



Schedulability analysis of dynamic priority real-time systems with contention

Ana Guasque¹ · José María Aceituno¹ · Patricia Balbastre¹ · José Simó¹ · Alfons Crespo¹

Accepted: 10 March 2022 / Published online: 8 April 2022
© The Author(s) 2022

Abstract

In multicore scheduling of hard real-time systems, there is a significant source of unpredictability due to the interference caused by the sharing of hardware resources. This paper deals with the schedulability analysis of multicore systems where the interference caused by the sharing of hardware resources is taken into account. We rely on a task model where this interference is integrated in a general way, without depending on a specific type of hardware resource. There are similar approaches but they consider fixed priorities. The schedulability analysis is provided for dynamic priorities assuming constrained deadlines and based on the demand bound function. We propose two techniques, one more pessimistic than the other but with a lower computational cost. We evaluate the two proposals for different task allocators in terms of the increased estimated utilization. The results show that both bounds are valid for ensuring schedulability although, as expected, one is tighter than the other. The evaluation also serves to compare allocators to see which one produces less interference.

Keywords Real-time · Scheduling · Dynamic priorities · Contention · Multiple sources contention

✉ Ana Guasque
anguaor@ai2.upv.es

José María Aceituno
aceituno@ai2.upv.es

Patricia Balbastre
patricia@ai2.upv.es

José Simó
jsimo@ai2.upv.es

Alfons Crespo
acrespo@ai2.upv.es

¹ Instituto de Automática e Informática Industrial (ai2), Universitat Politècnica de València, Camino de Vera, s/n, 46022 Valencia, Spain

1 Introduction

The use of multicore in embedded systems is already widespread. In critical real-time systems, ensuring the temporal requirements of multicore systems is much more complicated than in single-processor systems. Not only because there is one more dimension (the number of processors) but also because the execution of tasks no longer depends not only on their own computational time, but also on the hardware resources shared between processors. This sharing means interference between processors, so that there are additional delays in the execution of tasks because these resources may be being used by other tasks on other processors. Some works have identified the main sources of interference in multicore systems such as in [11, 16, 21]. Three main sources of interferences are identified: cache, memory bus and main memory. In a multicore platform with a shared memory model, the data in the cache must be kept coherent. To prohibit access to stale data, additional bus transactions are required. This increases indeterminism. In general, the main memory comprises multiple components such as ranks, banks, and buses that cause unpredictability due to its non-deterministic access time.

The study of such systems, in particular the analysis of interference, has been the focus of attention in recent years in the real-time systems community. Many works have focused on calculating this interference at a low level for each type of shared hardware resource. The idea is to add this estimated interference to the task model, either directly by adding it to the Worst Case Execution Time (WCET), or by adding it to the schedulability test. The advantage is that this estimation is very close to the real interference values. The problem is that this analysis is only valid for that type of hardware resource.

Another approach is to consider a more general task model, not directly linked to the type of shared hardware resource and therefore independent of it. This solution is the only one possible when the hardware vendor does not provide detailed information about the shared resource behaviour. The advantages and disadvantages are obvious: the model does not depend on the specific type of hardware, but by not modelling interference in detail, the temporal analysis is more pessimistic.

The recent work [1] proposes a general task model that considers the contention of the previous mentioned hardware shared resources. They also propose an allocation algorithm to minimize the interference due to contention of shared hardware resources. Nevertheless, their work lacks a schedulability test and assumes an implicit deadline task model for both fixed and dynamic priorities.

1.1 Contribution

This paper proposes a schedulability test for multicore real-time systems. We extend the model in [1] to consider constrained task deadlines. In particular, authors propose two contention-aware demand bound functions with different levels of pessimism. The novelty of the contribution is the consideration of dynamic priority scheduling in a model that considers interference due to shared hardware resources

and constrained deadlines. We use a general model that can be used with different types of shared hardware resources in contrast with other works that assume a very specific kind of resource. Other works assume only fixed priorities or the interference is only valid for a specific type of shared resource.

This work is organized as follows: The main contributions in the related research area are presented in Sect. 2. Section 3 presents the system model for constrained-deadline systems. Section 4 reviews the classical schedulability analysis for multi-core systems with dynamic priorities. Then, we propose two upper bounds to the *dbf* based on the classical analysis with their corresponding schedulability tests. The experimental evaluation in Sect. 5 demonstrates the acceptance of our schedulability tests. It also compares the different allocation techniques in terms of degree of approximation between the proposed algorithms in terms of utilisation and the actual utilisation.

2 Related works

There has been a trend towards using multicore platforms due to their high computing performance. From the key results in the field in 2006, there is a lot of research about real-time multicore systems. Some of the main surveys in the area are [14, 16, 17, 25, 26].

This work is focused on hard real-time systems, in which the non-compliance of temporal constraints could have dramatic consequences. Therefore, partitioned scheduling is considered in these systems, as migration of tasks between cores is not allowed. Then, the problem of scheduling in a multicore hard real-time system involves a first phase of task to cores allocation and a second phase of independent scheduling of each core.

However, in multicores, the schedulability analysis does not only consider the WCET of each task but also the interference produced when tasks are executing in parallel on other cores and access to the shared hardware resource. This way, the timing correctness of the hard real-time system becomes more complicated. In [25] it is presented a survey about timing verification techniques for multi-core real-time systems until 2018.

In order to integrate the effects of the interference in the schedulability analysis, each task is characterized by its WCET (running in isolation) and the effect of the contention of the shared resource in the response time of the task. Many works consider a single shared resource: memory bus [10, 23], scratchpad memories and DRAM [19, 22, 34], etc. For example, [19] focus on the analysis of memory contention delays in heterogeneous commercial-off-the-shelf (COTS) MPSoC platforms, where their goal is to derive a safe bound on these delays suffered by any critical task in a mixed-criticality system executing on these platforms upon accessing the off-chip DRAM.

However, other works consider multiple shared resources in the contention, which is on the scope of this work. Among the most relevant works of interference contending for multiple resources, [2] presented a Multicore Response Time Analysis (MRTA) framework, that provides a general approach to timing verification for multicore

systems. They omit the notion of WCET and instead directly target the calculation of task response times through execution traces. They start from a given mapping of tasks to cores and assume fixed-priority preemptive scheduling. Other works as [12] or [30] come from the MRTA framework.

In [20], a schedulability test and response time analysis for constrained-deadline systems is proposed. They analyse the amount of time for shared resource accesses and the maximum number of access segments, which is out of the scope of this work. They also assume fixed priorities.

[7] propose a conservative modeling technique of shared resource contention supporting dependent tasks, in contrast to our work, that considers independent tasks. They also assume fixed-priority scheduling. Another work that considers fixed priority scheduling is [3]. This work presents a task model for tasks with co-runner-dependent execution times that generalizes the notion of interference-sensitive WCET (isWCET). Their model considers constrained-deadline sporadic task sets and a fixed priority scheduling. Here, tasks are represented by a sequence of segments, each of which has execution requirements and co-runner slowdown factors with respect to sets of other segments that could execute in parallel with it.

In [1], the interference due to contention is added to the temporal model. Instead of adding it to the WCET, they propose a scheduling algorithm that computes the exact value of interference and an allocator that tries to reduce this total interference. This model considers implicit deadlines in the system and both fixed or dynamic priorities can be used. However, they do not propose any schedulability test to ensure the system's feasibility.

In [18], a partitioned scheduling that considers interference while making partition and scheduling decisions is presented. They present a mixed integer linear programming model to obtain the optimal solution but with a high computation cost and also they propose approximation algorithms. They only consider implicit deadline models. This paper differentiates between isolated WCET and the WCET with interference and overhead. They define an Inter-Task interference matrix, in which each element of the matrix is the interference utilisation between two tasks, considering the inflated WCET when two tasks run together. This work is similar to [1] but in [1] a general model is considered, valid for any type of shared hardware resource while in [18] only interference due to cache sharing is considered.

A similar work is presented in [15]. They define the Multicore Resource Stress and Sensitivity (MRSS) task model that characterises how much stress each task places on resources and its sensitivity to such resource stress. This work also considers a general model to cope with different hardware resources but only fixed priority scheduling policies are considered. The task-to-cores allocation is known *a priori*.

In this work, we proceed under the assumption of the task model in [1] (as fixed and dynamic priorities can be used) with an extension to a constrained-deadline model and we propose two schedulability tests for dynamic priorities.

3 Problem definition and task model

This task model is similar to the one presented in [1]. The only difference is the introduction of the parameter D in the temporal model since in this case we will assume that deadlines are less than or equal to periods. We suppose a multicore system with m cores $(M_0, M_1, M_2, \dots, M_{m-1})$ where a synchronous task set τ of n independent tasks should be allocated to these cores. Each task τ_i is represented by the tuple:

$$\tau_i = (C_i, D_i, T_i, I_i) \quad (1)$$

where C_i is the WCET, D_i is the relative deadline, T_i is the period and I_i is the interference. We assume constrained deadlines, so $D_i \leq T_i$. Tasks can be periodic or sporadic.

The term I_i is the time the task takes to access shared hardware resources. A typical case is the operation of reading and writing in memory. Although I_i is part of C_i , during the time the task is accessing the shared resource, other tasks on other cores will be delayed if they try to access the same resource. So this interference time is defined independently of C_i , as will be used to represent the delay caused to other tasks. A detailed description of this parameter can be found in [1].

When we refer to M_{τ_i} , we mean the core in which τ_i is allocated. Moreover, we will define as τ_{M_k} the subset of tasks in τ that belong to the core M_k . Therefore, $\tau_{M_0} \cup \dots \cup \tau_{M_{m-1}} = \tau$.

The hyperperiod of the task set, H , is the smallest interval of time after which the periodic patterns of all the tasks are repeated, and it is calculated as the least common multiple of the task periods. The utilisation of a task τ_i is calculated as the relation between the computation time and the period, $U_i = \frac{C_i}{T_i}$. The utilisation of a core M_k is the sum of the utilisation of all tasks that belong to this core: $U_{\tau_{M_k}} = \sum_{\tau_i \in M_k} U_i$. The total utilisation of the system is the sum of the utilisation of all cores: $U_\tau = \sum_{\forall k} U_{\tau_{M_k}}$.

We define A_i as the number of activations that τ_i has throughout H , $A_i = H/T_i$.

We will also need the following definitions:

Definition 1 [1] A task is defined as a receiving task when it accesses shared hardware resources and suffers an increase in its computation time due to the interference produced by other tasks allocated to other cores.

Definition 2 [1] A task is defined as a broadcasting task when it accesses shared hardware resources and provokes an increase in computation time in other tasks allocated to other cores due to contention.

If $I_i = 0$, τ_i is neither broadcasting nor receiving task. If $I_i > 0$, τ_i will be a broadcasting and receiving task if there is at least one task τ_j in other core whose $I_j > 0$.

Figure 1 represents the scheduling of a system when interference is considered. In the example, there is a set of three tasks allocated to a platform with two

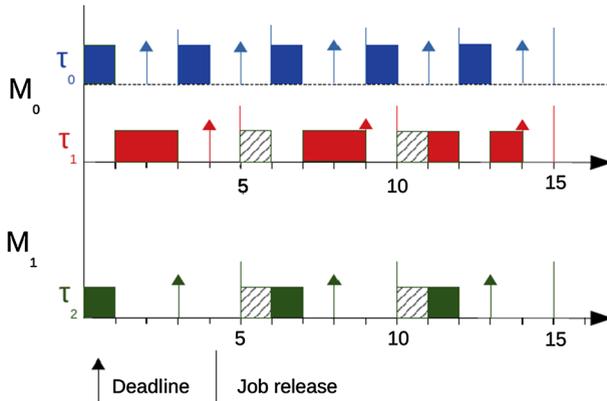


Fig. 1 Example. Execution of the task set with $\tau_0 = (1, 2, 3, 0)$, $\tau_1 = (2, 4, 5, 1)$, and $\tau_2 = (1, 3, 5, 1)$ allocated to a dual-core platform

cores. Tasks τ_0 and τ_1 are allocated to core 0 and τ_2 , to core 1. As $I_0 = 0$ is neither broadcasting nor receiving task, so it only executes its WCET. As $I_1, I_2 > 0$ and are allocated to different cores, both are broadcasting and receiving tasks. When they coincide in execution, the interference appears as extra units of execution due to accesses to shared hardware resources (depicted as rectangles with diagonal lines). Note that interference is not caused in all activations, only when two or more broadcasting tasks in different cores are executing.

4 Interference-aware schedulability analysis for dynamic priorities

There are two phases in order to obtain the schedulability plan of a partitioned multicore system: First, tasks are allocated to cores and then, each core schedules its tasks. Migration is not allowed, since the context of our problem is highly critical real-time systems. Then, once the allocation is performed, the scheduling can be solved as a multiple monocoresh scheduling problem.

In this section, first we present a well-known dynamic-priority schedulability analysis for constrained deadline task models and then we extend the study to consider the effect of the interference.

4.1 Earliest deadline first schedulability analysis

Dynamic priority-based schedulers do not assign an initial priority to the tasks but at runtime. Earliest Deadline First (EDF) in [24] is an optimal scheduling algorithm for dynamic priorities. EDF assigns the highest priority to the task with the earliest absolute deadline, which is $a \cdot T_i + D_i$ for the a^{th} activation in a periodic task τ_i .

With constrained deadlines task models, the demand bound function, dbf_{τ_i} , determines the schedulability of the system. It is a positive and increasing function that only increases in scheduling points i.e., when a deadline arrives. For

partitioned scheduling, this function is calculated for each core so if a task set τ is allocated to m cores, the system is characterised by m demand bound functions.

Definition 3 [4] The maximum cumulative execution time requested by a set of synchronous tasks τ_{M_k} over any interval of length t is:

$$dbf_{\tau_{M_k}}(t) = \sum_{\forall \tau_i \in M_k} C_i \left\lceil \frac{t + T_i - D_i}{T_i} \right\rceil \quad (2)$$

Therefore, the task set τ_{M_k} is schedulable by EDF if and only if [5]:

$$dbf_{\tau_{M_k}}(t) \leq t \quad \forall t \leq H \quad (3)$$

However, studying the demand bound function over all the hyperperiod H is a tedious process. Some schedulability tests as in [6] reduce the time interval in which the schedulability condition must be satisfied. In 1996, [31, 33] derived another upper bound for the time interval which guarantees the schedulability of the task set. This interval is called the *synchronous busy period*. It is a processor busy period in which all tasks are released simultaneously at the beginning of the processor busy period and ends by the first processor idle period. Its length is the maximum of any possible busy period in any schedule. The length of this interval is calculated by an iterative process [31, 33] Then, the schedulability condition is defined as:

Theorem 1 [31] A general task set τ_{M_k} is schedulable if and only if $U_{\tau_{M_k}} \leq 1$ and

$$dbf_{\tau_{M_k}}(t) \leq t, \quad \forall t \leq L_b$$

where L_b is the length of the synchronous busy period of the task set.

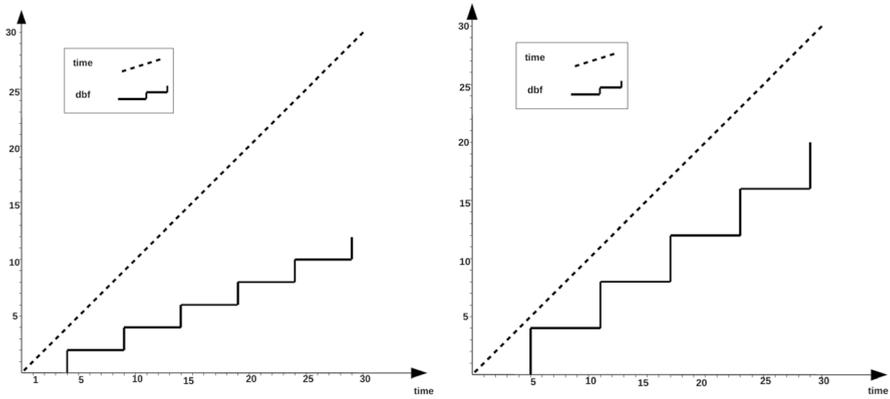
4.2 Interference-aware schedulability analysis for EDF

In this section, we propose a demand bound function that considers the interference so we can provide a schedulability test.

Let us start with the following task set, $\tau = [\tau_0, \tau_1]$ with $\tau_0 = (2, 4, 5, 1)$ and $\tau_1 = (4, 5, 6, 1)$, allocated to a dual-core platform. τ_0 is allocated to core M_0 , and τ_1 , to M_1 . Figure 2 shows the demand bound function for each core, according to Eq. 2. Note that this function only considers the execution times of the task set of each core and not the received interference.

In order to derive a demand bound function with interference considerations, some definitions need to be introduced.

Definition 4 Let $\overline{v_{j \rightarrow i}}$ be the activation pattern from a broadcasting task τ_j to a receiving task τ_i .



(a) Demand bound function *dbf* for τ_{M_0} (b) Demand bound function *dbf* for τ_{M_1}

Fig. 2 (a) Demand bound function *dbf* for τ_{M_0} (b) Demand bound function *dbf* for τ_{M_1} . Demand bound functions for the task set, $\tau = [\tau_0, \tau_1]$ with $\tau_0 = (2, 4, 5, 1)$ and $\tau_1 = (4, 5, 6, 1)$, allocated to a dual-core platform. τ_0 is allocated to core M_0 , and τ_1 , to M_1 . Interference is not considered

The meaning of this array is the number of activations of τ_j that fall within an activation a of τ_i . Note that this is an array that later we will use to represent the interference that a broadcasting task τ_j can cause to a receiving task τ_i . In this way, $\overline{v_{j \rightarrow i}}[a]$ coincides with the maximum overlapping between τ_j and τ_i in its a^{th} activation, etc. The length of this array coincides with the number of activations of τ_i .

The following example shows graphically how the array $\overline{v_{j \rightarrow i}}$ is characterised. Let us suppose a system with two tasks, $\tau' = [\tau'_0, \tau'_1]$ with $\tau'_0 = (1, 2, 3, 1)$ and $\tau'_1 = (1, 6, 7, 1)$, allocated to a dual-core platform. τ'_0 is allocated to core M_0 , and τ'_1 , to M_1 . For the sake of simplicity, computation times and deadlines are not depicted. As seen in Fig. 3, $\overline{v_{1 \rightarrow 0}} = [1, 1, 2, 1, 2, 1, 1]$. Equivalently, $\overline{v_{0 \rightarrow 1}} = [3, 3, 3]$.

Note that in the scheduling, not all activations receive the same interference from other tasks. As showed in Fig. 1, this will depend on whether the tasks coincide in execution.

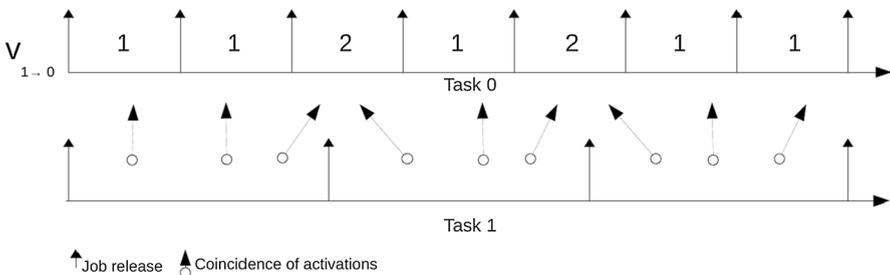


Fig. 3 Example of $\overline{v_{j \rightarrow i}}$ for a task set with $\tau'_0 = (1, 2, 3, 1)$ and $\tau'_1 = (1, 6, 7, 1)$ allocated to a dual-core platform

In the next theorem, we are going to provide an expression to calculate $\overrightarrow{v_{j \rightarrow i}}$ and we will demonstrate that this is the maximum number of activations of the broadcasting task τ_j that fall within each activation of τ_i .

Theorem 2 *The maximum number of activations of the broadcasting task τ_j that fall within a^{th} activation of τ_i is:*

$$\overrightarrow{v_{j \rightarrow i}}[a] = 1 + \sum_{t=aT_i+1}^{(a+1)T_i-1} g(t) \tag{4}$$

being

$$g(t) = \begin{cases} 1 & \text{If } t - T_j \left\lfloor \frac{t}{T_j} \right\rfloor = 0 \\ 0 & \text{Elsewhere} \end{cases} \tag{5}$$

Proof Let us assume that exists t_1 so that $t_1 = \alpha \cdot T_j$ and $aT_i \leq t_1 < (a + 1)T_i$.

In this case:

$$t_1 - T_j \left\lfloor \frac{t_1}{T_j} \right\rfloor = \alpha \cdot T_j - T_j \left\lfloor \frac{\alpha \cdot T_j}{T_j} \right\rfloor = 0$$

and then $g(t_1) = 1$.

Therefore, $g(t)$ is equal to 1 only when the broadcasting task τ_j is released in the interval $[aT_i + 1, (a + 1)T_i)$. Evaluating $g(t)$ all over the previous interval we get the number of activations that fall within activation a of τ_i .

It is not possible for the sum of $g(t)$ to be greater than this number of activations, so we can say that the maximum number of activations falling within the interval is correctly calculated with Eq. 4. □

Note that if τ_i is not a receiving task, $\overrightarrow{v_{j \rightarrow i}}[a] = 0 \quad \forall j, a$.

Listing 1 shows the pseudo-code (python-like) to calculate $\overrightarrow{v_{j \rightarrow i}}$.

Listing 1: Maximum interference array algorithm.

```

1 #variables definition and initialization
2  $\overrightarrow{v_{j \rightarrow i}}$  = [None]Ai
3 for a in range(Ai):
4     i = 1
5     for t in range(aTi+1, (a+1)Ti, 1):
6         if t % Tj == 0 and Ii ≠ 0 and Ij ≠ 0:
7             i = i + 1
8      $\overrightarrow{v_{j \rightarrow i}}$ [a] = i
    
```

In the next sections, we will use vector $\overrightarrow{v_{j \rightarrow i}}$ to obtain a demand bound function that incorporates the interference. We will present two proposals, one more pessimistic than the other but with a lower computational cost.

4.2.1 A first approximation

In order to deduce a demand bound function that contemplates the interference, a first approximation is to consider that all the activations of the receiving task τ_i receive the maximum possible interference from the broadcasting task τ_j . From the definition of $\overrightarrow{v_{j \rightarrow i}}$ array, this maximum interference is calculated as:

$$\max_{0 \leq a \leq A_i - 1} \overrightarrow{v_{j \rightarrow i}}[a] \cdot I_j \quad (6)$$

Then, the execution time of a receiving task will be defined as the sum of its own computation time and the worst-case interference received by the broadcasting tasks allocated in other cores.

$$C'_i = C_i + \sum_{\tau_j \notin M_k} \max_{0 \leq a \leq A_i - 1} \overrightarrow{v_{j \rightarrow i}}[a] \cdot I_j \quad (7)$$

Note that if τ_i is not a receiving task, $\overrightarrow{v_{j \rightarrow i}}[a] = 0 \quad \forall j, a$ and then $C'_i = C_i$.

Definition 5 From Eq. 2, we can propose a new definition of the demand bound function with interference considerations:

$$dbf'_{\tau_{M_k}} = \sum_{\forall \tau_i \in M_k} C'_i \left\lfloor \frac{t + T_i - D_i}{T_i} \right\rfloor \quad (8)$$

Consequently, the corresponding schedulability test is:

Theorem 3 A task set τ_{M_k} allocated to a core M_k with constrained deadlines is schedulable by dynamic priorities if and only if:

$$dbf'_{\tau_{M_k}}(t) \leq t \quad \forall t \leq L_b \quad (9)$$

Proof As $\overrightarrow{v_{j \rightarrow i}}[a]$ represents the maximum number of activations of τ_j that fall within an activation of τ_i , the maximum of this array multiplied by the interference factor I_j is the maximum interference that τ_i can receive from τ_j . There is no possibility for a task to receive more interference than $\max_a \overrightarrow{v_{j \rightarrow i}}[a] \cdot I_j$ in activation a so if by adding this value to the demand function the system is schedulable, no deadline will be lost in the execution. \square

By adding this maximum value to C_i , we are considering that the maximum interference occurs in all activations. On the one hand, introducing the same maximum value of interference in all activations makes the system still periodic. Then, the schedulability test must be satisfied in the synchronous busy period and not in the hyperperiod and only in the scheduling points. On the other hand, previous definition is very pessimistic as not all the activations of the receiving task receive its maximum value of interference. This is only a first approximation in order to provide a simple schedulability test. As $\overrightarrow{v_{j \rightarrow i}}[a] \cdot I_j$ exactly provides the maximum possible interference received from τ_j at each activation a of τ_i ,

next section presents a more accurate (less pessimistic) definition of the demand bound function.

4.2.2 A more accurate definition

In this section, let us present a schedulability test using the exact definition of the activation pattern array, $\overline{v_{j \rightarrow i}}$. We will estimate an upper bound of the interference received with this array and we will include it, not in the computation time but in the demand bound function.

Definition 6 The less pessimistic demand bound function of a task set allocated to a core M_k considering the interference is:

$$dbf''_{\tau_{M_k}}(t) = \sum_{\tau_i \in M_k} \left(C_i \left\lfloor \frac{t + T_i - D_i}{T_i} \right\rfloor + \sum_{\tau_j \notin M_k} \sum_{a=0}^{\left\lfloor \frac{t+T_i-D_i}{T_i} \right\rfloor - 1} \overline{v_{j \rightarrow i}}[a] \cdot I_j \right) \quad (10)$$

Corollary 1 $dbf''_{\tau_{M_k}}(t)$ is less pessimistic than $dfb'_{\tau_{M_k}}(t)$. Therefore:

$$dbf''_{\tau_{M_k}}(t) \leq dfb'_{\tau_{M_k}}(t)$$

Proof It is easy to see that:

$$\sum_{a=0}^{\left\lfloor \frac{t+T_i-D_i}{T_i} \right\rfloor - 1} \overline{v_{j \rightarrow i}}[a] \leq \max_{0 \leq a \leq A_i - 1} \overline{v_{j \rightarrow i}}[a] \cdot \left\lfloor \frac{t + T_i - D_i}{T_i} \right\rfloor$$

{To simplify the notation, let us define $\max_{0 \leq a \leq A_i - 1} \overline{v_{j \rightarrow i}}[a]$ as $maxV$.} Developing both sides of the equation:

$$\overline{v_{j \rightarrow i}}[0] + \overline{v_{j \rightarrow i}}[1] + \dots + \overline{v_{j \rightarrow i}} \left[\left\lfloor \frac{t + T_i - D_i}{T_i} \right\rfloor - 1 \right] \leq maxV + maxV + \dots + maxV$$

In both sides of the previous equation there are exactly $\left\lfloor \frac{t+T_i-D_i}{T_i} \right\rfloor$ terms. As $maxV \geq \overline{v_{j \rightarrow i}}[a] \forall a$, each term on the right side is equal or greater than each term on the left side. Therefore:

$$dbf''_{\tau_{M_k}}(t) \leq \sum_{\tau_i \in M_k} \left(C_i \left\lfloor \frac{t + T_i - D_i}{T_i} \right\rfloor + \sum_{\tau_j \notin M_k} \max_{0 \leq a \leq A_i - 1} \overline{v_{j \rightarrow i}}[a] \cdot \left\lfloor \frac{t + T_i - D_i}{T_i} \right\rfloor I_j \right)$$

□

The second term in Eq. 10 includes the upper bound of the total interference received by τ_i . It is calculated as the number of interferences that all the tasks allocated in other cores provoke to all activations of τ_i released until time t . If τ_i is

neither receiving nor broadcasting, the second term of this equation will be equal to 0 (as $\overrightarrow{v_{j \rightarrow i}}[a] = 0 \quad \forall j, a$). If τ_j is not broadcasting, $I_j = 0$ and also $\overrightarrow{v_{j \rightarrow i}}[a] = 0$.

This is an upper bound in the sense that the $\overrightarrow{v_{j \rightarrow i}}[a]$ array is maximum, as demonstrated in Theorem 2.

However, when interference is considered throughout the $\overrightarrow{v_{j \rightarrow i}}[a]$ array, the maximum cumulative execution time requested by the tasks may happen indifferently at any time during all the hyperperiod making the demand not periodic. With the following counterexample, we will show that studying the schedulability of the system in the synchronous busy period is not a valid approach.

4.2.3 Counterexample for dynamic priorities

Let us consider the task set defined in the beginning of Sect. 4.2, $\tau = [\tau_0, \tau_1]$ with $\tau_0 = (2, 4, 5, 1)$ and $\tau_1 = (4, 5, 6, 1)$, allocated to a dual-core platform. τ_0 is allocated to core M_0 , and τ_1 , to M_1 . Once tasks are allocated to cores, EDF algorithm schedules tasks in each core. The actual execution of the task set is shown in Fig. 4. From [31], proving that $dbf_{\tau_{M_k}}(t) \leq t$ during the first busy period is enough to assure the schedulability of the task set. As seen in Fig. 4, the first busy period in M_1 is $[0,5]$, as $t=5$ is the first instant when all requests have already been served and no additional requests have arrived yet. Then, it may be assumed that the system is schedulable. However, due to interferences, a deadline miss is produced in $t=10$. So, we can conclude that when interference is considered, the worst scenario may happen at any time during the hyperperiod.

Moreover, not always this interference will be received, it will depend on the real scheduling. For example, suppose that the WCET of τ_0 is 1 unit instead of 2. Then, the second activation of τ_0 will not interfere with τ_1 as it ends just when the second activation of τ_1 starts. However, as this work evaluates the worst-case scenario, it will consider that these activations interfere as one falls within the other.

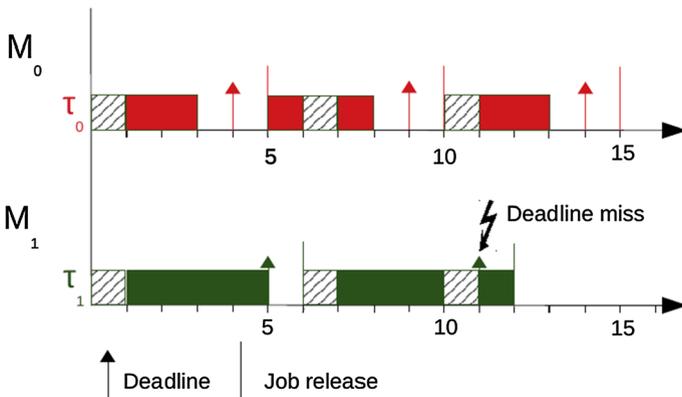


Fig. 4 Counterexample. Execution of the task set with $\tau_0 = (2, 4, 5, 1)$ and $\tau_1 = (4, 5, 6, 1)$ allocated to a dual-core platform

The schedulability condition for constrained-deadline synchronous and periodic task models based on the demand bound function was presented in Eq. 3. This equation is applied with periodic or sporadic tasks, whose requests happen every inter-arrival time, T_i . When interference is considered with the first approach $dbf'_{\tau_{M_k}}$, i.e., considering the maximum of the array $\overline{v_{j \rightarrow i}}$, the model is still periodic, as this maximum value is introduced in all the activations. However, if we consider the array $\overline{v_{j \rightarrow i}}$ per se in the $dbf''_{\tau_{M_k}}$, the behaviour of the task set is no longer periodic, as there is no repeatability. Other works as [5, 29] include the definition of the demand bound function for extended models, for example, those in which tasks are also defined by their start times, whose behaviour would be similar to ours with interference. Therefore, schedulability tests are evaluated by intervals, in particular, by the intervals of activation of each task, to ensure that all temporal constraints are met. For this reason and from now on, the demand bound function will be evaluated by intervals, $dbf_{\tau_{M_k}}(t_1, t_2)$, with $0 \leq t_1 < t_2 \leq H$. Note that $dbf_{\tau_{M_k}}(t_1, t_2) = dbf_{\tau_{M_k}}(t_2) - dbf_{\tau_{M_k}}(t_1)$. It can be applied to all the demand bound functions presented in this work.

Considering the interference in the demand bound function and the previous considerations, the schedulability test is now presented.

Theorem 4 *A task set τ_{M_k} allocated to a core M_k with constrained deadlines is schedulable by dynamic priorities if:*

$$dbf''_{\tau_{M_k}}(t_1, t_2) \leq t_2 - t_1 \quad 0 \leq t_1 < t_2 \leq H \tag{11}$$

Proof Similar to the proof of Theorem 3, $\overline{v_{j \rightarrow i}}[a]$ represents the maximum number of times a task τ_j can cause interference in another task τ_i at an activation a . Thus, it is not possible to receive in total more than:

$$\sum_{\tau_i \in M_k} \sum_{\tau_j \notin M_k} \sum_{a=0}^{\lfloor \frac{t+T_i-D_i}{T_i} \rfloor - 1} \overline{v_{j \rightarrow i}}[a] \cdot I_j \tag{12}$$

interference units. If the system slack ($t_2 - t_1 - dbf(t_1, t_2)$) is greater than or equal to this value, the set of tasks will be schedulable. □

Let us follow with the example in Sect. 4.2.3. Figure 5 shows the demand bound functions dbf , dbf' and dbf'' for both cores. First, the activation patterns are calculated: $\overline{v_{1 \rightarrow 0}} = [1, 2, 2, 2, 2, 1]$ and $\overline{v_{0 \rightarrow 1}} = [2, 2, 2, 2, 2]$. Figures 5a and b show the demand bound functions for cores 0 and 1, respectively. In Fig. 5b, $dbf' = dbf''$ as $\max_{0 \leq a \leq A_i - 1} \overline{v_{j \rightarrow i}}[a] = \overline{v_{j \rightarrow i}}[a] \quad \forall a$.

As seen in Fig. 5a, there is a deadline missed in the execution of the tasks in core M_1 . This infeasibility is demonstrated with the demand bound functions (Fig. 5b) as $dbf'_{\tau_{M_1}}(0, t) > t$ and $dbf''_{\tau_{M_1}}(0, t) > t$ for some instants of time during the hyperperiod.

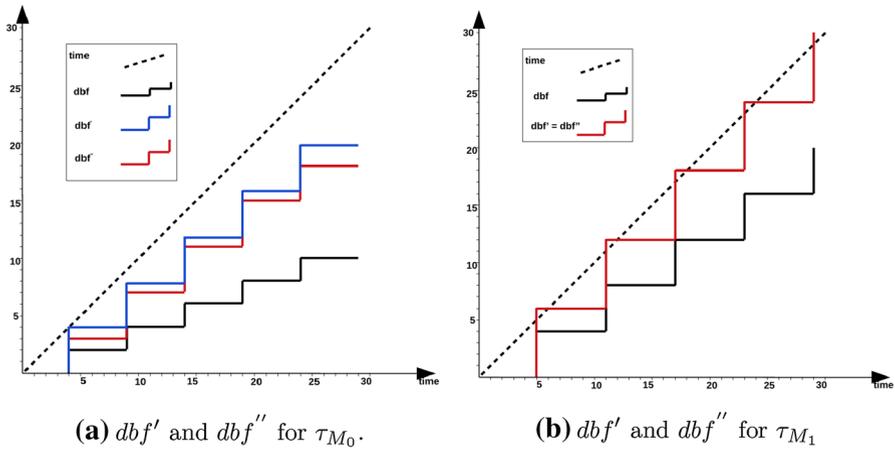


Fig. 5 (a) dbf' and dbf'' for τ_{M_0} (b) dbf' and dbf'' for τ_{M_1} . Relation between dbf' , dbf'' and schedulability of the task set in Sect. 4.2.3

5 Evaluation

5.1 Experimental conditions

In order to obtain the schedule in a multicore system, first the tasks are allocated to cores (allocation phase) and then each core is scheduled (scheduling phase) independently, as this work does not consider migration of tasks between cores.

Therefore, to validate the proposed technique, a simulator that considers both allocation and scheduling phases is implemented. The simulation scenario developed for this work is depicted in Fig. 6. It is divided into three steps:

- Generation of the load (see Sect. 5.1.1).
- Allocation phase (see Sect. 5.1.2).
- Scheduling phase (see Sect. 5.1.3).

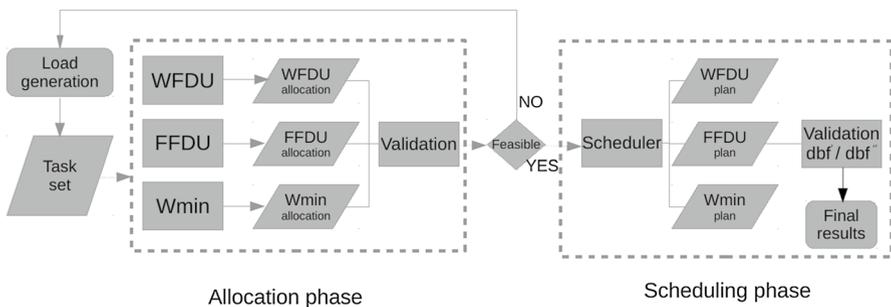


Fig. 6 Experimental evaluation overview

5.1.1 Load generation

The load is generated using a synthetic task generator. The number of tasks in each set and the total system utilisation depends on the number of cores in which tasks are allocated to. Given the total system utilisation and a number of tasks for each set, the utilisation is shared among the tasks using the UUniFast discard algorithm of [13]. Periods are generated randomly in [20,1000] and computation times are deduced from the system utilisation. Deadlines are constrained to be less or equal to periods and are set to $D_i \in [0.5T_i, T_i]$.

The experimental parameters of the evaluation process are specified in Table 1. To ensure the reproducibility of the results, these parameters coincide with those used in [1] (Table 2).

The theoretical utilisation varies between 50 and 75% of the possible maximum load of the system. For example, the maximum load of a system with 4 cores is 4, so for evaluation purposes, the initial utilisation is set to 2.1 ($\approx 50\%$) and 3 (75%).

Table 1 Definition of the experimental scenarios

Number of cores	Utilisation	Tasks	Broadcasting Tasks	Interference Over WCET (%)	Scenario				
2 cores	1.1	4	2	10	1				
				20	2				
				30	3				
	1.5			10	4				
				20	5				
				30	6				
4 cores	2.1	12	3	10	7				
				20	8				
				30	9				
	3			10	10				
				20	11				
				30	12				
				8 cores	4.1	20	5	10	13
								20	14
								30	15
6	10	16							
	20	17							
	30	18							
10 cores	5.1	28	7	10	19				
				20	20				
				30	21				
	7.5			10	22				
				20	23				
				30	24				

The number of broadcasting tasks is set to 25% of the total number of tasks, except for scenarios 1–6 (2 cores), which is 50%. This is due to the fact that, if only one task is broadcasting, no interference will be produced. Each combination of number of cores and utilisation is evaluated with 10, 20 and 30% of interference over the WCET. Note that not all the tasks in a task set have the same interference value, but the same percentage of interference over the WCET.

5.1.2 Allocation phase

Once the load is generated, it is shared among the cores using different algorithms. In the following, we briefly discuss several existing allocation techniques.

Bin packing heuristics such as Worst Fit (WF) and First Fit (FF) are typically used to solve the allocation problem [8, 28]. Moreover, task ordering such as decreasing utilisation (DU) directly affects the task allocation outcomes. In this sense:

- First Fit (FF) algorithm. Each item is allocated into the first bin that it fits into, without exceeding the maximum capacity of the bin. If there is no one available, a new bin will be opened. This algorithm results in an unbalanced allocation between cores.
- Worst Fit (WF) algorithm. WF allocates each item into the bin that leaves more remaining capacity, i.e., the emptiest bin. This algorithm results in a balanced allocation between cores.
- FFDU algorithm (WFDU algorithm). Arrange items i in the decreasing order of utilisation U_i and apply FF (WF) in the resulting order of i .

In addition to these heuristics, there are other bin packing algorithms used to solve the allocation problem. [9] presents a survey and classification of these algorithms.

However, these heuristic techniques do not consider the interference delays due to contention of shared hardware resources but only the utilisation of the tasks. In [1], authors present an allocation algorithm Wmin, whose objective consists of minimizing a binary matrix W that describes if there is (1) or not (0) interference between tasks. They consider that the contention-aware execution time C'_{ia} of τ_i in activation a is the sum of C_i plus the interferences caused by running tasks in other cores. This approximation is similar to the one presented in Sect. 4.2.1. Once these algorithms are introduced, let us continue with the description of the allocation phase in the simulation scenario. Tasks are allocated to cores following the three allocation methods: WFDU, FFDU and Wmin. Each allocator generates an allocation file, that contains the information about how tasks are allocated to cores. Then, the feasibility of this allocation plan must be checked. The validation of the allocation phase consists of assuring that the maximum capacity per core is not exceeded, i.e., $U_{M_k} \leq 1 \forall k = 0, \dots, m - 1$ and that all tasks are allocated. If these conditions are not satisfied, the corresponding task set is discarded and a new one is generated. Otherwise, the task set moves to the scheduling phase.

5.1.3 Scheduling phase

The contention aware scheduling algorithm proposed in [1] is applied in the scheduling phase. As any priority-based algorithm can be used as the basis for this algorithm, we selected EDF, following the proposal of this paper. The scheduler generates a temporal plan, that contains the information about how tasks are scheduled at each time at each core. This plan must also be validated, checking that all temporal constraints are satisfied. First, we need the following definitions:

The utilisation of the core M_k , U'_{M_k} , calculated from the definition of $dbf'_{\tau_{M_k}}$, is:

$$U'_{M_k} = \frac{dbf'_{\tau_{M_k}}(H)}{H} \quad (13)$$

As a consequence, the utilisation of all the system would be the sum of all the core utilisations:

$$U'_\tau = \sum_{\forall k} U'_{M_k} \quad (14)$$

The upper bound of the utilisation of the core M_k , U''_{M_k} , calculated from the definition of $dbf''_{\tau_{M_k}}$, is:

$$U''_{M_k} = \frac{dbf''_{\tau_{M_k}}(H)}{H} \quad (15)$$

As a consequence, the upper bound of the utilisation of all the system would be the sum of all the core utilisations:

$$U''_\tau = \sum_{\forall k} U''_{M_k} \quad (16)$$

The actual utilisation of the system is defined as U_τ^{real} and is determined after the scheduling phase, when the actual interference is measured. This utilisation is always lower or equal than U'_τ and U''_τ , as not always the estimated interference will be produced, as stated in Sect. 4.2.3. From previous sections (see Corollary 1), it is easy to deduce that:

$$U_\tau \leq U_\tau^{real} \leq U''_\tau \leq U'_\tau \quad (17)$$

5.2 Experimental results

After conducting the experiments, the evaluation phase consists of measuring the previous utilisation factors and making a comparison between them, for all the scenarios in Table 1. The objective is to compare the allocation algorithms and confirm that no set with $U_\tau^{real} > U''_\tau$ is schedulable. We also want to know how pessimistic dbf' and dbf'' are.

First, the difference in terms of utilisation between both demand bound functions presented in this work is evaluated. To do that, we measure two parameters:

- Difference between U'_τ and U_τ^{real} , measured as $\alpha' = \frac{U'_\tau - U_\tau^{real}}{U_\tau^{real}}$ (%).
- Difference between U''_τ and U_τ^{real} , measured as $\alpha'' = \frac{U''_\tau - U_\tau^{real}}{U_\tau^{real}}$ (%).

Figures 7, 8 and 9 depict the percentage difference α' and α'' for each scenario and different allocators. As expected, for all scenarios and all allocators $\alpha'' \leq \alpha'$, as a consequence of Eq. 17. As a general trend, the more cores in the system, the bigger α' vs α'' are. This is due to the fact that the estimated worst-case interference increases with the number of cores, as there is more contention between tasks allocated to different cores. However, this difference becomes zero in some cases from scenarios 13. One can differentiate between:

- *FFDU* (Fig. 7): This allocator unbalances the load among cores. It allocates the tasks in the less possible number of cores. Therefore, the used cores are overloaded. Then, when interference appears, the feasibility of the system decreases, as there is little scope to schedule this interference. Generally, FFDU presents low schedulability rates (10% and decreasing in systems from 8 cores [1]). For this reason, there are no values of the percentage difference from 8 cores on (scenario 13 and so on). Scenarios with 2 cores have a bigger percentage of broadcasting tasks as stated in Sect. 5.1.1, so α' and α'' are bigger than in the rest of scenarios.

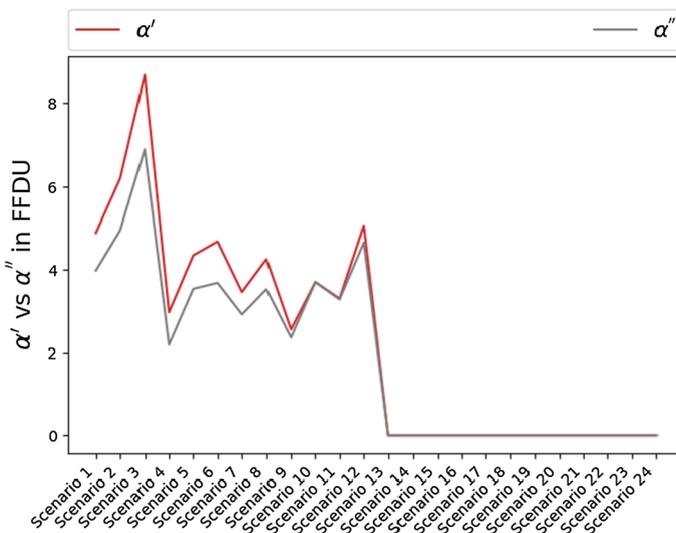


Fig. 7 Percentage difference α' vs α'' for each scenario with FFDU allocator

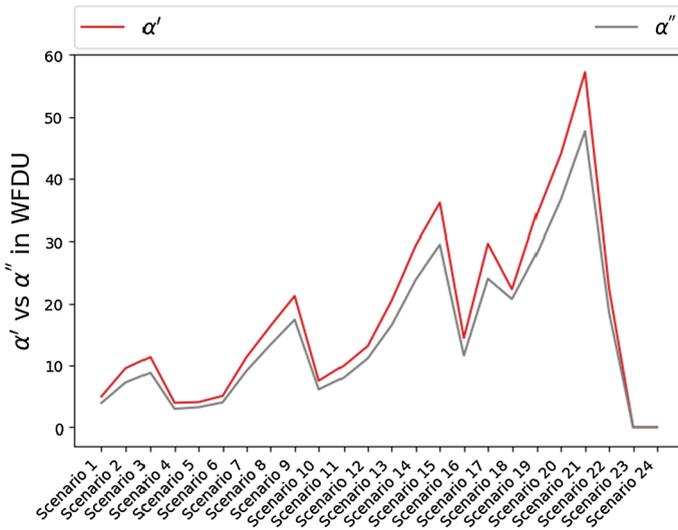


Fig. 8 Percentage difference α' vs α'' for each scenario with WFDU allocator

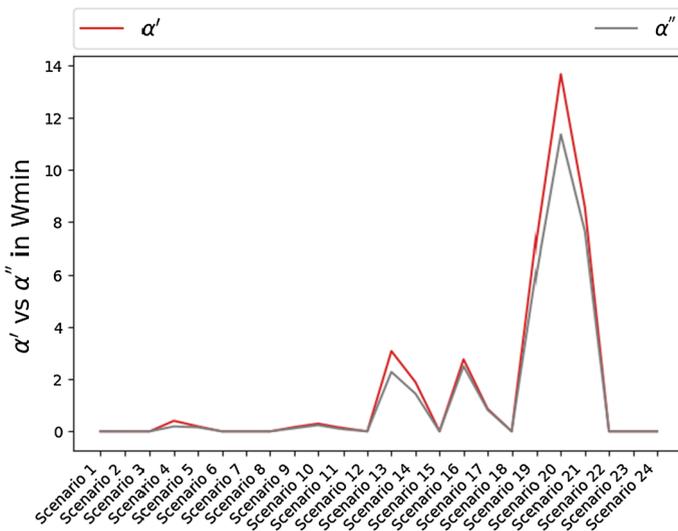


Fig. 9 Percentage difference α' vs α'' for each scenario with Wmin allocator

- WFDU** (Fig. 8): This allocator balances the load among cores. It maximizes the number of used cores so when interference appears, there is enough scope to schedule this interference. With this allocator, the schedulability ratio is high (almost 100% for all number of cores [1]) and we can observe the expected behaviour: the more cores in the system, the more α' and α'' are obtained, up to 60 and 50% in scenarios with many cores, respectively. Again, with 10 cores,

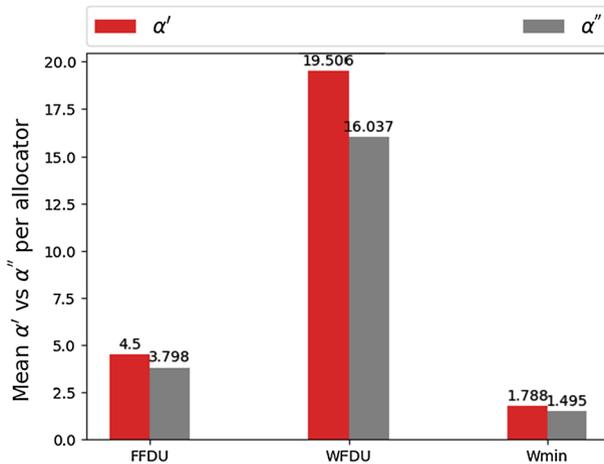


Fig. 10 Average percentage α' vs α'' for each allocator

75% of utilisation and 20-30% of interference over WCET, the schedulability ratio becomes zero so the percentage difference α' and α'' become zero.

- *Wmin* (Fig. 9): This allocator tries to group broadcasting tasks together, in order to reduce the overall provoked interference. In scenarios with few numbers of cores (and consequently few broadcasting tasks), the percentage difference is almost zero as interference is avoided in most of the cases by grouping broadcasting tasks in the same cores. As the number of cores increases, α' and α'' increase. Last scenarios (10 cores and 75% of utilisation), the feasibility is reduced and systems cannot be scheduled when interference appears. So α' and α'' are zero.

Figure 10 depicts the average values of all scenarios for each allocator. From this figure we can conclude that *Wmin* is the allocator in which α' and α'' are lower (U'_τ and U''_τ are closer to the actual utilisations) because it tries to decrease the interference. For *WFDU*, α' and α'' are the highest with respect to the studied allocators.

We can conclude that the proposed bounds $dbf'_{\tau_{M_k}}$ and $dbf''_{\tau_{M_k}}$ are valid to assure the schedulability but they are pessimistic as not in all activations the worst case interference may be produced. However, with $dbf''_{\tau_{M_k}}$ this pessimism is reduced. Moreover, depending on the allocation method, these upper bounds are more accurate, especially in those cases in which the interference is considered in the allocation phase (*Wmin* allocator).

6 Conclusions

This paper proposes two contention-aware schedulability analysis for real-time task models that consider constrained deadlines and dynamic priorities in hard real-time multicore systems. Both approaches are based on the demand bound function dbf to

determine the schedulability of the system. The first approximation, dbf' , is pessimistic in the sense that it considers that all activations of all tasks receive the maximum possible interference. The second approximation, dbf'' , reduces this pessimism as it considers specifically the maximum interference at each activation of the tasks. A schedulability test for each approach is proposed in this work.

We evaluate both approaches with different allocation techniques: *FFDU*, *WFDU* and *Wmin*. We measure the difference between both approaches proposed in this work (in terms of utilisation factors) and the actual utilisation factor measured after the scheduling phase for all the allocators. With this evaluation it is demonstrated that the dbf'' approach is much closer to the real value than the dbf' approach, as demonstrated mathematically in this work. Among all allocators, *Wmin* is the algorithm in which both approaches are more accurate, due to the fact that it considers the interference factor in the allocation process.

We plan to further investigate to use scheduling techniques that reduce interference as much as possible. Some variants of well-known scheduling algorithms such as Modified Least Laxity First [27] or Modified Maximum Urgency First [32], can achieve promising results with respect to interference.

We also plan to consider more task models such as mixed-criticality systems or partitioned systems.

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature. his work was supported under Grant PLEC2021-007609 funded by MCIN/ AEI/ 10.13039/501100011033 and by the “European Union NextGenerationEU / PRTR”.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. Aceituno JM, Guasque A, Balbastre P et al (2021) Hardware resources contention-aware scheduling of hard real-time multiprocessor systems. *J Syst Archit* 118:1–11
2. Altmeyer S, Davis RI, Indrusiak L et al (2015) A generic and compositional framework for multicore response time analysis. In: *Proceedings of the 23rd International Conference on Real Time and Networks Systems, RTNS '15*, pp 129–138
3. Andersson B, Kim H, Niz DD et al (2018) Schedulability analysis of tasks with corunner-dependent execution times. *ACM Trans Embed Comput Syst* 17(3):1–29
4. Baruah SK, Mok AK, Rosier LE (1990a) Preemptively scheduling hard-real-time sporadic tasks on one processor. In: (1990) *Proceedings 11th Real-Time Systems Symposium*, pp 182–190
5. Baruah SK, Rosier LE, Howell RR (1990) Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-Time Syst* 2(4):301–324
6. Baruah SK, Howell RR, Rosier LE (1993) Feasibility problems for recurring tasks on one processor. *Theor Comput Sci* 118:3–20

7. Choi J, Kang D, Ha S (2016) Conservative Modeling of Shared Resource Contention for Dependent Tasks in Partitioned Multi-core Systems. In: 2016 Design, Automation Test in Europe Conference Exhibition (DATE), pp 181–186
8. Coffman EG, Garey MR, Johnson DS (1996) Approximation algorithms for bin packing: a survey. PWS Publishing Co
9. Coffman EG Jr, Csirik J, Galambos G et al (2013) Bin packing approximation algorithms: survey and classification. Springer, New York
10. Dasari D, Andersson B, Nelis V, et al (2011) Response time analysis of cots-based multicores considering the contention on the shared memory bus. In: 2011 IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications, pp 1068–1075
11. Dasari D, Akesson B, Nélis V, et al (2013) Identifying the sources of unpredictability in cots-based multicore systems. In: 2013 8th IEEE international symposium on industrial embedded systems (SIES), pp 39–48
12. Davis R, Altmeyer S, Indrusiak L et al (2018) An extensible framework for multicore response time analysis. *Real-Time Syst* 54:607–661
13. Davis RI, Burns A (2009) Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. In: 2009 30th IEEE real-time systems symposium, pp 398–409
14. Davis RI, Burns A (2011) A survey of hard real-time scheduling for multiprocessor systems. *ACM Comput Surv* 43(4):1–44
15. Davis RI, Griffin D, Bate I (2021) Schedulability Analysis for Multi-core Systems Accounting for Resource Stress and Sensitivity. In: 33rd Euromicro Conference on Real-Time Systems, ECRTS 2021 Virtual Conference, LIPIcs, vol 196. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp 7:1–7:26
16. Fernandez G, Abella J, Quiñones E et al (2014) Contention in Multicore Hardware Shared Resources: Understanding of the State of the Art. In: Falk H (ed) 14th international workshop on worst-case execution time analysis, openaccess series in informatics (OASICs), vol 39. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. Dagstuhl, Germany, pp 31–42
17. Gracioli G, Alhammad A, Mancuso R et al (2015) A survey on cache management mechanisms for real-time embedded systems. *ACM Comput Surv* 48(2):1–36
18. Guo Z, Yang K, Yao F et al (2020) Inter-task cache interference aware partitioned real-time scheduling, association for computing machinery, p 218–226
19. Hassan M, Pellizzoni R (2020) Analysis of Memory-Contention in Heterogeneous COTS MPSoCs. In: Völpl M (ed) 32nd Euromicro Conference on Real-Time Systems (ECRTS 2020), pp 23:1–23:24
20. Huang WH, Chen JJ, Reineke J (2016) Mirror: symmetric timing analysis for real-time tasks on multi-core platforms with shared resources. In: Proceedings of the 53rd Annual Design Automation Conference, DAC '16
21. Karuppiah N (2016) The impact of interference due to resource contention in multicore platform for safety-critical avionics systems. *Int J Res Eng Appl Manag (IJREAM)* 02:39–48
22. Kim H, de Niz D, Andersson B et al (2014) Bounding memory interference delay in cots-based multi-core systems. In: 2014 IEEE 19th real-time and embedded technology and applications symposium (RTAS), pp 145–154
23. Lampka K, Giannopoulou G, Pellizzoni R et al (2014) A formal approach to the wcr analysis of multi-core systems with memory contention under phase-structured task sets. *Real-Time Syst* 50:736–773
24. Liu CL, Layland JW (1973) Scheduling algorithms for multiprogramming in a hard-real-time environment. *J ACM* 20(1):46–61
25. Maiza C, Rihani H, Rivas JM et al (2019) A survey of timing verification techniques for multi-core real-time systems. *ACM Comput Surv* 52(3):1–46
26. Mitra T, Teich J, Thiele L (2018) Time-critical systems design: a survey. *IEEE Design Test* 35(2):8–26
27. Oh SH, Yang SM (1998) A Modified Least-Laxity-First Scheduling Algorithm for Real-Time Tasks. In: Proceedings Fifth International Conference on Real-Time Computing Systems and Applications (Cat. No.98EX236), pp 31–36
28. Oh Y, Son SH (1995) Allocating fixed-priority periodic tasks on multiprocessor systems. *Real-Time Syst* 9(3):207–239
29. Pellizzoni R, Lipari G (2005) Feasibility analysis of real-time periodic tasks with offsets. *Real-Time Syst* 30:105–128
30. Rihani H, Moy M, Maiza C, et al (2016) Response Time Analysis of Synchronous Data Flow Programs on a Many-Core Processor. In: Proceedings of the 24th International Conference on Real-Time Networks and Systems, RTNS '16, p 67–76

31. Ripoll I, Crespo A, Mok AK (1996) Improvement in feasibility testing for real-time tasks. *Real-Time Syst* 11(1):19–39
32. Salmani V, Taghavi Zargar S, Naghibzadeh M (2005) A Modified Maximum Urgency First Scheduling Algorithm for Real-Time Tasks. *Proc Seventh World Enformatika Conference*
33. Spuri M (1996) Analysis of deadline scheduled real-time systems. Tech. rep
34. Xiao J, Altmeyer S, Pimentel A (2017) Schedulability analysis of non-preemptive real-time scheduling for multicore processors with shared caches. In: 2017 IEEE real-time systems symposium (RTSS), pp 199–208

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.