Technical Report

Resource Sharing Under a Server-based Semi-Partitioned Scheduling Approach

Alexandre Esper
Eduardo Tovar

CI-STER-TR-141008
10-08-2014
Resource Sharing Under a Server-based Semi-Partitioned Scheduling Approach

Alexandre Esper, Eduardo Tovar

CISTER Research Center
Polytechnic Institute of Porto (ISEP-IPP)
Rua Dr. António Bernardino de Almeida, 431
4200-072 Porto
Portugal
Tel.: +351.22.8340509, Fax: +351.22.8340509
E-mail: aresper@criticalsoftware.com, emt@isep.ipp.pt
http://www.cister.isep.ipp.pt

Abstract

The rapid evolution of commercial multicore platforms has raised the industry interest in developing and running applications independently on the same platform. However, in realistic industrial settings, tasks belonging to different applications share resources that are not limited to the CPUs. Applications share hardware components (e.g., co-processors and actuators), and need to access portions of code that may be protected by semaphores or mutexes. In this paper, we address the important challenge of resource sharing on multicore platforms through the use of servers, i.e., through a hierarchical scheduling approach, which is an effective technique to ensure the integration of independently developed applications on the same computing platforms as well as the isolation of tasks. To solve this problem, we adapt and extend the MrsP [6] resource sharing protocol and further combine it with the NPS-F scheduling algorithm [5], which employs a server-base approach. A schedulability analysis is then provided for the resulting framework.
Resource Sharing Under a Server-based Semi-Partitioned Scheduling Approach

Alexandre Esper †
Critical Software S.A.
Portugal
aresper@criticalsoftware.com

Eduardo Tovar ‡
CISTER/INESC-TEC
Portugal
emt@isep.ipp.pt

ABSTRACT

The rapid evolution of commercial multicore platforms has raised the industry interest in developing and running applications independently on the same platform. However, in realistic industrial settings, tasks belonging to different applications share resources that are not limited to the CPUs. Applications share hardware components (e.g., co-processors and actuators), and need to access portions of code that may be protected by semaphores or mutexes.

In this paper we address the important challenge of resource sharing on multicore platforms through the use of servers, i.e., through a hierarchical scheduling approach, which is an effective technique to ensure the integration of independently developed applications on the same computing platform as well as the isolation of tasks. To solve that problem we adapt and extend the MrsP [6] resource sharing protocol and further combine it with the NPS-F scheduling algorithm [5], which employs a server-base approach. A schedulability analysis is then provided for the resulting framework.

1. INTRODUCTION

The interest of industry in real-time embedded applications has recently gained strength. This has been mainly driven by the important need of taking as much benefit as possible of the processing power offered by multicore platforms, which can now be easily found in products ranging from portable cell phones and smartphones to large computer servers. Additionally, current software development processes often involve more than one independent development team (e.g., subcontractors) that produce software components, which are further integrated to form a final product.

In order to reach that goal, server-based techniques emerged as an intuitive solution to effectively ensure the temporal isolation and protection, while respecting all real-time constraints of the applications. Servers allow for the reservation of a portion of the embedded system capacity for a specific application. Therefore, it allows applications to run independently through time partitioning.

The research on real-time systems for uniprocessor (single core) is well established and consolidated, and there are multiple works extending the uniprocessor scheduling algorithms to multicore [7]. However, in practice, several challenges emerge when considering all the resources that must be accessed by tasks running in a multicore environment.

In a realistic industrial application, the resources shared by different tasks are not limited to the CPUs. Applications share hardware components and need to access data or execute portions of code that may be protected by semaphores or mutexes. This adds an additional layer of complexity to the scheduling problem and requires the introduction of Resource Sharing Protocols.

The objective of this work is to set the basis for the design of a framework that is able to effectively handle the hierarchical scheduling of tasks on multicore platforms whilst taking the shared logical resources into consideration. Hence, we adapt and combine the MrsP [6] resource sharing protocol with the NPS-F [5] scheduling algorithm.

2. SYSTEM MODEL

We consider the general sporadic task model, where each task \( \tau \) in a system \( \tau \) is characterized by its minimum inter-arrival time \( T_i \), relative deadline \( D_i \), and worst-case computation time, \( C_i \). That is, each task \( \tau \) can generate a potentially infinite number of jobs at least \( T_i \) time units apart, and each job must execute for at most \( C_i \) time units before its deadline occurring \( D_i \) time units after its release. Arbitrary deadlines are assumed, but jobs from the same task can never execute in parallel. The utilization \( u_i \) of a task \( \tau_i \) is defined as \( u_i = C_i / T_i \) and the system utilization, \( U(\tau) \), is defined as \( U(\tau) = \sum_{i=1}^{n} u_i \).

The execution platform is composed of \( m \) identical physical processors, uniquely numbered \( P_1 \ldots P_m \). We also consider a set of \( k \) servers, uniquely numbered \( S_1 \ldots S_k \). The tasks are first mapped to a server, which are then allocated to the processors. The server utilization \( U(S_q) \) is defined as:

\[
U(S_q) = \sum_{\tau_i \in S_q} u_i
\]

where \( \tau(S_q) \) is the set of tasks assigned to \( S_q \). We assume that the utilization of a server never exceeds 1 and that it never executes more than one processor at a time.

Shared Resources (denoted as \( r^i \)) are defined as the data structures that are shared between tasks. They are divided in two types; those that are shared by tasks mapped to the same server, called local resources, and those that are shared between tasks mapped to different servers, which are called global resources. The code associated with a resource is called a critical section and must be accessed under mutual exclusion. The blocking time experienced by a task \( \tau_i \) when accessing a locked local resource is defined as the local blocking time. Similarly, the blocking time experienced by \( \tau_i \) when it tries to access a locked global resource is defined as the global blocking time.

The relation between tasks and resources is given by two functions: \( F(\tau_i) \) and \( G(r^i) \). \( F(\tau_i) \) returns the set of re-
resources used by task $\tau_i$ and $G(r^j)$ returns the set of tasks that use resource $r^j$. The parameter $c^j$ is used to denote the worst case execution time of the resource $r^j$ when accessed by any task.

3. RELATED WORK

This section provides an overview of the MrsP protocol [6] and the NPS-F scheduling algorithm [5], which constitute the basis for the present work.

3.1 Review of MrsP

One of the most recent resource sharing protocols for multicore platforms is MrsP [6]. MrsP is restricted to fully partitioned systems where tasks are scheduled using fixed priorities. The general sporadic task model is employed, and each processor $P_k$ implements a local extension of the Stack Resource Policy (SRP) applied to the Priority Ceiling Protocol (PCP) [11] (denoted as PCP/SRP), where all resources $r^j$ are assigned a set of ceiling priorities, one for each processor $P_k$. The ceilings are defined as the maximum priority of all tasks allocated to $P_k$ that use $r^j$. Whenever a task $\tau_i$ attempts to access $r^j$, its priority is raised to the local ceiling of $r^j$. For local resources, MrsP behaves as an implementation of the uniprocessor PCP/SRP. For global resources, the access to a resource is granted through a FIFO queue. While waiting to gain access to resource $r^j$ that is already locked by another task $\tau_j$, $\tau_i$ remains active, i.e., it busy-waits for the lock to become available (spin-based locking).

The characteristics of MrsP reviewed so far are similar to MSRP [9], of which it is a variant. Its main difference is that tasks busy-waiting may use their “spin” time to undertake the execution of other waiting tasks. This means that although MrsP is defined for partitioned systems, the tasks still have the ability to migrate from one processor to another at run-time. If a task $\tau_i$ is preempted whilst accessing a resource $r^j$, then $\tau_i$ can migrate to any processor on which a task is waiting to gain access to $r^j$. The authors claim that this property effectively leads to a schedulability analysis that presents an identical form to the response-time analysis (RTA) [1] in the following way:

$$R_i = C_i + \max\{\hat{e}, \hat{b}\} + \sum_{r_j \in hpl(i)} R_j \frac{T_j}{\bar{T}_j} \times C_j$$  \hspace{1cm} (2)

where $hpl(i)$ is the set of local tasks with priority greater than $\tau_i$. The parameter $\hat{e}$ is the maximum execution time of a resource used by a local task with priority less than that of $\tau_i$ and a local task with an equal or higher priority than $\tau_i$. The parameter $\hat{b}$ is the maximum non-preemptive execution time induced by the Real-Time Operating System (RTOS).

The $C_i$ parameter for each task is given by:

$$C_i = WCET_i + \sum_{r_j \in P(\tau_i)} n_r \times c^j$$  \hspace{1cm} (3)

where $WCET_i$ is the worst-case execution time of the task, ignoring the time it takes to access resources (but including all time spent in the RTOS). The second term of Equation 3 accounts for the increased cost of the potential parallel access to resource $r^j$ due to tasks running on different processors. $n_r$ is the number of times $\tau_i$ uses $r_j$ and parameter $c^j$ is the maximum amount of time $\tau_i$ might need to execute $r^j$, including its spinning time, given by: $c^j = |\text{map}(G(r^j))| \times e^j$, where function $|\text{map}(G(r^j))|$ returns the number of processors onto which tasks that use resource $r^j$ can execute.

3.2 Review of NPS-F

NPS-F is a semi-partitioned scheduling algorithm [5]. The semi-partitioned approach allows the development of algorithms with a higher utilization bound than the partitioned approach (better work balance between processors) and also reduces the number of migrating tasks by avoiding the use of global shared queues for scheduling tasks to processors, when compared to the global scheduling approaches.

This algorithm employs a server-based approach, considering a set of $k$ servers. In the so-called flat-mapping, the tasks are assigned to the servers and each server is then assigned to one or two processors at most. A server has a utilization upper-bound by 1 and can never execute on more than one processor at a time. Hence, each server $S_k$ is equivalent to a unicore processor with a computing capacity $U(S_k)$. Each server serves one or more tasks using EDF as the internal scheduling policy.

The NPS-F algorithm is composed of 4 steps. The first step is the assignment of tasks to the servers, based on the task’s utilization ($u_t$). The second step is the computation of the capacity of each server $S_k$. NPS-F ensures the schedulability of all tasks allocated to $S_k$, even under the most unfavorable arrival phasings, i.e., when there are ready tasks, but their associated servers are not executing. This is achieved by inflating the utilization of the server, given by Equation (1) (see [5] [13] for more details). The third step of the off-line procedure is the allocation of the servers to the processors, following a semi-partitioned approach. Servers that are assigned to only one processor are called non-split servers, whereas servers that are assigned to two processors each are called split servers. Note that the servers that serve the split-tasks must be carefully positioned within the time slots in order to avoid their overlapping in time. The last step of the algorithm is performed at run-time, when the dispatching inside each server is performed under EDF policy.

Under NPS-F, it is the execution time of the servers which is split - not directly that of the underlying tasks served. In principle, this allows an improved efficiency in the utilization of a multiprocessor system. NPS-F has a utilization bound of 75% configurable up to 100% at the cost of more preemptions and migrations.

Recently, a new schedulability test for mapping tasks to servers for NPS-F has been proposed in [13]. But since we aim at defining a hierarchical scheduling framework that allows resource sharing between tasks, we first present some useful concepts defined in [12]. The Resource Demand of a task set $\tau(S_k)$ represents the collective workload resource requirements that the tasks in $\tau(S_k)$ request within a certain interval of time $t$. The Demand Bound Function (DBF) [4] of $\tau(S_k)$ calculates the maximum possible resource demands required to satisfy its timing requirements within a time interval of length $t$. The Resources Supply represents the amount of time the system can provide for $\tau(S_k)$’s execu-
tion, which is in fact the execution time provided by a server \( S_q \), onto which \( \tau(S_q) \) has been assigned. The Supply Bound Function (SBF) calculates the minimum possible resources supplies provided by a server \( S_q \) during a time interval of length \( t \). A server \( S_q \) is said to satisfy \( \tau(S_q) \)'s execution demand if:

\[
\text{DBF}(S_q, t) \leq \text{SBF}(S_q, t), \forall t > 0
\]  

(4)

Inequality (4) is then used as the new NPS-F schedulability test, meaning that the execution demand by all jobs assigned to a server (computed using the DBF) cannot exceed the amount of time supplied by the server for their execution, for every time interval of length \( t \). Since the test is based on the concept of the DBF (exact test for unicore platforms) rather than on the utilization, it allows to overcome many sources of pessimism that existed in the previous analysis [5].

Assuming sporadic task sets with arbitrary deadlines and ignoring all overheads (which can however be easily accounted for as shown in [13]), the DBF(\( S_q, t \)) is given by:

\[
\text{DBF}(S_q, t) = \sum_{\tau_i \in S_q} \max \left( 0, \frac{t - D_i}{T_i} \right) + 1 \times C_i
\]  

(5)

To perform the assignment of the tasks to the servers, NPS-F iterates over the set of all system tasks and attempts to fit each one of them (according to the bin-packing heuristic used, e.g., Next-Fit (NF) or First-Fit (FF)) in the servers. Each task \( \tau_i \) is provisionally added to the chosen server \( S_q \) and the length of the testing time interval \( t \) is calculated. The schedulability test defined by Equation (5) is then applied and if successful for some server \( S_q \), the task \( \tau_i \) is permanently mapped to it. Otherwise, a new server is opened and the task \( \tau_i \) is added to it. If the schedulability test fails for a server with only one task, then the task set is considered unschedulable.

4. ACCOUNTING FOR SHARED RESOURCES IN NPS-F

A current limitation of NPS-F is that it does not consider the interaction between tasks (i.e. the access to shared resources). The solution we propose in this paper complements NPS-F in that sense. It is based on an extension and further adaptation of the MrsP [6] resource sharing protocol that takes the particularities of NPS-F into account. In Section 4.1 we explain how the schedulability test of NPS-F can be adapted to introduce MrsP resource sharing protocol. Then, the problem of mapping tasks to servers is briefly addressed in Section 4.2.

We adapt MrsP to work with servers by instantiating the concept of bandwidth inheritance [8]. With that solution, one server may undertake the processing of a resource critical section on behalf of another task assigned to another server. In order to better visualize the impact of such bandwidth inheritance protocol, consider a simple system composed of two servers and four tasks. It is assumed that both servers are assigned to different physical processors and that the global resource \( r^g \) is shared between the servers. Tasks \( \tau_1 \) and \( \tau_2 \) are allocated to server \( S_1 \); task \( \tau_1 \) uses the local resource \( r^l \) and task \( \tau_2 \) does not use any shared resource (apart from processor). Tasks \( \tau_3 \) and \( \tau_4 \) are allocated to server \( S_2 \); task \( \tau_3 \) also uses the global resource \( r^g \) and task \( \tau_4 \) does not use any resource (apart from the processor).

Two simple example schedules (with and without bandwidth inheritance) for the system are provided in Fig.1 to illustrate the solution. In Fig.1(a), tasks \( \tau_1 \) and \( \tau_3 \) are executing at instant \( t_1 \). At instant \( t_2 \) task \( \tau_1 \) locks resource \( r^g \) and executes its critical section, represented by the notation \( \tau_{1,c} \). At instant \( t_3 \), task \( \tau_1 \) tries to access \( r^l \) but gets blocked (represented in gray) because the resource is already locked by \( \tau_2 \). Even worse, task \( \tau_1 \) is preempted by task \( \tau_2 \) at \( t_4 \). Task \( \tau_3 \) will only be able to continue its execution at \( t_7 \), after \( \tau_3 \) has finished its execution (\( t_5 \)) and after \( \tau_2 \) releases \( r^g \) (\( t_6 \)). The blocking interval of task \( \tau_3 \) is represented by \( B_3 \). In Fig.1(b), the notion of bandwidth inheritance is depicted. The main difference in relation to the previous schedule is that server \( S_2 \) is able to undertake the processing of task \( \tau_1 \) when it is preempted by \( \tau_2 \) at instant \( t_4 \). Task \( \tau_1 \) then executes the critical section and releases \( r^g \) at \( t = 5 \). Task \( \tau_3 \) is then able to lock \( r^g \) much earlier than in the schedule presented in Fig.1(a) and hence \( B_3 \) is reduced.

4.1 Adaptation of the NPS-F Schedulability Analysis

The main adaptation required is related to the restriction adopted in MrsP [6] where tasks are scheduled using fixed priority. Due to the fact that under NPS-F each server serves one or more tasks employing an EDF scheduling policy [10], the scheduling analysis defined by Equation (2) can no longer be applied. Considering this, a schedulability analysis that accounts for resources sharing under the EDF scheduling policy needs to be defined.

Thus we need to investigate how to account for the effects of (global and local) shared resources into Equation (4). As shown in [3], EDF-scheduled systems in which accesses to shared resources are arbitrated by SRP can be integrated into the analysis using a Blocking Function \( B(t) \). This function provides the longest blocking time a higher priority task can experience when blocked by a lower priority task. Based on [3], the Blocking Function \( B(t) \) can be approximated by a function \( B^r(S_q) \) and incorporated into Equation (4), resulting in the following schedulability test:

\[
\forall t : B^r(S_q) + \text{DBF}(S_q, t) \leq \text{SBF}(S_q, t)
\]  

(6)
where $B^2(S_0) = \max\{\hat{\epsilon}, \hat{b}\}$ is the blocking term due to local resources used by tasks served by $S_0$. In order to account for the global resources shared under the MrsP protocol, the DBF can be expanded as follows:

$$B^2(t) + \sum_{k=1}^{n} \max \left( 0, \frac{t - D_k}{T_k} + 1 \right) \times C_k \leq SBF(S_n, t)$$

where $C_k$ is given by:

$$C_k = WCET(\tau_k) + \sum_{r^j \in F(\tau_k)} e^j$$

where $e^j$ is now given by $e^j = |mapserv(G(r^j))| \times c^j$. Function $mapserv$ returns the set of servers onto which the tasks accessing $r^j$ are assigned. Therefore, similarly to Equation (4), the worst-case execution time of a task $\tau_i$ is augmented by the maximum number of parallel access to each resource used by $\tau_i$. Since global accesses are performed in a FIFO manner and because a priority ceiling protocol is used locally to each server, there may be at most one parallel access per server in which the resource is used. The $C_i$ of each task is therefore influenced by the number of servers accessing the resource rather than the number of processors as it was the case in Equation (4).

### 4.2 Mapping of tasks to servers

Equation (8) shows that the execution time $e^j$ of resource $r^j$ depends on the number of servers that have parallel access to $r^j$. This leads to one of the key challenges foreseen when applying Inequality (7), i.e., how to perform the mapping of tasks into servers under NPS-F. This mapping uses the schedulability test provided by Inequality (7). However, to perform that schedulability test on a task $\tau_i$, it is necessary to know where the tasks accessing the same resources than $\tau_i$ are assigned (through function $mapserv$). Therefore, this leads to a circular dependency between the calculation of the parallel access time to the resources and the assignment of the tasks to the servers.

One of the possible solutions to overcome this circularity issue for global resources is to assume a worst case scenario in terms of parallelism, when computing the schedulability test of each task $\tau_i$. This can be achieved by assuming that all the tasks sharing resources with $\tau_i$ are mapped to different servers. Under this scenario, two kinds of upper bounds on the amount of parallelism for the access to the resource can be considered: (i) the number of tasks that share resources with $\tau_i$; (ii) the maximum number of servers onto which the tasks can be allocated.

The smallest of these two values can then be used as an upper bound on $|mapserv(G(r^j))|$. In this way the circularity issue is broken. The allocation of the subsequent tasks can be improved by taking into consideration the mapping decisions already taken, and not the worst case any more. However, we still have to consider the worst case scenario for the tasks that have not been allocated to a server yet, but that share resources with $\tau_i$. Note that, through the definition of the parameter $\hat{\epsilon}$, local resources exhibit a similar circular dependency between the mapping decisions and the schedulability test. Again, this dependency could be broken by considering the worst-case value for $\hat{\epsilon}$, which can be computed taking the maximum blocking time that $\tau_i$ can suffer from tasks already assigned to the same server $S_0$ and those that are not yet assigned to any server.

### 5. CONCLUSIONS

In this paper we present a framework for scheduling real-time tasks in multicore platforms with resources sharing. The solution was based on an adaptation of MrsP resource sharing protocol to work with the server-based semi-partitioned scheduling algorithm NPS-F. The schedulability analysis of tasks assigned to a server is provided, taking into account the blocking time due to shared resources. The next foreseen step is the mapping of the tasks to the servers, which has circular dependencies with the schedulability test provided. The method designed for NPS-F could then be extended to any server based scheduling algorithm for multicore architectures and hence be used to design assignment techniques ensuring the isolation of different applications sharing the same computing platform.

### Acknowledgements

This work was partially supported by National Funds through FCT (Portuguese Foundation for Science and Technology) and by ERDF (European Regional Development Fund) through COMPETE (Operational Programme “Thematic Factors of Competitiveness”), within project PCOMP-01-0124-FEDER-037281 (CI3TER), and by National Funds through FCT and the EU ARTEMIS JU funding, within projects ref. ARTEMIS/0001/2012, JU grant nr. 333053 (CONCERTO) and ref. ARTEMIS/0001/2013 (JU grant nr. 621429 - EMC2).

### 6. REFERENCES