A capacity sharing and stealing strategy for open real-time systems

Luís Nogueira *, Luís Miguel Pinho

CISTER Research Centre, School of Engineering of the Polytechnic Institute of Porto (ISEP/IPP), Rua Dr. António Bernardino de Almeida 431, 4200-072 Porto, Portugal

ARTICLE INFO

Article history:
Received 28 November 2008
Received in revised form 11 January 2010
Accepted 22 February 2010
Available online 3 March 2010

Keywords:
Open real-time systems
Dynamic scheduling
Resource reservation
Residual capacity reclaiming
Reserved capacity stealing
Shared resources
Precedence constraints

ABSTRACT

This paper focuses on the scheduling of tasks with hard and soft real-time constraints in open and
dynamic real-time systems. It starts by presenting a capacity sharing and stealing (CSS) strategy that sup-
ports the coexistence of guaranteed and non-guaranteed bandwidth servers to efficiently handle soft
tasks’ overloads by making additional capacity available from two sources: (i) reclaiming unused
reserved capacity when jobs complete in less than their budgeted execution time and (ii) stealing
reserved capacity from inactive non-isolated servers used to schedule best-effort jobs.

CSS is then combined with the concept of bandwidth inheritance to efficiently exchange reserved band-
width among sets of inter-dependent tasks which share resources and exhibit precedence constraints,
assuming no previous information on critical sections and computation times is available. The proposed
Capacity Exchange Protocol (CXP) has a better performance and a lower overhead when compared
against other available solutions and introduces a novel approach to integrate precedence constraints
among tasks of open real-time systems.

© 2010 Elsevier B.V. All rights reserved.

1. Introduction

As an increasing number of users run both real-time and tradi-
tional desktop applications in the same system the issue of how to
provide an efficient resource utilisation in this highly dynamic,
open, and shared environment becomes very important. The need
arises from the fact that independently developed services can en-
ter and leave the system at any time, without any previous knowl-
edge about their real execution requirements and tasks’ inter-
arrival times.

For most of these systems, the classical real-time approach
based on a rigid offline design and worst-case execution time
(WCET) assumptions would keep resources unused for most of
the time. Usually, tasks’ WCET is rare and much longer than the
average case. At the same time, it is increasingly difficult to com-
pute WCET bounds in modern hardware without introducing
excessive pessimism [1]. Such a waste of resources can only be jus-
tified for very critical systems in which a single missed deadline
may cause catastrophic consequences.

A more flexible scheduling approach is then needed in order to
increase resource usage. Flexibility is particularly important for
small embedded devices used in consumer electronics, telecommu-
nication systems, industrial automation, and automotive sys-
tems. In fact, in order to satisfy a set of constraints related to
weight, space, and energy consumption, these systems are typi-
cally built using small microprocessors with low processing power
and limited resources.

Guarantees based on average estimations are typically accept-
able for soft real-time tasks since a deadline miss does not consti-
tute a system or application failure but it is only less satisfactory
for the user. Nevertheless, when scheduling soft tasks based on
average estimated needs any chosen approach must handle the
case when a task needs to execute more than its guaranteed
reserved time. Not only it is desirable to achieve temporal isolation
among soft tasks as well as the schedulability of hard tasks must
not be compromised.

In [2], Mercer et al. propose a scheme based on capacity
reserves to remove the need of knowing the WCET of each task
under the Rate Monotonic [3] scheduling policy. A reserve is a
couple \((C_s, T_s)\) indicating that a task \(t_i\) can execute for at most \(C_s\)
units of time in each period \(T_s\). If a task instance needs to execute
for more than \(C_s\), the remaining portion of the instance is scheduled
in background.

Based on a similar idea of capacity reserves, Abeni and Buttazo
[4] proposed the Constant Bandwidth Server (CBS) scheduler to
handle soft real-time requests with a variable or unknown execution
behaviour under the Earliest Deadline First (EDF) [3] scheduling
policy. To avoid unpredictable delays on hard real-time tasks,
soft tasks are isolated through a bandwidth reservation mecha-
nism, according to which each soft task gets a fraction of the CPU
and it is scheduled in such a way that it will never demand more
than its reserved bandwidth, independently of its actual requests.
This is achieved by assigning each soft task a deadline, computed
as a function of the reserved bandwidth and its actual requests.
If a task requires to execute more than its expected computation time, its deadline is postponed so that its reserved bandwidth is not exceeded. As a consequence, overruns occurring on a served task will only delay that task, without compromising the bandwidth assigned to other tasks.

However, with CBS, if a server completes a task in less than its budgeted execution time no other server is able to efficiently reuse the amount of computational resources left unused. To overcome this drawback, CBS has been extended by several reclaiming schemes [5–9] proposed to support an efficient sharing of computational resources left unused by early completing tasks. Such techniques have been proved to be successful in improving the response times of soft real-time tasks while preserving all hard real-time constraints.

Nevertheless, not all computational tasks in modern open real-time systems follow a traditional periodic pattern. For example, aperiodic complex optimisation tasks may take varying amounts of time to complete depending on the desired solution’s quality or current state of the environment [10–15]. Furthermore, the existing reclaiming schemes are unable to donate reserved, but still unused, capacities to currently overloaded servers.

Based upon a careful study of the ways in which unused reserved capacities can be more efficiently used to meet deadlines of tasks whose resource usage exceeds their reservations, our previous work [16] proposed the coexistence of the traditional isolated servers with a novel non-isolated type of servers, combining an efficient reclamation of residual capacities with a controlled isolation loss. The goal of the Capacity Sharing and Stealing (CSS) scheduler is to reduce the mean tardiness of periodic guaranteed jobs by handling overloads with additional capacity available from two sources: (i) by reclaiming unused allocated capacity when jobs complete in less than their budgeted execution time; and (ii) by stealing allocated capacities from inactive non-isolated servers used to schedule aperiodic best-effort jobs.

However, CSS assumes tasks to be independent. A challenging problem in open real-time systems is how to schedule inter-dependent tasks that share resources and exhibit precedence constraints without a complete previous knowledge about their actual runtime behaviour. The Capacity Exchange Protocol (CXP) [17] builds upon CSS and integrates its capacity sharing and stealing strategy with the concept of bandwidth inheritance [18] to mitigate the cost of blocking on soft real-time tasks whose actual execution behaviour is only known by executing tasks until completion. While preserving the isolation principles of independent tasks, upon blocking, a task is allowed to be executed on more than its dedicated server, efficiently exchanging reserved capacities among servers to reduce the undesirable effects caused by inter-task blocking.

In this paper we provide a complete and consistent description of these protocols and extend the conducted evaluation, simultaneously dealing with capacity sharing, stealing and exchanging. More important, the paper also provides a proof of correctness of the proposed runtime exchange of reserved capacities. Hard schedulability guarantees can be provided either for independent and inter-dependent task sets, even when hard and soft real-time tasks do share resources and exhibit precedence constraints in open real-time systems.

In the remainder of this paper, we describe the system model and used notation in Section 2. Section 3 analyses the most significant scheduling approaches proposed to improve the performance of soft real-time tasks and introduces the need for the novel capacity sharing and stealing approach described in Section 4. The correctness of the proposed runtime exchange of reserved capacities for independent task sets is proved in Section 5. CXP is described in detail in Sections 6 and 7, as a way to efficiently support shared resources and precedence constraints among inter-dependent task sets of open real-time systems. Although the goal of CXP is to minimise the cost of blocking among soft real-time tasks, Section 8 describes how hard schedulability guarantees can still be provided even when hard and soft real-time tasks share resources, at the expense of some pessimism on the computation of blocking times when tasks access (nested) critical sections. Section 9 presents and analyses the achieved evaluation results. Finally, Section 10 concludes this paper.

2. System model

This paper focuses on dynamic open real-time systems where all services execute on a single shared processor, the sum of the reserved capacities is no more than the maximum capacity of the processor, and the scheduler does not have any previous complete knowledge about the execution requirements of soft real-time tasks. We make the reasonable assumption that whenever a service arrives to the system it advertises its requirements on a certain amount of the system’s resources based on expected average needs for soft real-time tasks and WCET measures for hard real-time tasks. If, given the current system's load, the required amount can be guaranteed, the service is accepted and the requested amount is reserved.

A service is composed of a set of hard and/or soft real-time tasks. Each real-time task \( t_i \) can generate a virtually infinite sequence of jobs. The \( j \)-th job of task \( t_i \) arrives at time \( a_{ij} \), is released to the ready queue at time \( r_{ij} \), starts to be executed at time \( s_{ij} \) with deadline \( d_{ij} = r_{ij} + p_i \), with \( p_i \) being the period of \( t_i \), and finishes its execution at time \( f_{ij} \). These times are characterised by the relations

\[
\begin{align*}
\forall j: & \; a_{ij} \leq r_{ij} \leq s_{ij} \leq f_{ij}. \\
& \; \text{For a hard real-time task } t_i, \text{ the system must provide an a priori guarantee that every job must complete at a time } f_{ij} < d_{ij}. \text{ As such, } p_i \text{ refers to the minimum inter-arrival time between successive jobs of } t_i \text{ so that } a_{ij+1} > a_{ij} + p_i \text{ and its execution requirements } e_i \text{ are characterised by the task's WCET.} \\
& \; \text{For soft real-time tasks, the timing constraints are more relaxed. In particular, for a soft task } t_i, p_i \text{ represents the expected inter-arrival period between successive jobs. As such, the arrival time } a_{ij} \text{ of a particular job is only revealed at runtime and the exact execution requirements } e_i \text{ can only be determined by actually executing the job to completion until time } f_{ij}. \\
& \; \text{Each soft or hard real-time task } t_i \text{ is scheduled through an abstract entity } S_i \text{ called server. As such, all the jobs generated by task } t_i \text{ are dedicated to server } S_i. \text{ Each server } S_i \text{ is characterised by a pair } (Q_i, T_i), \text{ where } Q_i \text{ is the server’s maximum reserved capacity and } T_i \text{ its period. For a hard real-time task } t_i, \text{ its dedicated server } S_i \text{ has a reserved capacity } Q_i \text{ equal to the task’s WCET and a period } T_i \text{ equal to the task’s period. For soft-real-time tasks, } Q_i \text{ and } T_i \text{ are set based on the served tasks’ expected average values. It is important to point out that this paper does not deal with policies to optimally assign or dynamically change the servers’ parameters according to the actual needs of the served soft real-time tasks either based on some heuristic algorithms or feedback control schemes as appear, for example, in [19].} \\
& \; \text{At each instant, the following values are associated with a server } S_i: \text{ its currently assigned deadline } d_{ik}, \text{ its remaining execution capacity } 0 \leq c_i \leq Q_i, \text{ the amount of residual capacity } r_i \leq c_i \text{ that can be reclaimed by other servers, and its currently assigned replenishment time } h_i = d_{ik}. \text{ If at time } t, S_i \text{ finishes the execution of its currently served job without exhausting its reserved execution capacity } c_i \text{ and it has no pending work, the remaining amount } c_i > 0 \text{ sets the server’s residual capacity } r_i = c_i \text{ that can be reclaimed (} c_i \text{ is subsequently set to zero). By pending work we refer to the case when there exists at least a served job such that } r_i \leq t < f_{ij}. \text{ This paper considers two different types of servers: isolated servers used to schedule periodic and sporadic guaranteed tasks.} 
\end{align*}
\]
دریافت فوری متن کامل مقاله

امکان دانلود نسخه تمام متن مقالات انگلیسی
امکان دانلود نسخه ترجمه شده مقالات
پذیرش سفارش ترجمه تخصصی
امکان جستجو در آرشیو جامعی از صدها موضوع و هزاران مقاله
امکان دانلود رایگان ۲ صفحه اول هر مقاله
امکان پرداخت اینترنتی با کلیه کارت های عضو شتاب
دانلود فوری مقاله پس از پرداخت آنلاین
پشتیبانی کامل خرید با بهره مندی از سیستم هوشمند رهگیری سفارشات