorithms. Bandwidth Inheritance applies a priority inheritance mechanism to the Constant Bandwidth Server. However, a serious drawback is its unfairness. In fact, a task executing in a server can potentially steal the bandwidth of another server without paying any penalty. The main idea of the Clearing Fund Algorithm is to keep track of processor-time debts contracted by lower priority tasks that block higher priority ones and are executed in the higher priority servers by having inherited the higher priority. The proposed algorithm reduces the undesirable effects of those priority inversions because the blocked task can finish its execution in its own server or in the server of the blocking task, whichever has the nearest deadline. If demanded, debts are paid back in that way. Inheritors are therefore debtors. Moreover, at certain instants in time, all existing debts may be waived and the servers are reset making a clear restart of the system. The Clearing Fund Protocol showed definite better performances when evaluated by simulations against Bandwidth Inheritance, the protocol it tries to improve.

Keywords: Open systems; Soft real-time; Scheduling

1. Introduction

In the classical definition, Real-Time Systems are those in which results must be not only correct from an arithmetic–logical point of view but also produced before a certain instant called *deadline*. Because of that it is said that the system has time constraints. If no deadline can be missed, the system is said to be *hard real-time* as opposed to *soft real-time*, in which some deadlines may be missed. Scheduling theory addresses the problem of determining the necessary and sufficient conditions that a real-time system must meet in order that no deadline is missed (hard systems) or that as few as possible are missed (soft systems).

In some cases, tasks' parameters (execution and interarrival times, and deadlines) are not exactly known in advance. Because of this uncertainty, it may happen that a task exceeds its expected execution time or has a shorter interarrival time. In that case, interference with other tasks must be prevented. This can be done by the provision of temporal isolation implemented by means of dedicated constant bandwidth servers, each one serving only one application (Deng and Liu, 1997; Lipari and Butazzo, 2000; Caccamo and Sha, 2001) (a server is an entity used by the scheduler to reserve a fraction of processor-time to a task Marzario et al., 2004). In that way, a task can use only the time assigned to it and cannot use time assigned to others tasks. When tasks do not share resources, they

Q1 * Corresponding author. Tel.: +54 291 4595181; fax: +54 291 4595154.
E-mail addresses: ierms@criba.edu.ar (R. Santos), lipari@sssup.it (G. Lipari), iesantos@criba.edu.ar (J. Santos).

0164-1212/$ - see front matter © 2007 Published by Elsevier Inc.
doi:10.1016/j.jss.2007.07.004

are said to be spatially isolated. If they are not spatially isolated, undue priority inversions may take place and it is desirable not only to bound their duration but also to reduce the undesirable effects suffered by the task which execution was postponed in spite of having a higher priority. The Clearing Fund Protocol makes all this in the frame of an Open Dynamic System, ODS. It is open because new applications may enter the system at any time and dynamic because the acceptance test, even if the tasks' parameters are not exactly known, is performed on-line.

The main load is composed of sets of periodic or quasi-periodic preemptible soft real-time tasks related by precedence (STRP). Each STRP can be executed independently of the others by a series of linearly ordered computational steps on the single processor; in this way, logical concurrency is achieved. The tasks of one STRP may share resources with tasks of the same or of different STRPs. Hard real-time tasks and non-real-time tasks may, under certain circumstances, be admitted. After this introduction, related work is discussed in Section 2; in Section 3, the System Model and in Section 4 the Clearing Fund Protocol are respectively presented; Section 5 is devoted to the management of hard real-time STRPs sharing the processor with soft real-time STRPs; in Section 6, experimental results are analysed. Finally, conclusions are drawn in Section 7.

2. Related work

The problem of scheduling real-time systems with resource constraints is addressed, for instance, in Lipari and Butazzo (2000). When resources are shared, a lower priority task may block the execution of a higher priority one. This is called the priority inversion problem. In Sha et al. (1990), the authors proposed two protocols to deal with it, the Priority Inheritance and the Priority Ceiling protocols. Both work under Rate Monotonic Scheduling, a fixed priority discipline in which tasks are ordered by decreasing rates or, what is the same, by increasing periods. The Stack Resource Policy (Baker, 1990), is a concurrency control protocol that bounds the priority inversion phenomenon in static as well as in dynamic priority systems, expanding the previous results. However, it does not address the problem of scheduling ODSs.

Many algorithms designed to deal with aperiodic and sporadic tasks have been proposed. The Polling, Deferrable (Strosnider et al., 1995), Priority Exchange (Sprunt et al., 1988) and Sporadic (Sprunt et al., 1989) servers, are some examples. All these algorithms have in common that they work with fixed priorities, in particular Rate Monotonic. They are used in hybrid systems where there are hard real-time periodic tasks and non-real-time aperiodic ones. The servers are meant to reduce the mean response time of aperiodic tasks. In order to do it, the hard real-time system must be completely specified, in the sense that tasks' execution times, periods and deadlines must be known. This restriction prevents their use in ODSs, in which tasks' parameters are not exactly known. Some other algorithms, Slack Stealing (Ramos-Thuel and Lehoczky, 1994) and k-schedulability (Santos et al., 2004), have a better performance at a higher computational cost but, again, they cannot be applied to the ODSs case.

Other related works addressing the problem are Constant Bandwidth Servers with shared resources (Caccamo and Sha, 2001), Resource Kernel (Rajkumar et al., 2000), Cooperative Scheduling Server (Saewong and Rajkumar, 1999) and Constant Utilization and Total Bandwidth Servers (Deng and Liu, 1997), but they require a complete knowledge of the sistem's specification. This makes their use impossible in applications in which the parameters are not exactly known and resources are shared, which is precisely the problem this paper addresses.

The Bandwidth Inheritance algorithm, BWI, proposed in Lipari et al. (2004), extends the Constant Bandwidth Server, CBS, algorithm (Abeni and Buttazzo, 1998) to real-time tasks not spatially isolated, by using a technique derived from the Priority Inheritance Protocol (Sha et al., 1990). However, BWI presents some drawbacks. One problem is that a blocking task can capture most of the bandwidth of a blocked task, causing long priority inversions without paying any penalty. Besides, the computation of sets of precedence constrained tasks is not considered. In Santos and Lipari (2003) an Extended BWI was presented to deal with precedence constrained tasks and it corrected, in some way, the bandwidth distribution by penalizing the blocking server. This is not enough because the blocked server never receives back the time it has spent servicing inherited tasks and, consequently, it may miss deadlines that could otherwise be met.

The scheduling of resource-sharing precedence-related tasks is addressed here by means of the three layered Clearing Fund Protocol, CFP. Top down, the three layers are the Clearing Fund, the BWI, and the CBS algorithms. Temporal isolation is provided between spatially isolated sets of tasks by separate constant bandwidth servers. However, priority inversions may take place between groups of sets sharing resources. Their effect, however, is reduced by the possibility of executing the blocked task either in its own server or in the server of the blocking task.

The basic idea consists in the incorporation of a balancing fund similar to the clearing realized by banks at the end of the day. When a task of low priority blocks a task of higher priority, it migrates to the server of the blocked task and it is executed at the priority of the blocked task. Then the blocking server contracts a processor-time debt with the blocked server that, if the lender demands its payment, will have to be paid. Moreover, the mechanism reduces the postponements in the deadlines of the servers, which is a characteristic of the CBS algorithm. The main advantage is that the effects of priority inversions due to the lack of spatial isolation are reduced because the blocked STRP can finish its execution in its own server or in the server of the blocking STRP. This gives the option of choosing the one with the nearest deadline making possible to meet deadlines that would otherwise be missed. Finally a prescription law for debts is incorporated: at certain instants in the evolution of the system, called *singularities* (Santos et al., 2004), the last pending task released before the singularity has completed its execution. From the point of view of pending executions, the system is as in the initial state. Servers may update their parameters (partially consumed budgets and postponed deadlines) and a reinitialization of the system takes place. Consequently, debts may be forgiven. Although in an ODS paradigm, hard schedulability guarantees are not the main target, the CFP is also able to provide them although at the expense of certain limitations on the acceptance of soft applications.

The main ideas of this article have been presented at the 10th International Conference on Real-Time Computing Systems and Applications (Göteborg and Sweden, 2004) (Santos et al., 2004). The presentations, however, were only preprinted for the conference's attendants and are not commercially available nor can they be downloaded from the conference's web page. Besides, this paper enhances the original presentation by adding a second protocol using the Constant Bandwidth Server Algorithm with Hard Reservation as the first layer. This modified version of the CBS Algorithm was designed to cope with some problems that may arise when acyclic multimedia or interactive tasks are processed.

3. System model

The system model will now be described. The CFP is slated to schedule soft real-time STRP defined as sets of periodic or quasi-periodic preemptible soft real-time tasks related by precedence and sharing resources. Each set can be executed independently of the others by a series of linearly ordered computational steps. However, a variant of the protocol is proposed to handle also acyclic tasks, understood as non periodic tasks, active during long intervals of time. In certain applications, and although non-real-time in a strict sense, they are better processed if the rate of execution does not vary much along time. Finally, although not as efficiently as other methods, the CFP can also handle hard real-time tasks.

Since tasks are periodic, they can be viewed as a stream of jobs (or instances) requesting the execution of a computation on a shared processor. τ_{ip} shall denote the pth instantiation of τ_i. In the model, time is considered to be slotted and the duration of one slot is taken as the indivisible unit of time. Slots are notated t and numbered 1, 2, ... The expressions *at the beginning of slot t* and *instant t* mean the same. The execution of a task can be interrupted by a task of higher priority. This preemption is assumed to be possible only at the beginning of slots. An *empty* slot is a slot in which there is no task ready to be executed; the processor, then, goes idle. A *singularity*, s, is a slot in which all real-time tasks released in $[1, (s\text{-}1)]$ have been executed (Santos et al., 2004). Note that s-1 can be either an empty slot or a slot in which a last pending task completes its execution. s is a singularity even if at $t = s$, other tasks are released. The concept of singularity is similar to the end of a busy period as defined in Palencia and Gonzalez Harbour (1999).

When, in order to be executed, a task τ_j needs data produced by other task τ_i, a precedence relation, notated $\tau_i \prec \tau_j$, is established and determines a partial ordering of the tasks. If $\tau_i \prec \tau_j$ and there is no task τ_l such that $\tau_i \prec \tau_l \prec \tau_j$, τ_i and τ_j shall be called predecessor and successor, respectively.

A digraph G can be associated to the computation (Ramammritham, 1990). Each node of the graph represents a task and there is a directed arc from the node representing τ_i to the node representing τ_j only if they are predecessor and successor respectively.

As usual, a root is a task with no predecessor and a leaf is a task with no successor. The level of the task in the graph is its minimum distance to the root, measured in arcs. Although they may share resources, tasks of one STRP are not precedence-related to tasks of other STRPs. A STRP starts always at a root node and ends at a leaf node. Since the tasks that form a STRP are periodic, the STRP itself is periodic and will have successive instantiations. In what follows it will be assumed that all the tasks of a STRP have the same period which will also be the period of the STRP.

The STRP starting at node g shall be notated γ_g. The cardinality of the STRP, i.e., the number of tasks in the STRP, shall be notated Υ_g. Single independent tasks have $\Upsilon_g = 1$ (unitary STRPs).

Example. The set of tasks $\{\tau_1, \tau_2, \ldots \tau_{10}\}$ in Fig. 1 are arranged in three STRPs

$$\gamma_1 = \{\tau_1, \tau_2, \ldots \tau_6\} \quad \Upsilon_1 = 6$$
$$\gamma_7 = \{\tau_7, \tau_8, \tau_9\} \quad \Upsilon_7 = 3$$
$$\gamma_{10} = \{\tau_{10}\} \quad \Upsilon_{10} = 1$$

Tasks may share physical or logical resources, notated R_k. A shared resource may be, for instance, a section of memory. If the contents of the shared section are not

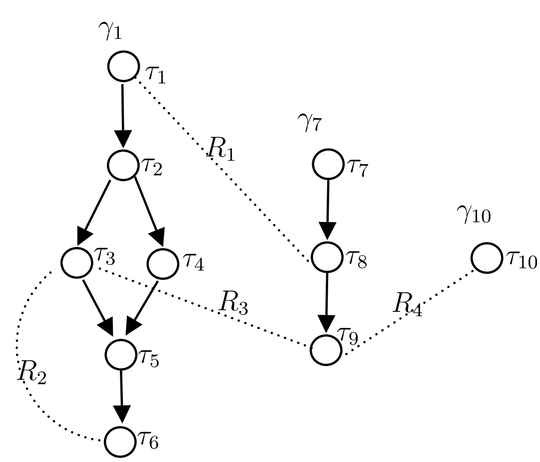


Fig. 1. $\gamma_1, \gamma_7, \gamma_{10}$, execute at servers S_1, S_7 and S_{10}, respectively. Nodes and arcs indicate tasks and precedence, respectively. Dashed lines indicate shared resources.

modified by the accessing tasks (e.g., a read-only portion of code), a mutual exclusion problem does not exist. However, if the contents, for instance data, can be modified, access must be serialized in order to maintain consistency. The section is then called critical and its access is controlled by the use of mutual exclusion semaphores with wait and signal operations, notated $P(R_k)$ and $V(R_k)$, respectively. When a task enters a critical zone, the semaphore is locked and no other task can enter the zone. Successive accesses to critical sections can be nested. In that case, it is assumed that they are always properly nested, in the sense that it is only possible to have sequences of the type $P(R_a)\cdots P(R_b)\cdots V(R_b)\cdots V(R_a)$. The times of use of a critical section may be different for different tasks.

Example. In Fig. 1, τ_1 and τ_8 share resource R_1; τ_3 and τ_6 share resource R_2; τ_3 and τ_9 share resource R_3; τ_9 and τ_{10} share resource R_4.

The relative deadlines (absolute deadline slot minus absolute release slot) of all the tasks of a STRP are assumed to be equal among them and equal to or greater than the period of the server.

Two STRPs that share a common resource are said to have an interference relation. The interference relation is reflexive, symmetric and transitive and it is therefore an equivalence relation partitioning the set of STRPs in equivalence classes. A STRP belonging to a class must not interfere with STRPs of other classes. Therefore, temporal isolation (in the CBS sense) must be guaranteed between STRPs of different classes.

The priority discipline to be used is Earliest Deadline First, EDF. It is a dynamic priority discipline in which the task to be executed at each unit of time is the one nearer its deadline. In Liu and Layland (1973) it has been formally proved that EDF is optimal in the sense that if a real-time system is not schedulable under EDF is not schedulable at all. The schedulability test boils down to verify that the sistem's utilization factor is less than, or equal to, 1. Because of its simplicity it may be performed on-line when deciding if a task may be accepted or not. If, because the tasks' parameters are not exactly known in the accepting process, while executing, the task actually exceeds the processing time alloted to it, its deadline may be missed but it certainly will not affect the execution of other tasks, isolated by their own servers.

4. The Clearing Fund Protocols

The Clearing Fund Algorithm, CFA, is the third layer of a three layered protocol. The second layer is the BWI algorithm. There are two versions of the first layer, the CBS algorithm: plain CBS and CBS with hard reservation, CBSHR. Consequently, there are two versions of the protocol, CFP and CFPHR. As illustrated in Fig. 2 the protocols run in an Open System Architecture.

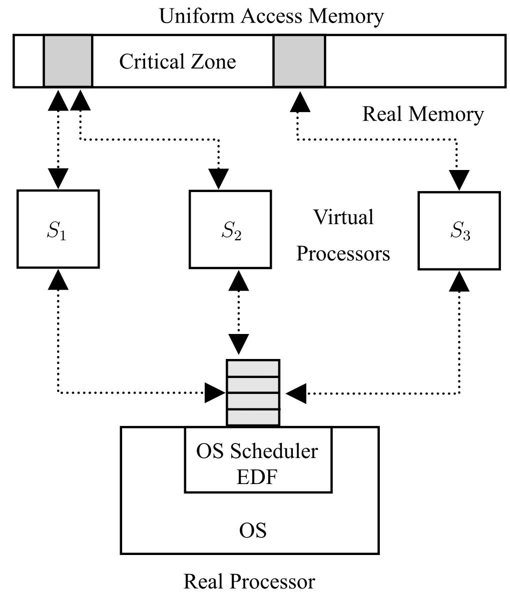


Fig. 2. Open System Architecture.

4.1. Open system architecture

The Open System Architecture resembles the one proposed by Deng and Liu (1997), in which:

(1) Each CBS is considered to be a virtual processor and holds only one STRP.
(2) Each virtual processor has its own bandwidth, which is the product of its utilization factor times the bandwidth of the real processor, e.g., in Mflops.
(3) There are two distinct schedulability architectural levels: STRPs must be scheduled in the virtual processor and virtual processors must be scheduled in the real processor.
(4) In case that a level in the STRP (its distance to the root) has more than one task, the tasks that do not share resources with other tasks outside the STRP are executed first; the others execute in reverse order of their declared execution time. The rationale behind this rule is that tasks imposing shorter delays to other tasks are taken out of the way first.
(5) Tasks can share resources with tasks of the same or of different STRPs, i.e., executed in the same or in different virtual processors, respectively.

4.2. How CFP improves BWI

A well known phenomenon of priority inversion may take place in the case of tasks sharing a resource. It happens when a lower priority task is executed instead of a higher priority one because of the lower priority task having accessed first, and not being able to release, the shared resource. Note that inversions may exist in a priority driven system regardless of the scheduling discipline used. In inheritance type protocols, the lower priority task inherits the priority of the higher one which is then said to be blocked. Combined with a proper use of semaphores guarding the entrance to the shared resources, the time that a task can be blocked may be bounded, and that is precisely what the BWI algorithm does. However, although BWI provides a bounding mechanism, it does not provide a compensating one such that the lower priority task (inheritor) pays back to the higher priority one the time spent in executing under a priority inversion.

CFP improves BWI by converting the inheritor into a debtor; it does so by incorporating a new queue and a dynamic variable to each server. The variable is incremented by one for each slot that the STRP is executed outside the server contracting a debt, and decremented by one for each slot used to pay back the debt. The blocked STRP that has been postponed in its execution, is enqueued in the high priority queue of the server of the blocking STRP. Depending on which is the nearest deadline, it may be executed in its own server or in the server of the blocking STRP until the debt is paid or prescribes in a singularity. A singularity means that the last pending task released before the singularity has completed its execution; collecting debts is no longer necessary for the lenders and debts may therefore prescribe and be cleared.

In the CFA, each STRP is assigned to a dedicated periodic server. Each server, S_g, is specified by two parameters: Q_g and P_g, where Q_g is the budget, time available for execution, and P_g is the period. The server is allowed to execute at least during Q_g out of every P_g units of time. The period of the server is assumed to be shorter or equal to the period of the STRP it hosts.

The relation between Q_g and P_g is the utilization factor of the server. Since there is only one real processor in the system, the total utilization factor can not be greater than 1, that is $\forall i \quad \sum Q_i/P_i \leqslant 1$.

In the CFA, the servers have three dynamic variables: q_g, δ_g and $v_{g,f}$. The first variable is the current available budget and keeps track of the portion already consumed. The second variable, δ_g, is the server's absolute deadline, that is the number of the slot or, what is the same, the instant before which the task is expected to be processed. The third variable, $v_{g,f}$, counts the borrowed budget and keeps track of the debts contracted by the server S_g with the server S_f of higher priority.

Each server has two STRP queues, each one with a different priority. The higher priority one is for external STRPs; it holds the blocked STRP after an inheritance has taken place. The lower one is for the STRP allocated to the server. Initially q_g is equal to Q_g and is decremented by one for each unit of time the server executes. When it reaches zero, the budget is recharged and the deadline is postponed.

The CBS presents some drawbacks when serving acyclic tasks, understood as non-periodic non-real-time tasks that are active for long intervals of time, covering therefore many periods of the sever. This is particularly negative for multimedia or for interactive tasks since it may lead to a loss of quality of service and interactivity. The main cause is the fact that in CBS, servers, recharged immediately after being exhausted, can be used immediately within the same period if the server's deadline, although postponed, is still the earliest. This leads to a temporal over execution that may be followed by a starvation altering the rate of the multimedia or the interactive application.

IRIS (Marzario et al., 2004) was proposed to solve this problem. Besides the budget and the period parameters of the server, a recharging time, r_i, is set in such a way that under no circumstances the server that exhausts its budget may execute again until r_i. In IRIS, r_i is the beginning of the next period of the server and in the interval between exhausting its budget and been able to use it again, the server is said to be in the recharging state. The method is called hard reservation and the modified CBS algorithm is notated CBSHR.

In CBSHR, servers have three states: idle (no task is demanding execution within the server), active (at least one task is demanding execution) or recharging (there is at least one task demanding execution but the server has consumed his budget). It may happen, however, that being no active servers in the system and some of them being in the recharging state, deadlines are missed. To avoid this, a simple rule to advance the activation instant is included.

Example. Suppose there is only one server in the system with parameters $(2,6)$ and a task requires an execution time of 4. Following CBS, the temporal evolution of the system will be the one shown in Fig. 3a. Following CBSHR instead, the evolution will be as shown in Fig. 3b. As can be seen, CBS produces an over execution followed by a starvation while CBSHR keeps processing the application at a constant rate.

4.3. Rules

The rules of the two versions of the Clearing Fund Protocol will now be presented.

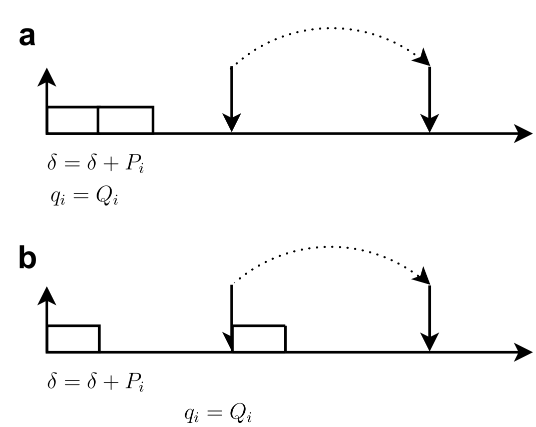


Fig. 3. CBS with and without Hard Reservation.

4.3.1. CFP (CFA/BWI/CBS)

Rules A–C correspond to CBS, rules D and E correspond to BWI and rules F to I are CFA specific:

A. When at time a_{gh} the hth instantiation of γ_g arrives, the server S_g checks the condition $q_g \leqslant Q_g(\delta_g - a_{gh})/P_g$. If it holds, the current pair (q_g, δ_g) stands, otherwise a new pair $q_g := Q_g$ and $\delta_g := a_{gh} + P_g$ is computed to be used.
B. If server S_g executes for Δt units of time, the budget is decremented accordingly, that is $q_g := q_g - \Delta t$.
C. Server S_g is allowed to execute while $q_g > 0$. When the budget is depleted, a new pair (q_g, δ_g) is computed: the absolute deadline is postponed to $\delta_g = \delta_g + P_g$ and the budget is recharged to $q_g := Q_g$. Since the scheduling deadline has changed, the EDF queue of the servers may have to be reordered and preemptions may occur.
D. If task $\tau_i \in \gamma_g$ is blocked when accessing a resource R_k that is locked by task $\tau_j \in \gamma_l$, then τ_j is added to the queue of the server S_g. If, in turn, τ_j is currently blocked on some other resource, then the chain of blocked tasks is followed, and server S_g adds all the tasks in the chain to its queue, until it finds a non blocked task. In this way, each server can have tasks belonging to more than one STRP, but only one of these tasks is not blocked.
E. If there is more than one task blocked in R_k, one of them is unblocked when the resource is released. All the servers that added it to their list must discard it.
F. After each singularity s, when a new instance of γ_g arrives, the server S_g updates its variables: $q_g := Q_g$ and $\delta_g := a_{gh} + P_g$.
G. If $\tau_j \in \gamma_l$ allocated to server S_l executes inside server S_g postponing the execution of STRP γ_g, then γ_g is incorporated to the high priority queue of server S_l with a priority higher than that of γ_l. γ_g is in the queue of both servers S_l and S_g and executes in the one with the closest relative deadline. Until the debt is paid or forgotten, each time it is released, Γ_g will remain on both queues.
H. For each unit of time that γ_g, allocated to server S_l, executes inside server S_g postponing the execution of STRP γ_g, the variable $v_{l,g}$ is incremented by one unit. It keeps track of the debt contracted by debtor S_l with lender S_g.
I. For each unit of time that γ_g, allocated to server S_g, is executed inside server S_l after being blocked by γ_l, the variable $v_{l,g}$ is decremented by one unit. The debt is being paid and the execution can go on until $v_{l,g}$ reaches value 0.

The first three rules, corresponding to the CBS mechanism, provide the important properties of temporal isolation and hard schedulability guarantee. However, a transient overload produced by an increment in the execution time of a task produces a postponement in its deadline. While this mechanism guarantees that the overload does not affect other tasks, it penalizes the server for the rest of the life of the system by differentiating the deadline of the STRP from the deadline of the server. The idea is to establish a prescription to this penalization whenever a singularity appears in the system. In this way, successive postponements in the deadlines of a CBS are forgotten and new values for the dynamic variables can be computed as if an initialization of the system is taking place; this is done by means of rule F. Rules D and E implement the Bandwidth Inheritance among servers when a STRP is blocked. The other rules describe the balancing mechanism when a debt is contracted and paid later.

The performance of CFP will generally improve the performance of BWI. To begin with, in order to complete its execution, a blocked task will never have less time, as proved in the following lemma:

Lemma 1. *A task blocked under CFP never has less available time to complete its execution than under BWI.*

Proof. From rules D and E, under BWI a blocked task can resume its execution only when its server has no more blocking tasks in its ready queue and has the earliest deadline among all active servers (giving it the highest priority). Under CFP, instead, rules G–I guarantee that a blocked task may execute on its own server or in the debtor server, whichever has the higher priority. In this way, the time available to complete its execution may be increased but never reduced. □

As a consequence, a schedule under CFP will not produce more deadlines' misses than a schedule under BWI, and it may produce less. This is because, with the exception of the case in which rules H and I are not used because debts are cancelled at singularities, the blocked task will have its own time plus more time available to complete its execution. Having more time available, the number of deadlines' misses cannot be increased and, on the contrary, it may be reduced.

Example. In order to illustrate the main characteristics of the CFA, an example is given. In Figs. 4 and 5, the evolution of a system operating under the BWI/CBS and the CFA/BWI/CBS protocols in the interval $[1, 30]$ are respectively depicted. The explanation about how the rules are applied is restricted to the interval $[1, 6]$. In Appendix, however, the explanation is extended to the interval $[1, 30]$. The system has three servers: S_1 $(2, 6)$, S_2 $(6, 18)$ and S_3 $(8, 24)$. Each one serves a task with period equal to the server-period and worst case execution time equal to the budget of the server. The first two servers share a critical section on resource R, S_1 for the duration of its execution time and S_2 for the first five slots of its execution time.

The rules of the BWI/CBS protocol as applied in the interval $[1, 6]$ are:

– $t = 1$. τ_{21} and τ_{31} arrive. Since $\delta_2 = 19 < \delta_3 = 25$ holds, the processor is assigned to S_2 (EDF policy). R is locked.

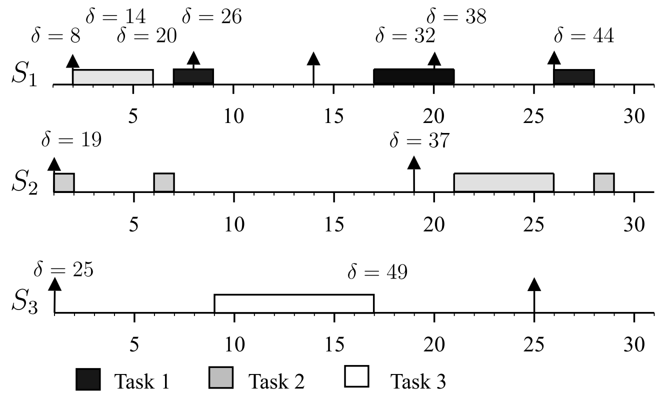


Fig. 4. System evolution under BWI. ↑, task arrival; δ, new deadline. In the three axes the activity of each server is depicted.

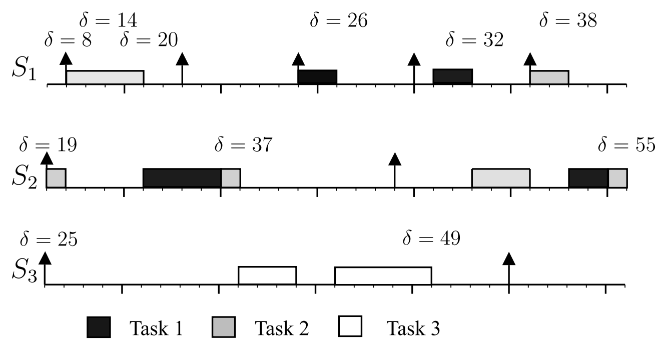


Fig. 5. System evolution under CFA. ↑, task arrival; δ, new deadline. In the three axes the activity of each server is depicted.

- $t = 2$. τ_{11} arrives. It cannot access R. Since τ_{11} has the earliest deadline, τ_{21} is transferred to S_1 (Rule D) and executed there.
- $t = 4$. S_1's budget is depleted and recharged to 2. Its deadline is postponed to 14 (Rule C).
- $t = 6$. S_1's budget is depleted and recharged to 2. Its deadline is postponed to 20 (Rule C). τ_{21} frees R but now, $\delta_2 = 19 < \delta_1 = 20$ holds. τ_{21} returns to S_2 (EDF policy) and finishes its execution there.

The rules of the CFA/BWI/CBS protocol as applied in the interval $[1, 6]$ are:

- $t = 1$. τ_{21} and τ_{31} arrive. Since $\delta_2 = 19 < \delta_3 = 25$ holds, the processor is assigned to S_2 (EDF policy). R is locked.
- $t = 2$. τ_{11} arrives. It cannot access R. Since S_1 has the earliest deadline, τ_{21} is transferred to S_1 (Rule D) and executed there. For every slot executed by τ_{21} in S_1, v_{21} is incremented in one unit (Rule H). τ_{11} is incorporated to the high priority queue of S_2 (Rule G).
- $t = 4$. S_1's budget is depleted and recharged to 2. Its deadline is postponed to 14 (Rule C). v_{21} is equal to 2 (Rule H).
- $t = 6$. S_1's budget is depleted and recharged to 2. Its deadline is postponed to 20 (Rule C). v_{21} is equal to 4 (Rule H). τ_{21}frees R but now, $\delta_2 = 19 < \delta_3 = 20$ holds. τ_{21} returns to S_2. τ_{11} starts execution in S_2 (Rule G) and locks R. For each slot executed in S_2, v_{21} is decremented in one unit (Rule I).

As can be seen, executing under BWI/CBS, in the interval $[1, 30]$ τ_1 misses the deadlines in the first four instantiations. This does not happen if CFA/BWI/CBS is used.

4.3.2. CFPHR (CFA/BWI/CBSHR)

For the CFP to work under the hard reservation scheme, rule C has to be modified.

C. Server S_g is allowed to execute while $q_g > 0$. When the budget is depleted, a new pair (q_g,δ_g) is computed: the absolute deadline is postponed to $\delta_g = \delta_g + P_g$ and the server goes into a recharging state. At $r_g = P_g - \delta_g$ the budget is recharged to $q_g := Q_g$ and the server becomes active again. In case no server is active and there is at least one server in the recharging state, the activation time for the servers can be advanced by determining $advance = \min\{r_i - t\}$ for every server in the recharging state. Then update the activation time of the servers: $r_i = r_i - advance$.

This modification does not alter the basic properties explained in the previous section. It must be noticed, however, that under hard reservation a server may not postpone its deadline more than once per period. Moreover, the maximum debt that an inheritor server may contract in a period is bounded by the budget of the lender server, as proved in the following theorem.

Theorem 2. *Under CBSHR the maximum debt a server S_i can contract with server S_g is limited to Q_g per period.*

Proof. A server S_g executing a blocking task can execute up to Q_g units of time before suspending itself and passing to the recharging queue. Only at the reactivation time the server will be able to continue with the execution. Thus, server S_i can only contract a debt of Q_g units of time out of every P_g. □

The previous theorem does not bound the total debt a server may contract but the rate at which it does it. In fact, the critical section of the blocking task may be so large compared to the period of the blocked one that it may take several periods of the lender to execute it.

In Fig. 6, the temporal evolution of the previous example is shown and explained.

At $t = 4$, the budget of S_1 is depleted and there is a deadline postponement. By the modified Rule C, S_1 will not be active until $t = 14$. As S_2 is the server with the nearest deadline, the scheduler dispatches it and continues with the execution of the critical section. The debt contracted is $v_{ig} = 2$. As soon as task 2 releases the semaphore at $t = 6$, τ_1 collects its debt by executing in S_2. At $t = 14$, S_1 becomes active again and has the higher priority so task 1 can be executed in it.

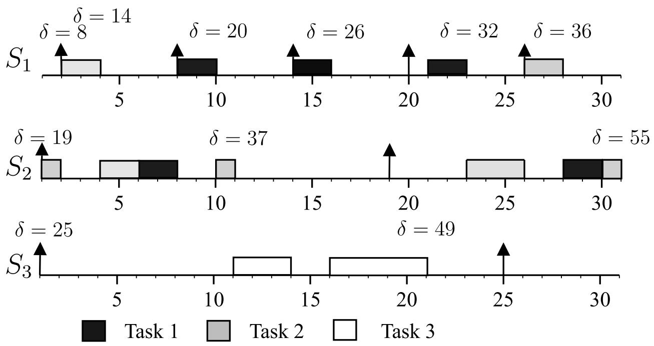


Fig. 6. System evolution under CFAHR. ↑, task arrival; δ, new deadline. In the three axes the activity of each server is depicted.

As can be seen, there are some differences between the temporal evolutions under both protocols. The debt contracted by S_2 with S_1 in the first period is only of 2 units of time instead of 4 and, as a consequence, at $t = 8$, server S_1 resumes the execution of task 1.

In general, the hard reservation approach may cause the loss of more deadlines than the simple one. This is because, once their budgets have been depleted, the servers are suspended until the next activation instant if other servers, even of lower priority, are in the active state and may execute. This is the price to pay for ensuring a more constant rate in multimedia applications. The designer may chose the proper protocol having in mind the intended applications.

5. Catering for HRT STRPs

As it was shown in the previous section, the Clearing Fund Protocol is specially apt to manage an Open Dynamic System of soft real-time STRPs. However, it may be used to schedule hard real-time systems, although with two caveats:

(i) If it handles only hard tasks, its use is not as efficient as for instance Rate Monotonic plus Priority Inheritance and Ceiling protocols.
(ii) If it handles a mix of hard and soft subsystems, the price to be paid is that a complete spatial isolation must exist between both subsystems.

If a soft application shares resources with a hard one, meeting hard deadlines cannot be guaranteed anymore. If, on the contrary, spatial isolation is preserved, the hard subsystem will meet all its time-constraints while the soft subsystem will receive the second best treatment.

Having that in mind, some definitions about blocking chains must be given. Then some properties of the algorithm used in determining the HRT schedulability will be proved. Based on them, the actual method to compute the duration that an HRT STRP may suffer is given. With them, the schedulability of the HRT subsystem may be tested.

5.1. Blocking chains

A blocking chain, notated $H_i = \{\tau_I, R_I, \tau_{II}, R_{II}, \ldots, \tau_k, R_k, \ldots, R_{z-1}, \tau_z\}$ is a set of tasks and shared resources ordered according to the following rules:

- $\tau_1 = \tau_i$
- $T_k \leqslant T_{k+1}$
- τ_k and τ_{k+1} share the resource R_k.
- Accesses are of type $P(R_{k-1})\cdots P(R_k)\cdots V(R_k)\cdots V(R_{k-1})$.
- $\tau_k \in \gamma_g \Rightarrow \nexists \tau_{h\in 1,2,\ldots\, z-k} \in \gamma_g$

Therefore:

(1) The subscript of the chain identifies its first task in the graph as defined in Section 3.
(2) Tasks are ordered by monotonically increasing periods.
(3) Each pair of adjacent tasks share the resource in between.
(4) Accesses are properly nested (the higher subscript inside the lower one).
(5) Only one task of each STRP belongs to a given blocking chain.

It must be noted that, according to the previous rules, τ_{II} will be the first task of a chain H_{II}, which is a subchain of H_I.

It is assumed that because of the precedence relation, a successor cannot start its execution until its predecessor finishes its execution. Therefore, although they can share a resource, no blocking may take place between tasks belonging to the same STRP.

Example. In the graph of Fig. 1, three blocking chains can be identified:

$$H_1 = \langle \tau_1, R_1, \tau_8 \rangle$$
$$H_3 = \langle \tau_3, R_3, \tau_9, R_4, \tau_{10} \rangle$$
$$H_9 = \langle \tau_9, R_4, \tau_{10} \rangle$$

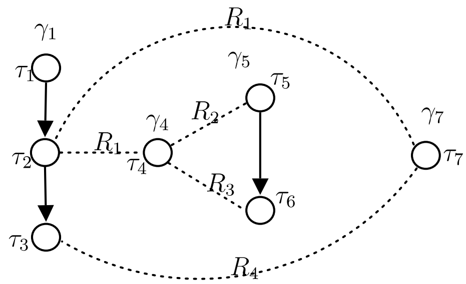
Note that H_9 is a subchain of H_3. Also that, although sharing R_2, τ_6 can not block τ_3 and therefore there is not H_2.

Each task can have more than one blocking chain. The hth blocking chain of task τ_i is notated H_i^h.

Example. In Fig. 7

$$H_2^1 = \langle \tau_2, R_1, \tau_4, R_2, \tau_5 \rangle$$
$$H_2^2 = \langle \tau_2, R_1, \tau_4, R_3, \tau_6 \rangle$$
$$H_2^3 = \langle \tau_2, R_1, \tau_7 \rangle$$
$$H_3^1 = \langle \tau_3, R_4, \tau_7 \rangle$$
$$H_4^1 = \langle \tau_4, R_2, \tau_5 \rangle$$
$$H_4^2 = \langle \tau_4, R_3, \tau_6 \rangle$$

Please cite this article in press as: Santos, R. et al., Improving the schedulability of soft real-time open dynamic ..., J. Syst. Software (2007), doi:10.1016/j.jss.2007.07.004



Q3

Fig. 7

Example. In Fig. 8

$$H_1^1 = \langle \tau_1, R_1, \tau_2 \rangle$$
$$H_1^2 = \langle \tau_1, R_1, \tau_3 \rangle$$
$$H_2^1 = \langle \tau_2, R_1, \tau_3 \rangle$$

Γ_i denotes the set of tasks of all the blocking chains or subchains starting at τ_i but excluding it. $\Gamma(\gamma_g)$ denotes the union of all $\Gamma_i | \tau_i \in \gamma_g$ and represents, therefore, the set of tasks that may interact with the STRP γ_g.

Example. In Fig. 7

$$\Gamma_2 = \{\tau_4, \tau_5, \tau_6\}$$
$$\Gamma_3 = \{\tau_7\}$$
$$\Gamma_4 = \{\tau_5, \tau_6\}$$
$$\Gamma(\gamma_1) = \Gamma_2 \bigcup \Gamma_3 = \{\tau_4, \tau_5, \tau_6, \tau_7\}$$
$$\Gamma(\gamma_4) = \{\tau_5, \tau_6\}$$

$\Gamma_{g,l}$ denotes the set of tasks belonging to Γ_g that may be blocked by γ_l such that the last task in the blocking chain belongs to γ_l.

Example. In Fig. 1, $\Gamma_{1,7} = \{\tau_1, \tau_3\}$, $\Gamma_{1,10} = \{\tau_3\}$. In Fig. 7, $\Gamma_{1,4} = \{\tau_2\}$, $\Gamma_{1,5} = \{\tau_2\}$, $\Gamma_{1,7} = \{\tau_2, \tau_3\}$. In Fig. 8, $\Gamma_{1,2} = \{\tau_1\}$, $\Gamma_{1,3} = \{\tau_1\}$.

5.2. Properties

Only properties that modify, or are added to, the properties of the CBS and the BWI algorithms presented in Abeni and Buttazzo (1998) and Lipari et al. (2004) shall be proved.

Lemma 3. *Only one task of γ_l can block γ_g in each instance.*

Proof. Assume that γ_g, in the present instance, is blocked a first time by a task belonging to γ_l. After that blocking ceases, γ_g will execute, either in S_g or in S_l, with higher priority than γ_l and therefore cannot be blocked again by γ_l in the present instance. □

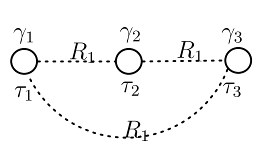


Fig. 8

Lemma 4. *A STRP can be blocked on the same resource more than once.*

Proof. Although, according to Lemma 1, only one task of each STRP can block another STRP, nothing precludes that two tasks of different STRPs block another STRP on the same resource. □

The interference time of STRP γ_g, notated I_g, is defined as the total blocking time that γ_g may suffer in the worst case. It will be the sum of the longest blocking times that each of its tasks may suffer. Because of the precedence relations, tasks of one STRP cannot block tasks of the same STRP or, in other words, a STRP cannot block itself.

Lemma 5. *Each STRP containing tasks belonging to $\Gamma(\gamma_g)$ can contribute to I_g for at most the longest execution among the critical sections shared by both STRPs.*

Proof. Immediate from Lemma 3 □

However, a STRP may be successively blocked by several other STRPs. Obviously, only tasks belonging to $\Gamma(\gamma_g)$, as defined in Subsection 5.1, can interfere and I_g results to be the maximum time that those tasks can execute inside S_g for each instantiation of γ_g.

$B_k(R_{k-1})$ shall denote the longest time that task τ_k can spend in the critical section of resource R_{k-1}, blocking therefore tasks $\tau_I, \tau_{II}, \ldots \tau_{k-1}$. $B(H_i^h) = \sum_k B_k(R_{k-1})$ is the sum of the longest times that each task of the chain H_i^h can block task τ_i. Of all the H_i^h chains, the one producing the maximum $B(H_i^h)$ shall be chosen. I_g will be the sum of the maximums for each task of the STRP.

5.3. Computation of the interference time

An algorithm to compute the interference time is presented in pseudo code. It is based on the previous definitions and lemmas. The computation is made for each STRP. Two sets are passed as parameters to the algorithm: $G = \{\Gamma_{gl}, l \neq g\}$ and $\Psi = \{\Psi_{gi}^l, \forall i, l\}$

The algorithm presented in pseudo code computes I_g, the interference time for STRP γ_g, exploring all the possible blocking chains. They are grouped according to the STRP of the last task. In this way all the blocking chains that end in γ_l are considered together (lines 6–9). In order to do this, for each task $\tau_i \in \gamma_g$ that can be blocked directly or indirectly by a task in γ_l, it computes the maximum blocking time.

Since γ_l can block γ_g only once, the longest blocking chain is considered (line 8). When all the tasks τ_i that can be blocked by γ_l have been considered, the maximum blocking time computed in line 8 is added to I_g (line 10) and a new blocking STRP is considered (lines 3–11). When all the STRPs have been considered I_g has the interference time that γ_g may suffer.

6. Experimental results

A comparative performance evaluation of the CFA/BWI/CBS and the CFA/BWI/ CBSHR protocols against BWI was carried out. Since BWI was taken as benchmark and it cannot handle STRPs of precedence related tasks, the evaluation was performed on sets of unitary STRPs.

Algorithm 1. Interference Time Computation Interference_Computation (G, Ψ)

```
{
I_g = 0;
for each Γ_gl ∈ G
{
    block = 0;
    for each τ_i ∈ Γ_gl
    {
    block = max(max_h(B(H_i^h)|H_i^h ∈ Ψ_gi^l), block);
    }
      I_g = I_g + block;
}
```

6.1. Setting the simulations

About one hundred thousand sets composed of 10 unitary STRPs each were run. The STRPs' periods were randomly generated with uniform distribution in the sample space $[10, 20, 30, \ldots, 100]$. The set utilization factor was forced to take values in the range $[0.54, 0.55, \ldots, 0.99]$. Nine of the ten STRPs (each one already with its defined period) were selected at random. Each one was assigned an execution time to produce a STRP utilization factor randomly selected in the interval $[0.03, 0.10]$ of the set of STRPs' utilization factor. The execution time of the tenth STRP was adjusted to produce the final sought set utilization factor. Also, for each set, the following parameters were randomly generated: (1) The number of shared resources (none, 1, 2 or 3). (2) Which STRPs access a shared resource. (3) For how long each sharing STRP uses the resource and at what instant it access it.

For each STRP in the system a CBS server with budget and period equal to the worst case execution time and period of the STRP, respectively, were assigned. Each set was run for about ten thousand slots, a run time long enough to produce a good variety of inheritances and preemptions among the servers holding the STRPs. Each generated set of ten STRPs was run under the three protocols. Whenever a task missed a deadline, a counter was incremented; when the run of the system was finished, this counter was stored together with the utilization factor of the system. Finally, a factor of demerit was computed. It is the Average Missed Deadline Ratio (AMDR), defined as the ratio between the sum of all the missed deadlines associated to a certain utilization factor and the number of systems run with that utilization factor. A metric for the QoS of the methods could be the reciprocal of the AMDR.

6.2. Results obtained

Results (AMDR *vs.* UF), plotted in a semilogarithmic graph, are presented in Fig. 9.

BWI, CFAHR and CFA, in that order, start losing deadlines at utilization factors of approximately 0.53, 0.57 and 0.73, respectively. As could be expected, the miss-ratio increases with utilization factors. For UF = 0.99, the miss-ratios differ roughly by one and two orders of magnitude.

As the difference between BWI and the other two protocols is of one and two orders of magnitude, the results are plotted in a semilogarithmic graph.

6.3. Results explained

The results obtained sustain the conclusions drawn from the examples. The BWI has the higher AMDR because the inheritance procedure has no compensating mechanism for the lender server; since, according to the CBS rules, the servers recharge their budget and postpone their deadline everytime the budget is depleted, the deadline of the server is soon very different (much later) than the deadline of the STRP, producing a degradation of the servers's priority. For the CFA/BWI/CBSHR case, things improve considerably. There is a compensating mechanism but the restriction on the activation of the server prevents sometimes the immediate restitution of the bandwidth consumed. Finally, the CFA/BWI/CBS has the best performance of the three protocols. This is due to the fact that there is no restriction for the servers to execute if there are ready tasks in their queues.

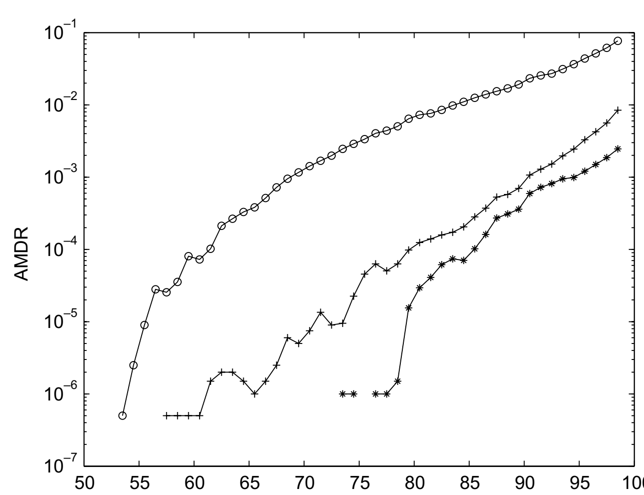


Fig. 9. Simulation Results, o – BWI, + – CFAHR, $*$ – CFA. Zero values have no representation on logarithmic scale.

Please cite this article in press as: Santos, R. et al., Improving the schedulability of soft real-time open dynamic ..., J. Syst. Software (2007), doi:10.1016/j.jss.2007.07.004

The AMDR for the BWI/CBS protocol at the higher utilization factor is approximately 0.1, which is quite good considering that it is a simple straightforward protocol. In the CFA/BWI/CBS case, things are much better because the ratio of missed deadlines is only slightly bigger than 0.001, certainly a good result specially for applications in which although some deadlines may be missed, it is better to lose as few as possible. CFA/BWI/CBSHR has a general performance in between the other two protocols and it would be useful in cases in which more uniform rates of execution of STRPs with long total execution times, necessarily partitioned in successive smaller budgets, are convenient.

7. Conclusions

In this paper, the Clearing Fund Protocol was presented. It is a three layered protocol, based on the CBS and the BWI algorithms, to schedule soft real-time STRPs in an open and dynamic environment. In the context of this paper, they are defined as sets of precedence-related sharing-resources tasks, each one executed in its own constant bandwidth server. A STRP may consist of only one task (unitary STRPs). The main idea behind the protocol is the facility of executing a blocked STRP either in its own server or in the server of the blocking STRP, whichever has the nearest deadline. For accounting purposes, a counter in the blocking server keeps tab of the time its STRP spent in the higher priority server. This acquired debt must be paid on demand but it can also be cleared at singularities, defined as instants in the evolution of the system in which the last pending task is executed. At that time, budgets and deadlines are restored to their original values, reinitializing the whole system with a clear start.

Extensive simulations were performed to obtain a comparative evaluation of the proposed protocols against BWI. The metric used is a factor of demerit, the Average Missed Deadlines Ratio, defined as the ratio between the number of deadlines that were missed while running different sets of STRPs with the same utilization factor, and the number of jobs generated in those sets. The results favour the two variants of the Clearing Fund Protocol, CFP and CFPHR, by two and one order of magnitude, respectively.

Acknowledgements

The authors express their sincere appreciation to the anonymous referees for all their comments and suggestions, which have substantially improved the paper.

Appendix

The example presented in Fig. 4 is now described completely. The rules of the BWI algorithm as applied in the interval $[1, 30]$ are:

- $t = 1$. τ_{21} and τ_{31} arrive. Since $\delta_2 = 19 < \delta_3 = 25$ holds, the processor is assigned to S_2 (EDF policy). R is locked.
- $t = 2$. τ_{11} arrives. It cannot access R. Since τ_{11} has the earliest deadline, τ_{21} is transferred to S_1 (Rule D) and executed there.
- $t = 4$. S_1's budget is depleted and recharged to 2. Its deadline is postponed to 14 (Rule C).
- $t = 6$. S_1's budget is depleted and recharged to 2. Its deadline is postponed to 20 (Rule C). τ_{21} frees R but now, $\delta_2 = 19 < \delta_1 = 20$ holds. τ_{21} returns to S_2 (EDF policy) and finishes its execution there.
- $t = 7$. τ_{11} starts execution in S_1 (EDF policy). R is locked.
- $t = 8$. τ_{12} arrives. τ_{11} keeps executing in S_1 but it misses its deadline.
- $t = 9$. S_1's budget is depleted and recharged to 2. R is unlocked. Its deadline is postponed to 26 (Rule C). τ_{31} starts executing in S_3 because $\delta_3 = 25 < \delta_1 = 26$ holds (EDF policy).
- $t = 14$. τ_{13}arrives, and τ_{12} misses its deadline. Rule A is applied and S_1 keeps its parameters unchanged.
- $t = 16$. τ_{31} finishes and S_3's budget is recharged to 8 and its deadline postponed to 49 (Rule C).
- $t = 17$. S_1 is the only active server so it begins to execute τ_{12}.
- $t = 19$. S_1's budget is depleted and recharged to 2. Its deadline is postponed to 32 (Rule C). τ_{13} misses its deadline. τ_{14} arrives and τ_{13} begins its execution. R is locked. τ_{22} arrives, S_2's parameters are updated (Rule A), budget recharged to 6 and deadline postponed to 37.
- $t = 21$. S_1's budget is depleted and recharged to 2. R is unlocked. Its deadline is postponed to 38 (Rule C). $\delta_2 = 36 < \delta_1 = 38$ so S_2 has the earliest deadline and begins to execute τ_{22}. R is locked.
- $t = 25$. τ_{32} arrives, S_3's parameters are kept unchanged (Rule A). R is unlocked.
- $t = 26$. τ_{22} finishes its execution, S_2's budget is depleted and recharged to 6 and its deadline postponed to 55. τ_{14} misses its deadline. τ_{15} arrives. Rule A is applied and S_1 keeps its parameters unchanged. $\delta_1 = 38 < \delta_3 = 49$, by EDF, the processor is granted to S_1 and τ_{14} executes. R is locked.
- $t = 28$. S_1's budget is depleted and recharged to 2 and its deadline postponed to 44. R is unlocked. $\delta_1 = 44 < \delta_3 = 49$ and S1 is granted the processor by EDF and τ_{15} executes. R is locked.
- $t = 30$.S_1's budget is depleted and recharged to 2 and its deadline postponed to 50. τ_1 meets its fifth deadline after loosing the first four. R is unlocked.

In Fig. 5 the evolution of the same example is presented for the case of the CFA. The rules applied in the interval [1, 30] are:

- $t = 1$. τ_{21} and τ_{31} arrive. Since $\delta_2 = 19 < \delta_3 = 25$ holds, the processor is assigned to S_2 (EDF policy). R is locked.
- $t = 2$. τ_{11} arrives. It cannot access R. Since S_1 has the earliest deadline, τ_{21} is transferred to S_1 (Rule D) and executed there. For every slot executed by τ_{21} in S_1,

v_{21} is incremented in one unit (Rule H). τ_{11} is incorporated to the high priority queue of S_2 (Rule G).

- $t = 4$. S_1's budget is depleted and recharged to 2. Its deadline is postponed to 14 (Rule C). v_{21} is equal to 2 (Rule H).
- $t = 6$. S_1's budget is depleted and recharged to 2. Its deadline is postponed to 20 (Rule C). v_{21} is equal to 4 (Rule H). τ_{21} frees R but now, $\delta_2 = 19 < \delta_3 = 20$ holds. τ_{21} returns to S_2. τ_{11} starts execution in S_2 (Rule G) and locks R. For each slot executed in S_2, v_{21} is decremented in one unit (Rule I).
- $t = 8$. τ_{11} finishes on time, R is unlocked. v_{21} is equal to 2 (Rule I). τ_{12} arrives and begins its execution in S_2 (Rules G–I). R is locked. S_1's parameters are kept unchanged (Rule A).
- $t = 10$. τ_{12} meets its deadline. v_{21} is equal to 0, τ_1 is removed from S_2 high priority queue. τ_{21} continues its execution in S_2 (Rules G–I).
- $t = 11$. S_2's budget is depleted and recharged to 6. Its deadline is postponed to 37. (Rule C). S_3 is the only active server so τ_{31} begins its execution.
- $t = 14$. τ_{13} arrives and S_1's parameters are kept unchanged (Rule A). $\delta_1 = 19 < \delta_3 = 25$ so S_1 is granted the processor (EDF policy) and τ_{13} begins its execution. R is locked.
- $t = 16$. τ_{13} finishes and S_1's budget is recharged to 2 and its deadline postponed to 26 (Rule C). R is unlocked. S_3 is the only active server and continues with the execution of τ_{31}.
- $t = 19$. τ_{21} arrives, S_2's parameters are kept unchanged. $\delta_3 = 25 < \delta_2 = 37$ so S_3 is granted the processor (EDF policy) and τ_{31} continues executing.
- $t = 20$. τ_{14} arrives, S_1's parameters are kept unchanged. $\delta_3 = 25 < \delta_1 = 26$ so S3 is granted the processor (EDF policy) and τ_{31} continues executing.
- $t = 21$. S_3's budget is depleted and recharged to 8, its deadline is postponed to 49 (Rule C). $\delta_1 = 26 < \delta_2 = 37$ so S_1 is granted the processor (EDF policy) and τ_{14} begins its execution. R is locked.
- $t = 23$. τ_{14} finishes, S_1's budget is depleted and recharged to 2 and its deadline postponed to 31 (Rule C). S_2 is the only active server so τ_{22} begins its execution.
- $t = 25$. τ_{32} arrives, S_3's parameters are kept unchanged (Rule A).
- $t = 26$. τ_{15} arrives. Rule A is applied and S_1 keeps its parameters unchanged. $\delta_1 = 31 < \delta_2 = 37 \quad < \delta_3 = 49$, by EDF, the processor is granted to S_1 and τ_{15} tries to lock R. It cannot access R. τ_{22} is transferred to S_1 (Rule E) and executed there. For every slot executed by τ_{22} in S_1, v_{21} is incremented in one unit (Rule H). τ_1 is incorporated to the high priority queue of S_2 (Rule G). v_{21} is equal to 0.
- $t = 28$. S_1's budget is depleted and recharged to 2 and its deadline postponed to 37. R is unlocked. $\delta_1 = 37 < \delta_2 = 37$ and S_2 is granted the processor by EDF and τ_{15} executes in S_2. R is locked. v_{21} is decremented in one for each slot τ_1 executes in S_2 (Rules H, I).
- $t = 31$. τ_{15} finishes, v_{21} is equal to 0. τ_{22} completes its execution, S_2's budget is depleted and recharged to 6 and its deadline postponed to 55.

References

Abeni, L., Buttazzo, G., 1998. Integrating multimedia applications in hard real-time systems. In: Proceedings of the 19th IEEE Real Time Systems Symposium, pp. 4–13.

Baker, T., 1990. A stack-based allocation policy for real-time processes. In: Proceedings of the 11th IEEE Real Time Systems Symposium, pp. 191–200.

Caccamo, M., Sha, L., 2001. Aperiodic servers with resource constraints. In: Proceedings of the 22nd IEEE Real Time Systems Symposium, pp. 161–170.

Deng, Z., Liu, J.W.S., 1997. Scheduling real-time applications in open environment. In: Proceedings of the 18th IEEE Real Time Systems Symposium, pp. 308–319.

Lipari, G., Butazzo, G., 2000. Analysis of periodic and aperiodic tasks with resource constraints. Journal of Systems Architecture 46, 327–338.

Lipari, G., Lamastra, G., Abeni, L., 2004. Task synchronisation in reservation-based real-time systems. IEEE Transactions on Computers 53 (12), 1591–1601.

Liu, G.L., Layland, J.W., 1973. Scheduling algorithms for multiprogramming in hard real time environment. ACM 20, 46–61.

Marzario, L., Lipari, G., Balbestre, P., Crespo, A., 2004. Iris: a new reclaiming algorithm for server-based real-time systems. In: Proceedings of the 10th IEEE Real-Time and Embedded Technology and Applications Symposium, pp. 211–218.

Palencia, J.C., Gonzalez Harbour, M., 1999. Exploiting precedence relations in the schedulability analysis of distributed real-time systems. In: Proceedings of the 20th IEEE Real Time Systems Symposium, pp. 328–339.

Rajkumar, R., Abeni, L., Niz, D.D., Gosh, S., Miyoshi, A., Saewong, S., 2000. Recent developments with linux/rk. In: Proceedings of the Real Time Linux Workshop.

Ramammritham, K., 1990. Allocation and scheduling of complex periodic tasks. In: Proceedings of the 10th International Conference on Distributed Computer Systems, pp. 108–115.

Ramos-Thuel, S., Lehoczky, J.P., 1994. Algorithms for scheduling hard aperiodic tasks in fixed-priority systems using slack stealing. In: Proceedings of the 15th IEEE Real Time Systems Symposium, pp. 22–33.

Saewong, S., Rajkumar, R., 1999. Cooperative scheduling of multiple resources. In: Proceedings of the 20th IEEE Real-Time Systems Symposium, pp. 90–101.

Santos, R., Lipari, G., 2003. Scheduling precedence constraint tasks in open dynamic systems. In: Proceedings of the WIP 15th Euromicro Conference on Real-Time Systems.

Santos, R.M., Urriza, J., Santos, J., Orozco, J., 2004. New methods for redistributing slack time in real-time systems: applications and comparative evaluations. Journal of Systems and Software 69 (1–2), 115–128.

Santos, R., Lipari, G., Santos, J., 2004. Scheduling open dynamic systems: The clearing fund algorithm. In: 10th International Conference on Real-Time Computing Systems and Applications.

Sha, L., Rajkumar, R., Lehoczky, J., 1990. Priority inheritance protocols: an approach to real time synchronization. IEEE Transactions on Computers 39 (9), 1175–1185.

Sprunt, B., Lehoczky, J.P., Sha, L., 1988. Exploiting unused periodic time for aperiodic service using the extended priority exchange algorithm. In: Proceedings of the 9th IEEE Real-Time Systems Symposium, pp. 251–258.

Sprunt, B., Sha, L., Lehoczky, J.P., 1989. Aperiodic scheduling for hard real-time system. Real-Time Systems 1 (1–2), 27–60.

Strosnider, J.K., Lehoczky, J.P., Sha, L., 1995. The deferrable server algorithm for enhanced aperiodic responsiveness in hard real-time environments. IEEE Transactions on Computers 44 (1), 73–91.