UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Programa de doctorado en Automática, Robótica e Informática Industrial

Instituto de Automática e Informática Industrial (ai2)

INVESTIGACIÓN DE NUEVAS METODOLOGÍAS PARA LA PLANIFICACIÓN DE SISTEMAS DE TIEMPO REAL MULTINÚCLEO MEDIANTE TÉCNICAS NO CONVENCIONALES

JOSÉ MARÍA ACEITUNO PEINADO

Directores:
Dra. Patricia Balbastre Betoret
Dra. Ana Guasque Ortega
Dr. José Enrique Simó Ten

Marzo, 2024

Contenido

Investigación de nuevas metodologías para la planificación de sistemas de tiempo real multinúcleo mediante técnicas no convencionales

José María Aceituno Peinado

Resumen

Los sistemas de tiempo real se caracterizan por exigir el cumplimento de unos requisitos temporales que garanticen el funcionamiento aceptable de un sistema. Especialmente, en los sistemas de tiempo real estricto estos requisitos temporales deben ser inviolables. Estos sistemas suelen aplicarse en áreas como la aviación, la seguridad ferroviaria, satélites y control de procesos, entre otros. Por tanto, el incumplimiento de un requisito temporal en un sistema de tiempo real estricto puede ocasionar un fallo catastrófico.

La planificación de sistemas de tiempo real es una área en la que se estudian y aplican diversas metodologías, heurísticas y algoritmos que intentan asignar el recurso de la CPU sin pérdidas de plazo.

El uso de sistemas de computación multinúcleo es una opción cada vez más recurrente en los sistemas de tiempo real estrictos. Esto se debe, entre otras causas, a su alto rendimiento a nivel de computación gracias a su capacidad de ejecutar varios procesos en paralelo.

Por otro lado, los sistemas multinúcleo presentan un nuevo problema, la contención que ocurre debido a la compartición de los recursos de hardware. El origen de esta contención es la interferencia que en ocasiones ocurre entre tareas asignadas en distintos núcleos que pretenden acceder al mismo recurso compartido simultáneamente, típicamente acceso a memoria compartida. Esta interferencia añadida puede suponer un incumplimiento de los requisitos temporales, y por tanto, la planificación no sería viable.

En este trabajo se proponen nuevas metodologías y estrategias de planificación no convencionales para aportar soluciones al problema de la interferencia en sistemas multinúcleo. Estas metodologías y estrategias abarcan algoritmos de planificación, algoritmos de asignación de tareas a núcleos, modelos temporales y análisis de planificabilidad.

El resultado del trabajo realizado se ha publicado en diversos artículos en revistas del área. En ellos se presentan estas nuevas propuestas que afrontan los retos de la planificación de tareas. En la mayoría de los artículos presentados la estructura es similar: se introduce el contexto en el que nos situamos, se plantea la problemática existente, se expone una propuesta para solventar o mejorar los resultados de la planificación, después se realiza una experimentación para evaluar de forma práctica la metodología propuesta, se analizan los resultados obtenidos y finalmente se exponen unas conclusiones sobre la propuesta.

Los resultados de las metodologías no convencionales propuestas en los artículos que conforman esta tesis muestran una mejora del rendimiento de las planificaciones en comparación con algoritmos clásicos del área. Especialmente la mejora se produce en términos de disminución de la interferencia producida y mejora de la tasa de planificabilidad.

Resum

Els sistemes de temps real es caracteritzen per exigir el compliment d'uns requisits temporals que garantisquen el funcionament acceptable d'un sistema. Especialment, en els sistemes de temps real estricte aquests requisits temporals han de ser inviolables. Aquests sistemes solen aplicar-se en àrees com l'aviació, la seguretat ferroviària, satèl·lits i control de processos, entre altres. Per tant, l'incompliment d'un requisit temporal en un sistema de temps real estricte pot ocasionar un error catastròfic.

La planificació de sistemes de temps real és una àrea en la qual s'estudien i apliquen diverses metodologies, heurístiques i algorismes que intenten assignar el recurs de la CPU sense pèrdues de termini.

L'ús de sistemes de computació multinucli és una opció cada vegada més recurrent en els sistemes de temps real estrictes. Això es deu, entre altres causes, al seu alt rendiment a nivell de computació gràcies a la seua capacitat d'executar diversos processos en paral·lel.

D'altra banda, els sistemes multinucli presenten un nou problema, la contenció que ocorre a causa de la compartició dels recursos de hardware. L'origen d'aquesta contenció és la interferència que a vegades ocorre entre tasques assignades en diferents nuclis que pretenen accedir al mateix recurs compartit simultàniament, típicament accés a memòria compartida. Aquesta interferència afegida pot suposar un incompliment dels requisits temporals, i per tant, la planificació no seria viable.

En aquest treball es proposen noves metodologies i estratègies de planificació no convencionals per aportar solucions al problema de la interferència en sistemes multinucli. Aquestes metodologies i estratègies comprenen algorismes de planificació, algorismes d'assignació de tasques a nuclis, models temporals i anàlisis de planificabilitat.

El resultat del treball realitzat s'ha publicat en diversos articles en revistes de l'àrea. En ells es presenten aquestes noves propostes que afronten els reptes de la planificació de tasques. En la majoria dels articles presentats l'estructura és similar: s'introdueix el context en el qual ens situem, es planteja la problemàtica existent, s'exposa una proposta per a solucionar o millorar els resultats de la planificació, després es realitza una experimentació per a avaluar de manera pràctica la metodologia proposada, s'analitzen els resultats obtinguts i finalment s'exposen unes conclusions sobre la proposta.

Els resultats de les metodologies no convencionals proposades en els articles que conformen aquesta tesi mostren una millora del rendiment de les planificacions en comparació amb algorismes clàssics de l'àrea. Especialment, la millora es produeix en termes de disminució de la interferència produïda i millora de la taxa de planificabilitat.

Abstract

Real-time systems are characterised by the demand for temporal constraints that guarantee acceptable operation and feasibility of a system. Especially, in hard real-time systems these temporal constraints must be respected. These systems are typically applied in areas such as aviation, railway safety, satellites and process control, among others. Therefore, a missed deadline in a hard-real time system can lead to a catastrophic failure.

The scheduling of real-time systems is an area where various methodologies, heuristics and algorithms are studied and applied in an attempt to allocate the CPU resources without missing any deadline.

The use of multicore computing systems is an increasingly recurrent option in hard real-time systems. This is due, among other reasons, to its high computational performance thanks to the ability to run multiple processes in parallel.

On the other hand, multicore systems present a new problem, the contention that occurs due to the sharing of hardware resources. The source of this contention is the interference that sometimes happens between tasks allocated in different cores that try to access the same shared resource simultaneously, typically shared memory access. This added interference can lead to miss a deadline, and therefore, the scheduling would not be feasible.

This paper proposes new non-conventional scheduling methodologies and strategies to provide solutions to the interference problem in multicore systems. These methodologies and strategies include scheduling algorithms, task allocation algorithms, temporal models and schedulability analysis.

The results of this work have been published in several journal articles in the field. In these articles the new proposals are presented, they face the challenges of task scheduling. In the majority of these articles the structure is similar: the context is introduced, the existing problem is identified, a proposal to solve or improve the results of the scheduling is presented, then the proposed methodology is experimented in order to evaluate it in practical terms, the results obtained are analysed and finally conclusions about the proposal are expressed.

The results of the non-conventional methodologies proposed in the articles that comprise this thesis show an improvement in the performance of the scheduling compared to classical algorithms in the area. In particular, the improvement is produced in terms of reducing the interference and a higher schedulability rate.

Agradecimientos

Agradezco a todas las personas que en mayor o menor grado me han ayudado durante la formación de mi doctorado: Ana Guasque, Patricia Balbastre, Jose Simó, Alfons Crespo, Edoardo Cittadini y Giorgio Buttazzo. Y también agradezco a mi familia por el apoyo durante estos años.

Tesis por compendio de artículos previamente publicados

Esta tesis doctoral se presenta con el formato de 'compendio de artículos'. Esta tesis incluye todos los artículos publicados en revistas indexadas o de reconocido prestigio sometidas a revisión por pares. En 3 artículos el doctorando aparece como autor y en 2 como coautor, todos los coautores son doctores. La Comisión Académica del Programa de Doctorado y el director de tesis autoriza la entrega en este formato.

A continuación, se presenta la lista de artículos que conforman esta tesis doctoral:

- *ARTÍCULO 1: Integer programming techniques for static scheduling of hard-real time systems*
  Autores: Ana Guasque Ortega, Hossein Tohidi, Patricia Balbastre Betoret, José María Aceituno Peinado, José Simó Ten y Alfons Crespo Lorente
  Revista: IEEE Access IEEE Access, vol. 8
  Identificador: (DOI) 170389-170403
  Año publicación: 2020

- *ARTÍCULO 2: Hardware resources contention-aware scheduling of hard real-time multiprocessor systems*
  Autores: José María Aceituno, Ana Guasque Ortega, Patricia Balbastre Betoret, José Simó Ten, Alfons Crespo Lorente
  Revista: Journal of Systems Architecture, Volume 118, 102223
  Identificador: (ISSN) 1383-7621
  Año publicación: 2021

- *ARTÍCULO 3: Interference-aware schedulability analysis and task allocation for multicore hard real-time systems*
  Autores: José María Aceituno Peinado, Ana Guasque Ortega, Patricia Balbastre Betoret, José Simó Ten, Alfons Crespo Lorente
  Revista: Electronics
  Identificador: (DOI) 10.1007/s11227-022-04446-y

Año publicación: 2022

- *ARTÍCULO 4: Schedulability analysis of dynamic priority real-time systems with contention*
  Autores: Ana Guasque Ortega, José María Aceituno Peinado, Patricia Balbastre Betoret, José Simó Ten, Alfons Crespo Lorente
  Revista: The Journal of Supercomputing. 78(12)
  Identificador: (DOI) 14703-14725
  Año publicación: 2022

- *ARTÍCULO 5: Optimized scheduling of periodic hard real-time multicore systems*
  Autores: José María Aceituno Peinado, Ana Guasque Ortega, Patricia Balbastre Betoret, Francisco Blanes Noguera y Luigi Pomante
  Revista: IEEE Access, vol. 11
  Identificador: (DOI) 30027-30039
  Año publicación: 2023

A lo largo de este documento de tesis doctoral, nos referiremos a los artículos como *ARTÍCULO 1*, *ARTÍCULO 2*, *ARTÍCULO 3*, *ARTÍCULO 4* Y *ARTÍCULO 5*.

Índice de Figuras

Índice de tablas

Capítulo 1

Introducción

En este capítulo se introduce el marco y se especifica el contexto en el que se sitúa esta tesis doctoral. Por tanto, en este capítulo introductorio se hace un análisis del ámbito de esta tesis. Este análisis empieza aportando una visión general del problema y acaba con los aspectos más específicos.

De ese modo, se comenzará a explicar el concepto de sistemas de tiempo real y los tipos de sistemas de tiempo real que existen. Después se entrará en el problema de la planificación de sistemas de tiempo real, donde se explicará brevemente los aspectos más importantes que se deben tener en cuenta.

A continuación se explicarán las fases de la planificación, para poner en contexto el asunto principal de esta tesis doctoral: cómo afrontar el problema de la interferencia. Por último, se explica brevemente cada uno de los artículos que conforman esta tesis y se analizará el ámbito y aporte de cada uno.

1.1. Objetivos

En esta sección se exponen los objetivos planteados en esta tesis doctoral. En primera instancia se establece un objetivo general y a continuación se desglosa en objetivos específicos.

Estos objetivos están alineados con dos proyectos de investigación de carácter nacional. Uno de ellos es el proyecto es PRECON-I4. Consiste en la búsqueda de sistemas informáticos predecibles y confiables para la industria 4.0. El otro proyecto es PRESECREL, que consiste en la búsqueda de modelos y plataformas para sistemas informáticos industriales predecibles, seguros y confiables. Tanto PRECON-I4 como PRESECREL son proyectos coordinados financiados por el Ministerio de Ciencia, Innovación y Universidades y los fondos FEDER (AEI/FEDER, UE). En ambos proyectos participa la Universidad Politécnica de Valencia, la Universidad de Cantabria y la Universidad Politécnica de Madrid. Además, en PRESECREL también participa IKERLAN S. COOP I.P. Además, parte de los resultados de esta tesis también han servido para validar la asignación de recursos temporales en sistemas críticos en el marco del proyecto METROPOLIS (PLEC2021-007609).

El objetivo general de esta tesis es la búsqueda de metodologías no convencionales para aplicarlas a la planificación de sistemas de tiempo real. La tesis se centra en los sistemas multinúcleo aunque en su fase inicial se utilizan sistemas mononúcleo para validar la aplicación de determinadas técnicas.

Para lograr el objetivo general, en esta tesis doctoral se han explorado diferentes vías que abarcan todas la fases de la planificación, incluyendo también diversos análisis de planificabilidad.

Este objetivo general se puede desarrollar sobre el siguiente conjunto de objetivos específicos:

- Objetivo 1. Explorar la viabilidad de la programación lineal en enteros (ILP) en sistemas mononúcleo.
- Objetivo 2. Proponer un modelo de tareas para sistemas multinúcleo.
- Objetivo 3. Desarrollar nuevas técnicas de planificación para sistemas multinúcleo utilizando técnicas y criterios de planificación no convencionales:
    - Objetivo 3.1. Proponer algoritmos de asignación de tareas a núcleos
    - Objetivo 3.2. Proponer técnicas para realizar análisis de planificabilidad
    - Objetivo 3.3. Proponer algoritmos de planificación

Seguidamente, se detalla cada objetivo.

Objetivo 1: Las técnicas de programación lineal permiten obtener la solución óptima de un problema que se plantea con una función objetivo, unas variables y una serie de restricciones.

Como primer paso, en esta tesis se explora la utilización de esta técnica para obtener planificaciones viables en sistemas de tiempo real mononúcleo.

Objetivo 2: Para evolucionar hacia los sistemas multinúcleo, es necesario desarrollar un modelo de tareas para sistemas multinúcleo. Este modelo de tareas debe diseñarse para que contemple el problema de la interferencia.

Objetivo 3: Una vez propuesto el modelo de tareas, se exploran nuevas técnicas y metodologías de planificación que obtengan mejor rendimiento en sistemas multinúcleo en términos de reducción de la interferencia. Para este objetivo se exploran tres enfoques distintos que corresponden a los objetivos 3.1, 3.2 y 3.3.

Objetivo 3.1: En primer lugar, los trabajos se focalizan en la fase de asignación de tareas a núcleos. Se proponen nuevos algoritmos que alojen las tareas a los núcleos. De modo que en la siguiente fase, la planificación sea viable mejorando el rendimiento respecto a la interferencia. Para ello se exploran tanto técnicas de ILP como otros métodos.

Objetivo 3.2: Una vez asignadas las tareas a núcleos de forma estática, y como paso previo a la planificación, se proponen métodos para realizar análisis de planificabilidad. Utilizando los algoritmos desarrollados, será posible determinar si un conjunto de tareas será o no será planificable en un sistema multinúcleo determinado antes de abordar la fase de planificación.

Objetivo 3.3: Finalmente se proponen nuevos algoritmos y heurísticas de planificación de tareas en sistemas multinúcleo. Se utilizan tanto técnicas de ILP como otros métodos. Además de la viabilidad de la planificación obtenida, el criterio a optimizar será la reducción de la interferencia.

1.2. Estado del arte

1.2.1. Ámbito de trabajo de la tesis doctoral

Una vez declarados los objetivos de esta tesis doctoral, se van a especificar los ámbitos en los que se enmarca esta tesis dentro de los sistemas de tiempo real. Para ello, se va a hacer un recorrido desde los conceptos más genéricos a los conceptos más específicos: en primer lugar se va a definir el concepto de sistemas de tiempo real, después, se detallará en que se diferencian los sistemas de tiempo real estrictos de los no estrictos y por último se definirán los tipos de núcleo.

En apartados posteriores se definirá el modelo de tareas básico desde el que se parte en esta tesis, la cuestión de la planificación de tareas y se entrará en el problema de la interferencia, una de las claves de esta tesis doctoral.

1.2.1.1. Ámbito: Sistemas de tiempo real

Desde un enfoque más amplio, esta tesis doctoral trabaja en el ámbito de los sistemas de tiempo real.

Según [17], se consideran sistemas de tiempo real aquellos en los que su exactitud no solo depende del resultado lógico del cálculo sino también del momento en el que los resultados se producen. También pueden entenderse como aquellos sistemas en los que la interacción con su entorno los obliga a proporcionar unas salidas o servicios que cumplan con una serie de exigencias temporales así como la realización completa de las tareas para las que fue diseñado.

Hoy en día, los sistemas de tiempo real son clave en nuestra sociedad dado el incremento de sistemas complejos que confían, parcial o totalmente, en el control computacional. Algunos ejemplos que utilizan la computación de sistemas de tiempo real son: el control de plantas nucleares, maniobras ferroviarias, sistemas de control de vuelo, sistemas de telecomunicación o robótica [3].

Los sistemas de tiempo real utilizan todo tipo de algoritmos y heurísticas para conseguir el objetivo de respetar los requisitos temporales. Estos sistemas, como cualquier sistema informático, están formados por un hardware y un software. Existen diversos elementos de software diseñados para los sistemas de tiempo real como por ejemplo los sistemas operativos de tiempo real.

1.2.1.2. Ámbito: Sistemas de tiempo real estrictos y no estrictos

Los sistemas de tiempo real estrictos pueden entenderse como aquellos que poseen unos requisitos temporales que son totalmente inviolables.

El fallo en un sistema de tiempo real estricto tiene el potencial de ser catastrófico. En estos sistemas las consecuencias de un fallo son enormemente mayores que cualquier beneficio que provea el sistema cuando carece de fallos [2]. Por tanto, los sistemas de tiempo real estrictos se diferencian de los sistemas de tiempo real no estrictos en la necesidad del cumplimiento de los requisitos temporales. Esta necesidad suele estar relacionada con la criticidad del sistema.

Ejemplos de sistemas de tiempo real estricto son el control de tráfico aéreo, el control de procesos o los sistemas *on-board* [2]. Ejemplos de sistemas de tiempo real no estricto son las redes telefónicas o redes digitales.

1.2.1.3. Ámbito: Sistemas multinúcleo y mononúcleo

Según el número de núcleos de procesamiento que tenga un sistema, podemos categorizar los sistemas de tiempo real en dos tipos: sistemas multinúcleo y sistemas mononúcleo. Los sistemas multinúcleo pueden ejecutar simultáneamente diferentes tareas.

Precisamente, la motivación de esta tesis doctoral en trabajar con sistemas multinúcleo es porque las prestaciones que aportan estos sistemas son mayores que las de los sistemas mononúcleo. Por ejemplo, en los sistemas embebidos la capacidad de procesamiento que proporcionan los sistemas multinúcleo permiten a los sistemas embebidos compartir una sola plataforma de hardware.

Actualmente, la literatura sobre la planificación de sistemas mononúcleo está bastante madura, aunque todavía existe investigación de un alcance significativo en esta cuestión [7].

El problema de la planificación de sistemas de tiempo real multinúcleo es intrínsicamente más complejo que en la planificación mononúcleo [7]. La compartición de recursos de hardware en los sistemas de tiempo real multinúcleo se presenta como un reto a resolver en la planificación de tareas. El alto rendimiento de los procesadores multinúcleo (en comparación con los mononúcleo) requiere que los planificadores sean adaptados a dicho sistema [1].

La arquitectura de los procesadores multinúcleo presenta difíciles desafíos en el análisis temporal [8]. En [6] se identifican diferentes elementos de hardware que producen indeterminismo en un sistema multinúcleo, como pueden ser los buses, las cachés o la memoria. La contención en el acceso a los recursos del procesador produce la aparición de los efectos de la interferencia [8]. Precisamente, esta tesis doctoral se ocupa de abordar el problema de la interferencia en los sistemas de tiempo real estrictos multinúcleo.

1.2.2. Modelo de tareas

En esta sección se va a detallar el modelo de tareas que se ha usado en esta tesis doctoral para el diseño y desarrollo de todos los algoritmos y heurísticas propuestas.

En la Figura 1.1 se representa un diagrama que contiene elementos básicos del modelo de tareas que se ha utilizado en esta tesis doctoral. A lo largo de la tesis se ampliará el modelo con otros elementos que ayuden a resolver los retos que se afrontan.

A continuación se defininen los elementos básicos del modelo de tareas usado [3] [7]:

- Tiempo de cómputo: Es el tiempo que necesita el procesador para ejecutar una tarea sin interrupción.

  Representamos el tiempo de cómputo como C.

- Plazo: Es el tiempo máximo antes del cual una tarea debe completar su ejecución.

  Es decir, la ejecución de una tarea no debe extenderse más allá del instante de tiempo máximo que establece el plazo. Lo representamos como D.

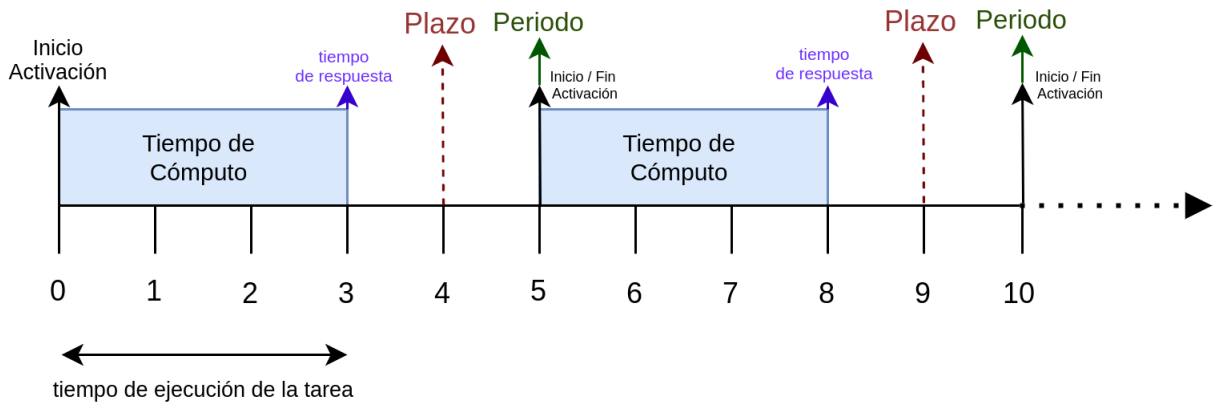


Figura 1.1: Elementos que intervienen en la planificación de una tarea

Los plazos pueden ser implícitos o restringidos. Se considera plazo implícito cuando el plazo de una tarea es igual a su periodos. Plazos restringidos son aquellos en los que todos los plazos de una tarea son iguales o menores a sus periodos.

- Periodo: En el caso de las tareas que se ejecutan cíclicamente, el periodo indica el tiempo mínimo entre dos activaciones consecutivas.

  El periodo supone el inicio de la activación de una tarea. El periodo se representa como T.

  - Hiperperiodo: es el intervalo mínimo de tiempo después del cual la planificación se repite. El hiperperiodo de un conjunto de tareas se calcula como el mínimo común múltiplo de los valores correspondientes a los periodos que cada tarea.

- Inicio de la activación: Es el instante de tiempo en el que una tarea se activa y por tanto ya está lista para ser ejecutada en el procesador.

  El inicio de la activación de una tarea no supone directamente la ejecución de la misma en el sistema. Dependiendo de la planificación, la activación de la tarea puede suponer su ejecución inmediata o bien puede suponer la inclusión de la tarea en una cola de espera.

- Tiempo de respuesta: es el tiempo de trabajo que utiliza la activación de una tarea desde que se activa hasta que completa su ejecución.

  Llamamos tiempo de respuesta en el peor de los casos, en inglés *WCRT (Worst Case Response Time)*, al tiempo de respuesta máximo que puede necesitar una tarea para su ejecución completa.

En función de la política de planificación o la prioridad de la tarea, una tarea puede sufrir una expulsión durante su ejecución, y por tanto, será ejecutada en dos bloques de tiempo o más. En cualquier caso no puede finalizar su ejecución más tarde que lo que marque su plazo.

Las tareas pueden ser clasificadas de diversas maneras. Según la regularidad de ejecución, las tareas pueden ser:

- Tareas periódicas: consisten en una secuencia infinita de actividades idénticas llamadas instancias o activaciones, que se ejecutan con una tasa de tiempo constante [3].
- Tareas aperiódicas: también consisten en una secuencia infinita de activaciones pero sus activaciones no son regulares en el tiempo [3].
- Tareas esporádicas: son tareas aperiódicas con un plazo estricto y un tiempo mínimo de llegada [14].

En esta tesis doctoral trabajamos en el ámbito de los sistemas de tiempo real críticos, por tanto, nos centraremos en trabajar con tareas periódicas y esporádicas dado que sus plazos deben respetarse siempre.

Además, según [3], dependiendo de las consecuencias de la pérdida de un plazo de una tarea, podemos distinguir entre tres tipos de tareas:

- Estrictas (*Hard*): una tarea es estricta si la pérdida de su plazo puede ocasionar consecuencias catastróficas en el sistema de control.
- Firmes (*Firm*): una tarea es firme si la pérdida de su plazo no causa daño al sistema pero su resultado no tiene valor.
- No estrictas (*Soft*): una tarea es no estricta si a pesar de la pérdida de su plazo todavía tuviese alguna utilidad para el sistema, aunque esto suponga cierta degradación en el rendimiento del sistema.

En esta tesis doctoral nos centraremos en trabajar con tareas estrictas ya que el marco de la tesis se centra en aplicaciones para el sector crítico.

1.2.3. Algoritmos de planificación de tareas

Los algoritmos de planificación son una parte esencial de los sistemas de tiempo real. Existen diversos algoritmos de planificación y elegir un algoritmo adecuado para un sistema concreto es crítico y está condicionado por el tipo de sistema [9]. Para un conjunto dado de tareas, el problema de la planificación consiste en determinar el orden de ejecución de las tareas de forma que se respeten las restricciones [13]. En los sistemas de tiempo real estrictos la planificación de tareas es el problema más importante debido a que el algoritmo de planificación es quien asegura que se cumplan las restricciones temporales [4].

Para la comparación del rendimiento de diferentes algoritmos se pueden usar distintas métricas tales como: la cantidad de cambios de contexto, el tiempo de respuesta, la predecibilidad (*predictability*) o la sobrecarga del planificador [9].

1.2.3.1. Clasificación de algoritmos de planificación

Los algoritmos de planificación pueden ser categorizados de diferentes maneras según la forma de planificar. En la Figura 1.2 se representa un diagrama jerárquico que clasifica distintos tipos de planificadores.

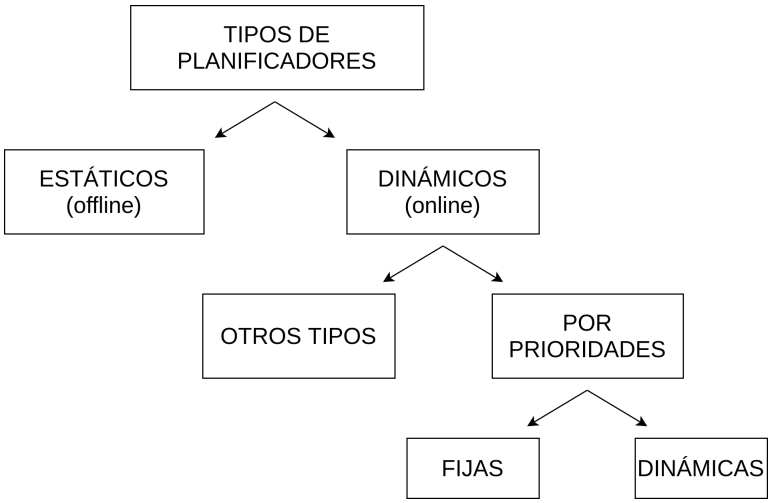


Figura 1.2: Clasificación de tipos de planificación en sistemas de tiempo real

A continuación, se van a describir los diferentes tipos de planificadores. Para ello, se va a seguir el orden jerárquico de la Figura 1.2, empezando desde arriba y continuando hacia abajo.

En función a cuándo se planifica el conjunto de tareas, podemos distinguir entre 2 tipos de planificaciones, la planificación estática y la planificación dinámica:

- La planificación estática es aquella en la que el orden de ejecución de las tareas se determina previamente a la ejecución de las mismas. También se conoce como planificación *off-line*.
- La planificación dinámica es aquella en la que el orden de ejecución de las tareas se determina durante la propia ejecución. También se conoce como planificación *on-line*.

Dentro de la familia de los planificadores dinámicos, existen distintas estrategias para planificar. Una de ellas es la planificación por prioridades. Esta consiste en dar distintas prioridades a las tareas de modo que la tarea con mayor prioridad se ejecutará antes que las otras.

En función de la manera de otorgar las prioridades, los planificadores dinámicos pueden ser clasificados en:

- Prioridades de tareas fijas: a cada tarea se le otorga una prioridad de manera permanente durante toda la ejecución.
- Prioridades de tareas dinámicas: las prioridades de las tareas pueden ir cambiando durante la ejecución.

Un ejemplo de planificador con prioridades fijas es Rate Monotonic (RM). Este algoritmo consiste en asignar las prioridades en función del tamaño de los periodos de las tareas. Cuanto menor sea el periodo, mayor será la prioridad de la tarea [12].

Otro ejemplo de planificador con prioridades fijas es Deadline Monotonic (DM). Este algoritmo de planificación otorga mayor prioridad a la tarea que posea el plazo más pequeño.

Un ejemplo de planificador con prioridades dinámicas es Earliest Deadline First (EDF). Este algoritmo de planificación otorga mayor prioridad a la tarea que posea el plazo absoluto más temprano. EDF es un algoritmo de prioridades dinámicas dado que en cada activación de una tarea el plazo absoluto es distinto.

En esta tesis doctoral se experimenta con distintas categorías de algoritmos por prioridades. Particularmente se utilizan los algoritmos EDF y DM como referencia comparativa de los planificadores y técnicas propuestas en los artículos.

Además de esta clasificación, podemos categorizar los algoritmos de planificación en función de la expulsividad [7]:

- Expulsivos: las tareas pueden ser expulsadas por otra tarea de mayor prioridad en todo momento.
- No expulsivos: una vez que una tarea empieza a ejecutarse, no podrá ser expulsada y se ejecutará hasta el final.
- Cooperativos: las tareas pueden ser expulsadas solo en ciertos momentos de la planificación durante su ejecución.

En esta tesis doctoral se trabaja con planificadores expulsivos. Para los objetivos planteados en esta tesis doctoral el dinamismo de los planificadores expulsivos ofrece más posibilidades que los no expulsivos.

Y por último, de acuerdo también con [7], en el ámbito de los sistemas multinúcleo, podemos categorizar los algoritmos de planificación en función a la migración entre núcleos:

- Sin migración: Una tarea solo puede realizar su ejecución en un mismo procesador durante todo el hiperperiodo.
- Con migración a nivel de tarea: las activaciones de una tarea pueden ejecutarse en distintos procesadores, sin embargo cada activación solo puede ejecutarse en un procesador.
- Con migración a nivel de activación: la misma activación de una tarea puede ejecutarse en diferentes procesadores. No obstante, la ejecución simultánea de una misma activación en distintos procesadores no está permitida.

Existen sistemas híbridos respecto a la migración, no obstante, en esta tesis doctoral trabajamos en el ámbito de los sistemas de tiempo real críticos. De tal modo, se ha optado por trabajar en el ámbito de los planificadores que no permiten la migración de tareas dado que son los más adecuados para ello.

Para el cumplimiento de los requisitos temporales en un sistema de tiempo real estricto, una buena técnica, metodología o algoritmo de planificación es crucial tanto para obtener la planificabilidad del conjunto de tareas como también para obtener el rendimiento de una planificación.

En los artículos de esta tesis doctoral se proponen nuevos algoritmos de planificación para sistemas de tiempo real estrictos. Los algoritmos propuestos son evaluados y se compara su rendimiento con planificadores clásicos como EDF o DM.

1.2.4. Fases de la planificación de tareas en sistemas multinúcleo

El proceso de planificación en sistemas multinúcleo es más complejo que en los sistemas mononúcleo. Cuando se trabaja en la planificación de sistemas multinúcleo sin migración de tareas, el problema de la planificación se subdivide en dos problemas: la asignación y la planificación [7].

1. El problema de la asignación: en qué procesador debe ejecutarse una tarea.
2. El problema de la planificación: cuándo, y en qué orden respecto a las activaciones de otras tareas, debe ejecutarse una activación.

Dado que en esta tesis doctoral trabajamos en el ámbito de la planificación sin migración de tareas, nos centramos tanto en la exploración de nuevas metodologías para la fase de asignación como para la fase de planificación. Esto se hace con el objetivo de optimizar ciertos parámetros temporales.

En la figura 1.3 podemos ver una representación gráfica de las dos fases mencionadas con un ejemplo genérico de N tareas y M núcleos.

1.2.4.1. Fase de asignación

En la planificación de sistemas multinúcleo la fase de asignación es ineludible. En esta fase se realiza el alojamiento o asignación de un conjunto de tareas entre los núcleos del sistema. La asignación de tareas a núcleos se realiza en función de la carga de utilización. La utilización de cada tarea se calcula como su tiempo de cómputo entre su periodo. Cuánto mayor sea la utilización de una tarea, más carga generará en el núcleo.

Existen diversos algoritmos para realizar esta función. En esta tesis proponemos nuevas metodologías para este proceso. El problema de asignación de tareas es análogo al problema de *bin-packing* y es conocido como un problema NP-complejo (*Non-deterministic Polynomial-time hardness*) en el sentido fuerte [10].

Los algoritmos de bin-packing más típicos en la literatura son los siguientes [5] [15]:

- First Fit (FF):
  Este algoritmo va asignando secuencialmente cada tarea en el primer núcleo que tenga

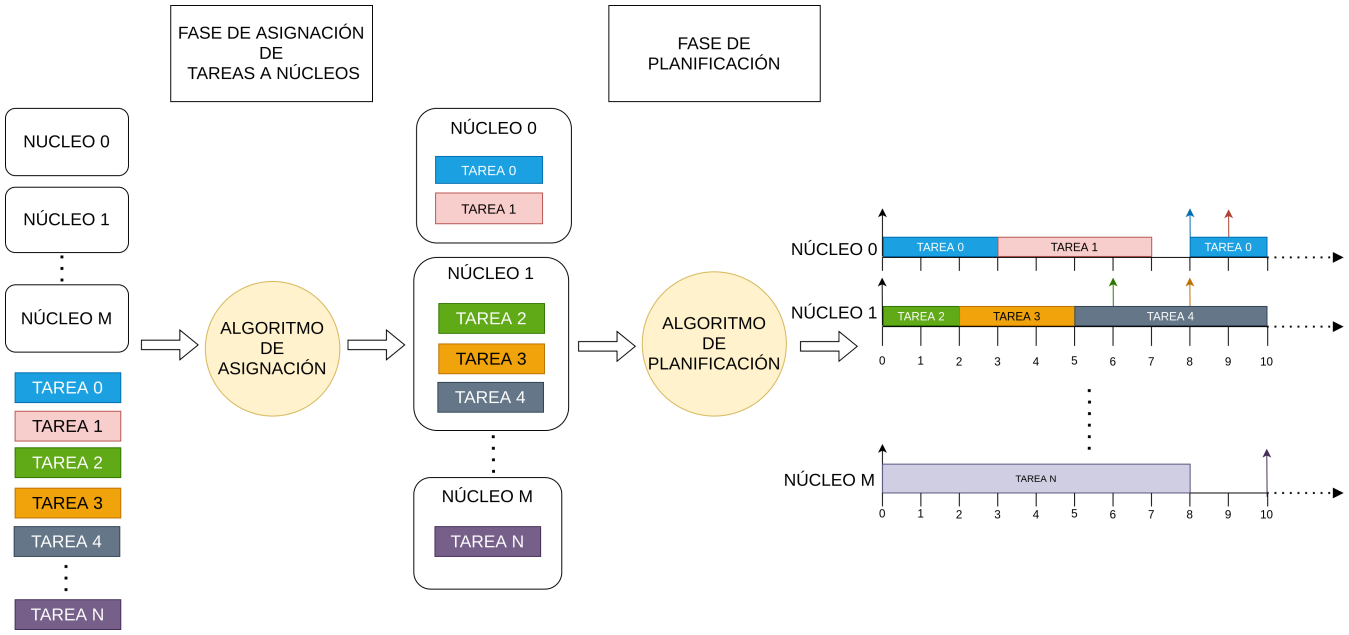


Figura 1.3: Esquema de las dos fases de la planificación de tareas en sistemas multinúcleo

suficiente tiempo de cómputo disponible. De ese modo, si una tarea no cupiese en el primer núcleo, el algoritmo intentaría asignar esa tarea al siguiente núcleo, y así sucesivamente.

FF tiende a minimizar el número de núcleos usados. Esto significa que puede provocar que algunos núcleos queden muy ocupados y otros con poca carga. Esta característica puede influir decisivamente en la posterior fase de planificación.

- Best Fit (BF): Este algoritmo asigna secuencialmente cada tarea al núcleo que esté más ocupado. En caso que una tarea no quepa en el núcleo más ocupado, se intenta asignar al siguiente núcleo con menos capacidad de cómputo, y así sucesivamente.

  Aunque BF usa una estrategia distinta a WF, ambos tienden a minimizar el número de núcleos usados.

- Worst Fit (WF): Este algoritmo asigna secuencialmente cada tarea al núcleo que tenga menor carga de utilización, es decir, al núcleo con más capacidad de cómputo libre. Puede considerarse como la política opuesta a BF.

  WF tiende a igualar la carga de utilización entre los núcleos. Es un algoritmo muy usado en los experimentos realizados en esta tesis para comparar su rendimiento con el de los algoritmos que se proponen.

Para hacer la asignación de tareas a núcleos, los algoritmos van asignando cada tarea secuencialmente, una tras otra. Por ello, es importante tener en cuenta el orden de las tareas. Normalmente, suele utilizarse el criterio de la utilización para ordenar las tareas y el tipo de orden usado es el decreciente. De ese modo, la tarea con mayor utilización será la primera en

ser asignada a un núcleo y la tarea con menor utilización será la última.

Por tanto, los algoritmos mencionados FF, BF y WF, en la fase de asignación de tareas incorporan el término *decreasing utilisation* (DU) en sus siglas, de ese modo, en el ámbito de la asignación de tareas estos algoritmos pasan a llamarse: FFDU (*First Fit Decreasing Utilisation*), BFDU (*Best Fit Decreasing Utilisation*) y WFDU (*Worst Fit Decreasing Utilisation*).

Además de estas técnicas clásicas, en esta tesis se proponen otras metodologías no convencionales para realizar la fase de asignación de tareas. Estas metodologías no convencionales se basan en asignar tareas a núcleos en función de parámetros como la interferencia o la diferencia de carga de las tareas (también llamada discrepancia). Dichas nuevas metodologías se expondrán más detalladamente en los artículos.

1.2.4.2. Fase de Planificación

La fase de planificación consiste en determinar en qué instante de tiempo se va a ejecutar cada tarea. En el caso de los sistemas mononúcleo esta es la única fase. En los sistemas multinúcleo es la segunda fase y debe realizarse dicho proceso para cada núcleo. Muchos de los algoritmos de planificación se consideran como problemas de tipo NP-completo (*Nondeterministic Polynomial-time complete*) [16]. En la literatura podemos encontrar diversos algoritmos de planificación clásicos. Algunos de ellos se han utilizado en la fase experimental de los artículos como EDF, RM, DM. Estos algoritmos son detallados en secciones posteriores.

En esta tesis doctoral hemos desarrollado nuevas metodologías de planificación que se explican en las próximas secciones. Estas metodologías abarcan sistemas mononúcleo y multinúcleo.

La planificación de tareas en sistemas multinúcleo, pese a hacer una asignación previa de tareas a núcleos, es más compleja que la planificación en sistemas mononúcleo porque presenta la problemática específica de la contención debida a la interferencia en el acceso a recursos. Esta cuestión es uno de los aspectos que trata esta tesis y está detallada en la siguiente sección.

1.2.4.3. Problema de la Interferencia en sistemas multinúcleo

En los sistemas de tiempo real multinúcleo existen diversas fuentes de indeterminismo que pueden producir un efecto de contención de los recursos en el sistema. Se puede entender la contención como la petición simultánea de un mismo recurso compartido por parte de varios procesos en diferentes núcleos. En la literatura se han implementado diferentes técnicas para lidiar con estas fuentes de indeterminismo.

Por ejemplo, en dispositivos de hardware multinúcleo de alto rendimiento como COTS (*Commercial Off-The-Shelf*), por razones de coste, energía y consumo se comparten componentes físicos como el bus, la memoria DRAM o las cachés, entre otros elementos. Sin embargo, los recursos compartidos solo pueden procesar una petición en cada instante [11].

La contención se produce a causa de la interferencia entre distintas tareas ejecutadas en distintos núcleos que simultáneamente solicitan el mismo recurso. Esta interferencia puede provocar el incumplimiento de algunos requisitos temporales, e incluso, podría impedir la planificabilidad de un conjunto de tareas en un sistema.

Muchos trabajos de investigación han profundizado en este aspecto. En ese sentido, han incorporado modelos, análisis y diversos algoritmos para tratar el problema de la interferencia. Para reducir la interferencia existen diversas técnicas y enfoques en la literatura.

En los artículos que comprenden esta tesis doctoral se han desarrollado diferentes enfoques para tratar el problema de la interferencia que podríamos agrupar en 4 categorías:

- Modelo temporal que contempla la interferencia que producen las tareas.
- Algoritmos de planificación para el modelo propuesto y análisis de planificabilidad.
- Algoritmos de asignación de tareas a núcleos que reducen la interferencia.
- Algoritmos de planificación que reducen la interferencia en la planificación final.

En secciones posteriores se desarrollará cada una de las estrategias aportadas en esta tesis para afrontar el problema de la contención de los sistemas multinúcleo producida por la interferencia entre las tareas.

1.3. Contribuciones

En las primeras páginas de este documento se ha presentado los artículos que conforman esta tesis. En esta sección se va a detallar el aporte de los artículos de esta tesis a la literatura de planificación de sistemas de tiempo real. Para ello, se presenta un pequeño resumen de cada artículo destacando sus aportaciones, un diagrama general con los aportes de cada artículo y se muestra una tabla de cobertura que muestra el ámbito de cada uno de los artículos.

A continuación vamos a realizar el resumen de cada artículo destacando sus aportes:

1.3.1. ARTÍCULO 1: Integer programming techniques for static scheduling of hard-real time systems

Este artículo se sitúa en el ámbito de los sistemas de tiempo real estrictos sobre plataformas mononúcleo. En el artículo se propone un modelo de tareas que trabaja con tareas periódicas en sistemas particionados. Además, se proponen tres algoritmos de planificación de tareas basados en técnicas de programación lineal en enteros. Las técnicas propuestas para planificar tratan de optimizar criterios como el número de cambios de contexto y el tiempo de respuesta.

Se evalúan las tres técnicas de planificación propuestas. Para ello, se miden y tienen en cuenta varios criterios como los cambios de contexto, los tiempos de respuesta obtenidos, el tiempo empleado para el cómputo de cada solución, la proximidad de cada solución respecto a la óptima y la variación que se produce en los tiempos de respuesta.

1.3.2. ARTÍCULO 2: Hardware resources contention-aware scheduling of hard real-time multiprocessor systems

Este artículo, a diferencia del anterior, se sitúa en el ámbito de los sistemas de tiempo real estrictos sobre plataformas multinúcleo. Se propone un nuevo modelo de tareas que tiene en cuenta el factor de la interferencia de manera genérica. En este artículo, al igual que en los siguientes, se trabaja sin contemplar la migración de tareas entre núcleos. También se propone un algoritmo de planificación que contempla y computa cuánta interferencia se produce en sus planificaciones. Por otro lado, este artículo propone tres algoritmos para realizar la fase de asignación de tareas a núcleos. El primer algoritmo de asignación minimiza la discrepancia entre la carga de núcleos, por tanto, minimiza la diferencia de utilización total que hay entre los núcleos tras realizar la asignación de tareas (UDmin). De ese modo, la carga entre los diferentes núcleos queda lo más igualada posible. El segundo algoritmo, al contrario que el primero, maximiza esta discrepancia (UDmax). Y por último, el tercer algoritmo de asignación (Wmin) minimiza la interferencia.

Los métodos propuestos son evaluados en términos de incremento de utilización y planificabilidad. Los tres algoritmos de asignación son comparados con otros algoritmos como FFDU, BFDU o WFDU. Los resultados son diversos según el criterio que se escoja pero Wmin muestra ser el único algoritmo de asignación que obtiene buenos resultados en los dos criterios.

1.3.3. ARTÍCULO 3: Interference-aware schedulability analysis and task allocation for multicore hard real-time systems

Este artículo se sitúa también en el ámbito de los sistemas de tiempo real estrictos sobre plataformas multinúcleo. Se usa el mismo modelo de tareas que se propone en el ARTÍCULO 2. Se propone un análisis de planificabilidad teniendo en cuenta la interferencia. En este análisis se estima el máximo de interferencia que cada tarea podría recibir de las otras durante la planificación.

Además se presenta otro algoritmo de asignación de tareas a núcleos (llamado Imin) que minimiza la interferencia.

Para realizar una evaluación del algoritmo de asignación Imin se compara con otros algoritmos de asignación clásicos y el algoritmo Wmin (propuesto en el ARTÍCULO 2). Los resultados muestran que el algoritmo de este artículo (Imin) obtiene el mejor rendimiento en términos de planificabilidad e incremento de la utilización.

1.3.4. ARTÍCULO 4: Schedulability analysis of dynamic priority real-time systems with contention

Este artículo se sitúa también en el ámbito de los sistemas de tiempo real estrictos sobre plataformas multinúcleo. El modelo de tareas utilizado en este artículo es una ampliación del propuesto en el ARTÍCULO 2. Se considera que los plazos sean iguales o menores a los periodos.

Este artículo propone dos métodos para estimar el límite en la demanda de uso del núcleo para un conjunto de tareas. El primer método se sitúa en el peor de los casos y por tanto es muy pesimista. El segundo tiene un cálculo más relajado.

La validez de ambos métodos se demuestra matemáticamente y ambos se evalúan en 24 escenarios diferentes.

En la evaluación, para la fase de asignación se utiliza los algoritmos de asignación típicos FFDU y WFDU, y también el algoritmo de asignación propuesto en el ARTÍCULO 2. Para la fase de planificación se utiliza el algoritmo propuesto en el ARTÍCULO 2.

Los resultados muestran que en las planificaciones generadas siempre se respeta el límite que establecen los dos métodos. Por tanto, la experimentación confirma que la teoría que exponen los dos métodos es válida. También se muestra que el segundo método se acerca más a la demanda real de tiempo de ejecución en el núcleo que finalmente tienen cada tarea.

1.3.5. ARTÍCULO 5: Optimized scheduling of periodic hard real-time multicore systems

Este artículo se sitúa también en el ámbito de los sistemas de tiempo real estrictos sobre plataformas multinúcleo. Se utiliza como base el modelo de tareas propuesto en el ARTÍCULO 2. Se tiene en consideración que los plazos son menores o iguales a los periodos.

En este artículo se propone un método de planificación de tareas compuesto internamente por dos técnicas distintas. Por un lado, una técnica basada en programación lineal en enteros. Y por otro lado, otra técnica que complementa a la anterior, basada en un método heurístico

(llamado planificador combinado) que genera una planificación en función de los intervalos de trabajo generados durante el hiperperiodo.

Para evaluar el método propuesto se compara con EDF. Para la parte de asignación de tareas a núcleos se utilizan los algoritmos WFDU y Wmin. Los resultados de la evaluación demuestran que el método propuesto obtiene planificaciones con menor cantidad de interferencia que EDF, e incluso, también muestra un ligero mejor porcentaje de planificabilidad que EDF.

1.3.6. Aportaciones de los artículos

En los cinco artículos que componen esta tesis se realizan diferentes aportaciones a la planificación de tareas en sistemas de tiempo real. La mayoría de los aportes intentan resolver el problema de la interferencia desde distintas perspectivas y estrategias, proponiendo modelos de tareas, análisis de planificabilidad, algoritmos para la fase de asignación o algoritmos para la fase de planificación.

En el diagrama de la Figura 1.4 pueden observarse todos los algoritmos, modelos y análisis aportados en esta tesis clasificados por tipos.

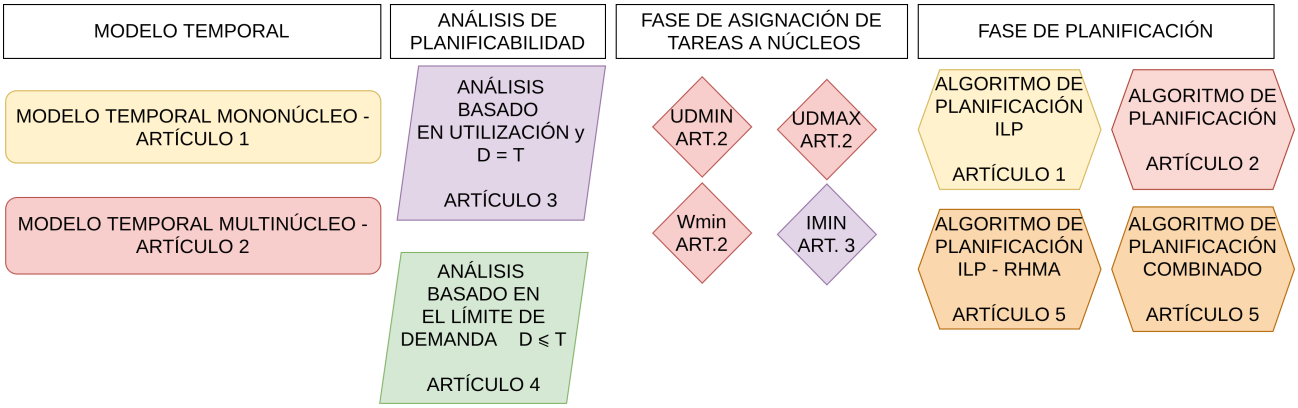


Figura 1.4: Diagrama que clasifica los aportes de los artículos por tipos

Los artículos se han agrupado por colores, de manera que todos los elementos de un mismo artículo, aunque sean de distinta tipología, comparten el mismo color.

Por tanto, los modelos propuestos en esta tesis son dos y están representados en rectángulos en la parte izquierda del diagrama. El modelo de tareas propuesto en el ARTÍCULO 1 es un modelo para sistemas mononúcleo, sin embargo el modelo de tareas propuesto en el ARTÍCULO 2 es un modelo para sistemas multinúcleo que contempla la interferencia.

En dos artículos se realizan aportes relacionados con el análisis de la interferencia en la planificabilidad de un conjunto de tareas, en el ARTÍCULO 3 y en el ARTÍCULO 4. Ambos análisis se basan en el modelo de tareas propuesto en el ARTÍCULO 2.

Respecto a la fase de asignación de tareas a núcleos, en total se han realizado hasta cuatro algoritmos de asignación, tres en el ARTÍCULO 2 y otro más en el ARTÍCULO 3.

Por último, respecto a la planificación de tareas a núcleos, se han realizado aportes hasta en tres artículos: en el ARTÍCULO 1, en el ARTÍCULO 2 y en el ARTÍCULO 5. En el ARTÍCULO 1 se proponen tres técnicas distintas para la planificación de sistemas mononúcleo. En el

ARTÍCULO 2 se propone un algoritmo de planificación que computa la interferencia producida en sistemas multinúcleo. En el ARTÍCULO 5 también se propone un método de planificación que reduce la interferencia en la planificación de tareas de sistemas multinúcleo.

1.3.7. Relación entre los artículos

En la Figura 1.5 se presenta un diagrama que muestra la relación existente entre los artículos de esta tesis. A continuación se va a detallar en qué consisten dichas relaciones y dependencias.

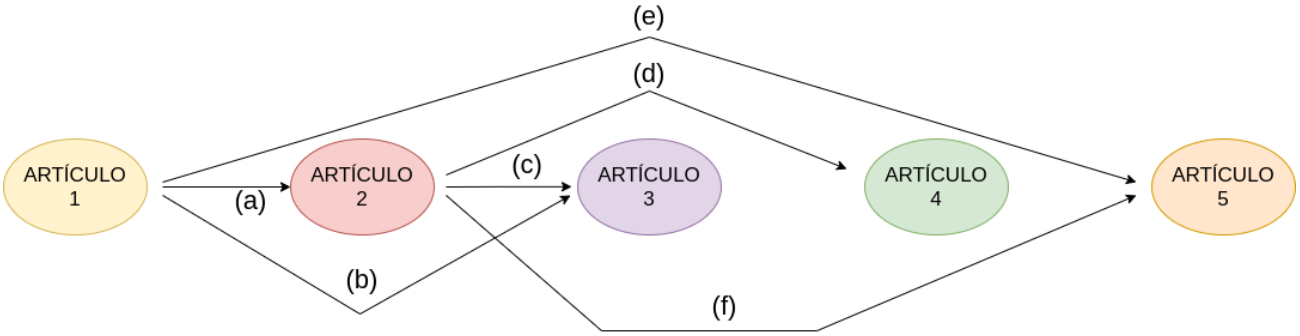


Figura 1.5: Diagrama que muestra la relación entre los distintos artículos

(a) En el artículo 1 se propone el uso de técnicas tipo *Integer Lineal Programming* (ILP) para la planificación de sistemas mononúcleo particionados. Tras obtener resultados satisfactorios, se pretende exportar esta técnica a sistemas multinúcleo. En el artículo 2 se establece un modelo de tareas para sistemas multinúcleo que servirá como base para una posterior aplicación de nuevas metodologías, entre ellas, usando técnicas de ILP.

(b) Tras la aplicación exitosa de técnicas ILP en el artículo 1, en el artículo 3 se propone un método de asignación de tareas a núcleos que también se basa en técnicas ILP.

(c) El artículo 3 se basa en el modelo de tareas propuesto en el artículo 2 para presentar un análisis de planificabilidad que estima el límite de interferencia generado entre las tareas. Además, en la parte experimental del artículo 3 se utiliza el algoritmo de asignación propuesto en el artículo 2 para evaluar el rendimiento del algoritmo presentado en el artículo 3.

(d) En el artículo 4 se extiende el modelo de tareas propuesto en el artículo 2. En este modelo extendido se considera que los plazos de las tareas pueden ser distintos a los periodos (*constrained deadlines*). A partir de este modelo extendido, se proponen dos análisis de planificabilidad. Además, en la parte experimental del artículo 4 se utiliza, junto a otros, el algoritmo de asignación propuesto en el artículo 2. De ese modo se evalúan los análisis de planificación propuestos.

(e) Tras la aplicación exitosa en el artículo 1 de técnicas ILP para utilizarlas en la planificación, en el artículo 5 se propone una metodología de planificación que también incluye técnicas ILP.

(f) El método de planificación que se propone en el artículo 5 se basa en el modelo de tareas propuesto en el artículo 2. Además, para la parte experimental del artículo 5 se utiliza el algoritmo de asignación propuesto en el artículo 2.

1.3.8. Tabla de cobertura

En la Tabla 1.1 se muestra una tabla de cobertura para situar ámbitos y propuestas de cada artículo. Aquellos ámbitos que son comunes a todos los artículos no son expuestos en la tabla. Por ejemplo, todos los artículos se sitúan en el ámbito de la planificación estática, por tanto esta característica no se indica en la tabla.

Artículo	Mono/Multi núcleo	Tipos de planificación	¿Plazos implícitos?	Propuesta
1	Mononúcleo	ILP	NO	Modelo temporal y métodos de planificación
2	Multinúcleo	Prioridades fijas o dinámicas	NO	Modelo temporal y métodos de asignación
3	Multinúcleo	Prioridades dinámicas	SÍ	Análisis de planificabilidad y métodos de asignación
4	Multinúcleo	Prioridades dinámicas	NO	Análisis de planificabilidad
5	Multinúcleo	ILP	NO	Método de planificación

Tabla 1.1: Tabla de cobertura de los artículos presentados en esta tesis

Como se observa en la Tabla 1.1, la mayoría de artículos se centran en el ámbito de sistemas multinúcleo. Esto responde a uno de los objetivos de esta tesis doctoral, la propuestas de nuevos enfoques y metodologías que afrontan el problema de la interferencia en sistemas multinúcleo.

1.3.9. Tabla de objetivos

En la sección 1.1 se presentaron los objetivos de esta tesis doctoral. A continuación se presenta la Tabla 1.2 que muestra en que artículo se ha alcanzado cada uno de los objetivos y subobjetivos planteados en esta tesis:

	Artículo 1	Artículo 2	Artículo 3	Artículo 4	Artículo 5
Objetivo 1	X				
Objetivo 2		X			
Objetivo 3.1		X	X		
Objetivo 3.2			X	X	
Objetivo 3.3		X			X

Tabla 1.2: Tabla que sitúa cada uno de los objetivos planteados en esta tesis con los artículos donde se abordan.

1.4. Siguientes secciones

A continuación se incluye una sección de referencias adicionales a las incluidas en los artículos y que se han utilizado en este resumen o parte introductoria.

Después de la sección de referencias adicionales, se van presentar los cinco artículos publicados que conforman esta tesis doctoral. A modo de guía, se puede usar la referencia tanto de la Tabla 1.1 como del diagrama de la Figura 1.4 para situar el contexto de cada uno de los artículos así como para situar el aporte realizado en cada uno de ellos.

Finalmente se expondrán las conclusiones de este trabajo de tesis doctoral. En esta sección se analizan las contribuciones de los diferentes artículos, la calidad de las diferentes publicaciones, un esbozo de la posible continuación de este trabajo en un futuro y una recopilación de las referencias utilizadas en los artículos.

Bibliografía

[1] Boyd-Wickizer, S., Morris, R., and Kaashoek, M. F. Reinventing scheduling for multicore systems. In Proceedings of the 12th Conference on Hot Topics in Operating Systems (2009), HotOS'09, USENIX Association, p. 21.

[2] Burns, A. Scheduling hard real-time systems: A review. Software Engineering Journal (1991), 116–128.

[3] Buttazzo, G. C. Hard real-time computing systems : predictable scheduling algorithms and applications, 3rd ed. ed. Springer, New York, 2011.

[4] Cheng, S., Stankovic, J. A., and Ramamritham, K. Scheduling algorithms for hard real-time systems–a brief survey.

[5] Coffman, E. G., Garey, M. R., and Johnson, D. S. Approximation Algorithms for Bin Packing: A Survey. PWS Publishing Co., USA, 1996, p. 46–93.

[6] Dasari, D., Akesson, B., Nélis, V., Awan, M. A., and Petters, S. M. Identifying the sources of unpredictability in cots-based multicore systems. In 2013 8th IEEE International Symposium on Industrial Embedded Systems (SIES) (2013), pp. 39–48.

[7] Davis, R. I., and Burns, A. A survey of hard real-time scheduling for multiprocessor systems. ACM Comput. Surv. 43, 4 (Oct. 2011).

[8] Fernandez, G., Abella, J., Quiñones, E., Rochange, C., Vardanega, T., and Cazorla, F. Contention in multicore hardware shared resources: Understanding of the state of the art. In WCET (2014).

[9] Hantom, W., Aldweesh, A., Alzaher, R., and Rahman, A. A survey on scheduling algorithms in real-time systems. 686–690.

[10] Johnson, D. Near-Optimal Bin Packing Algorithms. PhD thesis, Massachusetts Institute of Technology, Dept. of Math., 1973.

[11] Karuppiah, N. The impact of interference due to resource contention in multicore platform for safety-critical avionics systems. International Journal for Research in Engineering Application Management (IJREAM) 02 (01 2016), 39–48.

[12] Liu, C. L., and Layland, J. W. Scheduling algorithms for multiprogramming in a hard-real-time environment. J. ACM 20, 1 (jan 1973), 46–61.

[13] Mohammadi, A., and Akl, S. Scheduling algorithms for real-time systems.

[14] Mok, A. Fundamental design problems of distributed systems for the hard-real-time environment.

[15] Oh, Y., and Son, S. H. Allocating fixed-priority periodic tasks on multiprocessor systems. Real-Time Syst. 9, 3 (Nov. 1995), 207–239.

[16] Ramamritham, K., Stankovic, J. A., and Shiah, P. F. Efficient scheduling algorithms for real-time multiprocessor systems. IEEE Trans. Parallel Distrib. Syst. 1, 2 (apr 1990), 184–194.

[17] Stankovic, J. A., and Ramamritham, K. Hard Real-Time Systems. IEEE Computer Society Press, Washington, DC, USA, 1988.

Capítulo 2

Artículo: Integer Programming Techniques for Static Scheduling of Hard Real-Time Systems

2.1. Abstract

Hard real-time systems focus on obtaining a feasible schedule while satisfying different temporal requirements. In safety-critical applications, this schedule is generated offline. This paper explores different integer linear programming techniques (ILP) to schedule uniprocessor hard real-time systems. The goal is to efficiently obtain a static schedule for periodic tasks and partitioned systems where temporal and spatial isolation is crucial. The advantage of the proposed ILP techniques is the possibility of choosing the optimization criteria so that deadlines are met and better performance quality is achieved. The drawback is the time spent finding an optimal solution. We propose an ILP method that reduces by 70 % the time needed to obtain an optimal solution compared to basic approaches. This method is called the rolling task MILP approach and the optimization problem is addressed task by task. Experimental results show that our approach also achieves better results than heuristics when trying to reduce temporal parameters such as response times, context switches, and jitter. This makes our solution suitable for control systems and other applications.

2.2. Introduction

Modern real-time embedded systems comprise many applications, often of different criticalities, executing on the same computing platform. If a hard real-time task misses any temporal constraint in high criticality applications, it may suppose catastrophic results. As defined in [2.6], criticality is a designation of the level of assurance against failure needed for a system component. It is necessary to certify the system's safety and security to avoid any catastrophic or hazardous failure. Independent organizations certify the system if the collected evidence proves that it is behaving as expected. Conformance to a safety standard is of great help, or even

required, for certification. There are several such standards for different domains, such as electronic systems (IEC 61508), airborne civil avionics (DO-178B), nuclear power plants (IEC 880), medical systems (IEC 601-4), European railways (EN 50128]), European space (ECSS), etc. Temporal and spatial partitioning (TSP) is often a requirement to isolate faults and avoid recertifying the whole system when the requirements change. Partitioned systems are a way to ensure TSP in which applications with different criticalities are grouped into partitions that run in isolation from each other. This isolation prevents a failure in a low criticality application to propagate in more critical applications.

Both static and dynamic allocations might be used to obtain a feasible schedule of a hard real-time task set. Static allocation of resources is a major requirement for fulfilling certification requirements, for example, in standards such as ARINC-653 [2.1]. Static scheduling requires a priori knowledge of the characteristics of the tasks. The static scheduler generates an offline plan, a sequence of task executions, for a group of available tasks. This scheme is known as cyclic executive [2.3], [2.27]. This static plan is saved in a table and indicates when each task should be executed. The verification of the schedulability using this strategy must be carried out during the construction of the plan. Due to this determinism in the execution, static scheduling is widely used in real-time high integrity systems. Static scheduling presents advantages such as low cost at run time, and disadvantages, such as a lack of flexibility, and good knowledge of the task set is required.

Preemptive and non-preemptive periodic tasks scheduling is NP-hard in the strong sense [2.22]. Commonly used heuristics are mainly focused on obtaining a feasible schedule. However, optimizing certain temporal parameters, such as the best case response time, might be of interest, especially when timing constraints impose lower bounds on response times to events. For example, upon a collision, an airbag has to be inflated neither too early nor too late [2.33]. Jitter is the other critical parameter that measures the variation in response time. Especially in control systems, any delays can cause the system instability since these delays are not taken into account in the control law [2.9].

Exact optimization methods, such as integer linear programming (ILP) have been mostly used for scheduling tasks on multiprocessor systems rather than uniprocessor systems, as there are heuristics that can optimally schedule tasks on an uniprocessor system in polynomial time. By formulating the problem as an ILP, we can theoretically determine the optimal schedule in any system considering different objectives and constraints. The possibility of customizing an objective function that satisfies the needs of the designers is an interesting proposition that can be applied to different fields with different goals. However, solution times might not always remain tractable. Recent progress in optimization solvers diminishes this issue as they can effectively solve practical size instances.

In this work, we explore ILP for generating static schedules on uniprocessor applications for periodic and partitioned systems. Our goal is to find optimal static schedules with a customized objective rather than focusing on obtaining a feasible schedule. We formulate the problem for the periodic task model, both preemptive and non-preemptive, and propose an alternative ILP formulation that significantly improves the simple model performance. This new approach can be used to schedule a partitioned system to obtain an optimal solution in a reasonable time. The objective function can be changed to optimize any combination of temporal parameters.

Response times (worst and best), jitter and context switches can be minimized in order to obtain better a performance in a variety of applications. The context switch is one of the most significant attributes of a multi-task operating system. It occurs when the CPU switches from one task to another, and requires saving the context of the current task so that the CPU can restore and complete it later. If not properly controlled, context switches may lead to reduced responsiveness, unnecessary delays, energy wastage, and extra memory requirements. These can lead to a high overhead in real-time embedded systems. Therefore, this work provides a highly versatile scheduling technique, with customized objectives and feasible results.

The rest of the paper is organized as follows: Section 2.3 describes the related work. Section 5.4 defines the model used. Section 2.5 formulates the scheduling problem and presents the ILP approaches for both periodic tasks systems and partitioned systems. We evaluate the proposals in Section 2.6.

2.3. Related work

Initially, static scheduling was the most used method in real-time systems, with the cyclic executive approach being the most common according to [2.3]. The idea is based on building a cyclic system that is practically made to measure. Static planning is a widely used planning method in real-time high integrity systems according to [2.36], since determinism is the most valued characteristic as a safety feature.

In contrast, dynamic scheduling offers greater flexibility. However, static scheduling with cyclic executives continues to exist. Several certification standards, like those listed in the previous section, include a requirement for the use of static allocation of resources. Even in the proposed safety-critical Java specification (SCJ) [2.19], the most constrained level, which should be especially suited for certification, prescribes the use of a cyclic executive.

As mentioned, there are heuristic algorithms that can find the optimal real-time schedule for simple uniprocessor architectures. However, many researchers are recently converting the classical scheduling problem into an ILP problem in order to optimize more complex models.

The works that obtain feasible schedules with ILP techniques in complex real-time models includes, among others: multiprocessor systems [2.4]; power consumption optimization [2.20]; consideration of architectures with local instruction or data caches [2.30]; weakly hard real-time systems [2.34]; mixed criticality [2.17]; distributed systems [2.24]; etc. This is due to the absence of heuristic optimal algorithms for more complex models than the typical periodic task model.

In [2.28], the precedence relation between tasks is considered. This work uses relaxed ILP techniques to obtain an optimal priority/deadline assignment for preemptive dynamic priority scheduling under precedence constraints. Also, the response time calculation and priority assignment problem with an ILP is presented in [2.26], where the ceiling of the response time equation is reformulated as an ILP problem. An improved real-time schedulability test that allows an exact and efficient definition of the feasible region by fewer binary variables is provided in [2.13]. Our goal is not only to find a feasible schedulable plan, but we are seeking to find the optimal schedule considering different objectives.

Regarding optimal strategies but not based on ILP, in [2.11] offline scheduling strategies for

non-preemptive real-time tasks on uniprocessors are proposed. Using formal approaches such as supervisory control theory (SCT) or time discrete event systems (TDES), authors present an optimal scheduler for aperiodic and sporadic tasks. In [2.12], similar techniques have been used to reclaim unused task computation times.

2.4. Model definition

2.4.1. Periodic task model

In a hard real-time system, there is a set of n independent real-time tasks $\tau = [\tau_1, ..., \tau_n]$, where each task generates a set of infinite jobs $(\tau_{ij}, j \geq 0)$ that must be completed before the arrival of the due time. This work considers that all tasks are periodic, i.e., composed by a activations every specified time. Each task is characterized as $\tau_i = (C_i, D_i, T_i)$, where T_i is the period, D_i is the deadline and C_i is the worst case execution time. From now on, D_i is relative deadline, in contrast with absolute deadline, which is $d_{ij} = j \cdot T_i + D_i$ for activation j. The utilization of a task τ_i is calculated as the relation between the computation time and the period, $U_i = \frac{C_i}{T_i}$.

For notational convenience and without loss of generality, we assume that the tasks are given in order of increasing deadline. Under the same deadline conditions, an order of increasing period is assumed.

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 periods of the tasks.

The response time (w_{ij}) of task τ_i in activation j is the time between an activation being released and the end of its execution. The worst case response time [2.18], [2.23] is the maximum time interval between arrival and finish instants for each task ($WCRT_i = \text{máx}\{w_{ij}\}$). Similar to the worst case response time, the best case response time of a task is defined as the minimum time between any release of a task and its corresponding completion ($BCRT_i = \text{mín}\{w_{ij}\}$), where the minimum is taken over all executions of the tasks and all possible phasing of the task respect to each other. If, for each task, $BCRT_i \leq WCRT_i \leq D_i$, the task set will be then schedulable.

Usually, it is desirable to reduce the overhead introduced by the context switch [2.32]. Multiple works address this topic. For example, [2.31] develops a new scheduling model, which unifies the concepts of preemptive and non-preemptive scheduling, and proposes algorithms for optimal assignment of priorities and preemption threshold. [2.32] presents a solution for reducing the number of context switches in multi-task scheduling, with task sets with limited hyperperiods.

2.4.2. Partitioned model

Partitioned systems were developed to address security and safety problems and provide temporal and spatial isolation. A partition consists of an encapsulated group of applications that provide independent execution on a common platform. The operating system is in charge

of supporting the execution of the applications. Partitions are executed independently on the top of the hardware, which could be virtual or otherwise.

In a partitioned system, tasks are organized in a set of p partitions $P_1, .., P_p$.

Therefore, we can extend our periodic task model to consider a partitioned model in the following way:

$$\tau_i = (C_i, D_i, T_i, \rho_i) \tag{2.1}$$

being ρ_i the identifier of the partition it belongs to.

An illustrative example of a partitioned system is presented in 2.2 in Section 2.5.3.

2.5. MILP Scheduling approaches

This section proposes a mixed-integer linear programming (MILP) formulation to determine the optimal offline schedule of periodic tasks on a uniprocessor hard real-time system. The following approaches can be taken to solve the proposed scheduling problem:

- A basic MILP formulation for periodic task systems that provides the optimal scheduling plan according to a certain optimization criteria based on reducing response times and context switches (see Section 2.5.1); This problem (feasibility of periodic tasks on uniprocessors) is co-NP-complete [2.5] and intractable in most cases, but it will serve as a reference for the rest of the approaches. Preemptive and non-preemptive scheduling will be considered.
- A MILP formulation with a rolling task approach. In this approach the MILP problem is decomposed into a set of MILP problems, one for each task. Each task activation is allocated in priority order.
- A MILP formulation for partitioned systems based on the two previous approaches (see Section 2.5.3).

Section 2.4.1 has introduced the general notations used throughout this paper, and the rest of notations will be introduced when describing any particular approach.

2.5.1. MILP Model of uniprocessor periodic task systems

We will call this model the Simple MILP model. Table 6.2 introduces the different indices, parameters, and variables used in the model.

According to the problem statement, the objective function is defined in Equation (2.2), which is minimizing the total number of context switches and the total response time for all tasks in all system executions. The problem is considered as multiobjective because it tries to reduce context switches and response times. The range of values and the units of context switches and response times are different. There is a need for scaling. Context switches are dimensionless. Thus, we need to normalize both context switches and response time to be of a similar scale. As context switching variable (s) is binary (0-1), we scale response times by dividing by deadlines for each activation and task, to have a fair trade-off between competing

Tabla 2.1: MILP model notation

Sets and indices	
i	Tasks $\tau_i \in \{1, 2, ..., n\}$
j	Activations of $\tau_i \in \{1, 2, ..., N_i\}$
Parameters	
C_i	Worst case computation time of τ_i
D_i	Relative deadline of τ_i
T_i	Period of τ_i
H	Hyperperiod of the task set
N_i	Number of activations of τ_i (H/T_i)
d_{ij}	Due date of τ_i in activation j
R_{ij}	$[j \cdot T_i, (j+1) \cdot T_i]$ Possible execution time interval for task i in activation j
Decision variables	
x_{ijt}	Task execution matrix. 1 if τ_i in activation j is executed at time t and 0 otherwise.
s_{ijt}	Context switch matrix. 1 if τ_i in activation j is active at time t and not at time t + *1* or viceversa and 0 otherwise.
w_{ij}	Response time matrix. Response time of τ_i in activation j

objectives. Later, in section 2.5.4, we define the weights using a goal programming approach presented in [2.35] in order to optimize each objective by minimizing its deviation from their target value.

Therefore, Equation (2.2) minimizes the sum of multiple (normalized) objectives, depending on the levels of importance of each objective. Exact values for K_1 and K_2 will be provided in Section 2.6 to evaluate the proposed approaches.

$$\text{mín} \quad \text{Obj} = \sum_{\forall(i,j)} \sum_{\forall t} K_1 s_{ijt} + \sum_{\forall(i,j)} K_2 \frac{w_{ij}}{D_i} \tag{2.2}$$

s.t:

$$x_{ijt} = 0 \qquad \forall t, i, j | t \notin R_{ij} \tag{2.3}$$

$$\sum_{t \in R_{ij}} x_{ijt} = C_i \qquad \forall i, j \tag{2.4}$$

$$t \cdot x_{ijt} \leq d_{ij} - 1 \qquad \forall t \in R_{ij} \tag{2.5}$$

$$\sum_{(i,j)} x_{ijt} \leq 1 \qquad \forall t \in \{0, 1, ..., H\} \tag{2.6}$$

$$s_{ijt} \leq x_{ijt} + x_{ij(t+1)} \qquad \forall t, i, j | t \in \{0, 1, ..., H-1\} \tag{2.7}$$

$$s_{ijt} \geq x_{ijt} - x_{ij(t+1)} \qquad \forall t, i, j | t \in \{0, 1, ..., H-1\} \tag{2.8}$$

$$s_{ijt} \geq x_{ij(t+1)} - x_{ijt} \qquad \forall t, i, j | t \in \{0, 1, ..., H-1\} \tag{2.9}$$

$$s_{ijt} \leq 2 - x_{ijt} - x_{ij(t+1)} \forall t, i, j | t \in \{0, 1, ..., H-1\} \tag{2.10}$$

$$w_{ij} \geq t \cdot x_{ijt} - j \cdot T_i + 1 \forall t, i, j \tag{2.11}$$

$$x_{ijt}, s_{ijt} \in \{0, 1\} \tag{2.12}$$

$$w_{ij} \geq 0 \tag{2.13}$$

Constraints (2.3), (2.4), (2.5) and (2.6) are the key constraints and are the basic scheduling conditions. Constraint (2.3) does not allow a task to be executed outside its possible intervals of execution, $[j \cdot T_i, (j+1) \cdot T_i]$, for all each activation *a*. Constraint (2.4) assures that the task completes all its execution time. Equation (2.5) ensures real-time requirements, as tasks must end before the arrival of their deadlines. Constraint (2.6) ensures that only one task is being executed at each point in time.

Constraints (2.7 - 2.10) determine if a context switch s_{ijt} happens or not, i.e., if a task is activated in a time *t* and not in time *t+1* or vice versa. This behaviour is treated as an exclusive OR (XOR), which gives true only if one, and only one, of the inputs is true, $x_{ij(t+1)} \oplus x_{ijt}$. As stated before, s_{ijt} takes value 1 if there is a context switch.

Constraint (2.11) calculates the response time of each activation of all tasks. Equations (2.12) and (2.13) represent the decision variable domains.

2.5.1.1. Non-preemptive MILP Model

The previous approach provides an offline preemptive scheduling plan for a task set. It means that tasks can be interrupted by other tasks and restarted later. If non-preemptive behavior

is desired, i.e., if a task has to be executed until it is completed without interruption, a new constraint will be added in the model (Equation (2.14)).

$$\sum_{\forall(i,j)} s_{ijt} \leq 2 - x_{ij0} \quad \forall t \in \{0, 1, ..., H-1\} \tag{2.14}$$

2.5.1.2. Enhanced MILP Model of uniprocessor periodic task systems

A possible way to reduce solution times of MILP problems is via a *warm start*, i.e., to provide manually starting solution vectors of the problem to the solver [2.16].

If the MILP solver finds that the input solution is feasible, then the input solution provides an incumbent solution and a bound for the branch-and-bound algorithm. If the solution is not feasible, the MILP solver tries to repair it. When it is difficult to find a good integer feasible solution for a problem, a warm start can significantly reduce the solution time.

The effectiveness of the warm start in MILP solvers depends on many factors. Sometimes, warm starts do not help the solver to find solutions more quickly, but authors may consider providing feasible starting points to fix an upper bound on the objective value (in case of minimization) and thus can be used to prune nodes during the search.

The feasible starting point considered in this work is the optimal scheduling policy known as Deadline Monotonic Scheduling (DM) [2.25]. DM is a fixed-priority scheduling algorithm that assigns the highest priority to the task with the shortest deadline.

Therefore, an offline feasible scheduling plan for the task set will be generated using DM, and this plan will be the starting point of the Simple MILP model described in Section 2.5.1.

2.5.1.3. Decision variables size

In some cases, the solution time of MILP problems grows exponentially with respect to the problem size. For the simple MILP model, the size of the decision variables is the number of elements in matrices x_{ijt}, s_{ijt} and w_{ij}. Let be S_x, S_s and S_w the number of elements that differ from zero in x_{ijt}, s_{ijt} and w_{ij}, respectively. For all these matrices, the number of rows is equal to the number of tasks in the system, n.

As far as w_{ij} matrix is concerned, the size is (rows x columns):

$$S_w = \sum_{i=1,..,n} N_i \tag{2.15}$$

However, the number of zero elements in the matrices x_{ijt} and s_{ijt} is considerably large. Therefore, we only count the nonzero elements of those variables when computing the problem size.

For x_{ijt}, the number of nonzero elements in one row i and activation j is C_i. So, in row i for all activations, the number of nonzero elements is:

$$C_i \cdot N_i = C_i \frac{H}{T_i} = U_i \cdot H \tag{2.16}$$

As x_{ijt} has n rows, the set of points of x_{ijt} to calculate in the MILP problem is:

$$S_x = \sum_{i=1,..,n} U_i \cdot H = U \cdot H \tag{2.17}$$

In s_{ijt}, the same reasoning applies, but the maximum number of elements in a row is $C_i + 2$, considering the worst case in which a context switch occurs in each execution unit of C_i. Thus:

$$S_s = U \cdot H + 2 \cdot \sum_{i=1,..,n} N_i \tag{2.18}$$

The size S of the problem is then:

$$S = S_x + S_s + S_w = 2 \cdot U \cdot H + 3 \cdot \sum_{i=1,..,n} N_i \tag{2.19}$$

which is directly proportional to the number of tasks n, the total utilization U and the length of the hyperperiod H.

As it has been demonstrated, the simple MILP model is too computationally intensive to be used for large task sets with high utilization and long hyperperiods. For this reason, we propose an alternative MILP formulation in the next section.

2.5.2. Rolling task MILP model

In the rolling task scheme, the MILP problem consists of scheduling only one task in the hyperperiod. Once all activations of this task are allocated, the next task is chosen. The previous solution is an input to the new problem as a new constraint. This constraint implies that the new task can not be allocated in the same instants of time that the first one. This process is repeated until all tasks are allocated. The algorithm is explained in Fig. 2.1. For instance, let us define a set of three tasks $\tau_i = (C_i, T_i)$, with implicit deadlines ($D_i = T_i \ \forall i$), being $\tau_0 = (2, 5)$, $\tau_1 = (3, 9)$ and $\tau_2 = (3, 18)$. Note that tasks are ordered by increasing deadlines. First, τ_0 is allocated with the objective of minimizing its response time and context switches. As a result, plan σ_0 is defined. τ_1 is then allocated in the idle slots of time of plan σ_0. Note that, among all possibilities of allocation, the algorithm selects the one with the most balanced weighted sum of context switches and response times. As a result, plan σ_1 is defined. Finally, τ_2 is allocated in the idle slots of time of plan σ_1, following the same criteria.

If tasks are chosen in a priority scheme, the result is similar to the fixed priority non-preemptive scheduling algorithm with priorities equal to deadlines. Note that the reduction in time in rolling MILP approach comes from the a priori determined priority ordering.

The optimization criteria are maintained as in the simple MILP problem, minimizing both context switch and response time.

With this purpose, variations over the previous approach are done and are explained in Algorithm 1. The algorithm works as follows: the system receives the task set ordered by increasing deadlines. The first task, $i = 0$, is selected in order to be optimized, with the objective of reducing its context switch and response time. To avoid that other tasks are now included in the model, we force that all $x_{ijt} = 0$ for $i > 0$ (line 8). Once this is done, the variable x_{00t} is saved in a global auxiliary variable that contains the plan, σ_{ijt} (line 13). Previous constraints

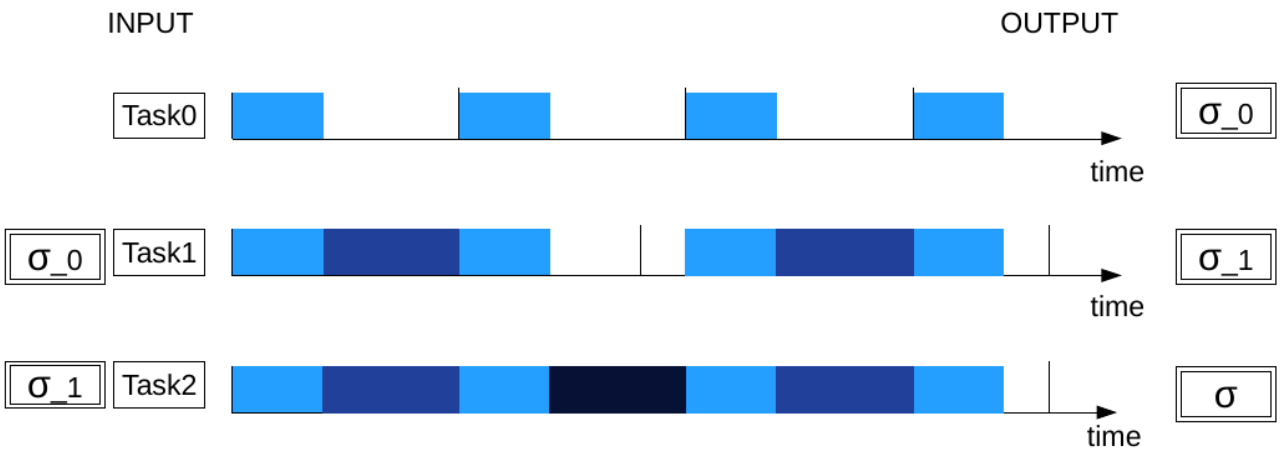


Figura 2.1: Functioning of the rolling task algorithm.

are then removed (line 17) and the model is updated (line 18). The algorithm moves to next task, $i = 1$. To maintain the previous assigned values of x, we force $x_{ijt} = \sigma_{ijt}$ for those tasks $i < 1$ and $x_{ijt} = 0$ for $i > 1$ (line 10). x_{11t} is then optimized. This process is repeated for all tasks and the optimized scheduling plan is saved in the variable σ_{ijt}[1].

The previous algorithm is translated into an MILP problem, which is similar to that stated in Section 2.5.1. For each task, the value of the objective function is calculated in Equation (2.20). The difference between this equation and the one in the simple MILP is that Equation (2.20) evaluates the objective function once for each task, while Equation (6.15) evaluates it for all the tasks in one step.

$$\text{mín} \quad \text{Obj} = \sum_{\forall j \in N_i} \sum_{\forall t} K_1 s_{ijt} + \sum_{\forall j \in N_i} K_2 \frac{w_{ij}}{D_i} \tag{2.20}$$

[1]In Fig. 2.1, variable σ_{ijt} omits subindexes j and t for improved clarity and only refers to the task i.

Algorithm 1 Rolling task MILP algorithm

```
INPUT: Task set with n tasks ordered by increasing deadlines
OUTPUT: Scheduling plan, σ
procedure ROLLING TASK MILP ALGORITHM(τ)
    Calculate H, d_ij, R_ij
    Plan σ_ijt → 0
    for τ_i ∈ τ do
        if i < n then
            x_kjt = 0    i < k ≤ n
        if i > 0 then
            x_kjt = σ_kjt
        add Constraints for task i
        set Objective and optimize
        if feasible then
            σ_ijt = x_ijt
        else
            Exit
        if i < n-1 then
            Remove constraints of the model
            Update the model
```

s.t:

$$x_{kjt} = \sigma_{kjt} \qquad \forall k,t|j\in[0,1,...,i-1],t\in\{0,1,...,H\} \tag{2.21}$$

$$x_{kjt} = 0 \qquad \forall k,t|j\in I-[0,1,...,i],t\in\{0,1,...,H\} \tag{2.22}$$

$$x_{ijt} = 0 \qquad \forall t,j|t \notin R_{ij}, j \in N_i \tag{2.23}$$

$$\sum_{t\in R_{ij}} x_{ijt} = C_i \qquad \forall j \in N_i \tag{2.24}$$

$$t \cdot x_{ijt} \leq d_{ij} - 1 \qquad \forall t,j|t \in R_{ij}, j \in N_i \tag{2.25}$$

$$\sum_{l} x_{ijt} \leq 1 \qquad \forall t \in \{0,1,...,H\} \tag{2.26}$$

$$s_{ijt} \leq x_{ijt} + x_{ij(t+1)} \qquad \forall t,i,j|t\in\{0,1,...,H-1\},j\in N_i \tag{2.27}$$

$$s_{ijt} \geq x_{ijt} - x_{ij(t+1)} \qquad \forall t,i,j|t\in\{0,1,...,H-1\},j\in N_i \tag{2.28}$$

$$s_{ijt} \geq x_{ij(t+1)} - x_{ijt} \qquad \forall t,i,j|t\in\{0,1,...,H-1\},j\in N_i \tag{2.29}$$

$$s_{ijt} \leq 2 - x_{ijt} - x_{ij(t+1)} \forall t,i,j|t\in\{0,1,...,H-1\},j\in N_i \tag{2.30}$$

$$w_{ij} \geq t \cdot x_{ijt} - j \cdot T_i + 1 \forall t,j|j \in N_i \tag{2.31}$$

$$x_{ijt}, s_{ijt} \in \{0,1\} \tag{2.32}$$

$$w_{ij} \geq 0 \tag{2.33}$$

In each pass of the algorithm (i.e., for each task) only that task is evaluated. Therefore, it inherits the execution plan of previous tasks (Constraint 2.21) and ignores next tasks (Cons-

traint 2.22). The other constraints are similar to the ones presented in Section 2.5.1, considering only a single task rather than a set of tasks. Note that the non-preemptive version of the rolling task MILP model is easy to obtain by adding constraint 2.14 to the previous model.

2.5.3. Partitioned systems MILP model

The model presented in Section 2.5.2 can be used to schedule a partitioned system, as defined in Section 2.4.2. The idea is to make a hybrid approach between the periodic and the rolling task model.

Given a task model where each task is defined as $\tau_i = (C_i, D_i, T_i, \rho_i)$, the rolling partition approach selects the first partition and schedules all tasks within this partition, following the rolling task approach defined in Section 2.5.2. The next partition is then selected and all its tasks are scheduled. The algorithm ends when all partitions are allocated. It is explained in Algorithm 2.

Algorithm 2 Rolling partition MILP algorithm

```
1: INPUT: Task set with n tasks ordered by increasing deadlines
2: OUTPUT: Scheduling plan, σ
3: procedure ROLLING PARTITION MILP ALGORITHM(τ)
4:     Calculate H, d_ij, R_ij
5:     Order tasks by number of partition → Partitions[ ]
6:     for ρ_i ∈ Partitions do
7:         for τ_i ∈ ρ_i do
8:             Rolling task MILP algorithm(τ_i)
```

We have made numerous simulations to investigate the feasibility of this method; however, we found some counter examples that make it necessary to propose another approach to solve the partitioned MILP scheduling problem. First, we are going to show one of these counter examples.

2.5.3.1. Counter example

Given a task set as shown in Table 2.2, we can schedule this system following a flat scheduling approach. But first, the hierarchical scheduling is presented. Partitioned software architectures define two levels of hierarchy: the global level, with a scheduler that allocates CPU time to partitions; and a local scheduler per partition, which schedules the tasks using the available time per partition. Flat scheduling is considered [2.8] among the strategies that can be followed to achieve hierarchical scheduling. This approach consists in removing all the barriers defined by the partitioned system and schedule and considering all the tasks at once. We then suppose that a single global scheduler is in charge of managing all the tasks and conducts the corresponding scheduling algorithm. The last step is to adapt the solution back to the partitioned system by grouping. With this, the final schedule will be very efficient and, sometimes, optimal. In this strategy, the timing knowledge of tasks should be previously detailed in depth in order to analyze the overall system and optimize the solution.

Fig. 2.2 shows the scheduling plan obtained with this approach. To schedule the flat system, we can use both the DM or the rolling task MILP algorithm. As seen in the temporal plan, all deadlines are met, so the system is schedulable.

Tabla 2.2: Task set parameters τ

	C_i	D_i	T_i	ρ_i
τ_0	2	5	5	0
τ_1	3	10	10	1
τ_2	4	20	20	0

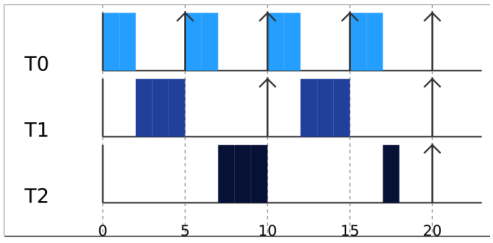
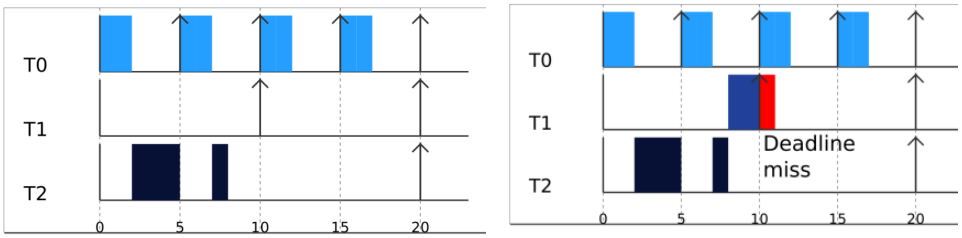


Figura 2.2: Task set τ scheduled with DM.

We are now applying the rolling partition MILP algorithm to the same set. According to Algorithm 2, partition 0 is initially selected. All tasks within that partition (τ_0 and τ_2) are then allocated in the core, using the rolling task algorithm. The result is shown in Fig. 2.3a. The algorithm then moves to partition 1 and tries to allocate the tasks within that partition (τ_1) into the available slots of the previous scheduling plan. As shown in Fig. 2.3b, it is impossible to execute τ_1 entirely, and the task misses the deadline in its first execution. Therefore, the system is not schedulable.



(a) Partition 0 allocation.

(b) Partition 1 allocation.

Figura 2.3: Task set τ scheduled with Rolling partition algorithm.

As shown in this example, the rolling task approach performs well in terms of feasibility, but its extension to the hybrid rolling partition algorithm does not provide a feasible result. Therefore, another approach for partitioned systems must be proposed.

2.5.3.2. Hierarchical MILP model for partitioned systems

As mentioned in the previous section, we will consider a hierarchical system with global and local levels. We consider a periodic server at the global level and the rolling task MILP approach at the local level.

At the global level, the scheduler periodically assigns the CPU to the partition, i.e., $\Gamma = (\theta, \pi)$ provides θ units of CPU every π units of time. Once time has been assigned for each partition, the local scheduler allocates the tasks that belong to the partition in their corresponding time slots, following the rolling task algorithm.

The method for calculating the global level periodic resource is explained hereafter: the first step consists of calculating the amount of CPU required for each partition P_i as the sum of the utilizations required for its tasks:

$$U_{P_i} = \sum_{\forall \tau_m \in P_i} \frac{C_m}{T_m} \tag{2.34}$$

We now have to fix θ_{P_i} or π_{P_i} and calculate the other using U_{P_i}. We will set the period of the partition to the shortest period of the tasks that belong to this partition. The credit of the periodic server for each partition θ_{P_i} is then calculated as:

$$\theta_{P_i} = U_{P_i} \cdot min(T_m) \, \forall \tau_m \in P_i \tag{2.35}$$

Algorithm 3 explains the hierarchical MILP model. First, the global level scheduling is evaluated (line 3). For each partition, the algorithm calculates the periodic resource through the parameters of the tasks that belong to that partition (lines 6-7). The output of this function is the periodic resource for each partition (line 8). The local level scheduling is then calculated for each partition (line 9). The function uses as an input parameter the periodic resource of each partition and schedules the tasks in this resource (line 13) to generate the temporal plan. If the partition does not completely use its credit to schedule tasks, the remaining time will be transferred to the next partitions (lines 14-15). As a result, the local plan for each partition is calculated (line 16).

For the task set defined in Table 2.2, the global level periodic resource is calculated as:

$$U_{P_0} = \frac{2}{5} + \frac{4}{20} = 0{,}60$$
$$U_{P_1} = \frac{3}{10} \qquad = 0{,}30$$

$$\theta_{P_0} = 0{,}60 \cdot 5 = 3$$
$$\theta_{P_1} = 0{,}3 \cdot 10 = 3$$

Therefore, the periodic resources for the partitions are modelled as: $\Gamma_{\rho_0} = (3, 5)$ and $\Gamma_{\rho_1} = (3, 10)$, assuming integer numbers for all parameters. The global scheduling plan is depicted in Fig. 2.4.

We followed the same approach for a small instance of the problem presented in Table 2.2. As shown in Fig. 2.5, the plan is schedulable as all the tasks at different activations are completed before their deadlines. Moreover, the obtained schedule is very efficient in terms of the context switch.

Algorithm 3 Hierarchical MILP algorithm

INPUT: Task set with n tasks ordered by increasing deadlines
OUTPUT: Scheduling plan, σ
procedure GLOBAL-LEVEL-SCHEDULING(τ)
    Group tasks by partition index $\rightarrow$ *Partitions*[ ]
    for $\rho_i \in$ Partitions **do** ▷ Calculate the periodic resource for each partition
        $\pi_{\rho_i} = \min(\mathrm{T}_j) \quad \forall\ \mathrm{task}_j \in \rho_i$
        $\theta_{\rho_i} = \mathrm{int}\left(\pi_{\rho_i} \cdot \sum_{task_j \in \rho_i} U_j\right)$
    return $\Gamma_{\rho_i}(\pi_{\rho_i}, \theta_{\rho_i}) \quad \forall \rho_i \in Partitions[\,]$;
procedure LOCAL-LEVEL-SCHEDULING($\Gamma(\pi, \theta)$, τ, *Partitions*[ ])
    for $\rho_i \in$ Partitions **do**
        Initialize plan σ
        for $\tau_j \in \rho_i$ **do** ▷ Application of rolling task MILP approach
            σ += Schedule τ_j in $\Gamma_{\rho_i}(\pi_{\rho_i}, \theta_{\rho_i})$
        Idle-time$_i$ = $\Gamma_{\rho_i}(\pi_{\rho_i}, \theta_{\rho_i})$ - σ
        $\sigma \quad \forall i \in Partitions[\,]$
    return $\Gamma_{\rho_i}(\pi_{\rho_i}, \theta_{\rho_i}) \quad \forall \rho_i \in Partitions[\,]$;

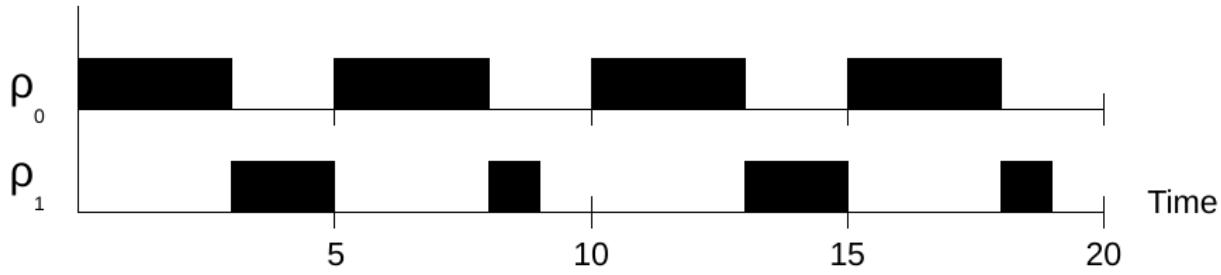


Figura 2.4: Periodic server in the global level.

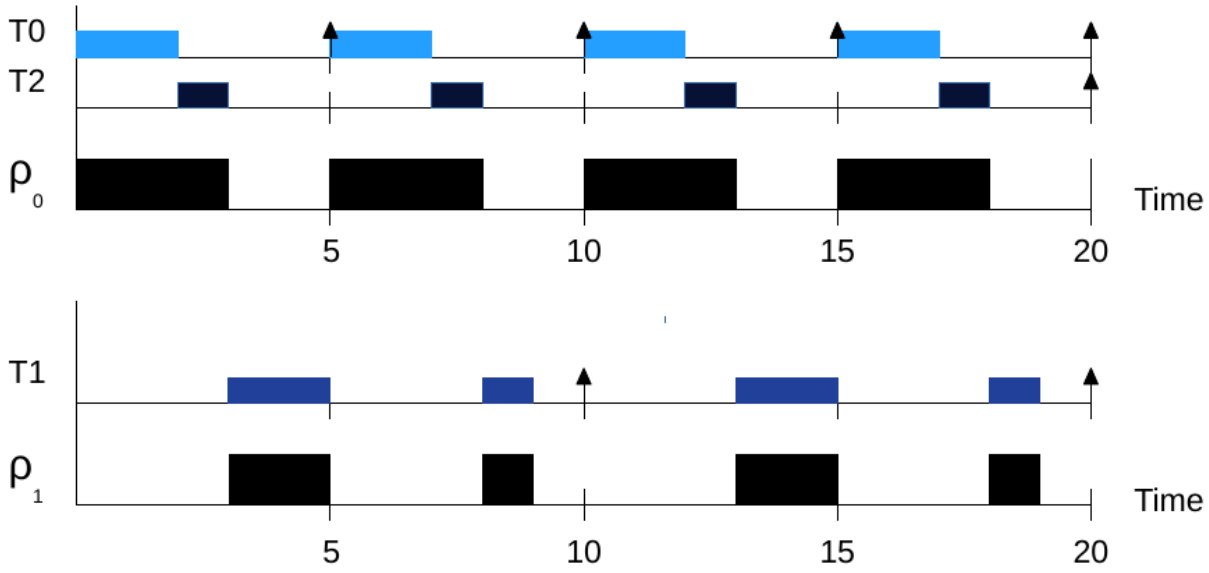


Figura 2.5: Rolling task MILP in the local level.

2.5.4. Selection of the weights in the multiobjective function.

A method to select the weights K_1 and K_2 in order to optimize both objectives is presented in [2.35] using a goal programming objective method. The procedure consists of solving the MILP problem twice, once for each objective and then obtain a function that closely approximates both optimal values. For this purpose, the task set defined in Table 2.2 is used and the target values are obtained:

Tabla 2.3: Target values for single objectives.

Objective	Response Time	Context switch
Minimize Response time	3.1	14
Minimize Context Switches	4.95	12

We then minimize the percentage deviations from the target values to optimize the response time and context switch, simultaneously (Equation (2.36)):

$$\text{mín} \quad K_1 \sum_{\forall(i,j)} \sum_{\forall t} \frac{s_{ijt} - 12}{12} + K_2 \sum_{\forall(i,j)} \frac{1}{D_i} \frac{WCRT_i - 3{,}1}{3{,}1} \tag{2.36}$$

Depending on each objective's relative importance, the value of K_1 and K_2 might vary in the range of [0,1] such that $K_1 + K_2 = 1$ (Fig. 2.6). It is the decision maker's responsibility to choose the weights that best fit the requirements.

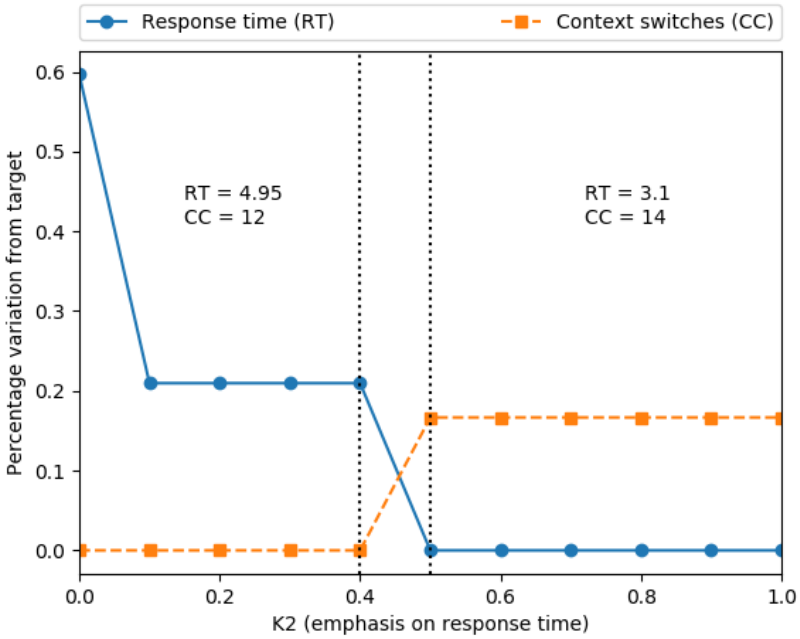


Figura 2.6: Optimal solutions for minimizing the weighted sum of percentage variations of the objectives, with different values for K_1 and K_2.

2.6. Experimental evaluation

This section evaluates both the quality of generated schedules and the required time for obtaining them. The assessed MILP models are:

- Simple (GS) presented in Section 2.5.1.1.
- Simple with start solution (GS2) presented in Section 2.5.1.2.
- Rolling task (GR) presented in Section 2.5.2.

As far as weights K_1 and K_2 is concerned, we have considered $K_1 = 1$ and $K_2 = 1$ for GR, GS and GS2. These weights are set in this way in order to provide a balanced solution in which context switch and WCRT are reduced equally. The relation between the response time of a task and its deadline is always less than or equal to one (a task cannot end after its deadline). Therefore, the weight of the context switch is usually slightly greater than the weight of the response time. Finally, we evaluate the preemptive versions of all MILP models.

2.6.1. Experimental conditions

The simulation scenario developed for this work is depicted in Fig. 2.7. It is divided into three steps:

- Generation of the load (see Section 5.6.1.1).

- Execution of MILP approaches and DM (see Section 2.5).
- Validation (see Section 3.7.1.4).

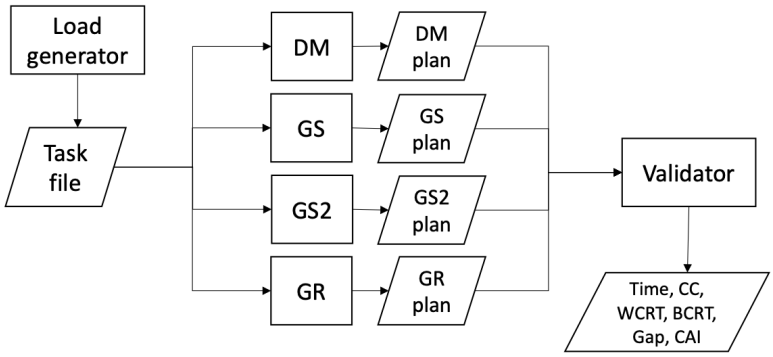


Figura 2.7: Experimental evaluation overview.

The automatic load generator generates a task set and the associated parameters with the process described in Section 5.6.1.1. This task set is the input for the 3 MILP approaches and the DM scheduler that generates the corresponding scheduling plans. If the obtained plans are then valid, their performance parameters will be stored. This sequence is repeated to complete a sufficient number of simulations.

We use Gurobi optimizer [2.2], from Gurobi Optimization, Inc., which is a powerful optimizer designed from scratch to run in multi-core and with capability to run in parallel mode. It has achieved performance improvement with each version and provides a Python interface. Version 9.0 [2.2] is selected for solving our proposed MILP formulations. All MILP approaches described in previous sections are executed on an Intel Core i7 CPU with 16GB of RAM.

2.6.1.1. Load generator

The load is generated using a synthetic task generator. Given the system utilization value and a number of tasks for each set, the utilization is shared among the tasks using the UUniFast discard algorithm [2.10]. Computation times are generated randomly in [2,10] and periods are deduced from the system utilization. We restricted the hyperperiod to be no larger than 5000, so we discard the sets that exceed this limit, repeating the generation again until enough feasible instances are found.

2.6.1.2. Validation of the results

The validation consists of two steps. Firstly, we must check feasibility to ensure all deadlines are met. Secondly, several performance parameters are obtained to compare different methods. Specifically, these are the parameters obtained for each set:

- Response times: We calculate best and worst case response times as defined in Section 2.4.1. In order to evaluate WCRT and BCRT of the task set, they will be calculated as the average between $\mathrm{WCRT}_i/\mathrm{D}_i$ and $\mathrm{BCRT}_i/\mathrm{D}_i$ for all tasks in the set, respectively.

- Number of context switches (CC): This parameter is computed by counting the number of times tasks are preempted by others.
- Solution time: For MILP approaches, we calculate the time spent to obtain a solution. As simple MILP models are too computationally-intensive for some task sets, the solver is limited to a certain time limit of solution time. After this time, the solution is stored.

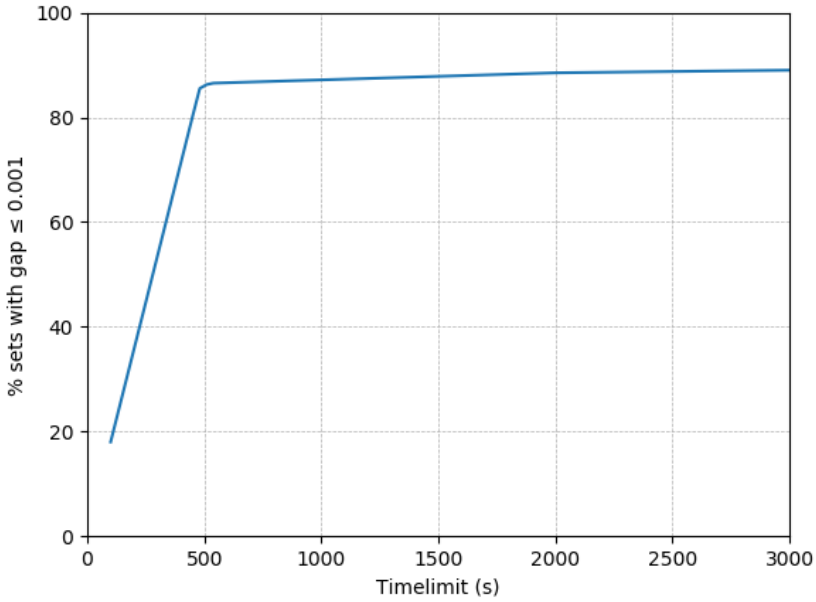


Figura 2.8: Experiment to select the timelimit parameter.

To pick the best time limit, we vary the GS algorithm time limits to between 100 and 3000 s to solve different instances of the problem. We then plot the percentage of the cases that have been solved to optimality (optimality gap <= 0,001) vs. the time limit. As depicted in Figure 2.8, up to 500 s, the percentage of optimal cases increases significantly. However, the improvement is negligible after that. Therefore, we conclude that 500 s is the right choice for the time limit.

- Optimality gap: If the previously chosen time limit is reached, the solution is not optimal. Therefore, the distance to the optimal solution should be analyzed. This distance is called the gap hereafter. It is the distance between the estimated lower bound (for minimization problem) and the best feasible solution that has been found so far. As looking for proven optimal solutions takes a long time to compute, a common practice is to look for a solution that guarantees not worse than x % (gap) from the optimal solution. A significant advantage is that most of the gap is often reduced quite early.
- Control action interval (CAI): It is the percentage variation of the control action delivery of a task relative to its period [2.9]. This parameter allows us to compare the performances

of a control system. The CAI of a task i is calculated as:

$$CAI_i(\%) = \frac{WCRT_i - BCRT_i}{T_i} \cdot 100 \tag{2.37}$$

2.6.2. Experimental results

This subsection evaluates the performance of the proposed MILP approaches and the scheduling policy DM using the performance parameters (i.e., quality of the solution and computation intensity) that are obtained in the evaluation process.

2.6.2.1. Comparison of ILP approaches

2.6.2.1.1. Solution time The results show that both the simple model and the simple model with a warm start take a long time to run. In fact, the median of all the simulations with GS and GS2 approximates the time limit (500 seconds), as depicted in Fig. 2.9. When the time limit is usually reached, only good solutions are found and not the optimal solution. In the case of a rolling approach, promising results are achieved. GR needs five times less solution time than GS and GS2.

In terms of solution time and computation intensity, it is clear that the best approach is GR. In Fig. 2.10, we can see the relationship between utilization, the number of tasks, and the solution time for GR. As shown in Section 2.5.1.3, the solution time increases with the number of tasks and utilization.

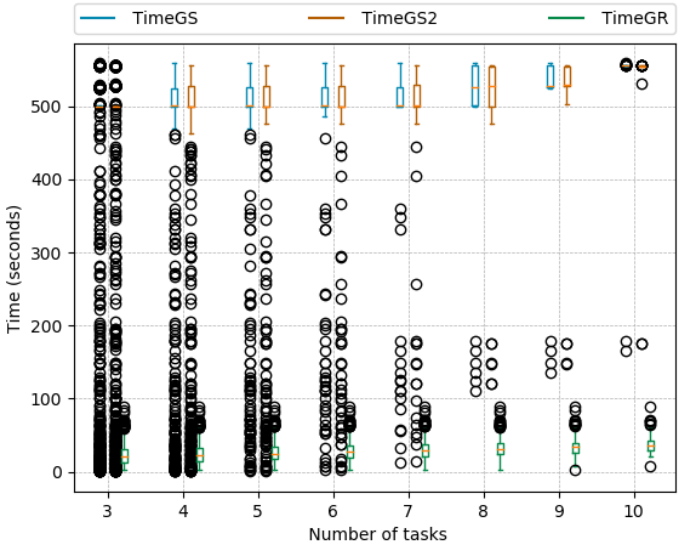
We then compare the rest of the performance parameters to check if it is worth spending computation time to obtain more significant results.

2.6.2.1.2. Gap This subsection studies the GS and GS2 optimality gaps as they could not find the optimal solution in the selected time limit. As observed in Fig. 2.11, the optimality gap is strikingly similar in GS and GS2. Therefore, providing a first solution (warm start) did not improve the solution time nor the solution quality. It is further observed that the gap increases with the number of tasks in the system, which is in line with the results in the previous section.

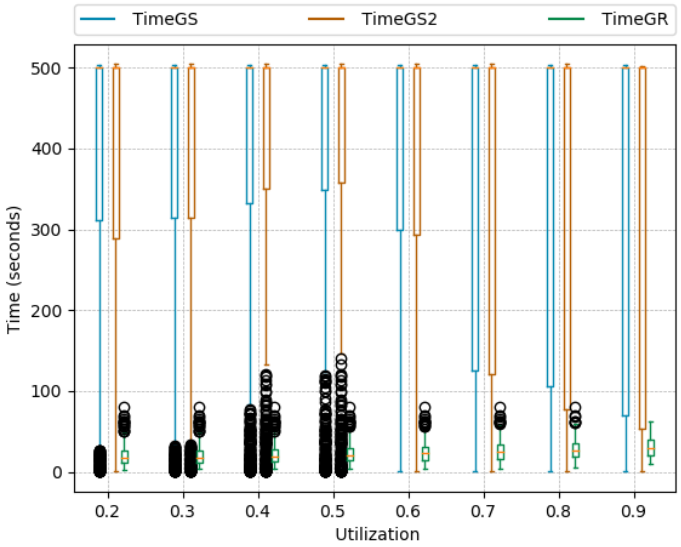
2.6.2.1.3. Worst case response time Regarding the WCRT, we can see in Fig. 2.12 that DM achieves the best result. This is expected as DM schedules tasks with shorter deadlines first. But regarding MILP approaches, GR achieves better results than GS or GS2. This is because GR obtains an optimal solution while GS and GS2 reach the time limit without finding an optimal solution in the major part of the task sets.

2.6.2.1.4. Context switches Regarding the number of context switches, as depicted in Fig. 2.13, GS and GS2 approaches produced fewer context switches than GR and DM. It should be noted that GS and GS2 coincide in this figure.

It is clear that, as far as CC is concerned, DM will obtain the worst results, since it is not a scheduling algorithm focused on reducing CC. Regarding GS, GS2 and GR, all have CC



(a) Number of tasks



(b) Utilization

Figura 2.9: Solution times depending on the utilizations and the number of tasks of the sets.

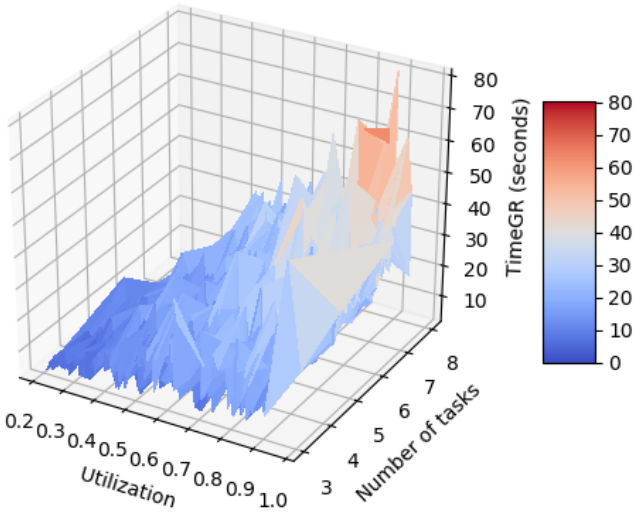


Figura 2.10: Influence of the number of tasks and system utilization in the solution time of the GR algorithm.

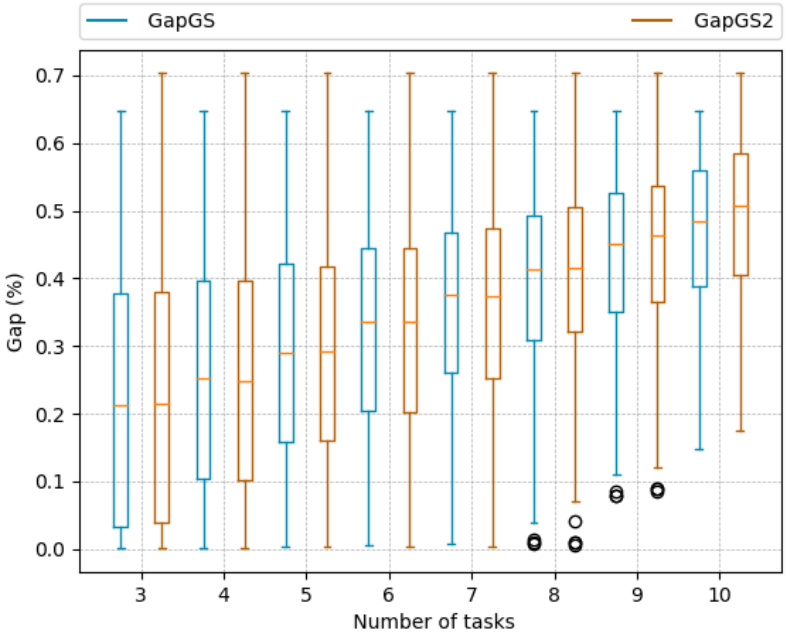


Figura 2.11: Influence of the number of tasks in the percentage of optimality gap.

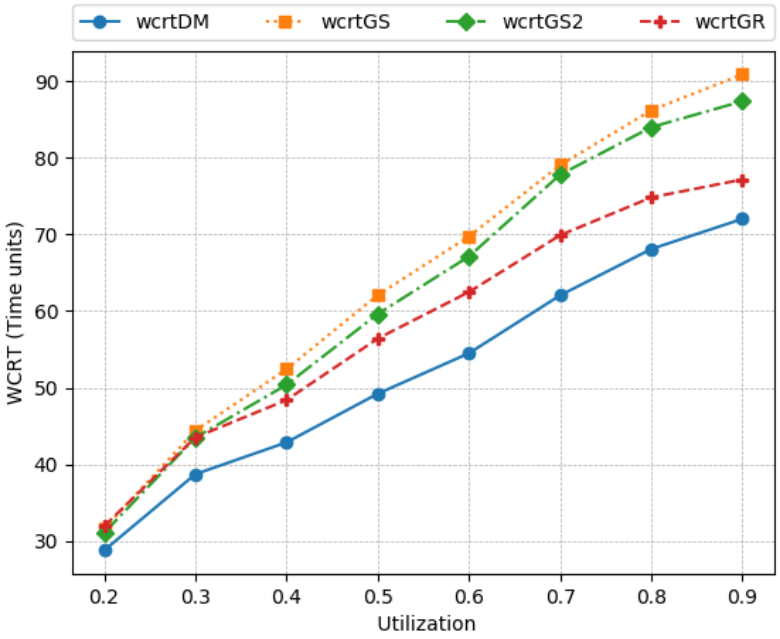


Figura 2.12: Influence of the system utilization in the worst case response times.

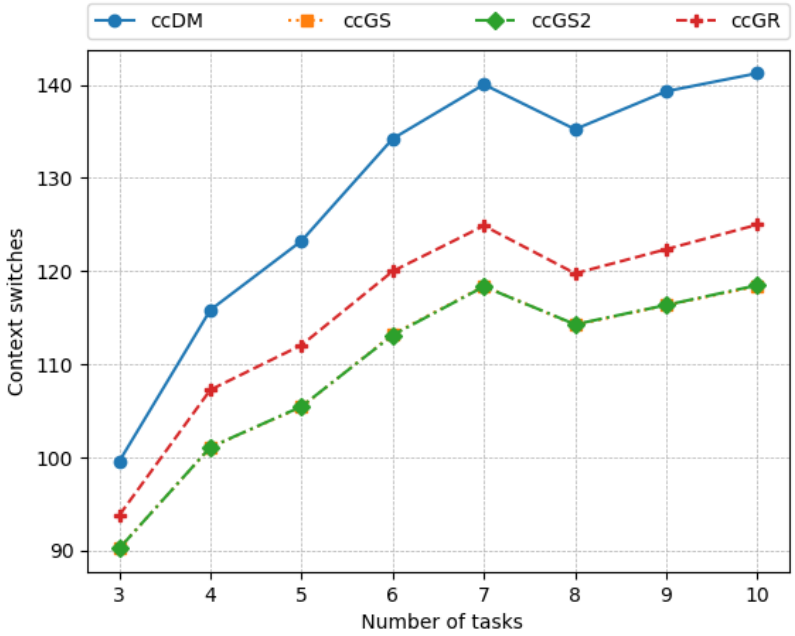
minimization as objective. However, the differences between GR and GS or GS2 are related to the way in which GR performs the optimization. As GR optimizes by following a rolling task approach, it means that tasks are scheduled in lower deadline order. This is very similar to DM and this is why GR achieves worse results in terms of CC than GS and GS2. Nonetheless, as GR has CC minimization in its objective, it achieves better results than DM.

As a reminder, the authors consider reducing the response time and the number of context switches as the main objectives of the problem. DM is the approach that generates a plan with the shortest response times and, in contrast, more context switches. GS is completely the opposite. This is due to the weights of the objectives in the multiobjective function. Depending on these coefficients, the results may change, as explained later in Section 2.6.2.2.

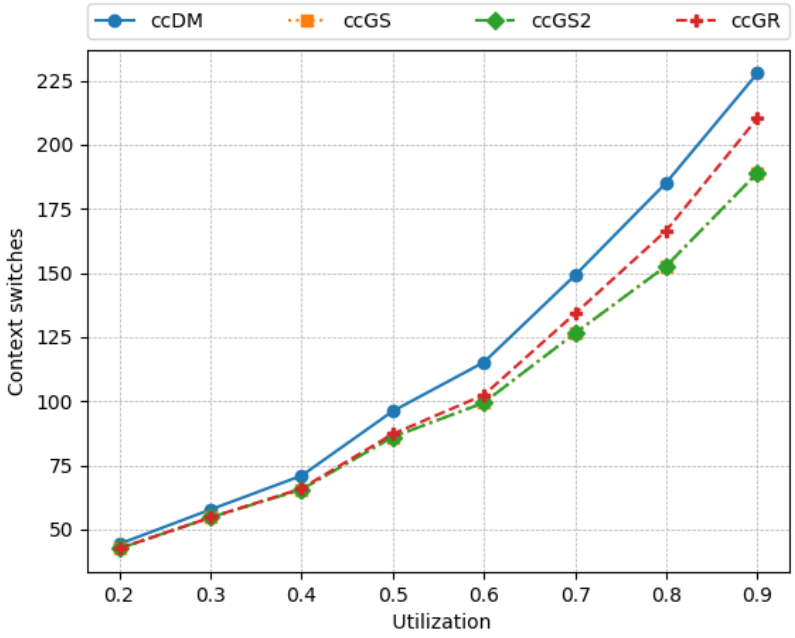
2.6.2.1.5. Best case response time In the case of the best case response time, GS is the approach with the shortest BCRT than other approaches. The ratio between BCRT and deadlines increases with the number of tasks and system utilizations, as seen in Fig. 2.14.

2.6.2.1.6. Control activation interval Finally, regarding to the percentage of CAI, DM is the approach that implies a better system performance, as observed in Fig. 2.15. This may be logical since GR and GS do not explicitly use a criterion to minimize CAI.

As a conclusion, we consider that GR the best approach in terms of time and performance and it will be studied in depth hereafter. Moreover, through an in-depth study, the GR approach is extended and applied to different objectives. This leads us to the next section, where we will investigate the effect of varying the constants K1 and K2.

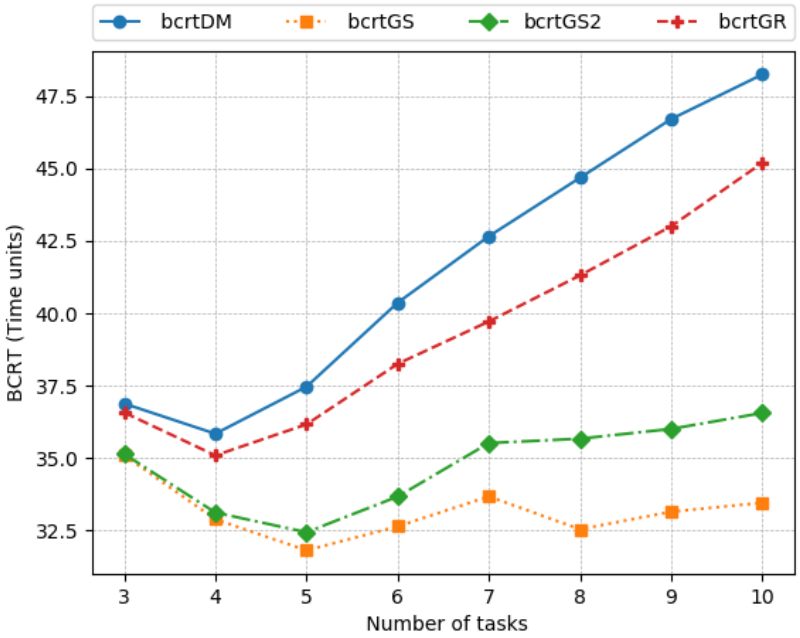


(a) Number of tasks

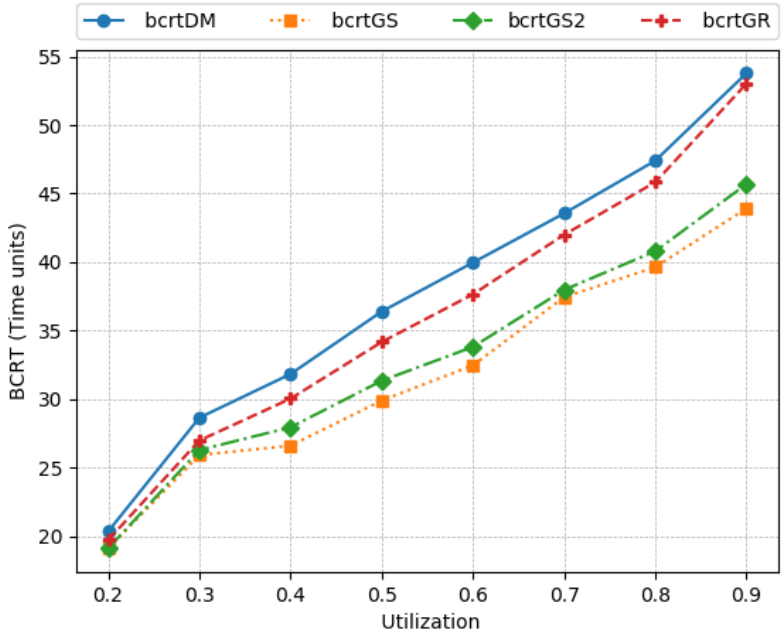


(b) Utilization

Figura 2.13: Impact of the system utilization and number of tasks in the context switches.

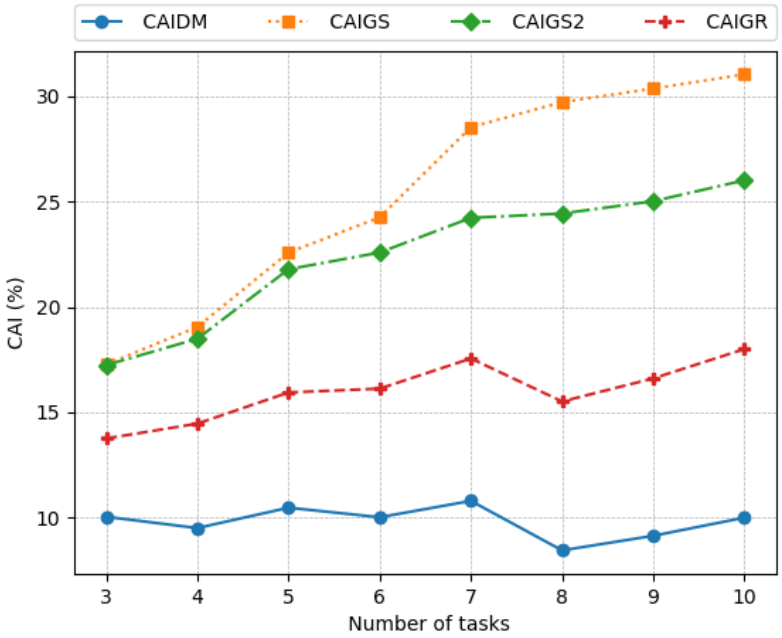


(a) Number of tasks

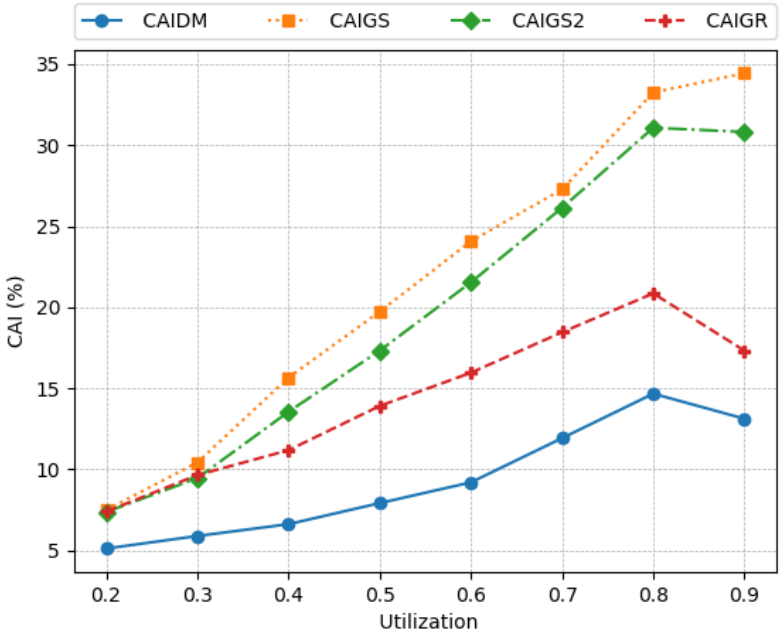


(b) Utilization

Figura 2.14: Influence of the system utilization and number of tasks in the best case response times.



(a) Number of tasks



(b) Utilization

Figura 2.15: Impact of the system utilization and number of tasks in the control activation interval.

2.6.2.2. Influence of the weights of the objectives in the objective function

As stated in Section 2.5.1, the objective function covers the total number of context switches, s, and the total response time, w, for all tasks.

By changing the weights K_1 and K_2, we can derive as many versions of GR as we want, depending on the system requirements. For example, a particular case of GR is GR2, which corresponds to the GR approach with a single objective: to reduce the response time of the system. Therefore, the context switch is removed from the objective function, i.e., $K_1 = 0$ and $K_2 = 1$ in Equation (2.2).

In this subsection, GR and GR2 approaches are compared, and DM is used as a reference in the evaluation of the results.

For this purpose, simulations with hyperperiods in $[0, 3500]$ are made and the solution time is limited to 1800 s. Furthermore, utilization factors are defined in range [0,1, 0,8] and the number of tasks in $[2, 10]$ per set.

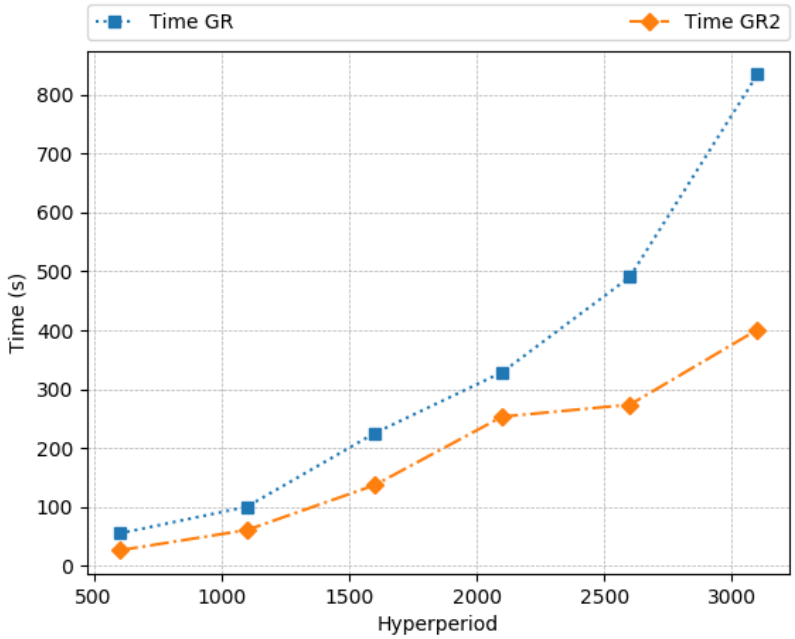
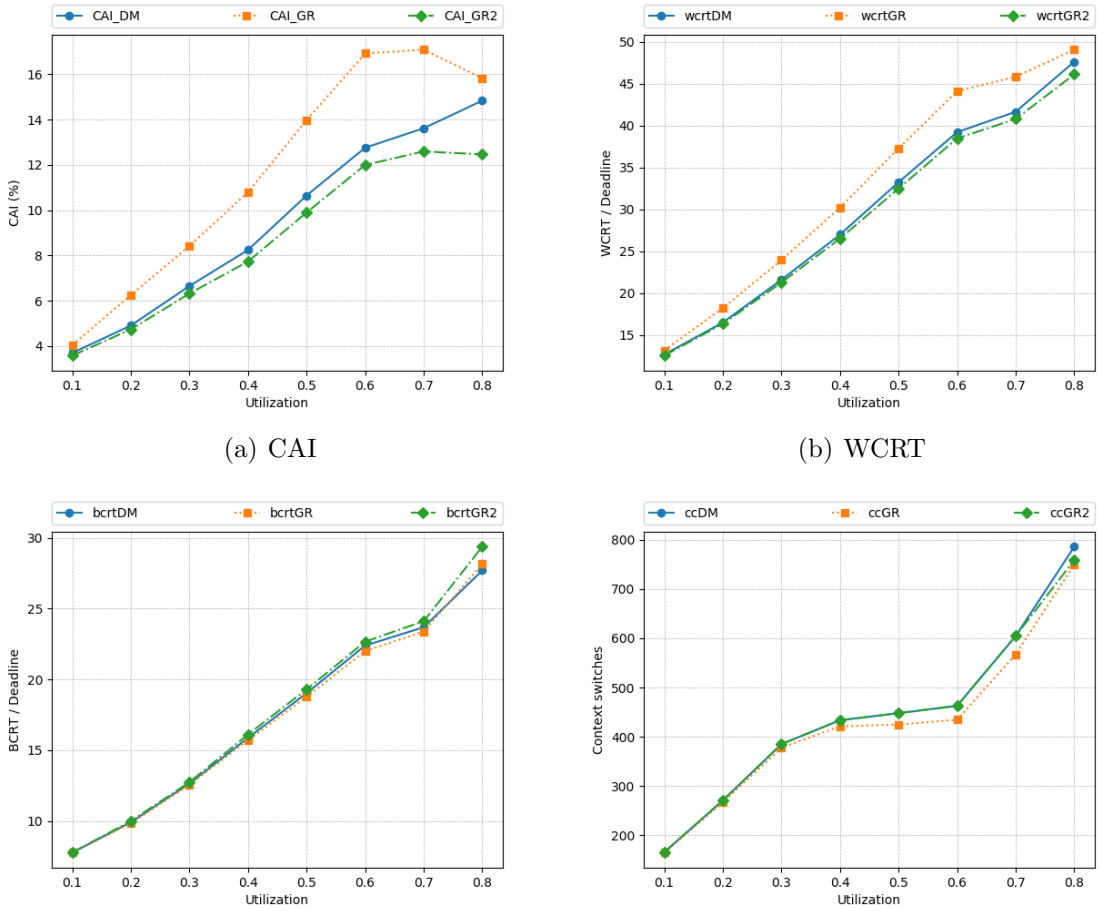


Figura 2.16: Solution time in GR and GR2 approaches depending on the system hyperperiod.

As seen in Fig. 2.16, the solution time grows exponentially for GR and more slowly for GR2 as the objective function has been simplified. Reducing the number of objectives in the objective function significantly reduces the solution time. Looking into Fig. 2.17, we can conclude that the GR2 approach is the most similar approach to the DM approach in terms of system performance. Although GR slightly reduces the number of context switches and BCRT compared to GR2 and DM, it implies worse performance in WCRT and CAI. It is possible to conclude that regarding WCRT, the GR2 approach reduces the response times even below the DM approach, and this is a favorable result.



(a) CAI

(b) WCRT

(c) BCRT

(d) Context switches

Figura 2.17: Performance parameters in GR and GR2 approaches depending on the system utilization.

As discussed, the system's performance parameters change when the objective function varies i.e., considering different terms in the objective function or changing the weights associated with each objective. Obviously, the weights of the objective function should be customized for the needs of the problem. For example, in control applications, it is desirable to minimize the computational delay in the controller, as well as the sampling jitter and the control jitter [2.7]. The implementation of the minimization of computational delay is explained in the next subsection.

2.6.3. Reducing Control Activation Interval

A variant form of the GR algorithm, GR3, is defined here with the CAI and response time in the objective function, i.e., Equation (2.38). The first part and the second parts of the equation minimize the CAI and the task's overall response time, respectively.

$$\text{mín} \quad \text{Obj} = \frac{1}{T_i}(WCRT_i - BCRT_i) + \sum_{\forall j \in N_i} \frac{1}{D_i} w_{ij} \tag{2.38}$$

In this subsection, GR, GR2 and GR3 approaches are compared, and DM is used as a reference in the evaluation of the results. Table 2.4 resumes the objective of each approach.

Tabla 2.4: Summary table of Rolling task approaches

Approach	Objective
GR	Minimize response time and context switches
GR2	Minimize response time
GR3	Minimize CAI and response time
DM	Deadline Monotonic Scheduling

As seen in Fig. 2.18, minimizing the CAI results in the best results among all approaches. GR3 approach provides slightly better results than GR2 in terms of CAI and both, GR2 and GR3 improve the system's performance compared with the DM approach. This is a promising result of the work.

2.6.4. Evaluation of hierarchical MILP model for partitioned systems

For the evaluation of partitioned systems, the proposed hierarchical MILP model has been applied to a real design case from the avionics domain [2.15], described in Table 2.5.

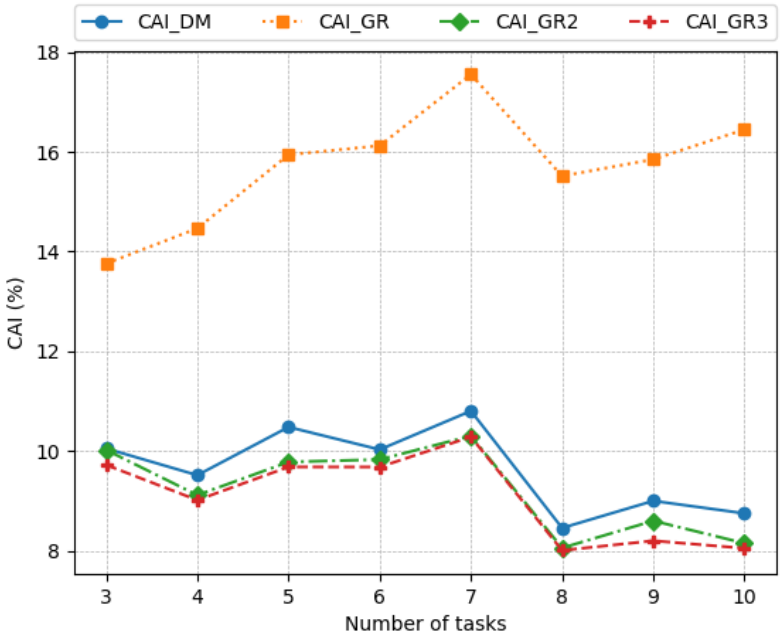


Figura 2.18: CAI for different approaches depending on the number of tasks of the system.

Tabla 2.5: Real design case from the avionics domain

	C_i	D_i	T_i	ρ_i
τ_0	1	25	25	ρ_0
τ_1	3	50	50	ρ_0
τ_2	2	50	50	ρ_1
τ_3	1	50	50	ρ_2
τ_4	1	25	25	ρ_3
τ_5	1	50	50	ρ_3
τ_6	2	100	100	ρ_3
τ_7	5	200	200	ρ_3
τ_8	1	50	50	ρ_4
τ_9	1	50	50	ρ_4

Firstly, the global level periodic resource is calculated as:

$$
\begin{aligned}
U_{P_0} &= \frac{1}{25} + \frac{3}{50} && = 0{,}10 \\
U_{P_1} &= \frac{2}{50} && = 0{,}04 \\
U_{P_2} &= \frac{1}{50} && = 0{,}02 \\
U_{P_3} &= \frac{1}{25} + \frac{1}{50} + \frac{2}{100} + \frac{2}{100} && = 0{,}105 \\
U_{P_4} &= \frac{1}{50} + \frac{1}{50} && = 0{,}04
\end{aligned}
$$

$$
\begin{aligned}
\theta_{P_0} &= 0{,}10 \cdot 25 && = 2{,}5 \\
\theta_{P_1} &= 0{,}04 \cdot 50 && = 2 \\
\theta_{P_2} &= 0{,}02 \cdot 50 && = 1 \\
\theta_{P_3} &= 0{,}105 \cdot 25 && = 2{,}625 \\
\theta_{P_4} &= 0{,}04 \cdot 50 && = 2
\end{aligned}
$$

Therefore, the periodic resources for the partitions are modelled as: $\Gamma_{\rho_0} = (3, 25)$, $\Gamma_{\rho_1} = (2, 50)$, $\Gamma_{\rho_2} = (1, 50)$, $\Gamma_{\rho_3} = (3, 25)$ and $\Gamma_{\rho_4} = (2, 50)$, assuming integer numbers for all parameters. The global scheduling plan that fulfills with this periodic resources is depicted in Fig. 2.19. Note that because of space limitations, the scheduling plan is limited to 130 units of time instead of all the hyperperiod.

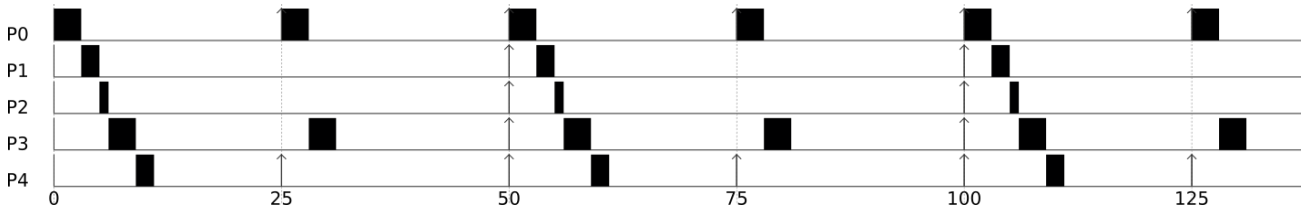


Figura 2.19: Periodic server in the global level for real design case described in Table 2.5.

Fig. 2.20 shows the local scheduling plan according to the rolling MILP approach.

Now, we are going to compare this approach with flat scheduling with DM algorithm [2.8]. Fig. 2.21 depicts the scheduling plan for the case defined in Table 2.5 following DM algorithm.

If the executions of the tasks that belong to the same partition are grouped, the execution in the global level is depicted in Fig.2.22:

Looking at Figs. 2.19 and 2.22, it is obvious that at the global level, hierarchical allocation achieves fewer context switches than flat scheduling. In fact, it is reduced by 35 %. This reduction

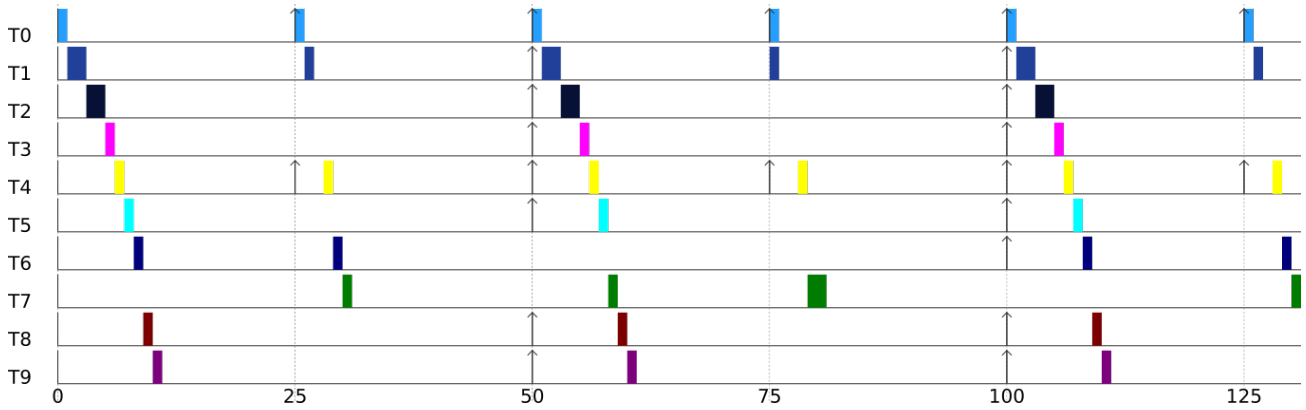


Figura 2.20: Rolling task MILP in the local level for real design case described in Table 2.5.

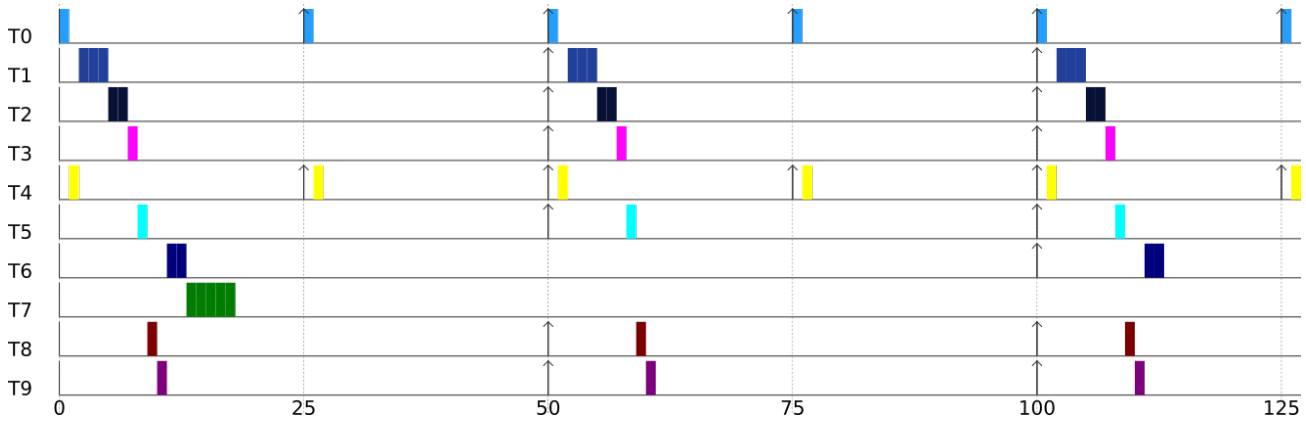


Figura 2.21: DM algorithm for real design case described in Table 2.5.

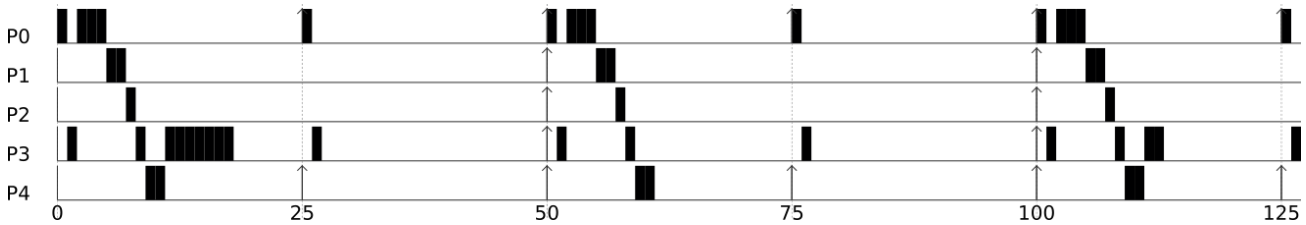


Figura 2.22: Periodic server in the global level for real design case described in Table 2.5.

is due to the fact that in the case of flat scheduling, tasks are scheduled independently of the partition to which they belong. The exact reduction will depend on how the credit θ and the period π in the global level are selected. The criteria to select these values are out of the scope of this paper, but there are several resource-partitioned models such as the regularity-based resource partitioning (RRP) model [2.29], the periodic model [2.21], and the explicit deadline periodic model [2.14] that can be used to calculate θ and π. This evaluation concludes that the hierarchical MILP model can successfully schedule both levels with better results regarding context switches. This is especially important in partitioned systems where there are two types of context switches: between partitions (global level) and between tasks inside a partition (local level).

2.7. Conclusion

We have explored different MILP techniques to schedule uniprocessor hard real-time systems. Our goal was to demonstrate that the MILP technique is an excellent alternative to heuristic scheduling algorithms, even in uniprocessor systems. We have proposed a MILP technique that achieves good performance in solution times and obtains better schedules than heuristics in terms of response times and jitter and generally in any desired performance parameter due to the possibility of customizing the optimization criteria. We have also proposed an MILP formulation for scheduling partitioned systems, considering them as two-level hierarchical systems. Further work involves evolving the approaches into more complex models and architectures, especially multicore architectures with more realistic models such as tasks with precedence relations, mixed-criticality systems, and power consumption reduction.

Bibliografía

[2.1] Avionics Application Software Standard Interface (ARINC-653). PART 1 – REQUIRED SERVICES, March 2006 2006. Airlines Electronic Eng. Committee.

[2.2] Gurobi optimizer reference manual. Inc. Gurobi Optimization, 2019.

[2.3] Baker, T. P., and Shaw, A. The cyclic executive model and ada. In Proceedings. Real-Time Systems Symposium (Dec 1988), pp. 120–129.

[2.4] Baruah, S. Feasibility analysis of preemptive real-time systems upon heterogeneous multiprocessor platforms. In 25th IEEE International Real-Time Systems Symposium (2004), pp. 37–46.

[2.5] Baruah, S. K., Howell, R. R., and Rosier, L. E. Feasibility problems for recurring tasks on one processor. Theoretical Computer Science 118, 1 (1993), 3 – 20.

[2.6] Burns, A., and Davis, R. I. A survey of research into mixed criticality systems. ACM Compututing Surveys 50, 6 (2017).

[2.7] Cervin, A. Improved scheduling of control tasks. In Proceedings of 11th Euromicro Conference on Real-Time Systems. Euromicro RTS'99 (1999), pp. 4–10.

[2.8] Crespo, A., Alonso, A., Marcos, M., de la Puente, J. A., and Balbastre, P. Mixed criticality in control systems. IFAC Proceedings Volumes 47, 3 (2014), 12261 – 12271. 19th IFAC World Congress.

[2.9] Crespo, A., Ripoll, I., and Albertos, P. Reducing delays in rt control: The control action interval. IFAC Proceedings Volumes 32, 2 (1999), 8527 – 8532. 14th IFAC World Congress 1999, Beijing, Chia, 5-9 July.

[2.10] Davis, R. I., and Burns, A. Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. In 2009 30th IEEE Real-Time Systems Symposium (Dec 2009), pp. 398–409.

[2.11] Devaraj, R., Sarkar, A., and Biswas, S. Real-time scheduling of non-preemptive sporadic tasks on uniprocessor systems using supervisory control of timed des. In 2017 American Control Conference (ACC) (2017), pp. 3212–3217.

[2.12] Devaraj, R., Sarkar, A., and Biswas, S. Exact task completion time aware real-time scheduling based on supervisory control theory of timed des. In 2018 European Control Conference (ECC) (2018), pp. 1908–1913.

[2.13] Di Natale, M., and Zeng, H. An efficient formulation of the real-time feasibility region for design optimization. IEEE Transactions on Computers 62 (04 2013), 644–661.

[2.14] Easwaran, A., Anand, M., and Lee, I. Compositional analysis framework using edp resource models. In 28th IEEE International Real-Time Systems Symposium (RTSS 2007) (2007), pp. 129–138.

[2.15] Easwaran, A., Lee, I., Sokolsky, O., and Vestal, S. A compositional framework for avionics (arinc-653) systems. Tech Report MS- CIS-09-04, University of Pennsylvania. (01 2009).

[2.16] Escoffier, B., Bonifaci, V., and Ausiello, G. Complexity and approximation in reoptimization. Computability in Context: Computation and Logic in the Real World (02 2011).

[2.17] Fleming, T., and Burns, A. Investigating mixed criticality cyclic executive schedule generation. In Proc. Workshop on Mixed Criticality (WMC) (2015).

[2.18] Harter, Jr., P. K. Response times in level-structured systems. ACM Trans. Comput. Syst. 5, 3 (Aug. 1987), 232–248.

[2.19] Henties, T., Ag, S., Hunt, J., Locke, D., Nilsen, K., Na, A., Schoeberl, M., and Vitek, J. Java for safety-critical applications. Electronic Notes in Theoretical Computer Science - ENTCS (01 2009).

[2.20] Hong, I., Kirovski, D., Gang Qu, Potkonjak, M., and Srivastava, M. B. Power optimization of variable-voltage core-based systems. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 18, 12 (1999), 1702–1714.

[2.21] Insik Shin, and Insup Lee. Periodic resource model for compositional real-time guarantees. In RTSS 2003. 24th IEEE Real-Time Systems Symposium, 2003 (2003), pp. 2–13.

[2.22] Jeffay, K. Scheduling sporadic tasks with shared resources in hard-real-time systems. In [1992] Proceedings Real-Time Systems Symposium (1992), pp. 89–99.

[2.23] Joseph, M., and Pandya, P. Finding response times in a real-time system. The Computer Journal 29, 5 (1986), 390–395.

[2.24] Kim, J., Oh, H., Ha, H., Kang, S.-H., Choi, J., and Ha, S. An ilp-based worst-case performance analysis technique for distributed real-time embedded systems. pp. 363–372.

[2.25] Leung, J. Y.-T., and Whitehead, J. On the complexity of fixed-priority scheduling of periodic, real-time tasks. Performance Evaluation 2, 4 (1982), 237 – 250.

[2.26] Lisper, B., and Mellgren, P. Response-time calculation and priority assignment with integer programming methods.

[2.27] Locke, C. D. Software architecture for hard real-time applications: Cyclic executives vs. fixed priority executives. Real-Time Syst. 4, 1 (Mar. 1992), 37–53.

[2.28] Mangeruca, L., Baleani, M., Ferrari, A., and Sangiovanni-Vincentelli, A. Uniprocessor scheduling under precedence constraints for embedded systems design. ACM Trans. Embed. Comput. Syst. 7, 1 (Dec. 2007).

[2.29] Mok, A. K., and Xiang Alex. Towards compositionality in real-time resource partitioning based on regularity bounds. In Proceedings 22nd IEEE Real-Time Systems Symposium (RTSS 2001) (Cat. No.01PR1420) (2001), pp. 129–138.

[2.30] Nguyen, V. A., Hardy, D., and Puaut, I. Cache-conscious off-line real-time scheduling for multi-core platforms: algorithms and implementation. Real-Time Systems 55, 4 (2019), 810–849.

[2.31] Nuth, P. R., and Dally, W. J. A mechanism for efficient context switching. In [1991 Proceedings] IEEE International Conference on Computer Design: VLSI in Computers and Processors (1991), pp. 301–304.

[2.32] Paul, A. A., and S. Pillai, B. A. Reducing the number of context switches in real time systems. In 2011 International Conference on Process Automation, Control and Computing (2011).

[2.33] Rivera-Verduzco, H. J., and Bril, R. J. Best-case response times of real-time tasks under fixed-priority scheduling with preemption thresholds. In Proceedings of the 25th International Conference on Real-Time Networks and Systems (New York, NY, USA, 2017), RTNS '17, Association for Computing Machinery, p. 307–346.

[2.34] Sun, Y., and Natale, M. D. Weakly hard schedulability analysis for fixed priority scheduling of periodic real-time tasks. ACM Trans. Embed. Comput. Syst. 16, 5s (Sept. 2017).

[2.35] Tompkins, M. F. Optimization techniques for task allocation and scheduling in distributed multi-agent operations. PhD thesis, Massachusetts Institute of Technology, Dept. of Electrical Engineering and Computer Science, 2003.

[2.36] Zamorano, J., Alonso, A., and [de la Puente], J. Building safety-critical real-time systems with reusable cyclic executives. Control Engineering Practice 5, 7 (1997), 999 – 1005.

Capítulo 3

Artículo: Hardware resources contention-aware scheduling of hard real-time multiprocessor systems

3.1. Abstract

In hard real-time embedded systems, switching to multicores is a step that most application domains delay as much as possible. This is mainly due to the number of sources of indeterminism, which mainly involve shared hardware resources, such as buses, caches, and memories. In this paper, a new task model that considers the interference that task execution causes in other tasks running on other cores due to memory contention is proposed. We propose a scheduling algorithm that calculates the exact interference. We also analyse and compare existing partitioning algorithms and propose three strategies to allocate tasks to cores to schedule as many tasks as possible and minimise total interference.

3.2. Introduction

The release of the first dual-core processor in 2001 began a migration of computing platforms from single-cores to multicore architectures. In response to the increased use of multicore processors, the Certification Authorities Software Team (CAST) published Position Paper CAST-32A named 'Multi-core Processors' [3.4]. This paper identifies topics that could impact the safety, performance and integrity of airborne software systems executing on multicore processors and provides objectives intended to guide the production of safe multicore avionic systems. For example, objective MCP_Software_1 requires that evidence is produced to demonstrate that all hosted software components function correctly and have sufficient time to complete their execution when operating in their multicore environment. In many domains such as avionics, space, or industrial control systems, hard real-time constraints, safety and security issues, and certification assurance levels are commonly required.

Hard real-time multiprocessor systems are commonly implemented using partitioned sche-

duling rather than global scheduling. In the partitioned approach, the tasks are statically partitioned among the processors, i.e., each task is assigned to a processor and is always executed on it. Static assignment is key to meeting the temporal requirements needed by certification authorities. Nevertheless, multiprocessor systems add many sources of indeterminism. Processors may contend for shared resources, such as memory. During the lifetime of a piece of software several processors may reach a part of the programs that lead to heavy bus loads and each access that the processors attempt may collide with accesses generated by another processor. It is necessary to analyse software images in the context of a multi-core system and not just analyse the software when it is running without contenders [3.5].

The timing behaviour of a task in a multicore system is affected not only by the software running on it and its inputs, but also by contention for resources such as buses, caches, and GPUs that are shared with tasks running on other cores. This contention causes interference in the timing behaviour of the task [3.1]. We can define the interference as a delay on the expected execution time of a set of tasks in a multicore system due to the contention produced for simultaneous accesses to the shared hardware.

Different techniques can be implemented to deal with these sources of indeterminism. The worst-case interference can be estimated and added to the worst-case execution time (WCET) of the task. This results in an over-estimated and very pessimistic model. Moreover, this value depends on the execution of other cores and so it is difficult to find a worst case. We can apply techniques to reduce or enhance the predictability of the interference due to memory contention. Our work is in the middle: we consider worst-case interference, but we do not assume that it is produced for every task. We propose a scheduling algorithm that counts the exact interference while assuming that the interference a task produces for other tasks due to contention is bounded. Based on the implementation of the scheduling algorithm we deduce a way to allocate tasks to cores so that we group tasks into cores to produce as little interference as possible.

It is important to note that the measurement about the time that a task spends on accessing shared hardware resources (i.e., reading and writing memory operations) is outside the scope of this paper. Considerable research has been done in this area. We assume that this is a value previously measured for the used hardware, and so we do not restrict our work to a specific type of hardware resource contention.

We centre our efforts on dealing with the limitations and requirements of the software, and it is important to note that, according to [3.13], the configuration of the software architecture can reduce the interference delay, and this means that indeterminism will be reduced as the number interruptions is reduced. On previous referenced work, interference is reduced with temporal and spatial isolation. But in our work, we centre on allocation and scheduling tasks in cores.

In summary, the main contributions of this paper are the definition of a task model that includes the interference due to contention for shared hardware resources, allocation strategies to reduce this interference, and a scheduling algorithm for the proposed model.

The rest of the paper is organised as follows: Section 3.3 presents the relevant works in partitioned multicore systems scheduling. In Section 3.4, the system model is described and the main contributions are highlighted. Section 3.5 describes the contention-aware scheduling algorithm, while in Section 3.6 three allocation algorithms are proposed. The evaluation of the

proposals is presented in Section 3.7, while conclusions and further work are given in Section 3.8.

3.3. Related works

Scheduling for multicore platforms was the subject of many research works, surveyed in [3.11]. As we know, in multi-core scheduling there are two main branches, partitioned scheduling, and global scheduling. In this work we are focusing on partitioned scheduling. To generate a schedule plan in a multicore partitioned system, the following steps are defined:

- Allocation of tasks to cores.
- Perform the schedule generation for each core.

Since our work is framed around hard real-time systems, we will focus on the state of the art of partitioned allocation (where) and static scheduling (when).

The task allocation problem is analogous to the bin packing problem and is known to be NP-hard in the strong sense [3.15].

A number of heuristics are available for solving the bin packing problem. Some of the most well-known [3.21] [3.6] are:

- First fit (FF). Each item is allocated into the first bin that it fits into without exceeding the maximum capacity of the bin. If there is no bin available, a new bin is opened.
- Best fit (BF). This algorithm allocates each item into the fullest bin where it fits, and as with FF, possibly opening a new bin if the item does not fit into any currently open bin.
- Worst fit (WF). WF allocates each item into the bin that leaves most remaining capacity, that is, the emptiest bin. It will also open a new bin if no bin is available to allocate the item.

In addition to these heuristics, there are other bin packing algorithms used to solve the allocation problem. Coffman et al. [3.7] presents a survey and classification of these algorithms. In [3.12] a state-of-the-art about contention delays is There are some works that try to reduce contention delays by using specific task models. The predictable execution model (PREM) is introduced in [3.22] which splits a task into a read communication phase and an execute phase. A similar technique is used in [3.25] that calculates task scheduling and contentions with the objective of minimising the schedule makespan by letting the technique decide when it is necessary to avoid or consider interference. The shared bus is arbitrated using a round-robin policy and the task model considers a DAG (direct acyclic graph) to separately read, execute, and write operations so the WCET of the execute phase can be measured in isolation. We do not split tasks and so our model is a classical periodic task model. For DAG task models, [3.14] proposes a scheduling method that applies the LET (Logical Execution Time) paradigm and considers communication timing between nodes to reduce contention. Other approaches, such as the one presented in [3.24], try to reduce interference costs using synchronisation-based

interference models and appropriate memory allocation schemes. In [3.9], a feedback control scheme is proposed to ensure the execution of critical cores in a mixed-criticality partitioned system. The controller limits at hypervisor level the use of the memory bus of non-critical cores when they reach a limit. Performance monitor counters are used to establish the number of bus accesses. In [3.3], Casini et al. propose an analysis of memory contention where an optimisation problem is formulated to bound the memory interference by leveraging a three-phase execution model and holistically considering multiple memory transactions issued during each phase. One of the first papers that introduced the interference parameter in the temporal model is presented in [3.13]. In this paper, for a partitioned system in the aerospace domain, the WCRA parameter is defined (Worst Case number of shared Resource Accesses). This value is added to the WCET and this results in a predictable but pessimistic scheduling plan. Similarly, the concept of interference-sensitive Worst-Case Execution Time (isWCET) is proposed in [3.20]. In [3.26] a dynamic approach is presented that safely adapts isWCET schedules during execution by relaxing or completely removing isWCET schedule dependencies (depending on the progress of each core). The concept of isWCET is similar to our work but the proposals are centred around minimising the effect of interference with new scheduling methods while our work focuses on allocation algorithms. The work lines of [3.16] and [3.17] deserve mention. In these approaches, interference is analysed in a similar way to our work, as it is also represented as a parameter. However, their work is based on the interference produced by the DRAM memory as a shared resource in a multicore system while our proposal is agnostic with respect to the hardware resource used. Their work also considers only fixed priority scheduling while our proposal can be used with fixed and dynamic priorities. Inpresented.

Previous algorithms are sensitive to the order of the items. For example, if lightweight items are allocated first, accommodating large items in the gaps they leave is a difficult task. There are several methods to order the items before allocating them into cores. The most used technique consists of ordering the items according to their weight, i.e., utilisation and decreasing utilisation is one of the main variants. The decreasing utilisation method (DU) puts the items in decreasing order by utilisation. In this way, WF, BF, and FF become WFDU (worst fit decreasing utilisation), BFDU (best fit decreasing utilisation) and FFDU (first fit decreasing utilisation).

All the previous results are highly theoretical and do not consider delays due to hardware resource contention. In [3.10] a deep study of the sources of unpredictability is analysed under two categories: primary sources (caches, FSB, memory, and memory controller); and secondary sources (hardware-prefetching, power saving strategies, translation look-aside buffer, misses, system management interrupts). This topic is also analysed in [3.19] where the sources of timing interference in single-core, multicore, and distributed systems are presented. As stated in this paper, memory interference can jeopardise system feasibility. It is shown in [3.23] that there are cases where memory interference can cause a worst case response time increase as high as 300 %, even for tasks that spend only 10 % of their time fetching memory in an eight-core system.

There are some works that try to reduce contention delays by using specific task models. The predictable execution model (PREM) is introduced in [3.22] which splits a task into a read communication phase and an execute phase. A similar technique is used in [3.25] that calculates task scheduling and contentions with the objective of minimising the schedule makespan by

letting the technique decide when it is necessary to avoid or consider interference. The shared bus is arbitrated using a round-robin policy and the task model considers a DAG (direct acyclic graph) to separately read, execute, and write operations so the WCET of the execute phase can be measured in isolation. We do not split tasks and so our model is a classical periodic task model. For DAG task models, [3.14] proposes a scheduling method that applies the LET (Logical Execution Time) paradigm and considers communication timing between nodes to reduce contention.

Other approaches, such as the one presented in [3.24], try to reduce interference costs using synchronisation-based interference models and appropriate memory allocation schemes.

In [3.9], a feedback control scheme is proposed to ensure the execution of critical cores in a mixed-criticality partitioned system. The controller limits at hypervisor level the use of the memory bus of non-critical cores when they reach a limit. Performance monitor counters are used to establish the number of bus accesses. In [3.3], Casini et al. propose an analysis of memory contention where an optimisation problem is formulated to bound the memory interference by leveraging a three-phase execution model and holistically considering multiple memory transactions issued during each phase.

One of the first papers that introduced the interference parameter in the temporal model is presented in [3.13]. In this paper, for a partitioned system in the aerospace domain, the WCRA parameter is defined (Worst Case number of shared Resource Accesses). This value is added to the WCET and this results in a predictable but pessimistic scheduling plan. Similarly, the concept of interference-sensitive Worst-Case Execution Time (isWCET) is proposed in [3.20]. In [3.26] a dynamic approach is presented that safely adapts isWCET schedules during execution by relaxing or completely removing isWCET schedule dependencies (depending on the progress of each core). The concept of isWCET is similar to our work but the proposals are centred around minimising the effect of interference with new scheduling methods while our work focuses on allocation algorithms.

The work lines of [3.16] and [3.17] deserve mention. In these approaches, interference is analysed in a similar way to our work, as it is also represented as a parameter. However, their work is based on the interference produced by the DRAM memory as a shared resource in a multicore system while our proposal is agnostic with respect to the hardware resource used. Their work also considers only fixed priority scheduling while our proposal can be used with fixed and dynamic priorities. In [3.17] memory interference is reduced by partitioning DRAM banks and by a BestFit-based allocation algorithm. Clearly, a model that considers a specific hardware resource and a specific scheduling algorithm will further reduce contention delays, but our aim is to provide a more general model that can be used for any shared hardware.

3.4. Task model and contributions

3.4.1. Periodic task model

In a multicore system, there is a set of m homogeneous cores M_1,...,M_m that enable different tasks to be executed at the same time. Each core has allocated N_k tasks. We assume that migration is not allowed so we will follow the partitioned approach. In a hard real-time system,

there is a set of n independent real-time tasks $\tau = [\tau_1, ..., \tau_n]$, where each task generates a set of jobs $(\tau_{ij}, 1 \leq i \leq n, j \geq 0)$ that must be completed before the due time. This work considers that all tasks are periodic and characterised by $\tau_i = (C_i, D_i, T_i, I_i)$, where C_i is the WCET, D_i is the deadline, T_i is the period and I_i is the worst-case interference time (explained in detail in 3.4.2). We assume a constrained deadline task model ($D_i \leq T_i$) and periodic or sporadic tasks.

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. Any task τ_i has A_i activations throughout H. Therefore, $A_i = \frac{H}{T_i}$.

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: $U_{M_k} = \sum_{\tau_i \in M_k} U_i$. The total utilisation of the system is the sum of the utilisation of all cores: $U_\tau = \sum_m U_{M_k}$.

3.4.2. Worst case interference time

In multicore systems, unlike monocore, the resources of the system are shared by different tasks at the same time and different cores may simultaneously need a particular resource, such as buses or memory because of the nature of the process they are executing. At this point, we can assert that during an execution of a multicore system there will probably be a contention or interference between different cores that will delay the expected execution of the processes. As remarked in [3.12] from contention to access to shared resources arises the interference effect between various tasks that are allocated and executed on different processors.

I_i is the worst-case time that τ_i uses for reading and writing memory operations. It is illustrated in Figure 3.1, where we can see that all the time that τ_0 spends on reading and writing operations is part of the interference parameter I_0. Task execution times are depicted in solid rectangles while interference is depicted in dashed rectangles. From the point of view of other tasks, I_i is the extra time that τ_i produces in other tasks executing at the same time on all other cores due to contention. For example, let's suppose that $\tau_i = \{10, 50, 75, 3\}$, this would mean that the computation time of τ_i is 10 time units, of which 3 are dedicated to memory accesses. In the worst case, we can assume that all the tasks running on other cores at the same time as τ_i will be delayed 3 units in their execution.

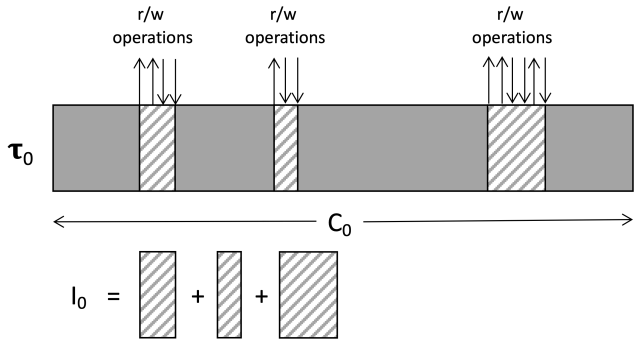


Figura 3.1: Example of task interference.

Figure 3.1 shows the interference parameter from the point of view of the task, and this interference implies a delay in the execution of tasks in other cores. Note that I_0 is represented as a whole piece when it affects other tasks while in Figure 3.1 it is represented as separate pieces. From now on, the interference will always be represented as a whole piece since we will investigate the effect it has on other tasks in other cores. The following example illustrates this effect.

Example. Consider a set of three tasks τ_0, τ_1, and τ_2 allocated on a platform with three cores as shown in Figure 3.2. Suppose that $I_0 = I_2 = 1$ and $I_1 = 0$, that is, τ_0 and τ_2 are the tasks that share the resources, so they provoke and receive interference. However, τ_1 does not use the resources and so suffers no interference. Every time that τ_0 is executed at the same time as τ_2, both tasks increase their computation times: τ_0 increases I_2 units due to the interference caused by τ_2 while τ_2 increases I_0 units due to the interference caused by τ_0.

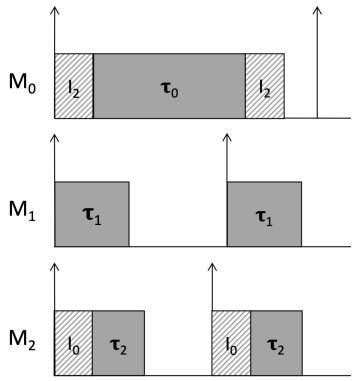


Figura 3.2: Example of influence of interference on scheduling.

From Figure 3.2, it is deduced that if contention is considered, the total utilisation of a task depends on its computation time and period, as well as the interference received from other tasks. Moreover, this interference does not have to be considered in all activations, and only in those where there is execution on both cores. Therefore, we define the equivalent real values for U_i, U_{M_k} and U_τ:

$$U_i' = U_i + U_i^{int} \tag{3.1}$$

being U_i^{int} the utilisation due to the interference caused by other tasks to τ_i.

Expressing U_i' with respect to the hyperperiod, we obtain:

$$U_i' = \frac{A_i C_i}{H} + \frac{I_i^T}{H} \tag{3.2}$$

being I_i^T the total interference that τ_i receives due to the execution of contenders in other cores.

In the same way, the real utilisation of a core M_k is:

$$U'_{M_k} = \sum_{\tau_i \in M_k} U_i' \tag{3.3}$$

And for the total real utilisation of task set τ:

$$U'_\tau = \sum_m U'_{M_k} \tag{3.4}$$

3.4.3. Contributions

As far as the authors are aware there is no scheduling algorithm that schedules a system like the algorithm presented in this paper where the interference is calculated at its exact value.

With this model, the interference plays a key role in determining the schedulability of multicore systems. In addition, we will use the knowledge of interference to propose new allocation algorithms and compare them with existing ones. Therefore, the main contributions of this work are:

- Definition of a task model that considers interference delays due to contention of shared hardware resources.
- Proposal of a new scheduling algorithm that copes with the new task model. This scheduling algorithm is formulated to be included in any priority-based scheduling algorithm for monocores.
- Proposal of three allocation algorithms and comparison with existing ones in terms of schedulability and real utilisation of the system.

3.5. Contention aware scheduling algorithm

This section describes the rules that a scheduling algorithm must follow to consider the exact interference that a task suffers in each activation. It is important to note that this scheduler does not try to reduce interference. We will do this in the allocation of tasks to cores in Section 3.6.

We initially need some definitions:

Definition 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 in other cores.*

Definition 2. *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 in other cores due to contention.*

If $I_i = 0$, τ_i is neither a broadcasting nor a 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 cores whose $I_j > 0$.

Interference is produced whenever two broadcasting tasks run at the same time in different cores. The instant in which an interference may occur is when one of the two following situations occurs:

- A receiving task τ_i is released. In this moment, active broadcasting tasks in other cores cause interference for τ_i.
- A broadcasting task τ_j is released. In this moment, τ_j causes interference for active receiving tasks in other cores.

Moreover, as a task can receive interference from more than one task (if there are more than two cores), and in different instants of time, it will be necessary to record the interference produced by each task in a matrix.

Definition 3. *Let W be a binary matrix of $n \times n \times H$. At each instant t the value of W_{ijt} indicates whether τ_i provokes interference for τ_j or not, in the following way:*

- $W_{ijt} = 1$*: there is interference.*
- $W_{ijt} = 0$*: there is not interference.*

This matrix will be used to establish when a task τ_i must increase its computation time because of the interference caused by other tasks running in different cores.

Property 1. *If two tasks τ_i and τ_j are allocated to the same core M_k then W_{ijt}=0 and W_{jit}=0 for all t=0,..,H.*

Property 2. *If a task τ_i has $I_i = 0$ then W_{ijt}=0 and W_{jit}=0 for all t=0,..,H and for all j= $1, .., n$.*

We are going to illustrate the behaviour of W with an example. Let us consider the following task set, $\tau = [\tau_0, \tau_1]$ with $\tau_0 = (1, 3, 3, 1)$ and $\tau_1 = (2, 5, 5, 1)$, allocated in a dual-core system, τ_0 is allocated in M_0 and τ_1, in M_1.

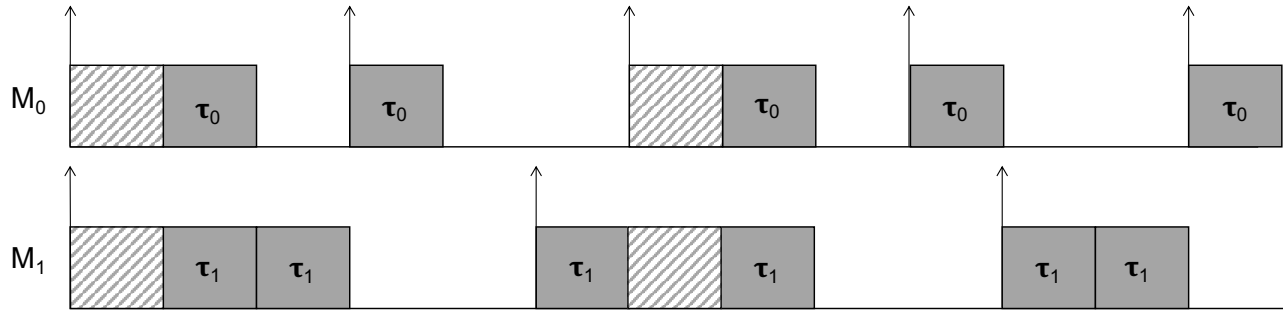


Figura 3.3: Execution chronogram of the example

Figure 3.3 shows the resulting rate monotonic [3.18] scheduling plan of the system that also considers interference due to memory contention. Rate monotonic is a static fixed-priority scheduling algorithm in which the priority of a task is inversely proportional to its period. As can be seen in the figure, there is an increase in the execution times of τ_0 and τ_1 by I_1 and

I_0 whenever other tasks are released. If τ_0 or τ_1 releases when the other task is not active, interference is not added.

The values of W for each time instant t are depicted in Figure 3.4. As is pointed in the comments, whenever there is a release of one of the tasks while the other is still active, the corresponding value of the matrix changes from 0 to 1. When a task τ_i finishes its activation, all the elements of W in the i-th file are 0. At each instant t, if W_{ijt} is 1, this means that τ_i is active at instant t and has caused an interference for τ_j.

t	W	Comment
0	$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$	τ_0 and τ_1 release
1	$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$	
2	$\begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}$	τ_0 finishes its first activation
3	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$	τ_1 finishes its first activation
4	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$	
5	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$	τ_1 releases, but τ_0 is not active
6	$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$	τ_0 releases while τ_1 is active
7	$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$	
8	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$	τ_0 and τ_1 finish
9->15	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$	τ_0 and τ_1 are not active at the same time

Figura 3.4: W values for the example

Looking at Figure 3.3, we see that the total interference suffered by τ_0 is 2, while the interference suffered by τ_1 throughout H, is also 2. Therefore, the real utilisation of each core is:

$$U'_{M_0} = \frac{C_0}{T_0} + \frac{I_0^T}{H} = \frac{1}{3} + \frac{2}{15} = 0{,}46$$

$$U'_{M_1} = \frac{C_1}{T_1} + \frac{I_1^T}{H} = \frac{2}{5} + \frac{2}{15} = 0{,}53$$

As a rule, from the study of W we can see that an interference is caused at time t when W_{ijt} changes from 0 to 1, that is, when W_{ijt}-$W_{ij(t-1)}$= 1 for all tasks τ_j not allocated in the same core as τ_i.

In this way, besides accounting for the total interference received I_i^T we can define and calculate this value for each activation, that is, the contention-aware execution time:

Theorem 1. *The contention-aware execution time C'_{is} of τ_i in activation s is the sum of C_i plus the interferences caused by running tasks in other cores and can be calculated as:*

$$C'_{is} = C_i + \sum_{\tau_j \notin M_k} \left(W_{jisT_i} + \sum_{t=sT_i+1}^{t=(s+1)T_i-1} \text{máx}\left(W_{jit+1} - W_{jit}, 0\right) \right) \cdot I_j \tag{3.5}$$

Demostración. In the interval of the s-th activation, $(sT_i, (s+1)T_i - 1)$ the number of interferences caused by a broadcasting task τ_j is equal to the number of times that W_{ji} changes from 0 to 1 from instant t to $t+1$, multiplied by I_j. If the change is the other way round, that is, from 1 to 0, then there is no interference. In this way, we must only consider low-to-high transitions of the W matrix. This is expressed in the term máx $(W_{jit+1} - W_{jit}, 0)$. The term W_{jisT_i} accounts for the interference at the initial instant in which τ_i releases. □

The relation between I_i^T and C'_{is} is then:

$$I_i^T = \sum_s (C'_{is} - C_i) \quad \forall s = 0, .., A_i - 1 \tag{3.6}$$

The interference matrix W is used to establish the exact computation time that a task should execute at each activation to consider its worst-case execution time C_i and the time added by the contention C'_{is}. This matrix must be calculated at each instant t as part of the online scheduling algorithm.

Listing 3.1 shows the pseudo-code (Python-like) of a priority (fixed or dynamic) scheduling algorithm, while Listing 3.2 shows the pseudo-code with the modifications needed to calculate C'_{is} added to the previous algorithm.

Listing 3.1: Priority based scheduling algorithm for m processors

```
1  #variables definition and initialisation
2  for t in range(H):
3      for k in range(m):
4          runningTask[k] = HigherPriority(M_k)
5      for k in range(m):
6          currentTask[k] = runningTask[k]
7          #account for execution
8          finished = Execute(currentTask[k], k)
9          if finished:
10             C'_i = C_i
```

Listing 3.2: Contention-aware scheduling algorithm

```
1  #variables definition and initialisation
2  for t in range(H):
3      for k in range(m):
4          runningTask[k] = HigherPriority(M_k)
5      for k in range(m):
6          τ_i = runningTask[k]
7          if τ_i != currentTask[k] && t % T_i == 0:
8              for s in range(m):
9                  if s != k and I_i > 0:
10                     τ_j = runningTask[s]
11                     W[j][i] = 1
12                     C'_i += I_j
13         else:
14             for s in range(m):
15                 if s != k and I_i > 0:
16                     τ_j = runningTask[s]
17                     if W[j][i] == 0:
18                         W[j][i] = 1
19                         C'_i += I_j
20     for k in range(m):
21         currentTask[k] = runningTask[k]
22         #account for execution
23         finished = Execute(currentTask[k], k)
24         if finished:
25             C'_i = C_i
26             for j in range(n):
27                 W[j][i] = 0
28                 W[i][j] = 0
```

The algorithm first selects the task to run on each core according to the selected algorithm. In this case, the task with the highest priority is chosen for each core (line 4). For each core, the condition to add the interference is then evaluated (line 7). If the task selected to run (τ_i) is different from the previous task (context switch) and τ_i has released in time t, then all the running tasks in the rest of the cores change from 0 to 1 the value in i column to indicate that these tasks cause interference for τ_i (line 11). C'_i then increases its value by the interference of the other cores running tasks (line 12). If there is not a context switch or the running task τ_i has resumed from preemption (line 13), it is possible that in the meantime (from preemption to resume) some tasks in other cores have been released, and so their interference must be considered. This can be seen if W_{ij} is 0, which means that some task τ_j has been released in another core while τ_i was preempted. In this case, the interference is added (lines 18 and 19) and recorded in W_{ij}. All the tasks are executed once all the cores have been checked for interference. Those that finish the execution of their activation, set their corresponding row (as a broadcasting task) and column (as a receiving task) in W to 0 (lines 27 and 28). Note that

it is not necessary to define W as a three-dimensional matrix (t dimension is removed) as the calculation of $C'_i s$ is done on the fly. The same happens with C'_{is}. This reduces notably the overhead of the algorithm.

After the completion of the scheduling on H, I_i^T is calculated for each task and, so can compute the real utilisation of the system U'_τ.

In terms of computational complexity, this algorithm consists of three main loops. The first one is inherited from Listing 3.1 and it corresponds with the complexity of RM or EDF algorithm by m cores. The second loop has computational complexity of $O(m^2)$. The third loop has computational complexity of $O(m \cdot n)$. Assuming that there are more tasks than cores, $n > m$ and then, the computational complexity of algorithm of Listing 3.2 is $O(m \cdot n)$.

3.6. Task allocation algorithms

Prior to scheduling, it is necessary to allocate tasks to cores. As commented in Section 3.3, different allocation heuristics for partitioned multiprocessor systems exist. These have the goal of maximising the number of tasks to allocate while assuring feasibility. However, with our model, the real utilisation of the core is no longer U_{M_k} but U'_{M_k} because the real utilisation of tasks U'_i is increased by the interferences.

The total interference received for each task I_i^T (and, equivalently, C'_{is} in each activation) is difficult to estimate since it depends on the scheduling of the tasks allocated to each core. For this reason, it is not obvious to estimate in advance I_i^T or C'_{is} to calculate the real utilisation per core, so we can know before scheduling whether a task allocation is going to be schedulable or not.

The goal of this section is to study W_{ijt} to derive allocation algorithms that try to take into account the possible interference generated and received so that real utilisation (U'_{M_k}) is decreased while ensuring feasibility.

At any instant t, the interference that a task τ_i allocated in core M_k causes all other tasks on other cores to look at their corresponding i row of W in which values are 1 and multiply it by I_i. To reduce the interference caused, we are interested in having as many zeros as possible in the W matrix. The elements of row i that are always zero are:

- when $j = i$
- when for j column, τ_j is allocated to the same core than τ_i
- when $I_i = 0$.

Therefore, a bound for the maximum interference that a core M_k can receive is:

$$maxW_k = \sum_{\substack{\tau_i \in M_k \\ I_i \neq 0}} \sum_{\tau_j \notin M_k} I_j \tag{3.7}$$

This value can not be reached at an instant because two tasks on the same core can not be active at the same time. But this bound can be reached in an interval $[s \cdot T_i, (s+1) \cdot T_i]$.

Finally, assuming that all values of W that can be 1 are indeed 1, we have that the maximum value of the sum of all elements of W is:

$$maxW = \sum_{\forall k} maxW_k = \sum_{k=1}^{m} \sum_{\substack{\tau_i \in M_k \\ I_i \neq 0}} \sum_{\tau_j \notin M_k} I_j \tag{3.8}$$

This is not a bound of $\sum_j I_j^T$ since this matrix changes as t changes. The total value of the interference can be greater if, for example, two activations of τ_j interfere with one activation of τ_i.

From the previous analysis of the W matrix we can intuitively sense that a task allocation algorithm that unbalances the load between cores, or that minimises Equation 3.8, will tend to have less real utilisation than an allocation algorithm that balances the load between cores. However, a task allocation algorithm that balances the load between cores will tend to schedule more task sets that an unbalanced algorithm.

An unbalanced load is defined as a load that tries to allocate the maximum number of tasks on the minimum number of cores is low. One approach to exactly measure how balanced a system is with respect to utilisation is by using the so-called utilisation discrepancy, that we define as follows [3.8]:

Definition 4. *The utilisation discrepancy UD_τ can be defined as the difference between the maximum and minimum utilisation of a multicore system.*

$$UD_\tau = \max_{k=0}^{m-1}(U_{M_k}) - \min_{k=0}^{m-1}(U_{M_k}) \tag{3.9}$$

To understand this concept, let's suppose the following task set: τ_0, τ_1, τ_2 and τ_3 with their corresponding utilisations: U_0, U_1, U_2, U_3, and two cores, M_0 and M_1. We do not provide all the temporal parameters because they are not relevant for this example. Figure 3.5 represents tasks and cores. Grey boxes are dimensioned depending on the task utilisation while cores are dimensioned with white boxes with a total length of one (maximum utilisation that can be reached by the core).

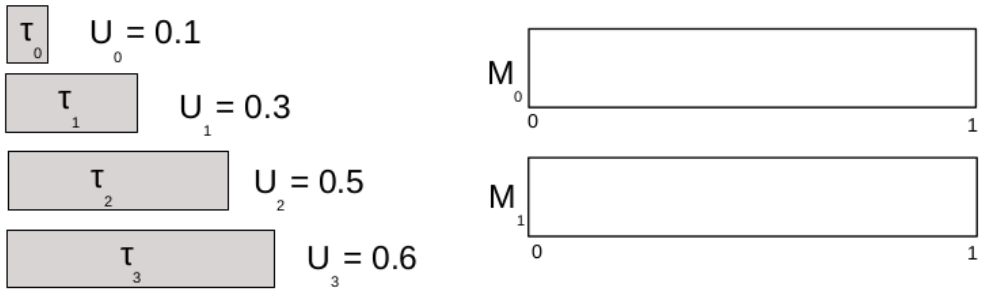


Figura 3.5: Task set and available cores in a system before allocation

We then make the allocation process and assign every task a core. We have different options to make the allocation, for example, we could assign τ_0 and τ_2 to M_0 and τ_1 and τ_3 to M_1. In this case, $U_{M_0} = 0{,}6$ and $U_{M_1} = 0{,}9$. Therefore, according to the definition of discrepancy, in this allocation case, the discrepancy would be 0.3. Figure 3.6 shows the resulting allocation.

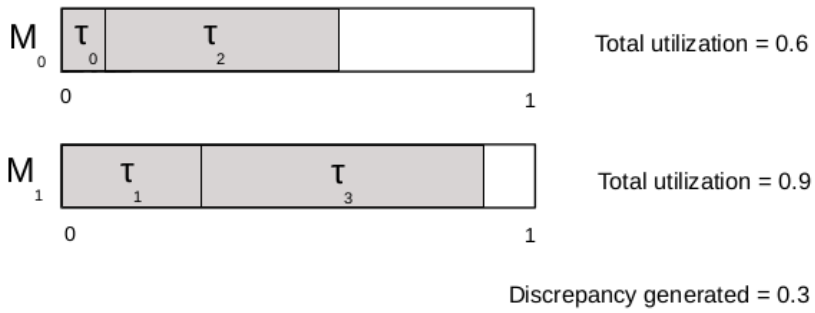


Figura 3.6: Allocation of 4 tasks in 2 cores

When tasks are allocated to different cores, the discrepancy would be maximised if there is no other combination that produces a greater discrepancy than this one. And in the opposite way, the discrepancy would be minimised if the task allocation produces the minimum possible discrepancy value. In this case, we can say that the cores have a balanced utilisation task load, and an unbalanced load on the contrary.

For the reasons explained above, we can derive new allocation algorithms for our model. We will propose three new allocation algorithms:

- Maximise utilisation discrepancy, UD_τ. From now on, this algorithm is called UDmax.
- Minimise utilisation discrepancy, UD_τ. From now on, this algorithm is called UDmin.
- Minimise Equation 3.8. From now on, this algorithm is called Wmin.

Below, we will describe the three allocation algorithms. We will check if the previous observations hold in Section 3.7. The allocators will be implemented as an integer programming formulation.

3.6.1. UDmin and UDmax allocators

Because of their similarities, we will describe UDmin and UDmax together. For that, we define a set of parameters and variables shown in Table 3.1, that also defines the parameters and variables for Wmin.

Both allocators have the same constraints, but the objective is to minimise or maximise the discrepancy.

We have that for UDmin the objective is:

$$\text{Minimise} \quad UD_\tau \tag{3.10}$$

Tabla 3.1: Model notation

SETS AND INDICES	
i	Tasks $\tau_i \in \{0, 1, 2, ..., n-1\}$
k	Cores $M_k \in \{0, 1, 2, ..., m-1\}$
PARAMETERS	
C_i	Worst case execution time of τ_i
T_i	Period of τ_i
U_i	Theoretical utilisation of τ_i
I_i	Interference factor of τ_i over other tasks
DECISION VARIABLES	
UDmin and UDmax models	
O_{ik}	Allocation matrix. 1 if τ_i allocated in M_k, 0 otherwise
U_{M_k}	Theoretical utilisation of core k.
UD_τ	Utilisation discrepancy of the task set τ.
Wmin model	
O_{ik}	Allocation matrix. 1 if τ_i is allocated in core k and 0 otherwise.
U_{M_k}	Theoretical utilisation of core k.
$maxWk$	Maximum value of the sum of all elements of W for core k.
$maxW$	Maximum value of the sum of all elements of W for all cores.

and for UDmax the objective is:

$$\text{Maximise} \quad UD_\tau \tag{3.11}$$

s.t:

$$\sum_{\forall k} O_{ik} = 1 \qquad \forall i \tag{3.12}$$

$$\sum_{i \in k} U_i \cdot O_{ik} = U_{M_k} \qquad \forall k \tag{3.13}$$

$$U_{M_k} \leq 1 \qquad \forall k \tag{3.14}$$

$$UD_\tau = \max_{k=0}^{m-1}(U_{M_k}) - \min_{k=0}^{m-1}(U_{M_k}) \tag{3.15}$$

$$O_{ik} \in \{0, 1\} \tag{3.16}$$

$$U_{M_k} \geq 0 \tag{3.17}$$

The model constraints are defined in equations (3.12), (3.13), (3.14) and (3.15). In constraint (3.12), we ensure that a task is allocated in one and only one core. The total utilisation per core is calculated as the sum of the task utilisations that belong to that core (Equation 3.13) and is less than or equal to 1 (Equation 3.14). In constraint (3.15) we calculate the utilisation discrepancy as the difference between the biggest and lowest task utilisation from all cores. Consequently, the objective function is to minimise the discrepancy in the case of UDmin and maximise the discrepancy in the case of UDmax. Finally, equations (3.16) and (3.17) represent the decision variable domains.

Constraint 3.15 is not linear but is easily expressed with this formulation that avoids a large set of linear and special-ordered set constraints, plus a number of auxiliary decision variables. It is directly supported by the solver API by performing the transformation to a corresponding Mixed Integer Programming formulation automatically and transparently during the solution process. As this constraint is not linear and the model has binary and integer variables, it is a Mixed Integer Non-Linear Programming (MINLP) problem.

3.6.2. Wmin allocator

To formulate the Wmin allocator through integer programming, let us also use the notation in Table 3.1. The objective function of this allocator is to minimise Equation (3.8).

$$\text{Minimise } maxW = \sum_{\forall k} maxW_k \tag{3.18}$$

s.t:

$$\sum_{\forall k} O_{ik} = 1 \qquad \forall i \tag{3.19}$$

$$\sum_{i \in k} U_i \cdot O_{ik} = U_{M_k} \qquad \forall k \tag{3.20}$$

$$U_{M_k} \leq 1 \qquad \forall k \tag{3.21}$$

$$\sum_{\substack{\tau_i \in M_k \\ I_i \neq 0}} \sum_{\tau_j \notin M_k} I_j = maxW_k \quad \forall k \tag{3.22}$$

$$O_{ik} \in \{0, 1\} \tag{3.23}$$

$$U_{M_k}, maxW_{_k} \geq 0 \tag{3.24}$$

Constraints defined in equations (6.20), (6.21), and (3.21) were explained in Section 3.6.1 and define the capacity of each core.

Equation (3.22) calculates the maximum interference provoked by all cores, following Equation (3.8) previously defined. Equations (3.23) and (3.24) represent the decision variable domains.

As the model has binary and integer variables, it is a Mixed Integer Linear Programming (MILP) problem.

3.7. Evaluation

3.7.1. Experimental conditions

The simulation scenario developed for this work is depicted in Figure 5.6. It is divided into five steps:

- Generation of the load (see Section 3.7.1.1).
- Allocation (see Section 3.6).
- Validation of the allocation phase (see Section 3.7.1.2).
- Scheduling (see Section 3.7.1.3).
- Validation of the scheduling phase (see Section 3.7.1.4).

The automatic load generator generates a task set with the process described in Section 3.7.1.1. This task set is the input for the six allocators: existing FFDU, BFDU, and WFDU (which correspond with the FF, BF and WF heuristics described in Section 3.3 and in which the items are ordered according to decreasing utilisation DU) and UDmin, UDmax, and Wmin proposed in this work. The result of the allocations is then evaluated to check their feasibility. The feasible allocations and the task set are the input for the scheduler, that generates the

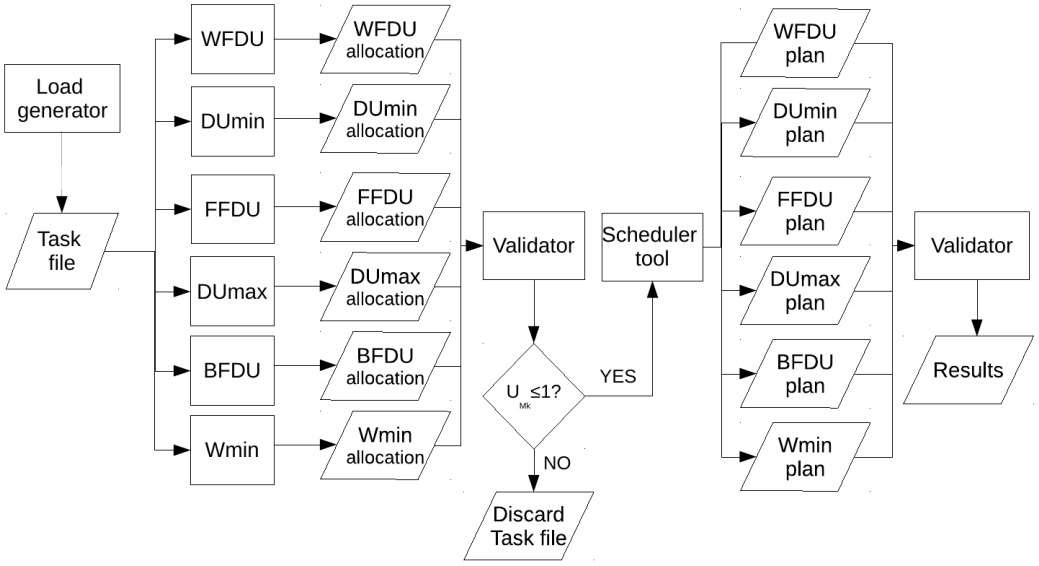


Figura 3.7: Experimental evaluation overview.

six scheduling plans. If they are schedulable, their performance parameters will be stored. This sequence is repeated to complete enough simulations.

We use the Gurobi optimiser 9.0 [3.2], from Gurobi Optimization, Inc., which is a powerful optimiser designed from scratch to run in multi-core and with the capability to run in parallel mode. It achieved performance improvements with each version and provides a Python interface. Since version 9.0, Gurobi can solve non-convex quadratic optimisation problems and also general constraints such as those described in Section 3.6.1.

All allocators described in previous sections are executed on an Intel Core i7 CPU with 16GB of RAM.

3.7.1.1. Load generator

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 they are allocated. As these experiments are conducted in 2, 4, 8, and 10 cores, we set a reasonable number of tasks and a feasible load for each number of cores.

Given the system utilisation value and the number of tasks for each set, the utilisation is shared among the tasks using the UUniFast discard algorithm [3.11]. Periods are generated randomly in [20,1000] and computation times are deduced from the system utilisation. Without loss of generality, deadlines are set to be equal to periods, although they could be constrained to be less than or equal to periods.

Table 3.2 defines the experimental parameters selected for the evaluation process. The total utilisation for each task set depends on the number of cores in which the set is allocated to and is equal to 50 percent of the maximum load of all cores. For example, the maximum load in a system with four cores is four ($U_\tau \leq m$), i.e., 400 %. Therefore, the load of a task set allocated

	Experimental parameters			
Number of cores	2	4	8	10
Theoretical utilisation	1	2	4	5
Number of tasks	4	12	20	28
Number of broadcasting tasks	2	3	5	7

Tabla 3.2: Experimental parameters.

to four cores is set to two. The number of broadcasting tasks is set to 25 % of the number of tasks in each set and $I_i = 1 \, \forall i$, being τ_i a broadcasting task. In other words, the extra time that τ_i produces for other tasks executing at the same time on all other cores due to contention is equal to one unit of time. As a reminder, if a task is broadcasting, it will interfere with the tasks allocated in other cores (τ_j) if their coefficient $I_j \neq 0$. The more broadcasting tasks, the greater the interference that may be produced. Note that in the case of 2 cores, the percentage of broadcasting tasks is set to 50 %. This is due to the fact that, if only 25 % is considered, only one task is broadcasting and then $U'_\tau = U_\tau$.

3.7.1.2. Validation of the allocation phase

The first validation phase consists of checking if all tasks have been allocated to cores and ensuring that the maximum capacity per core is not exceeded i.e. $U_{M_k} \leq 1 \quad \forall k = 0, .., m-1$. If any of the allocators cannot allocate the task set, this task set is discarded and a new one is generated. As we assume implicit deadlines for this evaluation, the previous condition is sufficient. In any event, we do not use it as a condition since U_τ is not the real utilisation of the system and will be increased by the interference in the scheduling phase. Therefore, in our evaluation, constrained deadlines can be used without loss of generality.

3.7.1.3. Scheduling phase

In this phase, the contention aware scheduling algorithm proposed in Section 3.7.1.3 is executed independently for all cores for the six allocations obtained in the allocation phase. In this work, the EDF scheduling algorithm [3.18] is used and, therefore, the task with the highest priority will be the task with the shortest relative deadline. Note that any other priority based algorithm is applicable (by implementing it in line 4 of Listing 3.2). As an output of this phase, U'_τ is obtained since the algorithm in Listing 3.2 obtains the exact interference.

3.7.1.4. Validation of the results

The validation of the scheduling plans involves two steps. Firstly, we must check feasibility to ensure that all deadlines are met in the hyperperiod. Secondly, some performance parameters are obtained to compare different methods. Specifically, we obtain the relation between the theoretical utilisation of the system and the real utilisation of the system for each set, measured

after the scheduling phase. Moreover, once the evaluation of all sets has finished, the parameters that must be evaluated are:

- Schedulability ratio. The percentage of task sets with feasible scheduling plans over total task sets with feasible allocations.
- Increased utilisation. The increase in utilisation with respect to the theoretical utilisation. This is measured as $1 - \sum_k \frac{U_{M_k}}{U'_{M_k}} = 1 - \frac{U_\tau}{U'_\tau}$.

Previous parameters are evaluated for a certain number of cores and also for a certain percentage of broadcasting tasks.

3.7.2. Experimental results

The selection of the initial utilisation of the system and the number of broadcasting tasks does affect the final outcomes. For example, in a set with four tasks allocated to a dual-core system in which the initial utilisation is under but near to two and three of the tasks are broadcasting, it is highly probable that the increase in utilisation due to interference makes the system infeasible. In this section, the results obtained in the evaluation are commented.

Figure 3.8 depicts the results of the experimental evaluation following the parameters in Table 3.2 for different numbers of cores. Figure 3.8 (left) shows the percentage of schedulable sets as a function of the number of cores in the system. As seen in the figure, as the number of cores increases, the schedulability ratio decreases. Allocators such as FFDU, BFDU, or UDmax minimise the number of used cores and, therefore, the load per core is high. The increase in load due to interference means that the cores with high utilisation cannot feasibly schedule the tasks and so the system becomes infeasible. Moreover, the greater the number of cores in the system, the more interference is produced. Therefore, unbalanced cores (FFDU, BFDU, or UDmax allocators) become infeasible when the number of cores increases. Likewise, balanced cores with similar loads (UDmin and WFDU allocators) can afford the increase of utilisation due to interference and the percentage of schedulability is almost 100 % for all experiments. The Wmin allocator presents very good schedulability ratios, even in those systems with a lot of cores.

Figure 3.8 (right) represents the increased utilisation of the system. As previously, the more cores in the system, the more interference is produced and, therefore, the more overloaded the cores are in comparison with the initial utilisation. WFDU, UDmin, and Wmin represent this behaviour. However, allocators as FFDU, UDmax, or BFDU differ from the previous allocators, especially in the case of ten cores. This is because they possess very low schedulability ratios (if any) and the increase in utilisation is calculated from a small feasible sample. We observe that, in the case of two cores, the overload is greater than in the case of four cores. This is because in two cores, the percentage of broadcasting tasks is 50 % and in other cases it is 25 % (see Section 3.7.1.1). Again, the more broadcasting tasks, the greater the utilisation.

From Figure 3.8 we can conclude that there is not an allocator that dominates the others in both schedulability and increased utilisation. The allocators that balance the load, such as UDmin and WFDU, almost always ensure the schedulability of the sets at the expense of the

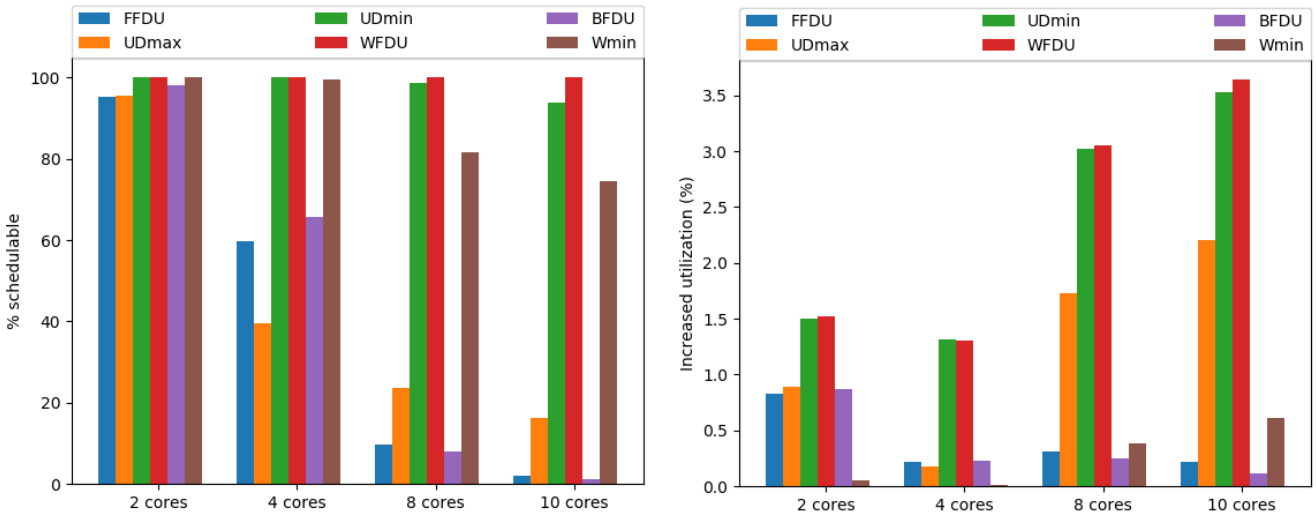


Figura 3.8: Experimental values of schedulability and increases in utilisation as a function of the number of cores and allocators: Percentage of schedulable task sets depending on the number of cores and allocators (left). Increased utilisation depending on the number of cores and allocators (right).

increase in load (about 3.5 % in the case of ten cores). Allocators with unbalanced loads, such as FFDU, BFDU, and UDmax, cannot ensure schedulability, especially for a large number of cores, but the increase in utilisation is less than UDmin and WFDU. However, Wmin presents a good ratio of schedulability and a small increase in utilisation in comparison with other allocators.

In all the cases, the percentage of schedulability decreases with the number of cores (and consequently, with the number of produced interferences) and the utilisation increases with the number of cores.

Figure 3.9 depicts the schedulability ratio and the increase in utilisation as a function of the allocators. It is calculated as the average of the values represented in Figure 3.8. From these figures we can conclude that FFDU, BFDU, or UDmax allocators achieved up to 43 % of schedulability with a small increase in system utilisation due to interference. However, WFDU or UDmin achieve almost 100 % of schedulability at the expense of a 2.3 % increase in utilisation. Previous allocators work in an opposite way: the former achieve an unbalanced load, which reduces the feasibility when interference appears, and the latter ensures feasibility but system utilisation increases. Finally, the Wmin allocator provides a high schedulability ratio (up to 89 %) with an increase in utilisation of only 0.266 %.

Figure 3.10 shows the influence of the number of broadcasting tasks in the schedulability ratio and the increase of utilisation, for each allocator evaluated in this work and following the evaluation parameters in Table 3.2. From Figure 3.10 (left) we can conclude that the schedulability ratio generally decreases with the number of broadcasting tasks. Only WFDU and UDmin achieve 100 % schedulability for the whole range of broadcasting tasks. FFDU, BFDU, and UDmax are the allocators with the lowest schedulability ratio.

In contrast, from Figure 3.10 (right) we can deduce that the more broadcasting tasks, the

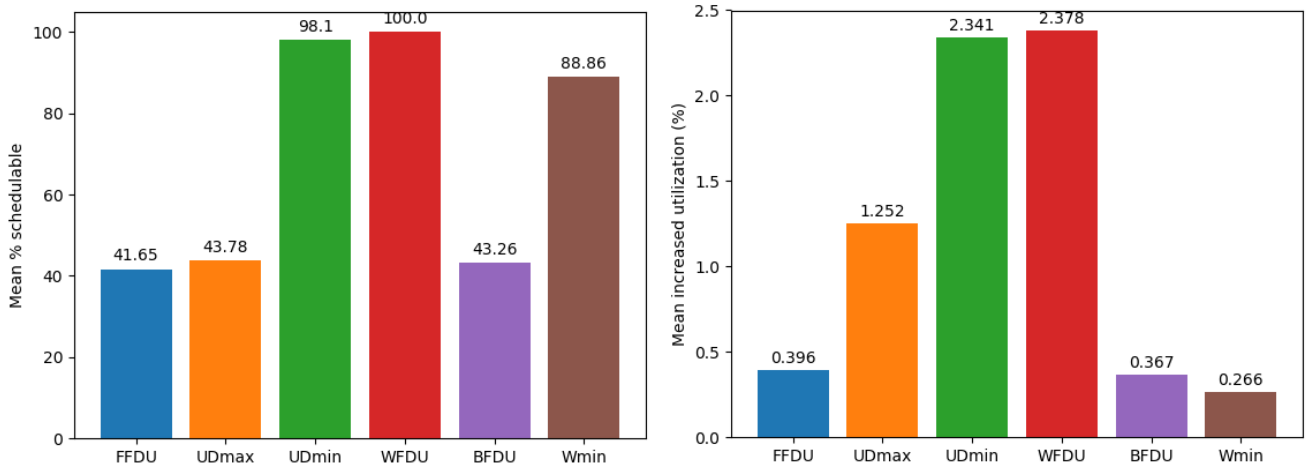


Figura 3.9: Experimental values of schedulability and increase in utilisation as a function of the allocators: Percentage of schedulable task sets depending on the allocators (left). Increased utilisation depending on the allocators (right).

greater is system utilisation. This is a common behaviour for all allocators. WFDU and UDmin are the allocators with the highest increase in utilisation, while FFDU and BFDU reveal the lowest increases. Wmin always presents an intermediate behaviour, i.e., its schedulability decreases depending on the number of broadcasting tasks and utilisation increases up to 0.6 % (which are considerable values in comparison with other allocators).

Therefore, it can be deduced that increasing the number of broadcasting tasks produces an increase in system utilisation and a reduction in the schedulability ratio for all allocators except for WFDU, and this ensures schedulability in the studied range of broadcasting tasks. With a major number of broadcasting tasks, WFDU will also reduce its schedulability but we can-not observe this behaviour in the studied range of broadcasting tasks.

Finally, the solution times for the proposed MILP approaches are measured and depicted in Figure 3.11 (note that the y-axis is represented on a logarithmic scale and the solution time values are included in the graph for ease of analysis and neatness). For this evaluation, we have conducted experiments with the experimental values described in Table 3.2.

Figure 3.11 shows that, as the number of cores increases, the solution time increases. This is because the more cores in the system, the more tasks and broadcasting tasks are considered. UDmin is the approach that takes longest to calculate the solution. This is because the algorithm tries to share the load as much as possible. On the contrary, UDmax unbalances the load, that is, at least one of the cores will remain empty (if possible) and at least one will be full, and so the discrepancy is maximised. Obtaining this solution is faster than obtaining a balanced solution. The Wmin allocator is again an intermediate proposal.

We can conclude that if a precise adjustment in terms of discrepancy is required, the UDmax and UDmin approaches are the most suitable solutions. In the remaining cases, their analogue heuristics FFDU, BFDU, and WFDU provide similar and faster solutions. As before, the Wmin

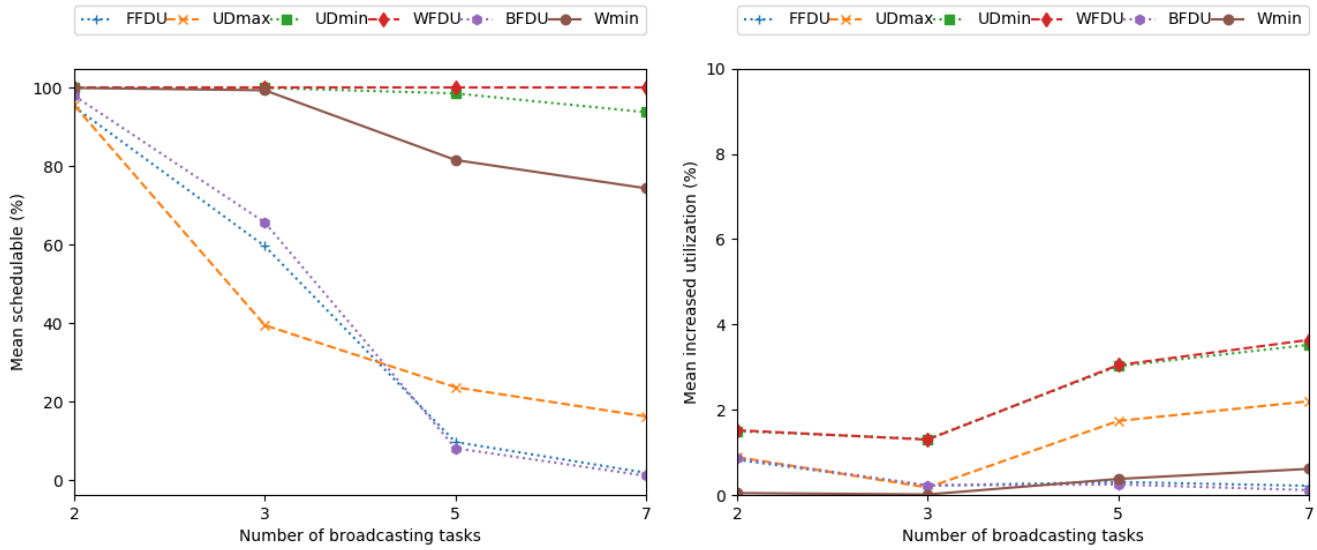


Figura 3.10: Experimental values of schedulability and increment of utilisation as a function of the number of broadcasting tasks: Percentage of schedulable task sets (left). Increased utilisation (right).

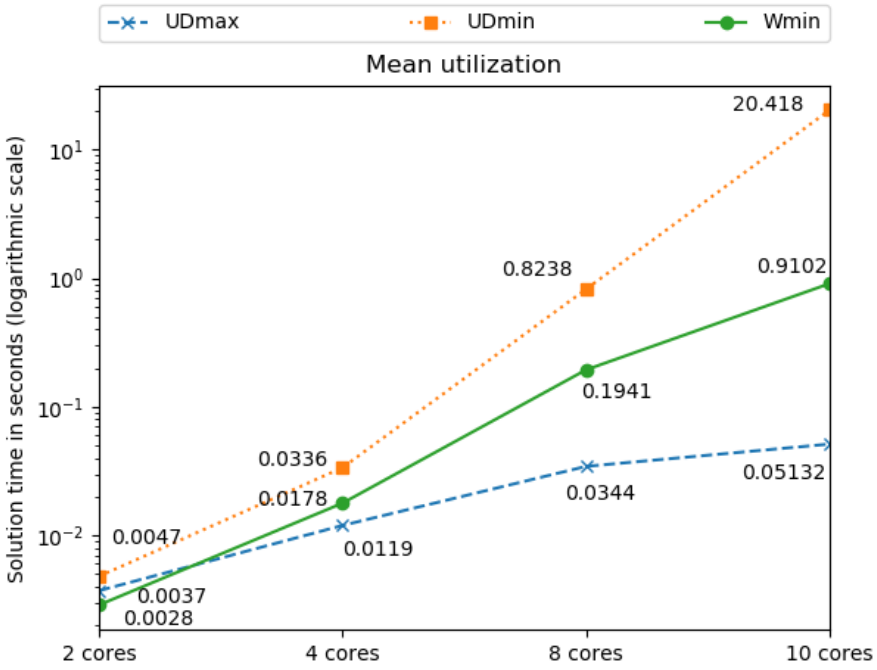


Figura 3.11: Solution time for MILP approaches.

approach provides intermediate solution times.

3.8. Conclusions

This paper has proposed a new task model that considers the delay produced by the contention of hardware shared resources in hard real-time multiprocessor systems. Along with the new model, a scheduling algorithm that considers the exact interference produced for each task is proposed. Until now, the problem was solved by significantly increasing the WCET to cope with the worst case. Our proposal is not so pessimistic without jeopardising feasibility. Moreover, three allocators have been proposed and compared with well-known existing allocators.

According to the experimental evaluation, we can conclude that FFDU, BFDU and UDmax allocator algorithms behave similarly, and UDmin and WFDU allocator algorithms also show similar behaviour. In the case of FFDU, BFDU and UDmax allocators, we can conclude that they reach a low rate of increase in utilisation (which is an advantage), but on the other hand, their rates of schedulability are very low. They may be the best choice for two core architectures.

The opposite case occurs with UDmin and WFDU allocators. They present an excellent rate of schedulability, but they are greatly affected by interference and so if a system needs to prioritise a low utilisation rate then UDmin and WFDU are not the best options. Wmin is clearly an intermediate option because it shows non-extreme rates in increases of utilisation and schedulability. Hence, according to the needs and requirements of a system, Wmin may be a suitable option.

We plan to further investigate the schedulability of task sets with worst-case interference time parameters by deriving an utilisation bound and proposing new scheduling algorithms to decrease interference.

Bibliografía

[3.1] Multicore Timing Analysis for DO-178C. `https://www.rapitasystems.com/downloads/multicore-timing-analysis-do-178c`. Rapita Systems.

[3.2] Gurobi optimizer reference manual. Inc. Gurobi Optimization, 2019.

[3.3] Casini, D., Biondi, A., Nelissen, G., and Buttazzo, G. A holistic memory contention analysis for parallel real-time tasks under partitioned scheduling. In 2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS) (2020), pp. 239–252.

[3.4] (CAST), C. A. S. T. Multi-core Processors—Position Paper CAST-32A, November 2016. Technical Report.

[3.5] Cobham. Multi-core software considerations. Available at `https://www.gaisler.com/doc/antn/GRLIB-AN-0005.pdf` (2015/10/28), 2015.

[3.6] Coffman, E. G., Garey, M. R., and Johnson, D. S. Approximation Algorithms for Bin Packing: A Survey. PWS Publishing Co., USA, 1996, p. 46–93.

[3.7] Coffman Jr., E. G., Csirik, J., Galambos, G., Martello, S., and Vigo, D. Bin Packing Approximation Algorithms: Survey and Classification. Springer New York, New York, NY, 2013, pp. 455–531.

[3.8] Crespo, A., Balbastre, P., Simo, J., and Albertos, P. Static scheduling generation for multicore partitioned systems. In Information Science and Applications (ICISA) 2016 (Singapore, 2016), K. J. Kim and N. Joukov, Eds., Springer Singapore, pp. 511–522.

[3.9] Crespo, A., Balbastre, P., Simó, J., Coronel, J., Gracia Pérez, D., and Bonnot, P. Hypervisor-based multicore feedback control of mixed-criticality systems. IEEE Access 6 (2018), 50627–50640.

[3.10] Dasari, D., Akesson, B., Nélis, V., Awan, M. A., and Petters, S. M. Identifying the sources of unpredictability in cots-based multicore systems. In 2013 8th IEEE International Symposium on Industrial Embedded Systems (SIES) (2013), pp. 39–48.

[3.11] Davis, R. I., and Burns, A. Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. In 2009 30th IEEE Real-Time Systems Symposium (Dec 2009), pp. 398–409.

[3.12] Fernandez, G., Abella, J., Quiñones, E., Rochange, C., Vardanega, T., and Cazorla, F. Contention in multicore hardware shared resources: Understanding of the state of the art. In WCET (2014).

[3.13] Galizzi, J., Vigeant, F., Perraud, L., Crespo, A., Masmano, M., Carrascosa, E., Brocal, V., Balbastre, P., Quartier, F., and Milhorat, F. Wcet and multicores with tsp. In DASIA 2014 DAta Systems In Aerospace (2014).

[3.14] Igarashi, S., Ishigooka, T., Horiguchi, T., Koike, R., and Azumi, T. Heuristic contention-free scheduling algorithm for multi-core processor using let model. In 2020 IEEE/ACM 24th International Symposium on Distributed Simulation and Real Time Applications (DS-RT) (2020), pp. 1–10.

[3.15] Johnson, D. Near-Optimal Bin Packing Algorithms. PhD thesis, Massachusetts Institute of Technology, Dept. of Math., 1973.

[3.16] Kim, H., de Niz, D., Andersson, B., Klein, M., Mutlu, O., and Rajkumar, R. Bounding memory interference delay in cots-based multi-core systems. In 2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS) (2014), pp. 145–154.

[3.17] Kim, H., De Niz, D., Andersson, B., Klein, M., Mutlu, O., and Rajkumar, R. Bounding and reducing memory interference in cots-based multi-core systems. Real-Time Syst. 52, 3 (May 2016), 356–395.

[3.18] Liu, C. L., and Layland, J. W. Scheduling algorithms for multiprogramming in a hard-real-time environment. J. ACM 20, 1 (Jan. 1973), 46–61.

[3.19] Mitra, T., Teich, J., and Thiele, L. Time-critical systems design: A survey. IEEE Design Test 35, 2 (2018), 8–26.

[3.20] Nowotsch, J. Interference-sensitive Worst-case Execution Time Analysis for Multi-core Processors. PhD thesis, November 2014.

[3.21] Oh, Y., and Son, S. H. Allocating fixed-priority periodic tasks on multiprocessor systems. Real-Time Syst. 9, 3 (Nov. 1995), 207–239.

[3.22] Pellizzoni, R., Betti, E., Bak, S., Yao, G., Criswell, J., Caccamo, M., and Kegley, R. A predictable execution model for cots-based embedded systems. In 2011 17th IEEE Real-Time and Embedded Technology and Applications Symposium (2011), pp. 269–279.

[3.23] Pellizzoni, R., Schranzhofer, A., Jian-Jia Chen, Caccamo, M., and Thiele, L. Worst case delay analysis for memory interference in multicore systems. In 2010 Design, Automation Test in Europe Conference Exhibition (DATE 2010) (2010), pp. 741–746.

[3.24] Reder, S., and Becker, J. Interference-aware memory allocation for real-time multi-core systems. 2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS) (2020), 148–159.

[3.25] Rouxel, B., Derrien, S., and Puaut, I. Tightening contention delays while scheduling parallel applications on multi-core architectures. ACM Trans. Embed. Comput. Syst. 16, 5s (Sept. 2017).

[3.26] Skalistis, S., and Kritikakou, A. Dynamic Interference-Sensitive Run-time Adaptation of Time-Triggered Schedules. In ECRTS 2020 - 32nd Euromicro Conference on Real-Time Systems (July 2020), pp. 1–22.

Capítulo 4

Artículo: Interference-Aware Schedulability Analysis and Task Allocation for Multicore Hard Real-Time Systems

4.1. Abstract

There has been a trend towards using multicore platforms for real-time embedded systems due to their high computing performance. In the scheduling of a multicore hard real-time system, there are interference delays due to contention of shared hardware resources. The main sources of interference are memory, cache memory and the shared memory bus. These interferences are a great source of unpredictability and they are not always taken into account. Recent papers have proposed task models and schedulability algorithms to account for this interference delay. The aim of this paper is to provide a schedulability analysis for a task model that incorporates interference delay, for both fixed and dynamic priorities. We assume an implicit deadline task model. 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. An allocation algorithm to minimise this interference (Imin) is also proposed and compared with existing allocators. The results show how Imin has the best rates in terms of percentages of schedulability and increased utilisation. In addition, Imin presents good results in terms of solution times.

4.2. Introduction

The use of embedded systems is nowadays spreading at an increasing speed, to all aspects of modern life as well as all phases of industrial production. The processing capability of multicore systems permits multiple embedded applications on a single shared hardware platform. Nevertheless, multicore systems add many sources of indeterminism, leading to a number of

execution delays. This sources of indeterminism mainly involve shared hardware resources, such as buses, caches, and memories. It is necessary to analyse the temporal model in the context of a multicore system and not just when it is running without contenders [4.14]. Specifically, the interference that appears when cores contend for these shared resources. In addition, in highly critical systems, the static plan must take the interference into account. If not, the system's feasibility may be jeopardized.

When this interference is taken into account, the model and the schedulability analysis is often limited to a particular type of hardware resource and even a particular type of memory.

Recently, in [4.2], a new task model is proposed that takes into account the interference of hardware shared resources in the context of multicore hard real-time systems. This model is general for any kind of hardware resource. However, there is no schedulability test for this new task model. This interference delay can be large and highly variable, posing a major challenge to the schedulability of real-time critical multicore systems.

Contribution. This paper proposes a schedulability test for multicore real-time systems for the model presented [4.2] since in this work they did not present any schedulability analysis.

An allocation algorithm to minimise this interference is also proposed and compared with existing allocators. Our proposal is valid for both fixed and dynamic priorities.

The novelty of the contribution is the consideration of dynamic priority scheduling in a model that considers interference due to shared hardware resources. 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. [4.2].

The paper is organized as follows: Section 4.3 presents the relevant works in partitioned multicore systems scheduling. In Section 4.4, the system model used is presented. Section 4.5 presents the contention aware utilisation factor that is the basis for the schedulability analysis in Section 4.6. In Section 4.7 the allocation algorithm that minimises the interference is proposed. The evaluation of the proposal is presented in Section 4.8, while conclusions and further work are given in Section 4.9.

4.3. Related works

There is a lot of research about real-time multicore systems, one of the most relevant surveys in this area is [4.12]. As we know, in multi-core scheduling there are two main branches, partitioned and global scheduling. In this work we are focusing on partitioned scheduling since we focus on highly critical real-time systems. Partitioned multi-core scheduling involves two phases: task allocation to cores and scheduling of each core.

The allocation of tasks to cores can be solved with bin packing techniques. We known that this problem is NP-hard in the strong sense, [4.19]. Some of the most popular heuristics, cited in [4.25] and [4.7] are Worst fit (WF), First fit (FF) and Best fit (BF). Coffman et al. [4.8] describes many of these algorithms in detail.

It is important to note that the previous allocators do not take into account the unpredictability produced by shared hardware resources. In [4.10] a comprehensive analysis of all possible

sources of indeterminism is presented. These sources are classified into primary and secondary. They consider primary sources such as caches, memory, FSB, and memory controller and secondary sources such as power saving strategies, hardware-prefetching, system management interrupts and translation look-aside buffer.

Additionally, in [4.14] a state-of-the-art about contention delays is presented. This topic is also analysed in [4.23] where the sources of timing interference in single-core, multi-core, and distributed systems are presented. As stated in this paper, memory interference can render a system infeasible. It is shown in [4.27] that it is possible that memory interference can cause a worst-case response time increase of almost 300 %, even for tasks that spend only 10 % of their time fetching memory on an eight-core system.

There are some works that reduce interference delays by using modified task models. In [4.26] it is proposed a new model called PREM (predictable execution model). This proposal divides one task into two phases: communication and execution. A similar technique is used in [4.29] that schedules the system with the goal of minimising the makespan by letting the scheduler decide when it is appropriate to avoid interference. For DAG task models, [4.18] proposes a scheduling method that applies the LET (Logical Execution Time) paradigm and considers communication timing between nodes to reduce interference delay due to shared hardware resources.

Other works, as [4.9], proposes a feedback control scheme where critical and non-critical tasks are separated and assigned to different partitions for ensuring the execution of the critical tasks, so a hypervisor manages the multicore hardware system in order to limit memory bus access in non-critical cores measured with Performance Monitor Counters. In [4.5], the authors propose an analysis of memory contention as an optimisation problem which tries to minimize memory interference. They split tasks intro three phases and consider multiple memory transactions issued during each phase. Other approaches, such as the one presented in [4.28], reduce contention using synchronisation-based interference models and specific memory allocation schemes.

Survey [4.22] provides an overview on timing verification techniques for multicore real-time systems until 2018. This survey considers single shared resources (memory bus, shared cache, DRAM) and also multiple resources, which is where this work focuses on. The most relevant works come from the Multicore Response Time Analysis framework in [4.3], that provides a general approach to timing verification for multicore systems. They omit the notion of worst case execution time (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 cited in the survey as [4.3] and [4.6] consider the amount of time for shared resources accesses and the maximum number of access segments, which is out of the scope of this work.

Regarding to mapping and scheduling, the survey presents several works grouped by the techniques that are used: bin-packing algorithms, genetic algorithms, ILP and MILP (Integer/Mixed Integer Linear Programming), etc. ILP and MILP techniques consider scratch-pad memories with different objectives: to minimise the initiation intervals of task graphs, to minimise the WCETs, to minimize the worst case response time (WCRT) of task graphs,etc. The presented techniques are not scalable to large numbers of cores in the system and authors present heuristics that are scalable.

There are some works that introduce the interference due to memory contention as a new parameter in the temporal model. In [4.15] WCRA (Worst Case number of shared Resource Accesses) is defined and added to the WCET. In the same way, [4.24] proposed the concept of interference-sensitive Worst-Case Execution Time (isWCET). A dynamic approach is presented in [4.30] so that, depending on the progress of each kernel, the dependencies of the isWCET schedules are reduced or completely eliminated. The concept of isWCET is similar to our work but the proposals are centred around minimising the effect of interference with new scheduling methods while our work focuses on allocation algorithms and schedulability conditions. In [4.20] memory interference is analysed in a similar way to our work, as it is also represented as a parameter of the temporal model. However, their work is based on the interference produced only by the DRAM memory while our proposal is agnostic with respect to the shared hardware resource used. Their work also considers fixed priority scheduling while our proposal considers dynamic priorities.

The work in [4.13] provides 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 considers different types of interference (limited, direct and indirect) and fixed priority scheduling policies. In contrast to this work, the task-to-cores allocation is known a priori.

In [4.2], the interference due to contention is added to the temporal model. Instead of add 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.

In [4.17], 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 [4.2] but in [4.2] a general model is considered, valid for any type of shared hardware resource while in [4.17] only interference due to cache sharing is considered.

Our work continues the research of [4.2], so we use the same temporal model to provide a schedulability bound. Therefore, our model does not only provide a schedulability analysis for both fixed and dynamic priorities with a general model for interference, but also a new allocation algorithm that minimizes this bound.

4.4. Problem definition and task model

The aim of this work is to provide a schedulability condition and an interference-conscious allocator for the task model presented below. This task model is the same as the one presented in [4.2].

We suppose a multicore system with m cores $(M_0, M_1, M_2, ..., M_{m-1})$ where a task set τ of n independent tasks should be allocated. Each task τ_i is represented by the tuple:

$$\tau_i = (C_i, T_i, I_i) \tag{4.1}$$

where C_i is the WCET, T_i is the period and I_i is the interference. We assume implicit deadlines, so deadlines are equal to periods. When we refer to M_{τ_i}, we mean the core in which τ_i is allocated.

The hyperperiod of the task set, H, is defined as the least common multiple (lcm) of the periods of all the periodic tasks:

$$H = lcm\left\{T_i \mid i = 0, ..., n-1\right\} \tag{4.2}$$

We define A_i as the number of activations that τ_i has throughout H:

$$A_i = H/T_i \tag{4.3}$$

Since the goal of this paper is to obtain a schedulability test, we will assume a synchronous task system.

To visualize the concept of I_i, we can observe Figure 4.1, where the computation time (C_0) of task τ_0 is represented. As part of this computation time, the time in which the task performs read and/or write operations in memory has been differentiated (depicted with diagonal lines), as an example of access to a shared hardware resource. It is possible that when the task wants to do one of these r/w operations, the requested resource is busy. Or the other way around, that when the task is accessing this hardware resource, other tasks are blocked from accessing this resource. In any case, this time is the time considered as interference caused to other tasks in other cores I_i. In general, the term I_i is the time the task takes to access shared hardware resources. 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. So this interference time is defined independently of C_i, as will be used to represent the delay caused to other tasks. Although the parameter I_i is defined in [4.2], it is worth describing in detail by means of an example.

Example of task set execution with interference: We have a task set composed of 3 tasks and 3 cores. Every task is allocated to a different core. τ_0 and τ_2 request access to the same shared hardware resource, as $I_0, I_2 > 0$. However, τ_1 will not be affected by the interference of the other tasks, as $I_1 = 0$.

$$\tau_0 = (3, 6, 1) \quad \tau_1 = (5, 8, 0) \quad \tau_2 = (6, 12, 1)$$

The scheduling of this system is represented in Figure 4.2.

As we can observe in Figure 4.2, on one hand, task τ_1 does not receive any interference from the other tasks, so its execution time is not affected by the contention. On the other hand, τ_0 and τ_2 suffer an increase in their execution time. At the beginning, both tasks (τ_0 and τ_2) suffer a delay of 1 unit because the interference between them. In t=6, τ_0 is released again and coincides with the first job of τ_2. In that moment, each task provokes a unit of interference in the execution of the other task. We represent the interference as unit times in unified blocks, and positioned in the beginning of the execution in order to facilitate the calculation but always preserving the real magnitude of the contention.

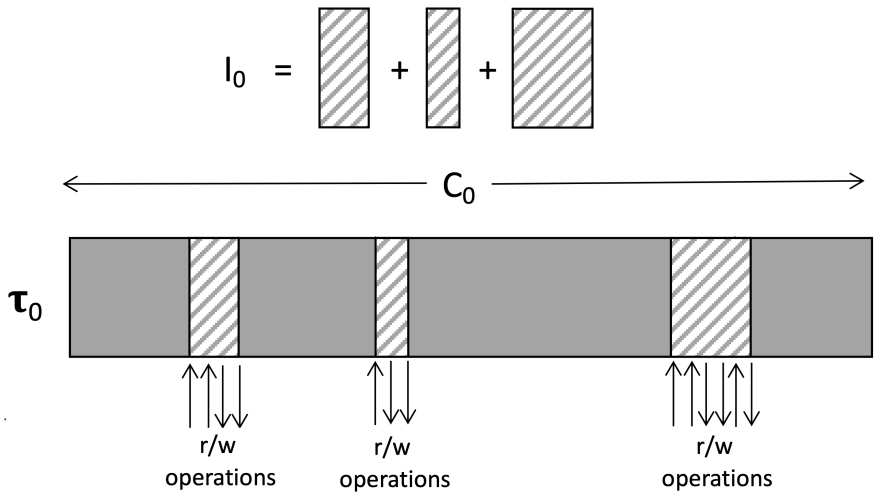


Figura 4.1: The interference time is included during the time execution

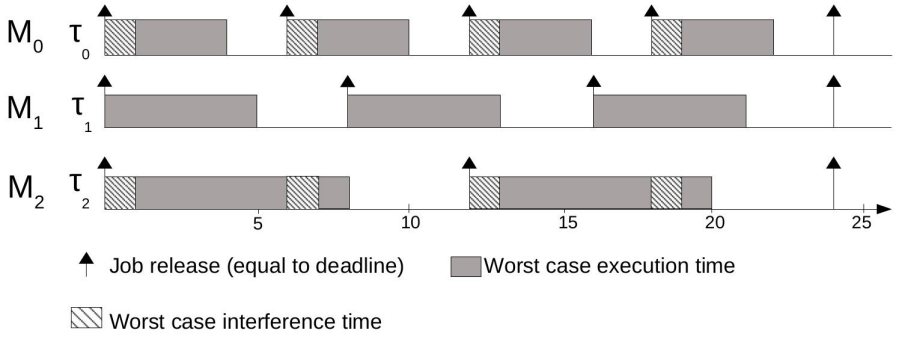


Figura 4.2: Example of interference

Once the model has been defined, we will now present some parameters necessary for the development of the following sections.

From Figure 4.2, it is deduced that, if contention is considered, the total utilisation of a task does not only depend on its computation time and period, but also on the interference received from other tasks.

We consider that the interference parameter refers to a single type of hardware resource. If there are several types of hardware resources the model would have to extend to as many interference parameters as there are hardware resources. Considering the parameter I for all interference types would result in an even more pessimistic model.

It should be taken into account that a task τ_j will cause interference to other task τ_i when the following conditions are met:

- τ_i and τ_j are not allocated to the same core ($M_{\tau_i} \neq M_{\tau_j}$).
- τ_i and τ_j are active at the same time.
- τ_i and τ_j have at least 1 unit of interference ($I_i > 0$, $I_j > 0$)

Therefore, the real utilisation[1] of a task, U'_i, is:

$$U'_i = U_i + U_i^{int} \tag{4.4}$$

being $U_i = C_i/T_i$ and U_i^{int} the utilisation due to the interference caused by other tasks to τ_i:

$$U_i^{int} = \frac{I_i^T}{H} \tag{4.5}$$

being I_i^T the total interference that a task τ_i receives from other tasks in a period T_i.
Therefore:

$$U'_i = \frac{C_i}{T_i} + \frac{I_i^T}{H} \tag{4.6}$$

Hence, the utilisation of a core would be the sum of all the task utilisations:

$$U'_{M_k} = \sum_{\tau_i \in M_k} U'_i \tag{4.7}$$

And the utilisation of all the system would be the sum of all the core utilisations:

$$U'_\tau = \sum_{\forall k} U'_{M_k} \tag{4.8}$$

[1]We refer to real utilisation to the utilisation that includes the interference delay.

4.5. Contention aware utilisation factor

In this section, we are going to provide an upper bound to U_i^{int}, so we will be able to provide a schedulability test. This bound will be called U_i^{iub}. First, we need two definitions, that were also introduced in [4.2]:

Definition 5. *[4.2] A task is defined as a receiving task when it accesses shared hardware resources and it suffers an increment of its computation time due to the interference produced by other tasks allocated to other cores.*

Definition 6. *[4.2] A task is defined as a broadcasting task when it accesses shared hardware resources and it provokes an increment of 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 $I_j > 0$.

To estimate U_i^{iub}, we will calculate the maximum total interference that will depend on the maximum number of activations of a broadcasting task that fall within a receiving task.

Definition 7. *Let $I_{j\to i}^{Tub}$ [2] be the maximum total interference that τ_j can cause to τ_i in a period of τ_i, T_i.*

Definition 8. *Let $A_{j\to i}$ be the maximum number of activations of the broadcasting task τ_j that fall within an activation of the receiving task, τ_i.*

From the two previous definitions, it is clear that:

$$I_{j\to i}^{Tub} = A_i \cdot A_{j\to i} \cdot I_j \tag{4.9}$$

Note that $I_{j\to i}^{Tub}$ is an upper bound because the maximum number of interferences does not have to occur in all activations. Besides, $I_{j\to i}^{Tub}$ only takes into account the interference caused by τ_j. To calculate U_i^{iub}, we need to consider all tasks allocated to other cores. Therefore:

$$U_i^{iub} = \frac{\sum_{\tau_j \notin M_{\tau_i}} I_{j\to i}^{Tub}}{T_i} \tag{4.10}$$

From Equation 4.10, it is easy to see that we have reduced the study of U_i^{iub} to the study of $I_{j\to i}^{Tub}$.

4.5.1. Worst case estimation of $I_{j\to i}^{Tub}$

To calculate $I_{j\to i}^{Tub}$, we need to know the exact value of $A_{j\to i}$. To do this, we will calculate $A_{j\to i}$ assuming the following conditions:

[2] In what follows, the expression $j \to i$ means that τ_j is a broadcasting task and τ_i is a receiving task

- τ_i and τ_j are allocated to different cores.
- $I_i, I_j > 0$
- $T_j \geq T_i$ (the period of the broadcasting task is greater than the period of the receiving task)

We will have further consideration of the case in which $T_j < T_i$.

Let us study the total interference received by τ_i. First, the maximum number of activations of τ_j that fall within an activation of τ_i is calculated:

$$A_{j \to i} = \left\lceil \frac{T_i - 1}{T_j} \right\rceil + K \tag{4.11}$$

being

$$K = \begin{cases} 0 & \text{If periods } T_i \text{ and } T_j \text{ are harmonics} \\ 1 & \text{Elsewhere} \end{cases}$$

Equation 4.11 expresses the relationship between the periods of broadcasting and receiving task modified by factor K and minus 1 to reflect the number of activations of the broadcasting task that fall within a period of the receiving task. With the division of periods, we obtain the whole times that a task is included in the other and we apply the ceiling function in order to obtain the greatest integer number of times. Moreover, the worst scenario happens if one of the tasks is shifted one unit of time. That is why the minus one is added to the formula.

Then, two cases are possible:

- Harmonic periods, with zero or small residues in the division of periods. Then, $K = 0$.
- Non-harmonic periods. Then, $K = 1$.

The above definition of $I_{j \to i}^{Tub}$ is very pessimistic in the sense that it considers that the maximum interference occurs equally in all activations. This is not always true, as it depends on how the activations of the two tasks coincide.

Once the definition of $I_{j \to i}^{Tub}$ is provided, let us consider the case in which $T_i > T_j$ (the period of the broadcasting task is shorter than the period of the receiving task) with the following example: let us suppose two tasks τ_i and τ_j allocated to a dual-core platform, with $I_i = 2I_j$. Figures 4.3 and 4.4 represent the maximum interference that may be produced when tasks τ_i and τ_j are executed simultaneously. Note that, for the sake of simplicity, we have not depicted the tasks computation times, but only the interference.

Figure 4.3 considers that τ_j is the broadcasting task and τ_i the receiving task. In this case, the period of the broadcasting task τ_j is greater than the period of the receiving task τ_i, and the total received interference is equal to $I_{j \to i}^{Tub} = 4$. This scenario has been studied previously.

Let us consider now that τ_i is the broadcasting task and τ_j is the receiving task, as depicted in Figure 4.4. In this case, the period of the broadcasting task is shorter than the period of the receiving task. Could we apply Equation 4.11? We are going to prove it numerically. Suppose

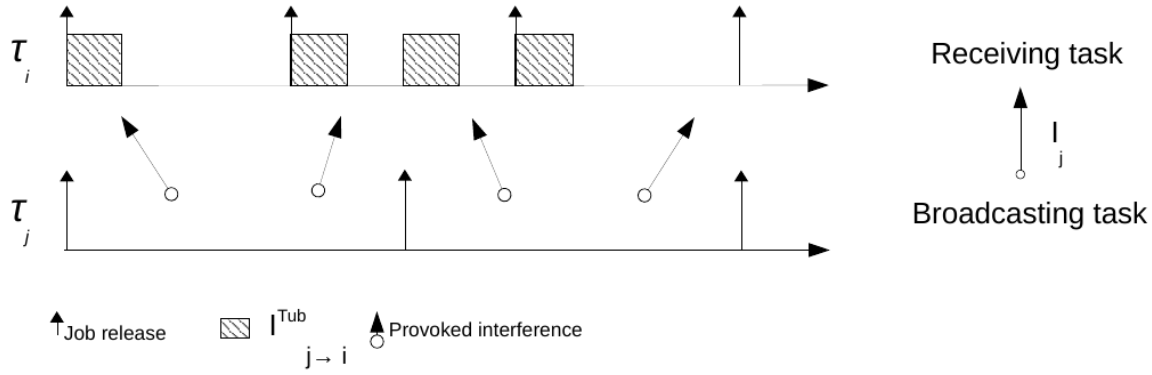


Figura 4.3: Example of interference from τ_j to τ_i

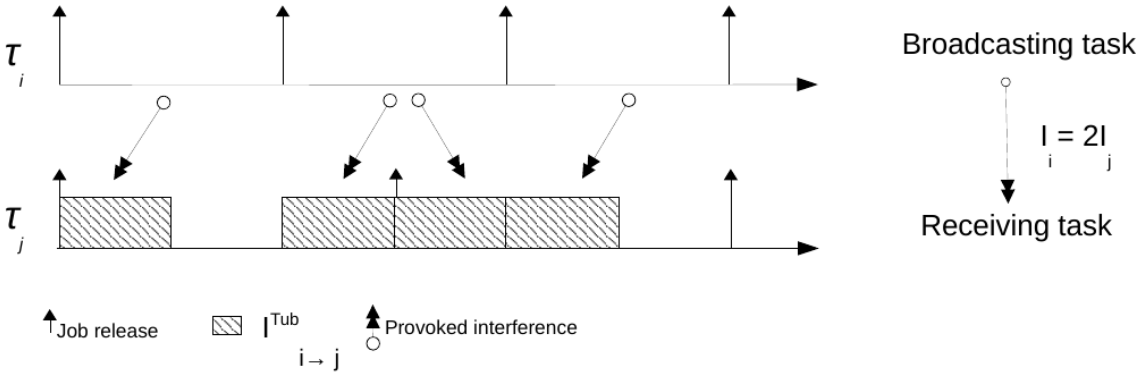


Figura 4.4: Example of interference from τ_i to τ_j

that $T_i = 4$ and $T_j = 6$. Applying Equation 4.11, the maximum number of activations of the broadcasting task that fall within an activation of the receiving task is $A_{i\to j} = \left\lceil \frac{T_j - 1}{T_i} \right\rceil + 1 = 3$. As seen in Figure 4.4, τ_i never falls three times within an activation of τ_j. So Equation 4.11 can not be applied when the period of the broadcasting task τ_j is shorter than the period of the receiving task.

Then, we are proposing a methodology to relate the total interference between a broadcasting and a receiving task, regardless of the lengths of their periods.

Theorem 2. *The ratio of interference received and broadcast by τ_j to τ_i is:*

$$I_{i\to j}^{Tub} = \frac{I_i}{I_j} I_{j\to i}^{Tub} \tag{4.12}$$

Demostración. At time 0, as both tasks are released, interference is always introduced. Moreover, every time any task is released, I_j units of interference may be introduced. In particular, it happens as many times as activations the tasks have in a hyperperiod minus one, because first activation has already been considered. Then, the total number of times is:

$$1 + \left(\frac{H}{T_i} - 1\right) + \left(\frac{H}{T_j} - 1\right) = \frac{H}{T_i} + \frac{H}{T_j} - 1$$

Then, the maximum number of interferences that a task τ_j provokes in τ_i is:

$$I_{j\to i}^{T} = \left(\frac{H}{T_i} + \frac{H}{T_j} - 1\right) I_j$$

Then, let us have a look at the total interference received by task j, $I_{i\to j}^{T}$. If we repeat the previous process, the maximum number of interferences that the task τ_i provokes in τ_j is:

$$I_{i\to j}^{T} = \left(\frac{H}{T_j} + \frac{H}{T_i} - 1\right) I_i$$

It is easy to deduce that the only difference between both situations is the coefficient of interference of the task that provokes the interference.

Therefore, we can conclude that total interference that task j provokes to task i is related with the total interference that task i provokes to task j as follows:

$$I_{i\to j}^{Tub} = \frac{I_i}{I_j} I_{j\to i}^{Tub}$$

□

Then, the total number of interferences between two tasks may be calculated interchangeably, from the point of view of both tasks. Thus, from now on and for calculus purposes, this work considers that, when two tasks interfere, the task with the biggest period provokes the interference on the task with the shortest period. If it is not the case, we will apply Equation 4.12.

To conclude with the example in Figures 4.3 and 4.4, as we deduced that $I_{j\to i}^{Tub} = 4$, from Equation 4.12:

$$I_{i\to j}^{Tub} = \frac{I_i}{I_j} I_{j\to i}^{Tub} = \frac{2I_j}{I_j} 4 = 8$$

and this result coincides with the depicted in Figure 4.4.

4.6. Schedulability analysis

Once an upper value is given for $I_{j\to i}^{Tub}$, we can estimate an upper bound of the utilisation of a receiving task τ_i, U_i^{ub}, taken into account interferences due to contention.

$$U_i^{ub} = U_i + U_i^{iub} = U_i + \frac{\sum_{\tau_j \notin M_{\tau_i}} I_{j\to i}^{Tub}}{T_i}$$

Note that, if τ_i is not a receiving task, $U_i^{iub} = 0$.

This upper bound is always greater or equal to the real utilisation of the receiving task i, U_i'. This results from the fact that:

$$I_i^T \leq \sum_{\tau_j \notin M_{\tau_i}} I_{j\to i}^{Tub}$$

Therefore, the upper bound of the utilisation of each core is defined as the sum of the upper bounds of the utilisations of the tasks that belong to that core, $U_{M_k}^{ub} = \sum_{\tau_i \in M_k} U_i^{ub}$. Similarly, the upper bound of the system utilisation is defined as the sum of the upper bound of the utilisations of all cores (or all tasks), $U_\tau^{ub} = \sum_{M_k} U_{M_k}^{ub}$. Then, the system will be schedulable if:

1. The upper bound of the utilisation of each task is less or equal to $n(2^{\frac{1}{n}} - 1)$ for fixed priorities and to one for dynamic priorities, [4.21]:

$$U_i^{ub} \leq \left\{ \begin{array}{ll} n(2^{\frac{1}{n}} - 1), & \text{for fixed priorities} \\ 1 & \text{for dynamic priorities} \end{array} \right\} \forall i = 0, ..., n-1$$

2. The upper bound of the utilisation of each core M_k is less or equal to one:

$$U_{M_k}^{ub} \leq 1 \quad \forall k = 0, ..., m-1$$

As a consequence of (i) and (ii), the upper bound of the system utilisation is less or equal to the number of cores [4.4], m:

$$U_\tau^{ub} \leq m$$

As utilisations U_i^{ub} are overestimated, we can conclude that if previous conditions are accomplished, the system will be schedulable.

Note that this is a necessary but no sufficient condition.

4.6.1. Example.

Let us show the procedure to be followed with an example: let us consider the following task set, $\tau = [\tau_0, \tau_1, \tau_2]$ with $\tau_0 = (2, 3, 0)$, $\tau_1 = (4, 8, 2)$, and $\tau_2 = (5, 12, 1)$, allocated to a system with 3 cores. τ_0 is allocated to core M_0, τ_1, in M_1, and τ_2, in M_2. Once tasks are allocated to cores, Earliest Deadline First (EDF) algorithm [4.21] schedules tasks in each core. The actual execution of the task set is shown in Figure 4.5.

Figura 4.5: $\tau = [\tau_0, \tau_1, \tau_2]$ with $\tau_0 = (2, 3, 0)$, $\tau_1 = (4, 8, 2)$, and $\tau_2 = (5, 12, 1)$ allocated to a system with 3 cores.

We are applying the method described in Section 4.5 to calculate the upper bound of the system utilisation. And then we will compare it with the actual system utilisation, shown in Figure 4.5.

- Step 1: Define τ_j and τ_i. The task with the greatest period is the broadcasting task τ_j, and the receiving task is the task with the shortest period, τ_i. In this example, $\tau_j = \tau_2$ and $\tau_i = \tau_1$. As $I_0 = 0$, τ_0 is neither receiving nor broadcasting.
- Step 2: Calculate the maximum number of times that task τ_2 falls within an activation of τ_1, $A_{2\to1}$. As depicted in Figure 4.6, τ_2 falls twice within the second activation of τ_1. It may be calculated applying Equation 4.11:

$$A_{2\to1} = \left\lceil \frac{T_1 - 1}{T_2} \right\rceil + 1 = \left\lceil \frac{8 - 1}{12} \right\rceil + 1 = 1 + 1 = 2$$

- Step 3: To calculate the maximum total interference that τ_2 can cause to τ_1. First, we calculate the number of activations of task τ_1 in a hyperperiod, $A_1 = H/T_1 = 3$. Following Equation (4.5):

$$I_{2\to1}^{Tub} = A_1 \cdot A_{2\to1} \cdot I_2 = 3 \cdot 2 \cdot 1 = 6$$

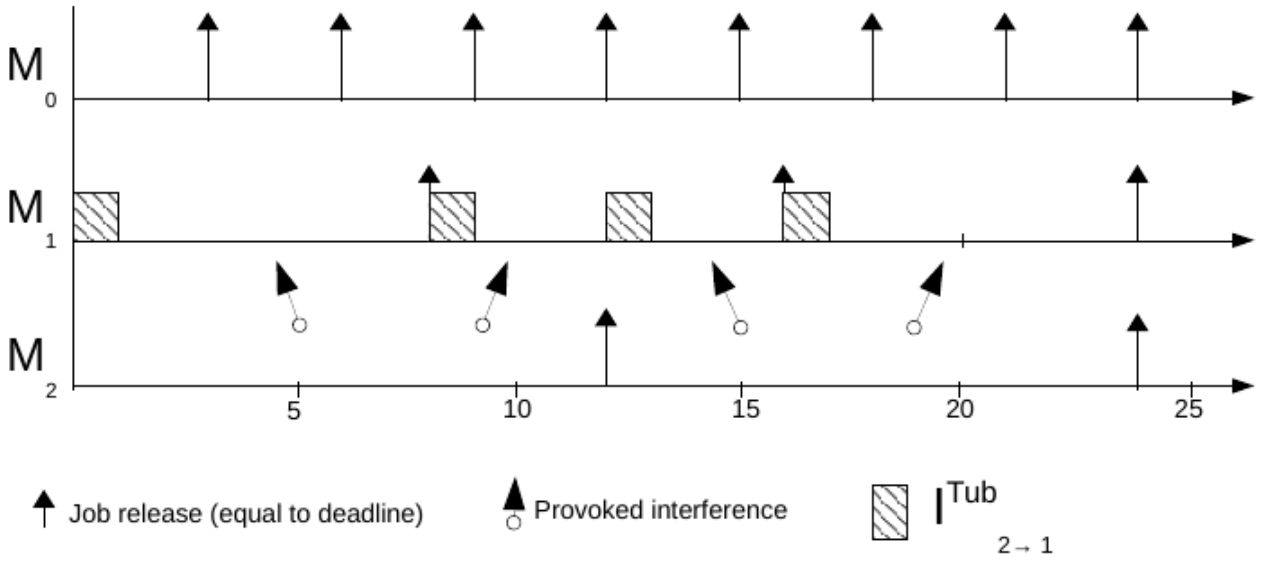


Figura 4.6: Upper bound of the interference received by τ_1, $I_{2\to1}^{Tub}$.

The maximum total interference is only received in the second activation of τ_1, as shown in Figure 4.6, but we consider the worst case, in which all activations receive the maximum of interference from τ_2.

- Step 5: Calculate the total interference received by the other receiving tasks, τ_2. As the period of τ_2 is greater than the period of τ_1, we apply the Equation 4.12:

$$I_{1\to2}^{Tub} = \frac{I_1}{I_2} I_{2\to1}^{Tub} = \frac{2}{1} \cdot 6 = 12 \tag{4.13}$$

- Step 6: Calculate the upper bound of the utilisation of each task.

$$U_0^{ub} = \frac{C_0}{T_0} = \frac{2}{3} = 0{,}667$$
$$U_1^{ub} = \frac{C_1}{T_1} + \frac{I_{2\to1}^{T}}{H} = \frac{4}{8} + \frac{6}{24} = 0{,}75$$
$$U_2^{ub} = \frac{C_2}{T_2} + \frac{I_{1\to2}^{T}}{H} = \frac{5}{12} + \frac{12}{24} = 0{,}91667$$

From Figure 4.5, the actual utilisations of the tasks are:

$$U_0^{'} = \frac{16}{24} = 0{,}667 \leq U_0^{ub}$$
$$U_1^{'} = \frac{14}{24} = 0{,}5833 \leq U_1^{ub}$$
$$U_2^{'} = \frac{14}{24} = 0{,}5833 \leq U_2^{ub}$$

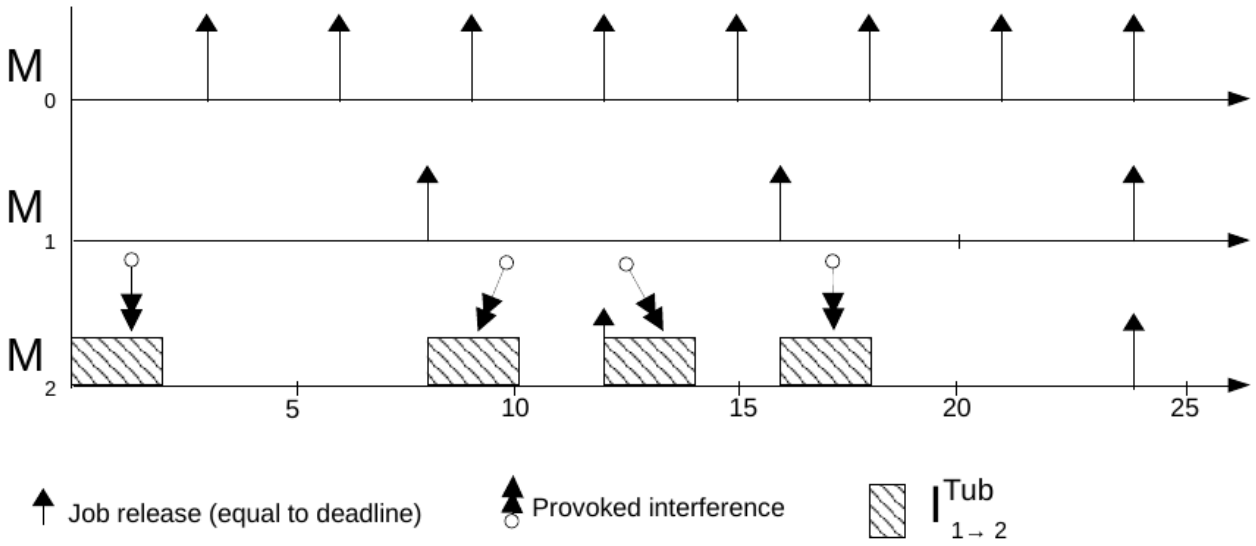


Figura 4.7: Upper bound of the interference received by τ_2, $I_{1\to 2}^{Tub}$.

It is deduced that the actual utilisations of the tasks are equal or less than the upper bounds estimated in this paper. Therefore, the system is schedulable.

4.7. Task allocation algorithms

In the following, we first briefly discuss several existing allocation techniques. We then propose a new mapping strategy. Allocation is a problem that appears with multicore systems. It comes to answer the question: which processor will execute each task?

The main disadvantage of the partitioning approach to multicore scheduling is that the task allocation problem is analogous to the bin packing problem and is known to be NP-Hard [4.16].

4.7.1. Overview of existing heuristic bin-packing algorithms.

In [4.25] and [4.7] are detailed some of the most well-known bin-packing heuristics:

- First Fit (FF). Each item is always packed into the first bin where it fits. A new bin is open if any item does not fit in the open bin.
- Worst Fit (WF). Each item is placed into the open bin with the largest amount of room remaining. If it does not fit any bins, a new bin is opened.

One problem with the previous mentioned heuristics is that, if items are not ordered properly, large items could not be packed efficiently. Therefore, items should be ordered effectively to avoid these problems. One way is to pack the items in order of decreasing weights or utilizations. This way, WF becomes WFDU (Worst Fit Decreasing utilisation), and FF becomes FFDU (First Fit Decreasing utilisation).

4.7.2. Overview of Aceituno's method.

In [4.2], a task model that takes into account interference delays due to contention of shared hardware resources is proposed. Moreover, three tasks-to-cores allocation methods are also presented.

In that paper, a discrepancy-based method is presented, which is defined as the difference between the maximum and minimum utilisation of a multicore system, UD_τ. One of the algorithms is focused on reducing the discrepancy, UDmin, and the other on maximizing it, UDmax. On the one hand, UDmin behaves as the heuristic WFDU, as both balance the load among cores. They present excellent schedulability ratios (98.1 % and 100 % respectively) but reach high rates of increment of utilisation due to the system interference (2.3 %). On the other hand, UDmax behaves as FFDU, as it unbalances the load among cores. In contrast to UDmin/WFDU, it reaches a low rate of increment of utilisation but their rates of schedulability are very low (around 43 %). Wmin allocator accounts for the possible interference produced for each task and it provides a low increment of utilisation (0.266 %) and high schedulability ratio (up to 89 %), with respect to previous algorithms.

4.7.3. Proposed allocator considering the interference: Imin

Previous bin-packing and discrepancy-based algorithms do not take into account the interference produced when two or more tasks allocated to different cores coincide in execution. However, Wmin does take interference into account, although not exactly, but based on a binary matrix (W) of possible interference between tasks on other cores. This allocator tries to minimise the number of 1s in the matrix, which represent possible interference.

In this subsection, a new allocation algorithm to minimise the interference is proposed, Imin. As it has been analysed in previous sections, U_i^{ub} is the upper bound of the utilisation taking into account the maximum possible interference. The allocator Imin obtains the task allocation to cores that minimises this upper bound as much as possible. This allocator is based on integer linear programming to obtain an optimal solution. For that, we define a set of parameters and variables shown in Table 4.1.

As it has said before, the objective is:

$$\text{Minimise} \sum_{\forall i} U_i^{ub} \tag{4.14}$$

SETS AND INDICES	
i, j	Task index $i, j \in \{0, 1, 2, ..., n-1\}$
k	Core index $k \in \{0, 1, 2, ..., m-1\}$
PARAMETERS	
C_i	Worst case execution time of τ_i
T_i	Period of τ_i
U_i	Utilisation of τ_i
H	Hyperperiod of the task set τ
$I_{j \to i}^{Tub}$	Maximum interference that τ_j can cause to τ_i during (0,T_i)
DECISION VARIABLES	
U_i^{iub}	Upper bound of the utilisation due to interference of τ_i
U_i^{ub}	Upper bound of the total utilisation of τ_i
U_{M_k}	Utilisation of M_k
O_{ik}	Allocation matrix. 1 if τ_i is allocated to core k and 0 otherwise.

Tabla 4.1: Model notation of the implicit deadline task model

s.t:

$$\sum_{\forall k} O_{ik} = 1 \qquad \forall i \tag{4.15}$$

$$U_i^{iub} = \frac{\sum_{\tau_j \notin M_{\tau_i}} I_{j \to i}^{Tub}}{T_i} \qquad \forall i \tag{4.16}$$

$$U_i^{ub} = U_i + U_i^{iub} \qquad \forall i \tag{4.17}$$

$$U_{M_k} = \sum_{\forall i} O_{ik} \cdot U_i \qquad \forall k \tag{4.18}$$

$$U_{M_k} \leq 1 \qquad \forall k \tag{4.19}$$

$$O_{ik} \in \{0, 1\} \tag{4.20}$$

$$U_i^{iub}, U_i^{ub}, U_{M_k} \geq 0 \tag{4.21}$$

Constraint 4.15 assures that a task is allocated to one and only one core. The equation 4.16 sets the value of the extra utilisation of a task provoked by the interference in its upper bound. The equation 4.17 sets the value of the total utilisation taking account the interference in its upper bound, it means, in its maximum value as it has been explained and established in this paper. The total utilisation per core is calculated as the sum of the utilisations of the tasks that belong to that core (Equation 4.18) and its value should be less or equal to 1 (Equation 4.19). Equations 4.20 and 4.21 represent the decision variable domains.

4.8. Evaluation

4.8.1. Experimental conditions

To validate our proposed technique, we implemented a synthetic task generator, to generate up to 5400 random feasible task sets with different configurations. The task generator works by calculating the following parameters:

- U_τ: utilisation of the task set calculated using the UUniFast discard algorithm [4.11].
- T_i: task periods, which are generated randomly in [20,1000].
- C_i: computation times are deduced from the periods and the utilisations, as $C_i = U_i \cdot T_i$.

The experimental parameters of the evaluation are specified in Table 4.2. To ensure the reproducibility of the results, these parameters coincide with those used in [4.2]. The only difference is that a wide range of theoretical utilisations and interference factors are also studied.

The theoretical utilisation of the maximum load of all cores varies between 50 % and 75 % of the maximum load of the system. For example, the maximum load of a system with 8 cores is 8, so for evaluation purposes, the initial utilisation is set to 4 (50 %) and 6 (75 %).

As it is as well shown in Table 4.2, the number of broadcasting tasks is 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 in 3 cases, with 10 %, 20 % and 30 % of interference over the WCET. It means that, for example, if the percentage of interference in a task set is established to 20 % and a broadcasting task of this set has 10 units of WCET, then the interference of this broadcasting task should be 2 units.

With this setup, we apply the proposed Imin allocator and the already existing allocation methods (FFDU, WFDU and Wmin) to the synthetic task sets. Once the allocation phase is complete, it must be checked if all tasks have been allocated to cores assuring that the maximum capacity per core is not exceeded, i.e. $U_{M_k} \leq 1 \quad \forall k = 0, .., m-1$. If any of the allocators can not allocate the task set, this task set will be discarded and a new one is generated.

To perform the allocators Imin and Wmin, we use Gurobi optimizer 9.0 [4.1], from Gurobi Optimization, Inc., which is the fastest and most powerful mathematical programming solver available for ILP, MILP and other problems. It provides a Python interface.

The evaluation process is executed on an Intel Core i7 CPU with 16GB of RAM.

Once all allocations are validated, the task sets are scheduled following the contention aware scheduling algorithm proposed in [4.2] that takes into account the interference. The selected priority-based algorithm for the mentioned scheduling algorithm is EDF [4.21] but any fixed-priority algorithm could also be used. As an output of this phase, the real utilisation of the system, U'_τ, is obtained.

After the scheduling phase, the scheduling plans must be validated in order to ensure that temporal constraints are met throughout the hyperperiod. In addition, the following performance parameters are measured to compare different methods:

	Experimental parameters																	
Number of cores	2						4						8					
Number of tasks	4						12						20					
Number of broadcasting tasks	2						3						5					
Theoretical utilisation	1.1			1.5			2.1			3			4			6		
Number of sets	900			900			900			900			900			900		
Interference (%) over WCET	10	20	30	10	20	30	10	20	30	10	20	30	10	20	30	10	20	30
Scenario	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Tabla 4.2: Experimental parameters selected for the evaluation process.

- Schedulability ratio. It is calculated as the relation between the number task sets with feasible scheduling plans and the number task sets with feasible allocations, expressed as a percentage.
- Increased utilisation. It is calculated as the relation between The increase in utilisation with respect to the theoretical utilisation. This is calculated as $1 - \sum_k \frac{U_{M_k}}{U'_{M_k}} = 1 - \frac{U_\tau}{U'_\tau}$.
- Relation between the upper bound of utilisation, U_τ^{ub}, and the actual utilisation, U'_τ.

4.8.2. Experimental results

Depending on the experimental parameters specified in Table 4.2, different scenarios are defined (numbered from 1 to 18).

Figures 4.8 and 4.9 depict the comparison of schedulability ratio and the increased utilisation for each scenario and each allocator. In Figure 4.8 it is shown that, as it was expected, generally, as the number of cores and tasks increases, the schedulability ratio decreases. Regarding the percentage of interference, it is shown that, as a general rule, with the same number of cores and tasks, the higher percentage of interference, the lower is the schedulability for all the allocators, even though there are some exceptions to this rule, as FFDU in scenarios 7, 8, 9 or scenarios 10, 11 and 12. After an analysis of the data with the exception cases, we realized that these exceptions are just provoked by the randomness of the tasks generated as input data. In general terms, the allocators FFDU and WFDU are affected by the increase of interference percentage more than Wmin and Imin. This property can be clearly shown, for example, in scenarios 7, 8

and 9 of Figure 4.8, where the percentage of interference is 10, 20 and 30 % respectively. In these cases, FFDU and WFDU have very different results depending on the amount of interference but Wmin and Imin maintain almost the same values in spite of the variations in the amount of interference.

Figure 4.9 shows the increased utilisation for each scenario. The results varies a lot depending on the scenario and the allocator. In the case of FFDU, the increased utilisation with 2 cores is clearly higher than 4 and 8 cores. In the case of WFDU the increased utilisation is always the highest respect other allocators. In the case of Wmin and Imin the increased utilisation is significantly lower with respect to WFDU.

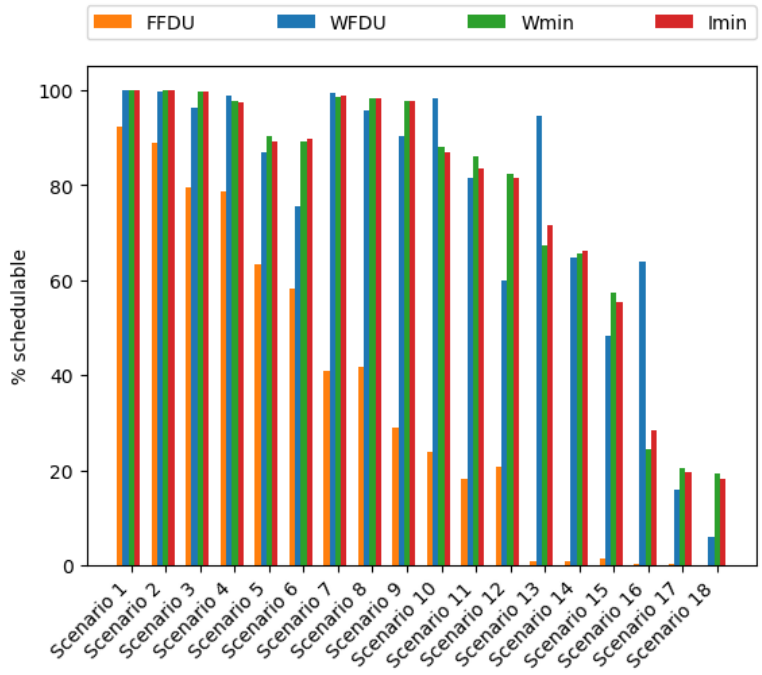


Figura 4.8: Percentage of schedulability task sets for each allocator depending on the scenario.

In Figures 4.8 and 4.9, average values of all scenarios of schedulability and increased utilisation are represented for each allocator. In Figure 4.10 it is shown that Imin has the highest rate of schedulability from all the allocators with a 76.83 %. We should be aware that Wmin also has a high index of schedulability, almost the same that Imin, 76.8 %. The FFDU allocator has the worst percentage of schedulability with 35.56 % and WFDU has an acceptable rate of schedulability with 76.48 %.

It is important to note that the results in [4.2] showed that WFDU could always ensure schedulability for all the task sets (100 %). In our case, this behaviour has not been reproduced. This difference lies in the interference coefficient selected in the evaluation process. In [4.2], $I_i = 1 \ \forall \tau_i \in \tau$. In this work, $I_i \geq 0{,}1 \cdot C_i \ \forall \tau_i \in \tau$, which is always greater or equal than 1. Adding small interferences does not affect the schedulability of allocators that balance the load, as WFDU does. In fact, when more interference is considered, these results vary. With bigger interferences, the percentage of schedulability is reduced.

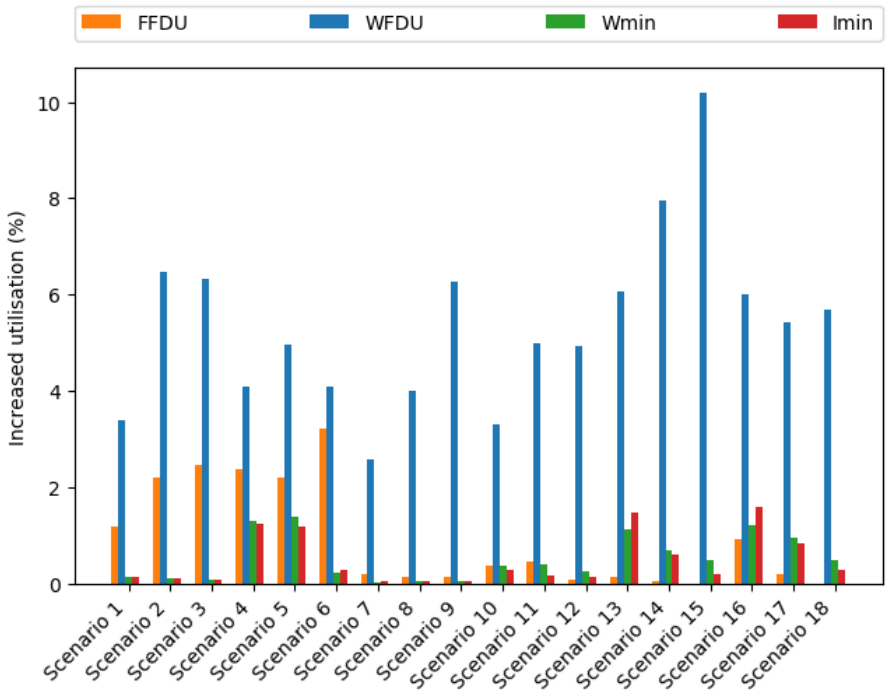


Figura 4.9: Increased utilisation resulted after the scheduling for each allocator depending on the scenario.

In Figure 4.11 it is shown that the WFDU algorithm has by far the highest increase in utilisation while Wmin and Imin have the best results. For WFDU and FFDU schedulability ratio and increased utilisation are directly proportional, which is not the desired behaviour. This is not the case of Wmin and Imin in which a high schedulability ratio does not suppose an increment in the interference and then, in the utilisation of the system.

The comparison between the upper bound (U_{τ}^{ub}) and the real utilisation (U'_{τ}) for the best allocators is depicted in Figures 4.12 (Imin), and 4.13 (Wmin). They are expressed in terms of utilisation system percentage. As it was expected, in both graphics, the real utilisation is always equal or less than its upper bound utilisation. Also in both graphics, the more cores and tasks in the scenario, the more percentage of system utilisation is estimated.

In Figure 4.14, the comparison between the utilisation upper bound of Wmin and Imin is depicted. We can see that the results are very similar as in previous figures but Imin achieves a slight lower value for U_i^{ub} because Imin minimises the upper bound while Wmin minimises W matrix. Even thought, results are very similar since both represent the possible interference between tasks in other cores.

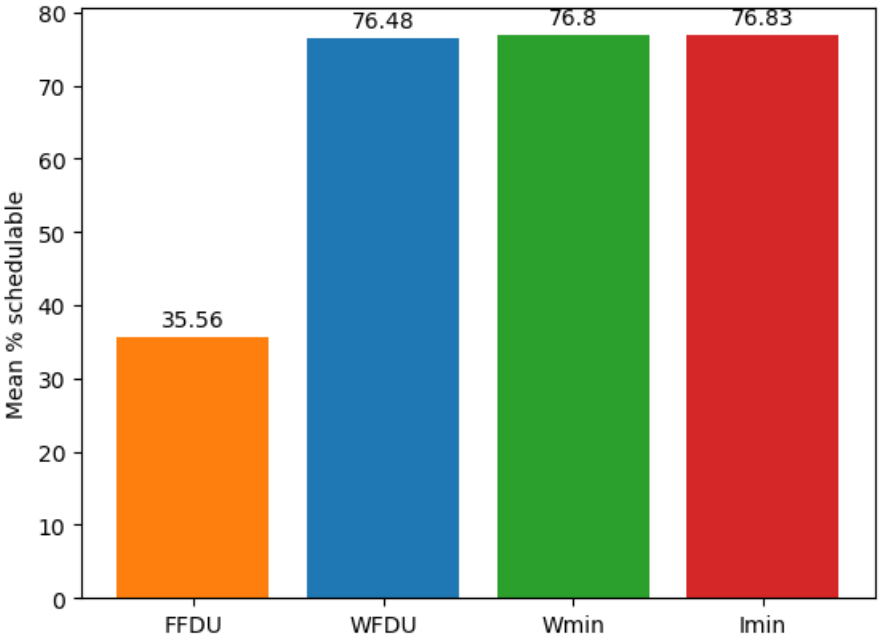


Figura 4.10: Average of percentage of schedulable task sets depending on the allocators.

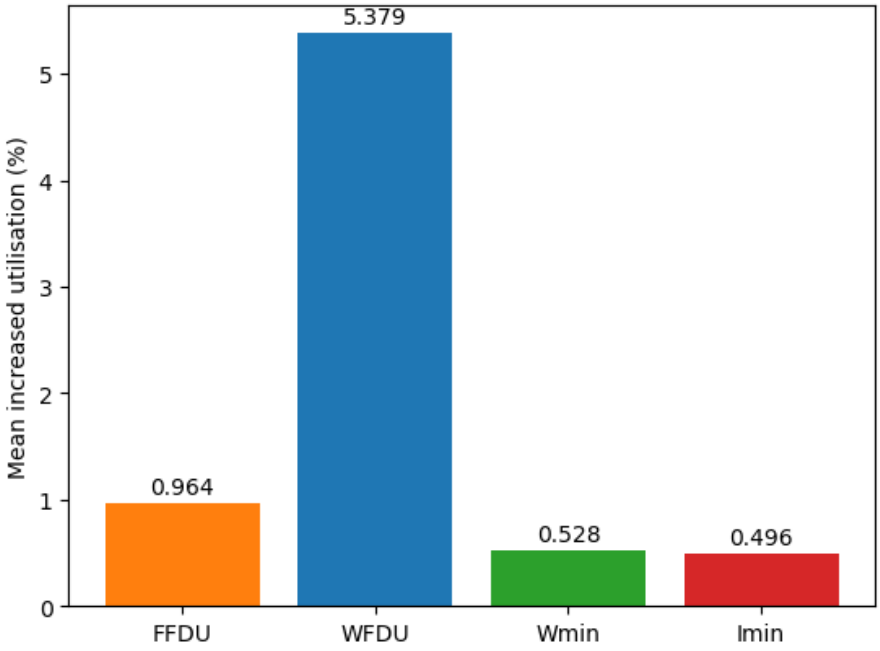


Figura 4.11: Average of increased utilisation depending on the allocators.

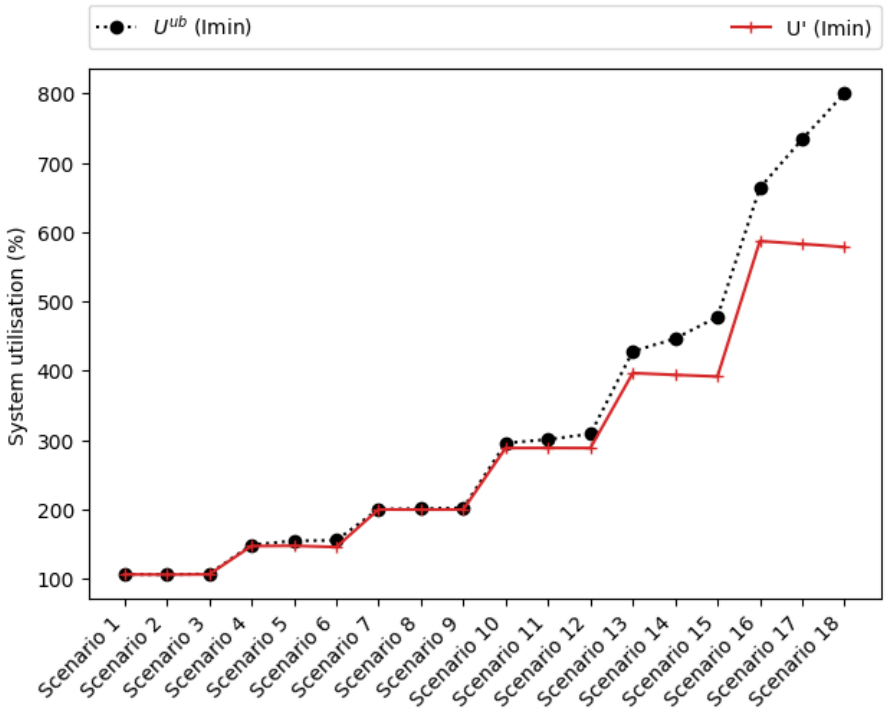


Figura 4.12: Imin algorithm: theoretical utilisation upper bound values compared to the real utilisation values measured after the scheduling.

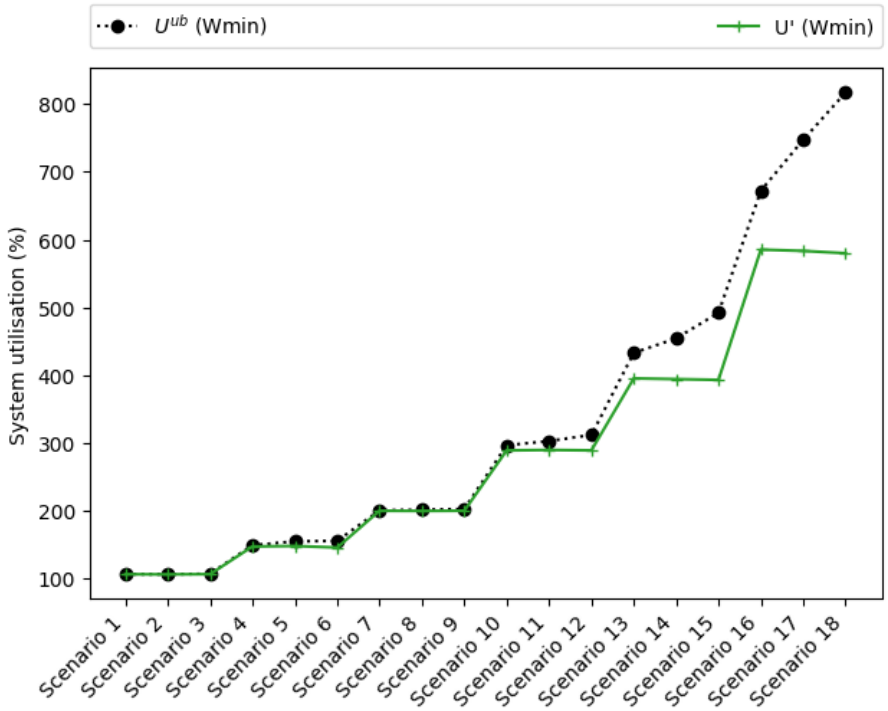


Figura 4.13: Wmin algorithm: theoretical utilisation upper bound values compared to the real utilisation values measured after the scheduling.

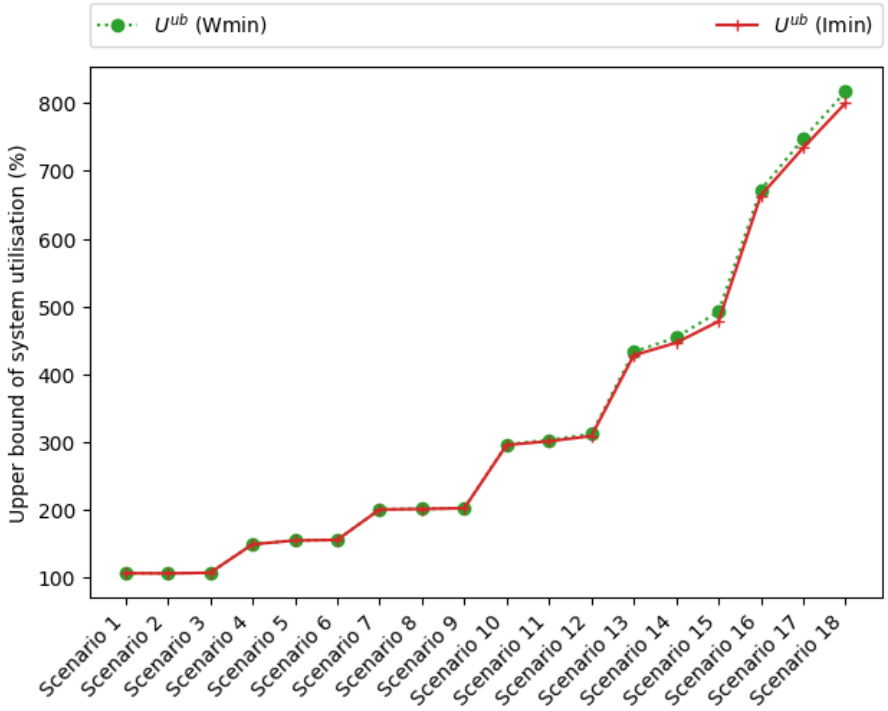


Figura 4.14: Comparison of Wmin and Imin utilisation upper bounds values.

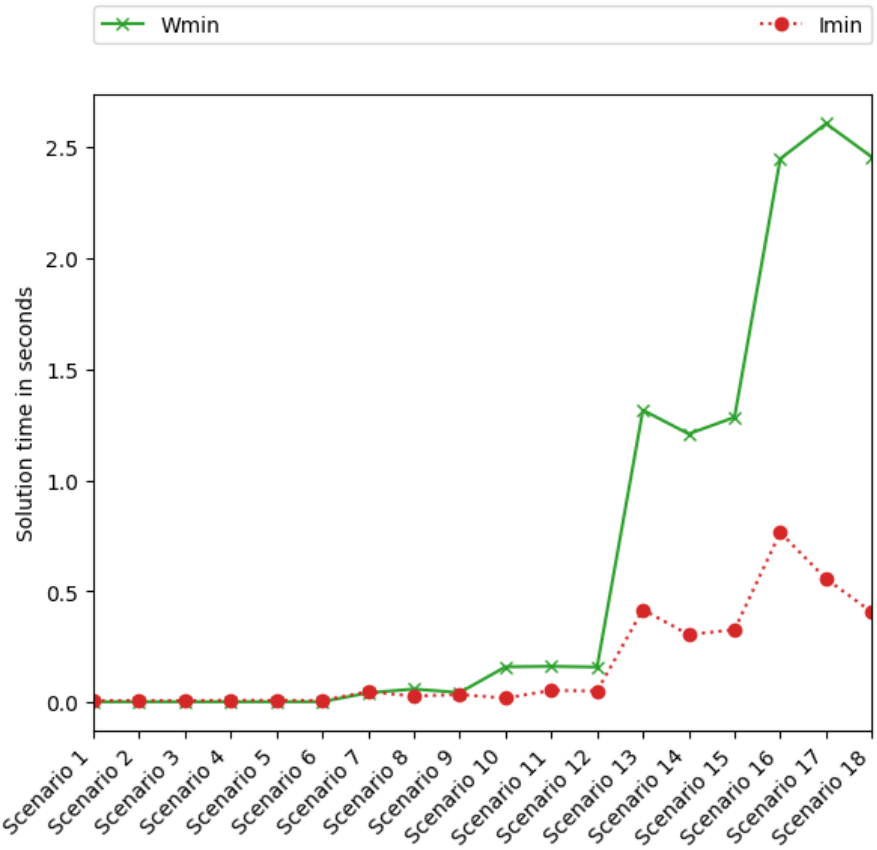


Figura 4.15: Solution times for MILP approaches depending on the experimental scenario.

Finally, the solution times for the MILP approaches are evaluated in Figure 4.15. It shows that, generally, as the number of cores increases, the solution time increases. This is more obvious in the cases of 8 cores scenarios (scenarios 13 to 18). This is because when the number of cores increases (and consequently, the number of tasks and broadcasting tasks), the complexity of the search of the optimal solution also increases. Moreover, it is observed that, specially in the 8 cores scenarios, the solution time for Imin is clearly lower than the solution time needed for Wmin. This is a clear advantage of Imin allocator.

4.9. Conclusions

This paper has proposed a schedulability analysis for task models that consider the delay produced by the contention of hardware shared resources in a hard real-time multicore system. Then, an allocation algorithm that minimises this contention, Imin, has been proposed and evaluated by comparison with other existing allocators.

After the results of the experimental evaluation, we can conclude that the allocation algorithm proposed in this paper, Imin, has the best rates in terms of percentages of schedulability and increased utilisation. In addition, Imin presents good results in terms of solution times. So

it is reasonable to affirm that Imin may be an eligible option as an allocator for the implementation of a multicore system, specially in those systems where the priority is to maximize the scheduling rate or to minimise the contention produced in hardware shared resources.

We plan to further investigate how to improve Imin in order to achieve a lower utilisation rate without decreasing schedulability. We also plan to find a less pessimistic upper bound for the utilisation and extend the model to constrained deadlines.

Bibliografía

[4.1] Gurobi optimizer reference manual. Inc. Gurobi Optimization, 2019.

[4.2] Aceituno, J. M., Guasque, A., Balbastre, P., Simó, J., and Crespo, A. Hardware resources contention-aware scheduling of hard real-time multiprocessor systems. Journal of Systems Architecture 118 (2021).

[4.3] Altmeyer, S., Davis, R. I., Indrusiak, L., Maiza, C., Nelis, V., and Reineke, J. A generic and compositional framework for multicore response time analysis. In Proceedings of the 23rd International Conference on Real Time and Networks Systems (New York, NY, USA, 2015), RTNS '15, Association for Computing Machinery, p. 129–138.

[4.4] Baruah, S., and Fisher, N. The partitioned multiprocessor scheduling of sporadic task systems. In 26th IEEE International Real-Time Systems Symposium (RTSS'05) (Dec 2005), pp. 9 pp.–329.

[4.5] Casini, D., Biondi, A., Nelissen, G., and Buttazzo, G. A holistic memory contention analysis for parallel real-time tasks under partitioned scheduling. In 2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS) (2020), pp. 239–252.

[4.6] Choi, J., Kang, D., and Ha, S. Conservative modeling of shared resource contention for dependent tasks in partitioned multi-core systems. In 2016 Design, Automation Test in Europe Conference Exhibition (DATE) (2016), pp. 181–186.

[4.7] Coffman, E. G., Garey, M. R., and Johnson, D. S. Approximation Algorithms for Bin Packing: A Survey. PWS Publishing Co., USA, 1996, p. 46–93.

[4.8] Coffman Jr., E. G., Csirik, J., Galambos, G., Martello, S., and Vigo, D. Bin Packing Approximation Algorithms: Survey and Classification. Springer New York, New York, NY, 2013, pp. 455–531.

[4.9] Crespo, A., Balbastre, P., Simó, J., Coronel, J., Gracia Pérez, D., and Bonnot, P. Hypervisor-based multicore feedback control of mixed-criticality systems. IEEE Access 6 (2018), 50627–50640.

[4.10] Dasari, D., Akesson, B., Nélis, V., Awan, M. A., and Petters, S. M. Identifying the sources of unpredictability in cots-based multicore systems. In 2013 8th IEEE International Symposium on Industrial Embedded Systems (SIES) (2013), pp. 39–48.

[4.11] Davis, R. I., and Burns, A. Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. In 2009 30th IEEE Real-Time Systems Symposium (Dec 2009), pp. 398–409.

[4.12] Davis, R. I., and Burns, A. A survey of hard real-time scheduling for multiprocessor systems. ACM Comput. Surv. 43, 4 (Oct. 2011).

[4.13] Davis, R. I., Griffin, D., and Bate, I. Schedulability Analysis for Multi-Core Systems Accounting for Resource Stress and Sensitivity. In 33rd Euromicro Conference on Real-Time Systems (ECRTS 2021) (Dagstuhl, Germany, 2021), B. B. Brandenburg, Ed., vol. 196 of Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl – Leibniz-Zentrum für Informatik, pp. 7:1–7:26.

[4.14] Fernandez, G., Abella, J., Quiñones, E., Rochange, C., Vardanega, T., and Cazorla, F. Contention in multicore hardware shared resources: Understanding of the state of the art. In WCET (2014).

[4.15] Galizzi, J., Vigeant, F., Perraud, L., Crespo, A., Masmano, M., Carrascosa, E., Brocal, V., Balbastre, P., Quartier, F., and Milhorat, F. Wcet and multicores with tsp. In DASIA 2014 DAta Systems In Aerospace (2014).

[4.16] Garey, M. R., and Johnson, D. S. Computers and Intractability; A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York, NY, USA, 1990.

[4.17] Guo, Z., Yang, K., Yao, F., and Awad, A. Inter-task cache interference aware partitioned real-time scheduling. In Proceedings of the 35th Annual ACM Symposium on Applied Computing (New York, NY, USA, 2020), SAC '20, Association for Computing Machinery, p. 218–226.

[4.18] Igarashi, S., Ishigooka, T., Horiguchi, T., Koike, R., and Azumi, T. Heuristic contention-free scheduling algorithm for multi-core processor using let model. In 2020 IEEE/ACM 24th International Symposium on Distributed Simulation and Real Time Applications (DS-RT) (2020), pp. 1–10.

[4.19] Johnson, D. Near-Optimal Bin Packing Algorithms. PhD thesis, Massachusetts Institute of Technology, Dept. of Math., 1973.

[4.20] Kim, H., De Niz, D., Andersson, B., Klein, M., Mutlu, O., and Rajkumar, R. Bounding and reducing memory interference in cots-based multi-core systems. Real-Time Syst. 52, 3 (May 2016), 356–395.

[4.21] Liu, C. L., and Layland, J. W. Scheduling algorithms for multiprogramming in a hard-real-time environment. J. ACM 20, 1 (Jan. 1973), 46–61.

[4.22] Maiza, C., Rihani, H., Rivas, J. M., Goossens, J., Altmeyer, S., and Davis, R. I. A survey of timing verification techniques for multi-core real-time systems. ACM Comput. Surv. 52, 3 (jun 2019).

[4.23] Mitra, T., Teich, J., and Thiele, L. Time-critical systems design: A survey. IEEE Design Test 35, 2 (2018), 8–26.

[4.24] Nowotsch, J. Interference-sensitive Worst-case Execution Time Analysis for Multi-core Processors. PhD thesis, November 2014.

[4.25] Oh, Y., and Son, S. H. Allocating fixed-priority periodic tasks on multiprocessor systems. Real-Time Syst. 9, 3 (Nov. 1995), 207–239.

[4.26] Pellizzoni, R., Betti, E., Bak, S., Yao, G., Criswell, J., Caccamo, M., and Kegley, R. A predictable execution model for cots-based embedded systems. In 2011 17th IEEE Real-Time and Embedded Technology and Applications Symposium (2011), pp. 269–279.

[4.27] Pellizzoni, R., Schranzhofer, A., Jian-Jia Chen, Caccamo, M., and Thiele, L. Worst case delay analysis for memory interference in multicore systems. In 2010 Design, Automation Test in Europe Conference Exhibition (DATE 2010) (2010), pp. 741–746.

[4.28] Reder, S., and Becker, J. Interference-aware memory allocation for real-time multi-core systems. 2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS) (2020), 148–159.

[4.29] Rouxel, B., Derrien, S., and Puaut, I. Tightening contention delays while scheduling parallel applications on multi-core architectures. ACM Trans. Embed. Comput. Syst. 16, 5s (Sept. 2017).

[4.30] Skalistis, S., and Kritikakou, A. Dynamic Interference-Sensitive Run-time Adaptation of Time-Triggered Schedules. In ECRTS 2020 - 32nd Euromicro Conference on Real-Time Systems (July 2020), pp. 1–22.

Capítulo 5

Artículo: Schedulability analysis of dynamic priority real-time systems with contention

5.1. 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.

5.2. 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 [5.10, 5.16, 5.21].

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 [5.1] 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.

Contribution. This paper proposes a schedulability test for multicore real-time systems. We extend the model in [5.1] 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 Section 5.3. Section 5.4 presents the system model for constrained-deadline systems. Section 5.5 reviews the classical schedulability analysis for multicore 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 Section 5.6 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.

5.3. 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 [5.14], [5.16], [5.17], [5.25] and [5.26].

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 [5.25] 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 ([5.11], [5.23]), scratchpad memories and DRAM ([5.22], [5.34], [5.19]), etc. For example, [5.19] 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, [5.2] 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 [5.12] or [5.30] come from the MRTA framework.

In [5.20], 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.

[5.7] 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 [5.3]. 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 [5.1], 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 [5.18], 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 [5.1] but in [5.1] a general model is considered, valid for any type of shared hardware resource while in [5.18] only interference due to cache sharing is considered.

A similar work is presented in [5.15]. 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 [5.1] (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.

5.4. Problem definition and task model

This task model is similar to the one presented in [5.1]. 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 $(M_0, M_1, M_2, ..., M_{m-1})$ 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:

$$\tau_i = (C_i, D_i, T_i, I_i) \tag{5.1}$$

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 [5.1].

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, $\tau_{M_0} \cup ... \cup \tau_{M_{m-1}} = \tau$.

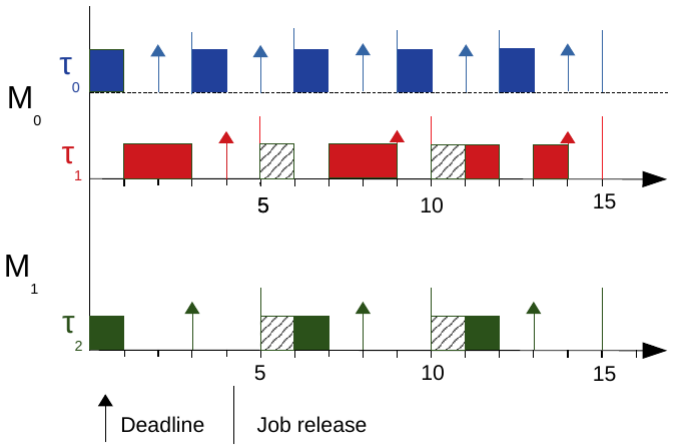


Figura 5.1: Example. Execution of the task set with $\tau_0 = (1, 2, 3, 0)$, $\tau_1 = (2, 4, 5, 1)$, and $\tau_2 = (1, 3, 5, 1)$ allocated to a dual-core platform.

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 9. *[5.1] 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 10. *[5.1] 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 5.1 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 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.

5.5. 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 monocore 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.

5.5.1. Earliest Deadline First schedulability analysis

Dynamic priority-based schedulers do not assign an initial priority to the tasks but at run-time. Earliest Deadline First (EDF) in [5.24] 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, 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 11. *[5.5] The maximum cumulative execution time requested by a set of synchronous tasks τ_{M_k} over any interval of length t is:*

$$dbf_{\tau_{M_k}}(t) = \sum_{\forall \tau_i \in M_k} C_i \left\lfloor \frac{t + T_i - D_i}{T_i} \right\rfloor \tag{5.2}$$

Therefore, the task set τ_{M_k} is schedulable by EDF if and only if [5.6]:

$$dbf_{\tau_{M_k}}(t) \leq t \quad \forall t \leq H \tag{5.3}$$

However, studying the demand bound function over all the hyperperiod H is a tedious process. Some schedulability tests as in [5.4] reduce the time interval in which the schedulability condition must be satisfied. In 1996, [5.33] and [5.31] 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 [5.33], [5.31] Then, the schedulability condition is defined as:

Theorem 3. *[5.31] 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.

5.5.2. 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 5.2 shows the demand bound function for each core, according to equation 5.2. Note that this function only considers the execution times of the task set of each core and not the received interference.

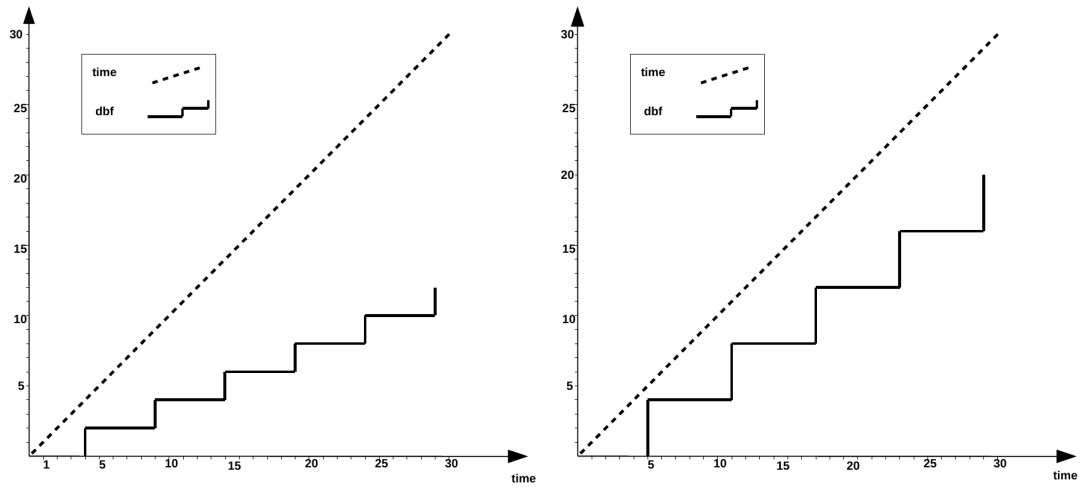


Figura 5.2: Demand bound functions for the task set, $\tau = [\tau_0, \tau_1]$ with $\tau_0 = (2, 4, 5, 1)$ and $\tau_1 = (4, 5, 6, 1)$, allocated to a dual-core platform. τ_0 is allocated to core M_0, and τ_1, to M_1. Interference is not considered: Demand bound function *dbf* for τ_{M_0} (left). Demand bound function *dbf* for τ_{M_1} (right).

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

Definition 12. *Let $\overrightarrow{v_{j \to i}}$ be the activation pattern from a broadcasting task τ_j to a receiving task τ_i.*

The meaning of this array is the number of activations of τ_j that fall within an activation a of τ_j. 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, $\overrightarrow{v_{j \to i}}[a]$ 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.

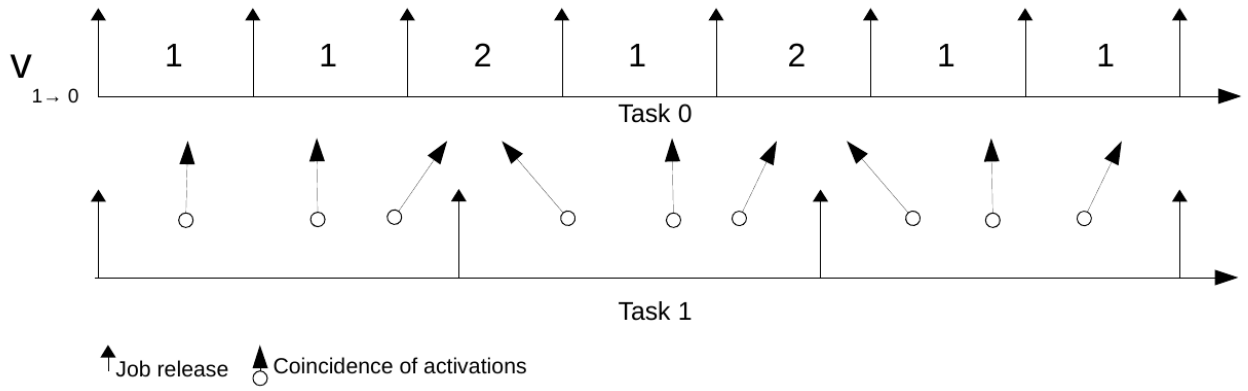


Figura 5.3: Example of $\overrightarrow{v_{j\to i}}$ for a task set with $\tau'_0 = (1, 2, 3, 1)$ and $\tau'_1 = (1, 6, 7, 1)$ allocated to a dual-core platform.

The following example shows graphically how the array $\overrightarrow{v_{j\to i}}$ 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 Figure 5.3, $\overrightarrow{v_{1\to 0}} = [1, 1, 2, 1, 2, 1, 1]$. Equivalently, $\overrightarrow{v_{0\to 1}} = [3, 3, 3]$.

Note that in the scheduling, not all activations receive the same interference from other tasks. As showed in Figure 5.1, this will depend on whether the tasks coincide in execution.

In the next theorem, we are going to provide an expression to calculate $\overrightarrow{v_{j\to i}}$ 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 4. *The maximum number of activations of the broadcasting task τ_j that fall within a^{th} activation of τ_i is:*

$$\overrightarrow{v_{j\to i}}[a] = 1 + \sum_{t=aT_i+1}^{(a+1)T_i-1} g(t) \tag{5.4}$$

being

$$g(t) = \begin{cases} 1 & \textit{If } t \textit{ - } T_j \left\lfloor \frac{t}{T_j} \right\rfloor = 0 \\ 0 & \textit{Elsewhere} \end{cases} \tag{5.5}$$

Demostración. 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 equation 5.4.

□

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

Listing 5.1 shows the pseudo-code (python-like) to calculate $\overrightarrow{v_{j\to i}}$.

Listing 5.1: Maximum interference array algorithm.

```
#variables definition and initialization
v_{j→i} = [None]A_i
for a in range(A_i):
    i = 1
    for t in range(aT_i+1,(a+1)T_i, 1):
        if t % T_j == 0 and I_i ≠ 0 and I_j ≠ 0:
            i = i + 1
    v_{j→i}[a] = i
```

In the next sections, we will use vector $\overrightarrow{v_{j\to i}}$ 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.

5.5.2.1. 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 $\overrightarrow{v_{j\to i}}$ array, this maximum interference is calculated as:

$$\max_{0\le a\le A_i-1} \overrightarrow{v_{j\to i}}[a] \cdot I_j \tag{5.6}$$

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.

$$C'_i = C_i + \sum_{\tau_j \notin M_k} \max_{0\le a\le A_i-1} \overrightarrow{v_{j\to i}}[a] \cdot I_j \tag{5.7}$$

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

Definition 13. *From Equation 5.2, we can propose a new definition of the demand bound function with interference considerations:*

$$dbf'_{\tau_{M_k}}(t) = \sum_{\forall \tau_i \in M_k} C'_i \left\lfloor \frac{t + T_i - D_i}{T_i} \right\rfloor \tag{5.8}$$

Consequently, the corresponding schedulability test is:

Theorem 5. *A task set τ_{M_k} allocated to a core M_k with constrained deadlines is schedulable by dynamic priorities if and only if:*

$$dbf'_{\tau_{M_k}}(t) \leq t \quad \forall t \leq L_b \tag{5.9}$$

Demostración. As $\overrightarrow{v_{j\to i}}[a]$ 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 $\max_a \overrightarrow{v_{j\to i}}[a] \cdot I_j$ 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.

□

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 $\overrightarrow{v_{j\to i}}[a] \cdot I_j$ 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.

5.5.2.2. A more accurate definition

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

Definition 14. *The less pessimistic demand bound function of a task set allocated to a core M_k considering the interference is:*

$$dbf''_{\tau_{M_k}}(t) = \sum_{\tau_i \in M_k} \left(C_i \left\lfloor \frac{t + T_i - D_i}{T_i} \right\rfloor + \sum_{\tau_j \notin M_k} \sum_{a=0}^{\left\lfloor \frac{t+T_i-D_i}{T_i} \right\rfloor - 1} \overrightarrow{v_{j\to i}}[a] \cdot I_j \right) \tag{5.10}$$

Corollary 5.1. *$dbf''_{\tau_{M_k}}(t)$ is less pessimistic than $dfb'_{\tau_{M_k}}(t)$. Therefore:*
$dbf''_{\tau_{M_k}}(t) \leq dfb'_{\tau_{M_k}}(t)$

Demostración. It is easy to see that:

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

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

$$\overrightarrow{v_{j\to i}}[0]+\overrightarrow{v_{j\to i}}[1]+...+\overrightarrow{v_{j\to i}}\left[\left\lfloor\frac{t+T_i-D_i}{T_i}\right\rfloor-1\right]\leq maxV+maxV+...+maxV$$

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

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

□

The second term in Equation 5.10 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 $\overrightarrow{v_{j\to i}}[a]=0\quad\forall j,a$). If τ_j is not broadcasting, $I_j=0$ and also $\overrightarrow{v_{j\to i}}[a]=0$.

This is an upper bound in the sense that the $\overrightarrow{v_{j\to i}}[a]$ array is maximum, as demonstrated in Theorem 4.

However, when interference is considered throughout the $\overrightarrow{v_{j\to i}}[a]$ 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.

5.5.2.3. Counterexample for dynamic priorities.

Let us consider the task set defined in the beginning of Section 5.5.2, $\tau=[\tau_0,\tau_1]$ 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 Figure 5.4. From [5.31], 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 Figure 5.4, 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.

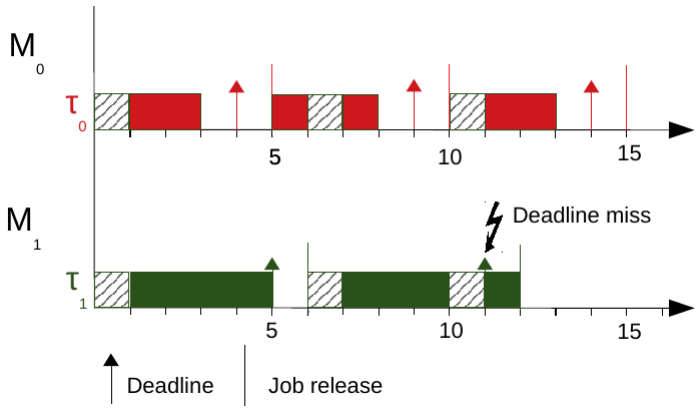


Figura 5.4: Counterexample. Execution of the task set with $\tau_0 = (2, 4, 5, 1)$ and $\tau_1 = (4, 5, 6, 1)$ allocated to a dual-core platform.

The schedulability condition for constrained-deadline synchronous and periodic task models based on the demand bound function was presented in Equation 5.3. 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 $\overrightarrow{v_{j\to i}}$, the model is still periodic, as this maximum value is introduced in all the activations. However, if we consider the array $\overrightarrow{v_{j\to i}}$ 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 [5.6] and [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 6. *A task set τ_{M_k} allocated to a core M_k with constrained deadlines is schedulable by dynamic priorities if:*

$$dbf''_{\tau_{M_k}}(t_1, t_2) \leq t_2 - t_1 \quad 0 \leq t_1 < t_2 \leq H \tag{5.11}$$

Demostración. Similar to the proof of Theorem 5, $\overrightarrow{v_{j\to i}}[a]$ represents the maximum number of times a task τ_j can cause interference in another task τ_j at an activation a. Thus, it is not possible to receive in total more than:

$$\sum_{\tau_i \in M_k} \sum_{\tau_j \notin M_k} \sum_{a=0}^{\left\lfloor \frac{t+T_i-D_i}{T_i} \right\rfloor - 1} \overrightarrow{v_{j\to i}}[a] \cdot I_j \tag{5.12}$$

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 Section 5.5.2.3. Figure 5.5 shows the demand bound functions dbf, dbf' and dbf'' for both cores. First, the activation patterns are calculated: $\overrightarrow{v_{1\to 0}} = [1, 2, 2, 2, 2, 1]$ and $\overrightarrow{v_{0\to 1}} = [2, 2, 2, 2, 2]$. Figure 5.5 show the demand bound functions for cores 0 and 1. In Figure 5.5, in the right graph, $dbf' = dbf''$ as $\max_{0\leq a\leq A_i-1} \overrightarrow{v_{j\to i}}[a] = \overrightarrow{v_{j\to i}}[a] \quad \forall a$.

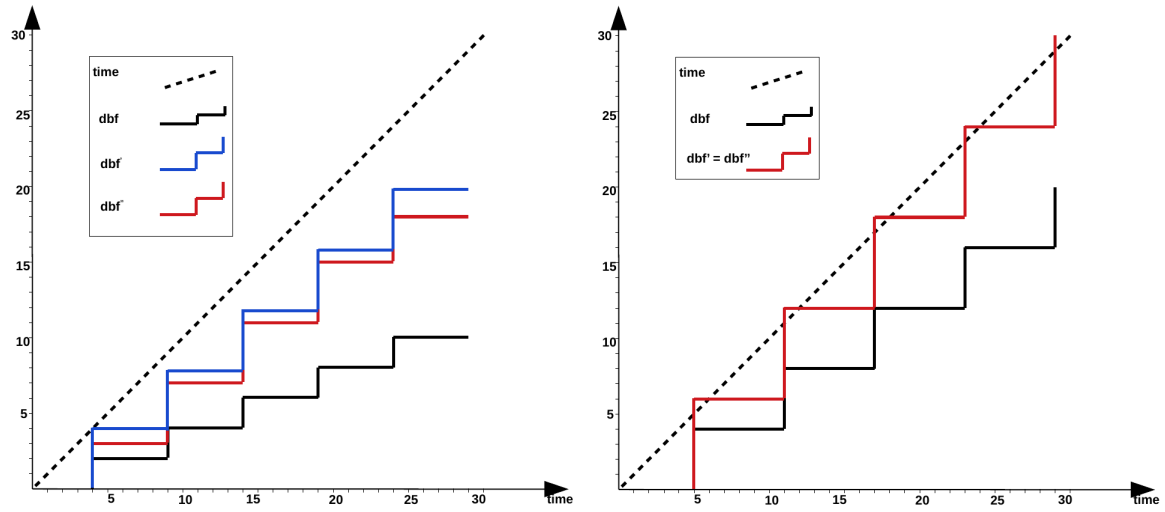


Figura 5.5: Relation between dbf', dbf'' and schedulability of the task set in Section 5.5.2.3: dbf' and dbf'' for τ_{M_0} (left). dbf' and dbf'' for τ_{M_1} (right).

As seen in Figure 5.4, there is a deadline missed in the execution of the tasks in core M_1. This infeasibility is demonstrated with the demand bound functions (Figure 5.5, in the right graph) as $dbf'_{\tau_{M_1}}(0, t) > t$ and $dbf''_{\tau_{M_1}}(0, t) > t$ for some instants of time during the hyperperiod.

5.6. Evaluation

5.6.1. 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 Figure 5.6. It is divided into three steps:

- Generation of the load (see Section 5.6.1.1).
- Allocation phase (see Section 5.6.1.2).

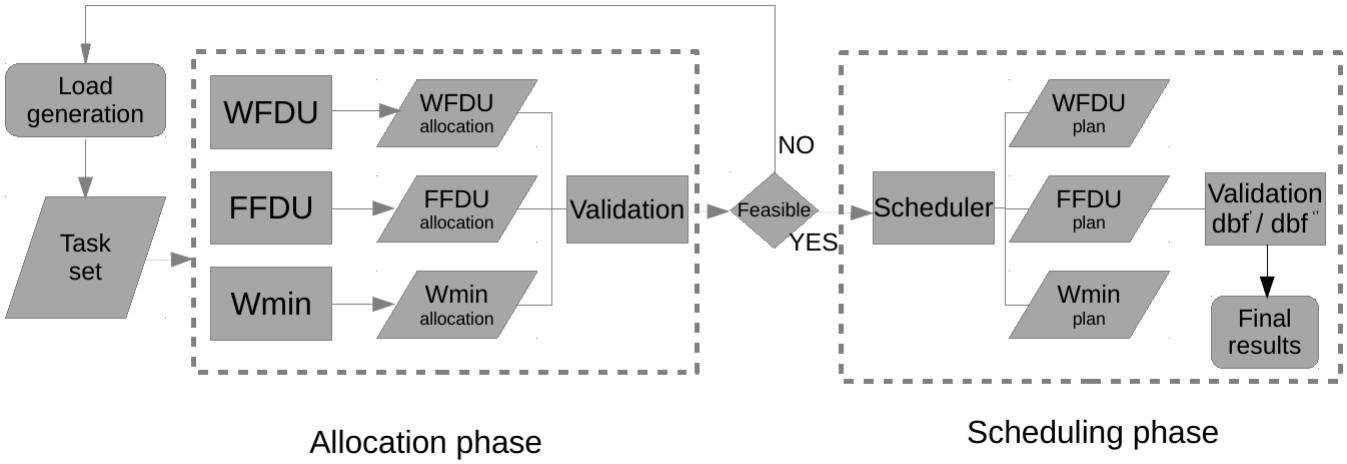


Figura 5.6: Experimental evaluation overview.

- Scheduling phase (see Section 5.6.1.3).

5.6.1.1. 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 [5.13]. 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 5.1. To ensure the reproducibility of the results, these parameters coincide with those used in [5.1](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 (≈50 %) and 3 (75 %).

The number of broadcasting tasks is set to 25 % of the total number of tasks, except for scenarios 1 to 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.

5.6.1.2. 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 ([5.8, 5.28]). Moreover, task ordering such as decreasing utilisation (DU)

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

Tabla 5.1: Definition of the experimental scenarios.

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. [5.9] 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 [5.1], 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 Section 5.5.2.1. 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., $U_{M_k} \leq 1 \forall k = 0, .., m-1$ 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.

5.6.1.3. Scheduling phase

The contention aware scheduling algorithm proposed in [5.1] 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:

$$U'_{M_k} = \frac{dbf'_{\tau_{M_k}}(H)}{H} \tag{5.13}$$

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

$$U'_{\tau} = \sum_{\forall k} U'_{M_k} \tag{5.14}$$

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

$$U''_{M_k} = \frac{dbf''_{\tau_{M_k}}(H)}{H} \tag{5.15}$$

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

$$U''_{\tau} = \sum_{\forall k} U''_{M_k} \tag{5.16}$$

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 $U_\tau^{''}$, as not always the estimated interference will be produced, as stated in Section 5.5.2.3. From previous sections (see Corollary 5.1), it is easy to deduce that:

$$U_\tau \leq U_\tau^{real} \leq U_\tau^{''} \leq U_\tau' \tag{5.17}$$

5.6.2. 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 5.1. The objective is to compare the allocation algorithms and confirm that no set with $U^{real} > U_\tau^{''}$ 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 $U_\tau^{''}$ and U_τ^{real}, measured as $\alpha^{''} = \frac{U_\tau^{''} - U_\tau^{real}}{U_\tau^{real}}(\%)$.

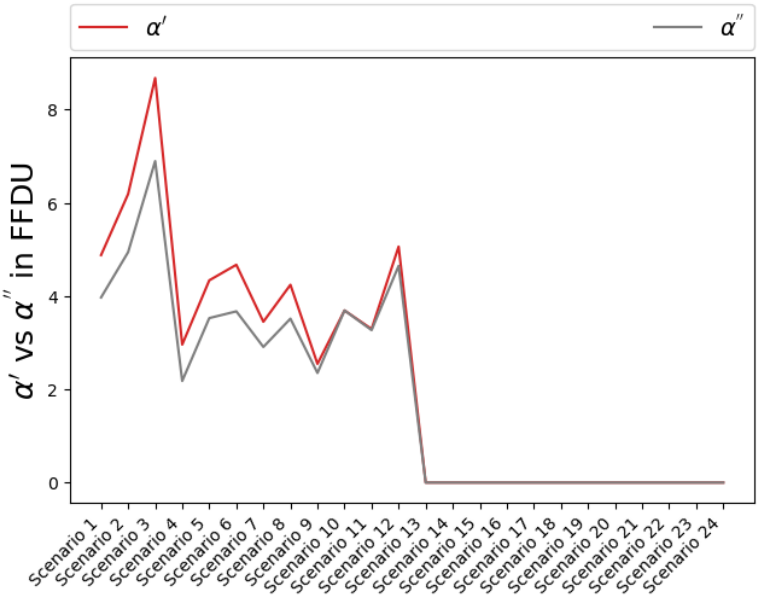


Figura 5.7: Percentage difference α' vs $\alpha^{''}$ for each scenario with FFDU allocator.

Figures 5.7, 5.8 and 5.9 depict the percentage difference α' and $\alpha^{''}$ for each scenario and different allocators. As expected, for all scenarios and all allocators $\alpha^{''} \leq \alpha'$, as a consequence of Equation 5.17. As a general trend, the more cores in the system, the bigger α' vs $\alpha^{''}$ 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:

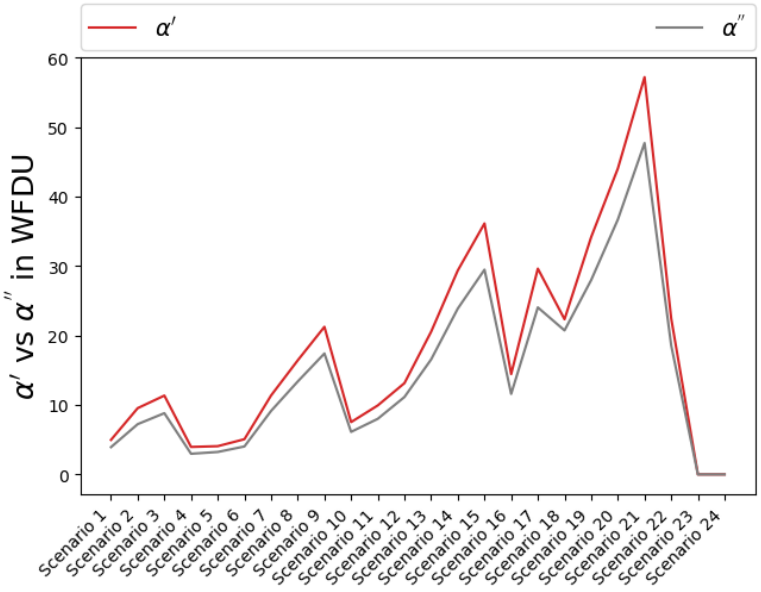


Figura 5.8: Percentage difference α' vs α'' for each scenario with WFDU allocator.

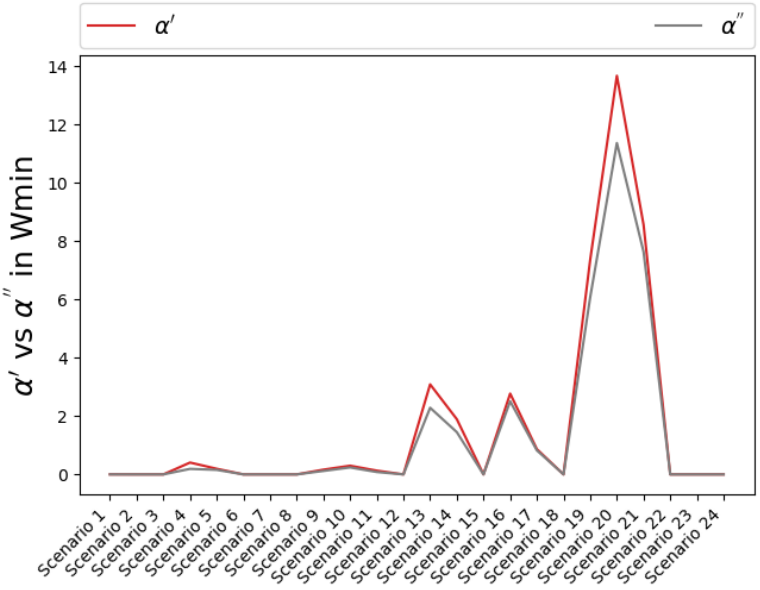


Figura 5.9: Percentage difference α' vs α'' for each scenario with Wmin allocator.

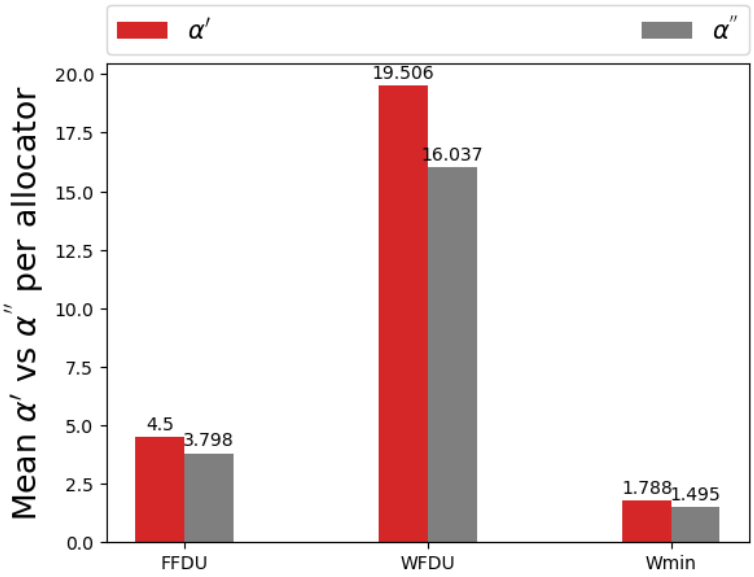


Figura 5.10: Average percentage α' vs α'' for each allocator.

- FFDU (Figure 5.7): 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 [5.1]). 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 Section 5.6.1.1, so α' and α'' are bigger than in the rest of scenarios.

- WFDU (Figure 5.8): 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 [5.1]) 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, 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 (Figure 5.9): 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 can not be scheduled when interference appears. So α' and α'' are zero.

Figure 5.10 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).

5.7. 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 ([5.27]) or Modified Maximum Urgency First ([5.32]), can achieve promising results with respect to interference.

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

Bibliografía

[5.1] Aceituno, J. M., Guasque, A., Balbastre, P., Simó, J., and Crespo, A. Hardware resources contention-aware scheduling of hard real-time multiprocessor systems. Journal of Systems Architecture 118 (2021).

[5.2] Altmeyer, S., Davis, R. I., Indrusiak, L., Maiza, C., Nelis, V., and Reineke, J. A generic and compositional framework for multicore response time analysis. In Proceedings

of the 23rd International Conference on Real Time and Networks Systems (New York, NY, USA, 2015), RTNS '15, Association for Computing Machinery, p. 129–138.

[5.3] Andersson, B., Kim, H., Niz, D. D., Klein, M., Rajkumar, R. R., and Lehoczky, J. Schedulability analysis of tasks with corunner-dependent execution times. ACM Trans. Embed. Comput. Syst. 17, 3 (may 2018).

[5.4] Baruah, S. K., Howell, R. R., and Rosier, L. E. Feasibility problems for recurring tasks on one processor. Theoretical Computer Science 118, 1 (1993), 3 – 20.

[5.5] Baruah, S. K., Mok, A. K., and Rosier, L. E. Preemptively scheduling hard-real-time sporadic tasks on one processor. In [1990] Proceedings 11th Real-Time Systems Symposium (Dec 1990), pp. 182–190.

[5.6] Baruah, S. K., Rosier, L. E., and Howell, R. R. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. Real-Time Syst. 2, 4 (Oct. 1990), 301–324.

[5.7] Choi, J., Kang, D., and Ha, S. Conservative modeling of shared resource contention for dependent tasks in partitioned multi-core systems. In 2016 Design, Automation Test in Europe Conference Exhibition (DATE) (2016), pp. 181–186.

[5.8] Coffman, E. G., Garey, M. R., and Johnson, D. S. Approximation Algorithms for Bin Packing: A Survey. PWS Publishing Co., USA, 1996, p. 46–93.

[5.9] Coffman Jr., E. G., Csirik, J., Galambos, G., Martello, S., and Vigo, D. Bin Packing Approximation Algorithms: Survey and Classification. Springer New York, New York, NY, 2013, pp. 455–531.

[5.10] Dasari, D., Akesson, B., Nélis, V., Awan, M. A., and Petters, S. M. Identifying the sources of unpredictability in cots-based multicore systems. In 2013 8th IEEE International Symposium on Industrial Embedded Systems (SIES) (2013), pp. 39–48.

[5.11] Dasari, D., Andersson, B., Nelis, V., Petters, S. M., Easwaran, A., and Lee, J. Response time analysis of cots-based multicores considering the contention on the shared memory bus. In 2011IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications (2011), pp. 1068–1075.

[5.12] Davis, R., Altmeyer, S., Indrusiak, L., Maiza, C., Nelis, V., and Reineke, J. An extensible framework for multicore response time analysis. Real-Time Systems 54 (07 2018).

[5.13] Davis, R. I., and Burns, A. Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. In 2009 30th IEEE Real-Time Systems Symposium (Dec 2009), pp. 398–409.

[5.14] Davis, R. I., and Burns, A. A survey of hard real-time scheduling for multiprocessor systems. ACM Comput. Surv. 43, 4 (Oct. 2011).

[5.15] Davis, R. I., Griffin, D., and Bate, I. Schedulability Analysis for Multi-Core Systems Accounting for Resource Stress and Sensitivity. In 33rd Euromicro Conference on Real-Time Systems (ECRTS 2021) (Dagstuhl, Germany, 2021), B. B. Brandenburg, Ed., vol. 196 of Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl – Leibniz-Zentrum für Informatik, pp. 7:1–7:26.

[5.16] Fernandez, G., Abella, J., Quiñones, E., Rochange, C., Vardanega, T., and Cazorla, F. Contention in multicore hardware shared resources: Understanding of the state of the art. In WCET (2014).

[5.17] Gracioli, G., Alhammad, A., Mancuso, R., Fröhlich, A. A., and Pellizzoni, R. A survey on cache management mechanisms for real-time embedded systems. ACM Comput. Surv. 48, 2 (nov 2015).

[5.18] Guo, Z., Yang, K., Yao, F., and Awad, A. Inter-task cache interference aware partitioned real-time scheduling. In Proceedings of the 35th Annual ACM Symposium on Applied Computing (New York, NY, USA, 2020), SAC '20, Association for Computing Machinery, p. 218–226.

[5.19] Hassan, M., and Pellizzoni, R. Analysis of Memory-Contention in Heterogeneous COTS MPSoCs. In 32nd Euromicro Conference on Real-Time Systems (ECRTS 2020) (2020), M. Völp, Ed., vol. 165 of Leibniz International Proceedings in Informatics (LIPIcs), pp. 23:1–23:24.

[5.20] Huang, W.-H., Chen, J.-J., and Reineke, J. Mirror: Symmetric timing analysis for real-time tasks on multicore platforms with shared resources. In Proceedings of the 53rd Annual Design Automation Conference (New York, NY, USA, 2016), DAC '16, Association for Computing Machinery.

[5.21] Karuppiah, N. The impact of interference due to resource contention in multicore platform for safety-critical avionics systems. International Journal for Research in Engineering Application Management (IJREAM) 02 (01 2016), 39–48.

[5.22] Kim, H., de Niz, D., Andersson, B., Klein, M., Mutlu, O., and Rajkumar, R. Bounding memory interference delay in cots-based multi-core systems. In 2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS) (2014), pp. 145–154.

[5.23] Lampka, K., Giannopoulou, G., Pellizzoni, R., Wu, Z., and Stoimenov, N. A formal approach to the wcrt analysis of multicore systems with memory contention under phase-structured task sets. Real-Time Systems 50 (11 2014), 736–773.

[5.24] Liu, C. L., and Layland, J. W. Scheduling algorithms for multiprogramming in a hard-real-time environment. J. ACM 20, 1 (Jan. 1973), 46–61.

[5.25] Maiza, C., Rihani, H., Rivas, J. M., Goossens, J., Altmeyer, S., and Davis, R. I. A survey of timing verification techniques for multi-core real-time systems. ACM Comput. Surv. 52, 3 (jun 2019).

[5.26] Mitra, T., Teich, J., and Thiele, L. Time-critical systems design: A survey. IEEE Design Test 35, 2 (2018), 8–26.

[5.27] Oh, S.-H., and Yang, S.-M. 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) (1998), pp. 31–36.

[5.28] Oh, Y., and Son, S. H. Allocating fixed-priority periodic tasks on multiprocessor systems. Real-Time Syst. 9, 3 (Nov. 1995), 207–239.

[5.29] Pellizzoni, R., and Lipari, G. Feasibility analysis of real-time periodic tasks with offsets. Real-Time Systems 30 (05 2005), 105–128.

[5.30] Rihani, H., Moy, M., Maiza, C., Davis, R. I., and Altmeyer, S. 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 (2016), RTNS '16, p. 67–76.

[5.31] Ripoll, I., Crespo, A., and Mok, A. K. Improvement in feasibility testing for real-time tasks. Real-Time Syst. 11, 1 (July 1996), 19–39.

[5.32] Salmani, V., Taghavi Zargar, S., and Naghibzadeh, M. A modified maximum urgency first scheduling algorithm for real-time tasks. Proc. Seventh World Enformatika Conference (01 2005).

[5.33] Spuri, M. Analysis of deadline scheduled real-time systems. Tech. rep., 1996.

[5.34] Xiao, J., Altmeyer, S., and Pimentel, A. Schedulability analysis of non-preemptive real-time scheduling for multicore processors with shared caches. In 2017 IEEE Real-Time Systems Symposium (RTSS) (2017), pp. 199–208.

Capítulo 6

Artículo: Optimized Scheduling of Periodic Hard Real-Time Multicore Systems

6.1. Abstract

Multicore systems were developed to provide a substantial performance increase over monocore systems. But shared hardware resources are a problem as they add unpredictable delays that affect the schedulability of multicore hard real-time systems. In recent years much effort has been put into modelling interference and proposing scheduling techniques to mitigate its negative effect. Using one of these models, we propose a scheduling technique, based on Integer Linear Programming (ILP) that, in combination with a task to core allocator, not only achieves a feasible schedule but also reduces the interference produced by shared hardware resources in the context of hard real-time multicore systems. The evaluation shows how both the interference is reduced and the schedulability is increased compared to traditional heuristics.

6.2. Introduction

The use of embedded systems is widespread, not only in the industrial sector, but in all aspects of modern life. The processing power of multicore systems allows multiple embedded applications to be used on a single shared hardware platform.

In sectors where applications are highly critical, no failure is allowed as it can have catastrophic consequences. In such applications, due to certification requirements, the allocation of all resources must be static [6.4]. In this paper we will focus on the static allocation of temporal resources, i.e. on the scheduling of multicore systems for highly critical applications.

Existing theory in the field of multicore systems shows that scheduling on such systems is complex (NP-Hard). The generation of cyclic plans from the temporal requirements of several applications in a multicore system requires the use of techniques and heuristics that attempt to achieve feasible schedules in limited time. A large number of additional elements such as the

different criticality of the applications, the assignment of tasks to cores, the management of energy consumption, the optimisation of the operating system performance, etc., adds to the complexity and difficulty of generating feasible and efficient schedules.

But there is not only difficulty in generating a feasible plan in multicore systems, it is also difficult to estimate the computation time of the tasks. In multicore systems, there are some sources of indeterminism due to the use of shared hardware resources such as memory, memory bus or cache [6.6]. This causes contention between tasks in different cores, which is reflected in delays in task execution. These delays, also called interferences, are non-deterministic and pose a challenge in multicore scheduling techniques. The position paper CAST32A on multicore processors [6.4] identifies topics that could impact the safety, performance and integrity of airborne software systems and lists a set of objectives to help addressing multicore certification challenges. One of these challenges is interference due to contention, that is absorbing considerable research efforts in both real-time industry and academic community.

In recent years much effort has been put into modelling interference and proposing scheduling techniques to mitigate its negative effect. Two different approaches are commonly used to model interference: using a model specific to the type of shared hardware or proposing a general model that is valid for any type of hardware. The former gives a more accurate interference value but it is only valid for the hardware for which it has been calculated. Moreover, this interference value is added to the Worst Case Execution Time (WCET), making it a very pessimistic solution. The latter obtains a higher value of the interference but by adding this parameter to the temporal model independently of the WCET, it is possible to obtain a less pessimistic model. But it is necessary to propose a temporal model to incorporate this new interference parameter.

Regarding the scheduling of such systems, there are also numerous works that have addressed the scheduling and schedulability of partitioned real-time multicore systems (migration is not allowed, as the allocation must be static). This scheduling is done in two phases: first, the tasks are assigned to the cores, and then each core schedules its tasks. If interference is not taken into account, the scheduling of each core is independent. But if interference is taken into account, the execution of tasks in one core affects the scheduling of the other cores. This way, the timing correctness of the hard real-time system becomes more complex.

In this work we propose a scheduling technique, based on Integer Linear Programming (ILP) that, in combination with a task to core allocator, not only obtains a feasible schedule but reduces the interference produced by shared hardware resources in the context of hard real-time multicore systems.

This paper is organised as follows: Section 6.3 briefly comments relevant papers in the area, Section 6.4 defines the temporal model used and the problem to solve while in Section 6.5 a first approximation of the ILP model is presented. In order to improve the presented model, Section 6.6 presents the concept of busy period in partitioned multicore systems and Section 6.7 proposes a scheduling technique to reduce interference. With the results of the previous sections, in Section 6.8, we present a ILP technique that obtains a scheduling plan that minimizes interference. The evaluation is presented in Section 6.9 and conclusions and further work is detailed in Section 6.10.

6.3. 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 [6.9] and [6.11].

This work is focused in partitioned hard real-time systems that take into account interference due to contention. Two surveys about interference and mitigation of its effect in the scheduling can be found in [6.21] (until 2018) and in [6.20] (until 2021).

Regarding contention models (specific or general), many works consider a specific shared resource: memory bus ([6.7, 6.17]), scratchpad memories and DRAM ([6.16, 6.26]), etc. However, other works consider multiple shared resources in the contention (general model), which is the approach followed in this work. Dasari et al [6.6] study the sources of unpredictability produced by shared hardware resources. Altmeyer et al. [6.2] 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 [6.8] or [6.24] come from the MRTA framework. In [6.15], 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 that task priority levels are assigned *a priori*. Choi et al. in [6.5] 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. In [6.1], 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, no action is taken in the scheduling phase to reduce interference, only in the phase of allocating tasks to cores. Our aim is to extend this work to obtain a schedule hat minimizes interference. A similar work is presented in [6.10]. 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 (preemptive and non-preemptive). They propose a schedulability analysis and a priority assignment to maximize schedulability but no actions are taken to reduce interference.

Regarding the use of ILP techniques in real-time systems, Guasque et al. [6.13] proposes a technique based on rolling horizons for monocore systems. In multicore systems, most works that use ILP for scheduling do so with the objective of minimising power consumption [6.14, 6.22]. In [6.3] a new method for solving complex scheduling problems of real-time in multiprocessor platforms is proposed using a directed acyclic graph (DAG) to represent the scheduling of the workload, where each vertex represents a processor of the system.

6.4. Task model and problem statement

6.4.1. Periodic task model

Let us suppose a multicore system with m cores $(M_0, M_1, M_2, ..., M_{m-1})$ where a task set τ of n independent tasks should be allocated to. Each task τ_i is represented by the tuple:

$$\tau_i = (C_i, D_i, T_i, I_i) \tag{6.1}$$

where C_i is the worst-case execution time, D_i is the relative deadline, T_i is the period, and I_i is the interference factor over other tasks. Constrained deadlines are considered, so $D_i \leq T_i \quad \forall i$.

A detailed description of the interference factor parameter can be found in [6.1] but, because of its importance, we will illustrate how to schedule the task model with a simple example. Let us assume a system with 2 cores: M_0 and M_1, and 3 tasks: τ_0, τ_1, τ_2 and τ_3 with the following parameters: $\tau_0 = (1, 3, 3, 1)$, $\tau_1 = (1, 7, 7, 1)$, $\tau_2 = (1, 21, 21, 0)$. We say that τ_0 and τ_1 are broadcasting tasks since its $I_i \neq 0$.Tasks τ_0 and τ_2 are allocated to core M_0 and τ_1 is allocated to core M_1. Figure 6.1 shows the scheduling of the example under Earliest Deadline First (EDF) algorithm. The novelty is that when two tasks in different cores coincide in execution, they cause each other a delay that is the interference factor, if it is not 0. That is why τ_0 suffers 1 unit of interference (blue dash lines) that corresponds with I_1. And vice versa, τ_1 suffers 1 unit of interference (red dash lines) that corresponds with I_0.

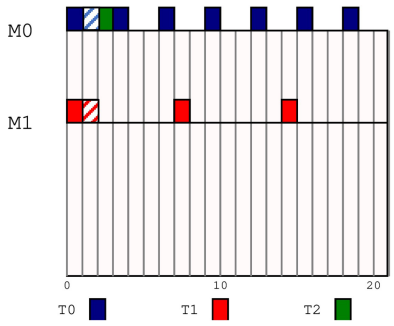


Figura 6.1: Example of chronogram under EDF

The theoretical utilisation of a core M_k, U_{M_k}, is the fraction of processor time spent executing the tasks allocated to this core. It is calculated by summing up the theoretical utilisation of each task τ_i that belongs to that core ($U_i = C_i/T_i$). In a multicore system, the utilisation of a core does not only depend on the processor time spent executing the computation time of the tasks but also on the interference produced when tasks are executed simultaneously on different cores. For this reason, this paper defines the actual utilisation of a core M_k, U'_{M_k}, as the sum of the actual utilisations (U'_i) of the tasks that belong to that core. Therefore, the theoretical system utilisation U_τ is the sum of the utilisations of all cores and the actual system utilisation is denoted by U'_τ and takes into consideration the interference.

The hyperperiod of the task set, H, is the smallest interval of time after which the periodic patterns of all the tasks are repeated. It is calculated as the least common multiple of the tasks'

periods and it is denoted as lcm. During the hyperperiod, each task τ_i is activated $N_i = lcm/T_i$ times.

6.4.2. Problem statement

As explained in the previous section, tasks can suffer delays due to interference of other co-runners tasks. This interference is unpredictable in the sense that depends on the tasks that are running in different cores at each instant of time. We model this delay in the worst case, so it can cause that a system becomes unfeasible if not taken into account. Different scheduling decisions can cause different amounts of interference. For example, Figure 6.2 shows the same example of Figure 6.1 but with a slightly different order of execution. Specifically, τ_2 is executed before τ_0. This means that now, τ_0 and τ_1 are not co-runners so they do not caused interference to each other. The result is that, by slightly changing the execution order, we have reduced the interference to 0. The resulting schedule does not correspond to any optimal priority assignment.

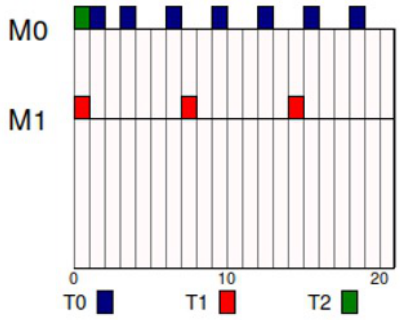


Figura 6.2: Example of chronogram

The aim of this work is to generate a static scheduling plan for multiprocessor systems. Integer Linear Programming (ILP) techniques will be used to find a scheduling plan that reduces as much as possible the total system interference due to accesses to the shared hardware. We will assume that tasks are already allocated to processors and they cannot migrate.

6.5. Multiprocessor LP scheduling

The first approximation to generate a static scheduling plan for multicore systems is to propose a complete LP model. This model addresses the multicore scheduling problem, assuming that the allocation phase has already been done.

Every optimization model has an objective function, which is the function on the decision variables that we wish (in this case) to minimize. The decision variables capture the results of the optimization. In a feasible solution, the computed values for the decision variables satisfy all of the model constraints. Finally, constraints capture a restriction on the values that a set of variables may take.

If we now apply it to our problem, the decision variables are: the tasks that are executed at each time on each core (binary, executed or not), the interference (binary, exists or not), and the response time of each task (continuous, time units). Then, the model we are dealing with is of the mixed integer type because it copes with integer and continuous variables. Our constraints are based on schedulability criteria: tasks must end before the arrival of their deadlines, multiple tasks can not be executed simultaneously in the same processor, etc. All constraints are linear and therefore, the problem is categorized as Mixed Integer Linear Problem (MILP).

In the following, we propose a complete MILP model that solves the multiprocessor scheduling core with interference. First, in Table 6.1 the notation is proposed.

The objective function to be solved is to minimize the interference and the response time of the tasks. The reason why the combination of two parameters has been chosen is because if we minimise interference alone, plans with many context switches could be produced:

$$\text{mín} \quad \text{Obj} = \frac{1}{maxI} \sum_{\substack{\forall i,k \in \tau \\ \forall a \in N_i \\ \forall b \in N_j}} m_{iakb} + \sum_{\forall i,a} \frac{w_{ia}}{D_i} \tag{6.2}$$

According to the problem statement, the objective function is defined in Equation 6.2, which is minimizing the total number of interferences (first term) and the total response time for all tasks in all system executions (second term). The problem is considered as multiobjective because it tries to reduce both interferences and response times.

Therefore, it is important to note that minimization is not only about interference, but also about optimising interference and response times. Therefore, it would not be correct to talk about interference minimisation because we could obtain a plan with less interference but with many context switches, which is not desirable. Moreover, as we will see below, there will be cases where a solution cannot be found in a reasonable time, which is why alternatives are proposed in the following sections.

The range of values of interferences and response times are different. Thus, we need to normalize both variables to be of a similar scale. We scale response times by dividing by deadlines for each activation and task. The sum of interferences is scaled by the maximum possible interferences. In this way, both values are in the range [0,1] and there is a fair trade-off between competing objectives.

The maximum possible interference in the system is calculated from the array $\overrightarrow{v_{j \to i}}$, which is the activation pattern from a broadcasting task τ_j to a receiving task τ_i and was defined in [6.12]. As a consequence of calculating the value of this array in all activations in all receiving tasks, the value of the maximum total received interferences, $maxI$ is obtained (Equation 6.3). As this is a pessimistic value, dividing the real number of interferences between $maxI$ scales the interference objective in the range [0,1].

$$maxI = \sum_{\substack{\forall i,k \in \tau \\ M_{\tau_i} \neq M_{\tau_k} \\ I_i \neq 0}} \sum_{0 \leq a \leq N_i - 1} \overrightarrow{v_{k \to i}}[a] \cdot I_k \tag{6.3}$$

Tabla 6.1: Model notation of the MILP problem

Sets and indices	
i	Tasks $\tau_i \in \{0, 1, 2, ..., n-1\}$
a	Activations of $\tau_i \in \{0, 1, 2, ..., N_i - 1\}$
j	Cores $M_j \in \{0, 1, 2, ..., m-1\}$
Parameters	
C_i	Worst case computation time of τ_i
D_i	Deadline of τ_i
d_{ia}	Due date of τ_i in activation a
T_i	Period of τ_i
I_i	Interference factor of τ_i over other tasks
N_i	Number of activations of τ_i
R_{ia}	$[a \cdot T_i, (a+1) \cdot T_i]$ Possible execution time interval for τ_i in activation a
o_{ij}	1 if τ_i is allocated to core j and 0 otherwise.
lcm	Hyperperiod of the task set
$maxI$	Maximum possible received interferences
Decision variables	
x_{iajt}	1 if τ_i in activation a is allocated to core j and executed at time t and 0 otherwise.
m_{iakb}	Interference matrix. 1 if execution of τ_i in activation a coincides with the execution of τ_k in activation b and 0 otherwise.
w_{ia}	Response time matrix. Response time of τ_i in activation a

Algorithm 4 MILP algorithm

```
1: INPUT: Task set and task-to-cores allocation
2: OUTPUT: Scheduling plan, σ
3: procedure MILP ALGORITHM
4:     Calculate parameters
5:     add Variables
6:     add Constraints
7:     set Objective and optimize
8:     if Optimal or Feasible then
9:         σ = x_iajt (Save scheduling plan)
10:    else
11:        Solution not found
```

The constraints of the problem are defined as follows:

$$x_{iajt} = o_{ij} \quad \forall i, a, j, t \quad \text{if } o_{ij} = 0 \tag{6.4}$$

$$m_{iakb} = 0 \quad \forall i, a, k, b | k \in \tau, b \in N_k \text{ if } k > i \text{ and if } R_{ia} \cap R_{kb} = \emptyset \tag{6.5}$$

$$\sum_{t \in R_{ia}} x_{iajt} = C_i \cdot o_{ij} + \sum_{\substack{k \in \tau \\ k \neq i \\ o_{ij} \neq o_{kj}}} m_{iakb} \cdot I_k \cdot o_{ij} \quad \forall i, a, j, k, b | k \in \tau, b \in N_k \text{ if } I_i, I_k \neq 0 \tag{6.6}$$

$$t \cdot \sum_{\forall j} x_{iajt} \leq d_{ia} - 1 \forall i, a, t | t \in R_{ia} \tag{6.7}$$

$$\sum_{\forall i, a} x_{iajt} \leq 1 \qquad \forall j, t \tag{6.8}$$

$$x_{iajt} = 0 \qquad \forall i, a, j, t | t \notin R_{ia} \tag{6.9}$$

$$m_{iakb} \geq x_{iajt} + x_{kblt} - 1 \quad \forall i, a, k, b, j, t | \forall k \in \tau, b \in N_k, t \in R_{ia} \cap R_{kb}, \text{ if } i \neq k \text{ and if } j \neq l \tag{6.10}$$

$$m_{iakb} = m_{kbia} \qquad \forall i, a, k, b | k \in \tau, b \in N_k, \text{ if } k \neq i \tag{6.11}$$

$$w_{ia} \geq t \cdot x_{iajt} - aT_i + 1 \ \forall i, a, j, t | t \in R_{ia} \tag{6.12}$$

$$x_{iajt}, m_{iakb} \in \{0, 1\} \tag{6.13}$$

$$w_{ia} \geq 0 \tag{6.14}$$

Constraints 6.4 and 6.5 reduce some variables because of the known information. For example, as the allocation is known *a priori*, x matrix can be reduced. Same happens with m matrix. m_{iakb} is equal to zero when tasks τ_i and τ_k do not coincide in execution. Because of the periods, there are activations that can not possibly overlap. In particular, this can not happen when the execution times for τ_i at activation a do not intersect with the execution times for τ_k at activation b.

Constraint 6.6 ensures that all activations a of τ_i are executed in its activation intervals R_{ia}. Inside each of these intervals, the WCET and (if exists) the received interference must be executed. Equation 6.7 ensures that all tasks end before the arrival of their deadlines. Constraint 6.8 ensures that only one task is being executed at each point in time at each core. Equation 6.9 ensures that tasks are not executed outside their activation intervals. Equations 6.10 and 6.11 calculate the produced interference. When activation a of τ_i and activation b of τ_k do coincide in execution, then m_{iakb} is equal to one.

Constraint 6.12 calculates the response time of each activation of all tasks. Equations 6.13 and 6.14 represent the decision variable domains.

Algorithm 4 shows the methodology used to obtain the scheduling plan.

The previous model solves the problem of scheduling a multicore hard real-time system. However, as stated in [6.13] the problem can become intractable, as its size is directly proportional to the number of tasks, task utilisation and hyperperiod. In this case, an additional dimension is added, namely the number of cores.

To overcome this drawback, we propose an alternative to efficiently solve the problem. The primary idea is to decompose the whole scheduling window (hyperperiod) into many short subproblems, called rolling horizons. The Rolling Horizon approach is used to reduce the computational time to solve big problems with many variables. In this work, each of these rolling horizons corresponds with a busy period that will be explained in the following section.

6.6. System busy periods

A busy period BP_i is an interval of time in which the processor has ready tasks to execute. The concept was first introduced in [6.18] for monocore. A way to calculate the busy periods in EDF is presented in [6.25].

We extend the definition of busy periods applied to multicore systems:

Definition 15. *In partitioned multicore systems, a busy period (BP_i) is the time interval in which there is no idle time in all the cores simultaneously.*

Note that this definition is different for global multicore scheduling in [6.27] in which an *all busy period* is a consecutive time interval in which there are always ready jobs or waiting to be executed on every core.

According to the above definition, busy periods can be derived using the monocore busy periods.

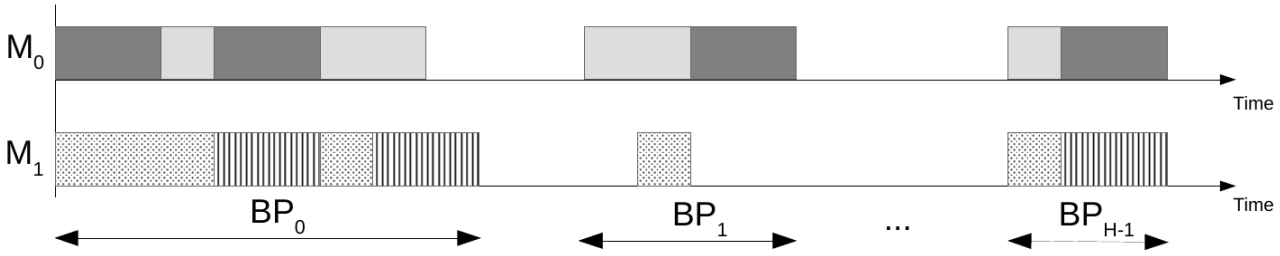


Figura 6.3: Busy period definition.

For example, Figure 6.3 shows the characterization of the busy periods for a specific scheduling plan. The end of each busy period is the time in which all cores in the system are in idle state simultaneously.

Algorithm 5 shows how to calculate the busy periods in multicore systems from the busy periods in monocore systems. It is obtained from all the execution intervals that compose the

plan, including the executions in all cores. It is a matter of combining all intervals until a point in which there is no execution. It is important to note that monocore busy periods take into account the interference caused by other cores.

Algorithm 5 Obtaining Busy Periods (BP)

```
1: INPUT: Monocore busy periods (MBP) of all tasks in all cores from the scheduling plan
   → [MBP_0, MBP_1, ...]
2: OUTPUT: Set of busy periods → [BP_0, ..., BP_{H-1}]
3: procedure OBTAINING BUSY PERIODS
4:     Temporarily ordering of the execution intervals (by starting times)
5:     Initialise list of busy periods → BP = [MBP_0]
6:     Initialise variables i←1, j←0
7:     for MBP_i ∈ Execution Intervals [MBP_1, ...] do
8:         if BP_j^e ≥ MBP_i^s then
9:             if MBP_i^e > BP_j^e then
10:                BP_j^e = MBP_i^e
11:        else
12:            j = j + 1
13:            BP_j ← MBP_i
14:        i = i + 1
```

Therefore, during a complete hyperperiod, cores alternate between idle and busy states that is, between busy and idle periods. Busy periods are independent of each other. This means that we can apply different scheduling strategies in each busy period and this does not affect the others.

Now that we have split the hyperperiod into busy periods and we know that we can schedule differently in each one without affecting the rest of the intervals, we are going to schedule each busy period with several conventional scheduling algorithms and we will choose the one that obtains the least interference. This is useful for two reasons:

- to have the calculation of each busy period that we need to develop the rolling horizon method (INPUT of Algorithm 6),
- to have a non-optimal solution in case the ILP method is not able to find one.

Note that in multicore scheduling with interference, the length of a busy period depends on the chosen scheduling algorithm. In the following section, we explain the technique of obtaining the scheduling of each busy period, which we have called the combined scheduler.

6.7. Combined Scheduler

The Combined Scheduler (CS) is a scheduling algorithm that uses or combines different scheduling policies such as EDF or Deadline Monotonic (DM) [6.19] in a single schedule. The

key of the CS is that it schedules interval by interval, and in each interval it can apply a different scheduling policy. The selected policy is the one that generates the best results for that interval at certain temporal parameters.

In the present case, the CS selects the policy that reduces the interference due to accesses to the shared hardware in each busy period. The CS chooses between the following scheduling policies: EDF, DM and their variants. The variants covered in this work are:

- Variant 1: consists of non-preempting a running task when a higher priority task is activated under certain conditions. The task is not preempted if the remaining time for the running task to finish is less than the computation time of the higher priority task. In this case, the running task would not be preempted, inheriting the priority of the task that wants to preempt it.

- Variant 2: consists of non-preempting a running task for N time units either in the case that the task is about to start execution or that the task has been previously preempted.

In sort, there are six schedulers to be chosen: classic EDF, classic DM, EDF variant 1, DM variant 1, EDF variant 2 and DM variant 2.

For a better understanding of the combined scheduler, an example is presented here. Let us assume a system with 2 cores: M_0 and M_1, and 4 tasks: τ_0, τ_1, τ_2 and τ_3 with the following parameters: $\tau_0 = (2, 6, 6, 0)$, $\tau_1 = (3, 10, 10, 1)$, $\tau_2 = (2, 7, 7, 1)$ and $\tau_3 = (3, 9, 9, 0)$. To simplify the example, we will assume that CS can only choose between classic EDF and EDF variant 1. As a first step, the CS schedules the task sets with classic EDF for the first busy period. As can be seen in Figure 6.4, it obtains a busy period from instant 0 to instant 5 and 0 units of interference. Thus, the CS schedules the task sets for the same busy period again, but now with EDF variant 1. The result is also 0 units of interference as it can be seen in Figure 6.4, so in this busy period both policies obtain the same result. Since all options have the same result, the CS will choose the assigned policy by default, in this case, classic EDF. Once the first busy period is solved, CS continues the scheduling at instant 6 and the process is repeated again.

For the second busy period, we can see in Figure 6.5 that classic EDF obtains 2 units of interference while in Figure 6.6 EDF variant 1 obtains 0 units of interference. It is relevant to note that the size of the second busy period is different, from instant 6 to 16 (instead of 17 of classic EDF), and that is fully associated by the interference produced between the tasks. Therefore, the CS chooses EDF variant 1 as the appropriate policy for the second busy period. The CS would then continue with the same process to solve the next busy period in the schedule and so on until the end is reached.

6.8. Rolling horizon MILP model

The previous approach basically applies different known scheduling policies at each busy period and selects the one that provides better results in terms of interference. As each busy period is isolated from its neighbours by idle instants, each one can be scheduled independently as the CS does. However, the CS does not obtain an optimized solution in terms of minimum

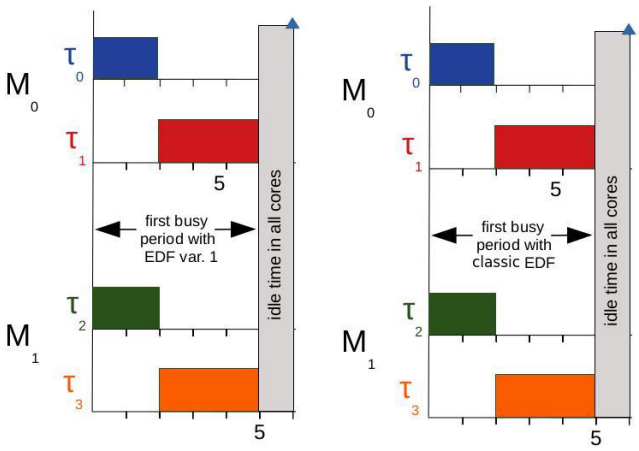


Figura 6.4: Resulting chronogram after scheduling the first busy period with EDF in variant 1 (left) and classic EDF (right).

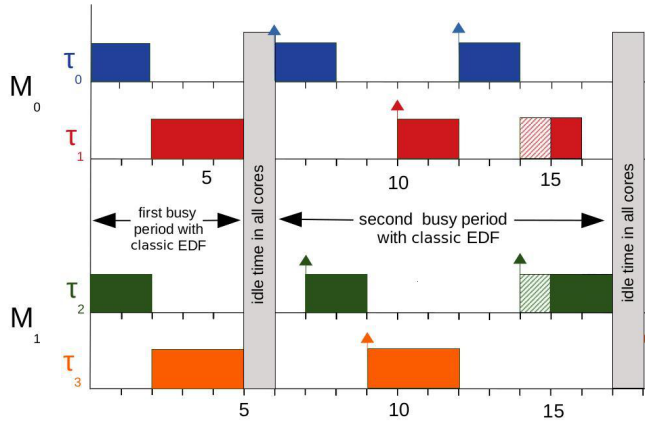


Figura 6.5: Resulting chronogram after scheduling the second busy period with classic EDF.

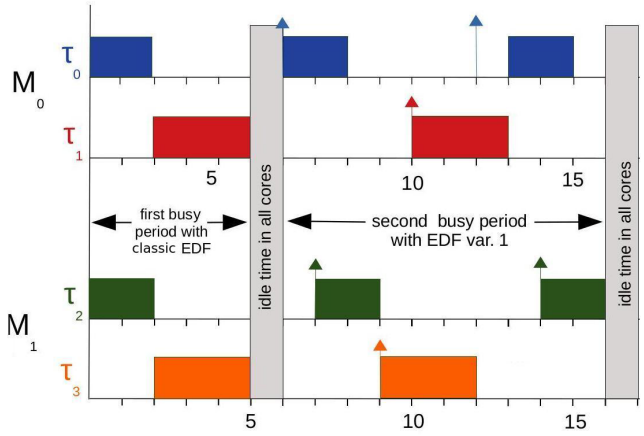


Figura 6.6: Resulting chronogram after scheduling the second busy period with EDF in variant1.

interference and this interference could be further reduced. In this section, we propose an ILP technique to obtain a static schedule with the optimization criteria of minimize interference.

We use the concept of *rolling horizon*. Figure 6.7 shows the main idea. The goal is to find a schedule for all the hyperperiod which is the scheduling horizon. As trying to solve the problem for the entire horizon is computationally very expensive, the problem is divided into smaller horizons in a "divide and conquer"strategy. In our case, each smaller horizon is a busy period.

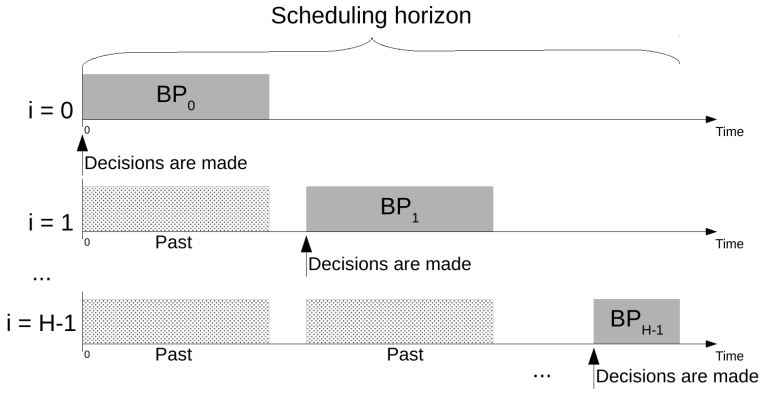


Figura 6.7: Rolling horizon description.

The methodology is depicted in Figure 6.8. As seen in the Figure, once the CS computes the feasible scheduling plan for the task set, we deduce the system busy periods or rolling horizons[1]. In each rolling horizon we will solve a MILP problem, the Rolling Horizon MILP Algorithm (RHMA) listed in Algorithm 6.

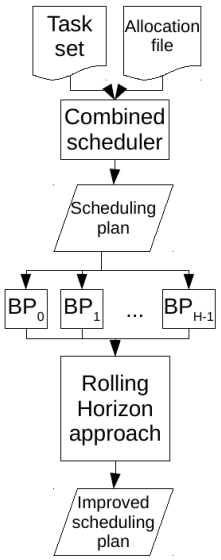


Figura 6.8: Rolling horizon schema.

[1]The terms "busy periods.ªnd rolling horizons.ªre used interchangeably throughout this text.

Algorithm 6 Rolling horizon MILP algorithm (RHMA)

```
1: INPUT: Busy periods from CS, Task set and task-to-cores allocation
2: OUTPUT: Scheduling plan, σ
3: procedure ROLLING HORIZON MILP ALGORITHM
4:     Temporarily ordering of busy periods → [BP_0, ..., BP_{H-1}]
5:     for BP_h ∈ Busy Periods [BP] do
6:         Calculate parameters
7:         Define and reduce variable matrix x
8:         Define and reduce variable matrix m
9:         Define variable matrix w
10:        add Constraints
11:        set Objective and optimize
12:        if Optimal or Feasible then
13:            σ_h = x_iajt (Save scheduling plan for BP_h)
14:        else
15:            Usage of the CS schedule in this BP
16:        if h < H-1 then
17:            Remove constraints and variables
18:            Update the model
```

The RHMA algorithm works as follows: it receives a task set, with a known task-to-core allocation and the set of busy periods from the plan generated by the CS. As the problem is to be posed as MILP, different parameters and variables must be introduced. They are defined in Table 6.2. For each rolling horizon (BP), all parameters from Table 6.2 are calculated from the data input (line 6)). Then, the variable matrices are declared. In order to improve the efficiency of the algorithm, both variables and constraints are declared at the beginning and removed at the end of the interval. In this way, the size of the problem is significantly reduced. Moreover, some of the variables may be simplified because of known information from the input data, as happened in the complete model presented in Section 6.5. Then, the constraints and the objective are added and the solver starts the optimization (lines 10 and 11). If it reaches the optimal or a feasible solution in the specified time (a feasible scheduling plan in that rolling horizon), the scheduling plan will be saved (line 13), all the constraints and variables will be removed (line 17), the model will be updated (line 18) and then the system will move to the next rolling horizon. This is repeated in all rolling horizons. If, at some point, the solver can not feasible schedule the tasks at any rolling horizon, this interval will be scheduled by the combined scheduler (line 15).

It is important to note that the difference between Algorithm 6 and Algorithm 4 is that in Algorithm 6 the optimisation is performed in each interval and in Algorithm 4 it is performed over the entire hyperperiod.

In the following, we are describing the objective function and the constraints for the MILP model. As happened in the complete model in Section 6.5, the objective function consists of minimizing the total number of interferences and the total response time for all tasks in all

Tabla 6.2: Model notation of the RHMA

Sets and indices	
i	Tasks $\tau_i \in \{0, 1, 2, ..., n-1\}$
a	Activations of $\tau_i \in \{0, 1, 2, ..., N_i - 1\}$
j	Cores $M_j \in \{0, 1, 2, ..., m-1\}$
h	Busy periods $BP_h \in \{0, 1, 2, ..., H-1\}$
Parameters	
C_i	Worst case computation time of τ_i
D_i	Deadline of τ_i
d_{ia}	Due date of τ_i in activation a
T_i	Period of τ_i
I_i	Interference factor of τ_i over other tasks
N_i	Number of activations of τ_i
o_{ij}	1 if task τ_i is allocated to core j and 0 otherwise.
S_{ih}	Activations of task i executed in the busy period BP_h
R_{iah}	Possible execution time interval for task i in its activation a in the busy period BP_h
T_h	Execution times for the busy period BP_h
lcm	Hyperperiod of the task set
$maxI$	Maximum possible received interferences
Decision variables	
x_{iajt}	1 if task i in activation a is allocated to core j and executed at time t and 0 otherwise.
m_{iakb}	Interference matrix. 1 if execution of τ_i in activation a coincides with the execution of τ_k in activation b and 0 otherwise.
w_{ia}	Response time matrix. Response time of τ_i in activation a

system executions.

$$\text{mín} \quad \text{Obj} = \frac{1}{maxI} \sum_{\substack{\forall i,k \in \tau \\ \forall a \in S_{ih} \\ \forall b \in S_{kh}}} m_{iakb} + \sum_{\forall i, a \in R_{iah}} \frac{w_{ia}}{D_i} \tag{6.15}$$

The constraints of the problem are defined as follows, similarly to the model presented in Section 6.5. The difference lies in the fact that the previous problem covered the entire hyperperiod and this model moves from interval to interval. Then, for each busy period BP_h:

$$x_{iajt} = o_{ij} \quad \forall i, a, j, t | i, a \in R_{iah}, t \in T_h \quad \text{, if } o_{ij} = 0 \tag{6.16}$$

$$m_{iakb} = 0 \quad \forall i, a \in R_{iah}, \forall k, b \in R_{kbh} \text{ if } k > i \text{ and if } R_{iah} \cap R_{kbh} = \emptyset \tag{6.17}$$

$$\sum_{\substack{a \in S_{ih} \\ t \in R_{iah}}} x_{iajt} = \text{len}(S_{ih}) \cdot C_i \cdot o_{ij} + \sum_{\substack{k \neq i \\ a \in S_{ih} \\ b \in S_{kh}}} m_{iakb} \cdot I_k \cdot o_{ij}$$
$$\forall i, j, \text{ if } o_{ij} \neq o_{kj} \text{ and if } I_i, I_k \neq 0 \text{ and if } \text{len}(S_{ih}), \text{len}(S_{jh}) > 0 \tag{6.18}$$

$$\sum_{t \in R_{iah}} x_{iajt} = C_i \cdot o_{ij} + \sum_{\substack{k \neq i \\ b \in S_{kh}}} m_{iakb} \cdot I_k \cdot o_{ij}$$
$$\forall i, j, a | a \in S_{ih}, \text{ if } o_{ij} \neq o_{kj} \text{ and if } I_i, I_k \neq 0 \text{ and if } \text{len}(S_{ih}), \text{len}(S_{jh}) > 0 \tag{6.19}$$

$$t \cdot \sum_{j \in J} x_{iajt} \leq d_{ia} - 1 \forall i, a, t | a \in S_{ih}, t \in T_h \tag{6.20}$$

$$\sum_{\substack{\forall i \\ a \in S_{ih}}} x_{iajt} \leq 1 \qquad \forall j, t | t \in T_h \tag{6.21}$$

$$m_{iakb} \geq x_{iajt} + x_{kblt} - 1$$
$$\forall i, a, k, b, j, l, t | a \in S_{ih}, b \in S_{kh}, t \in T_h$$
$$\text{, if } i \neq k \text{ and if } j \neq l \tag{6.22}$$

$$m_{iakb} = m_{kbia}$$
$$\forall i, a, k, b | a \in S_{ih}, b \in S_{kh}, \text{ if } k \neq i \tag{6.23}$$

$$w_{ia} \geq t \cdot x_{iajt} - a \cdot T_i + 1 \forall t, i, a, j | a \in S_{ih}, t \in T_h \tag{6.24}$$

$$x_{iajt}, m_{iakb} \in \{0, 1\} \tag{6.25}$$

$$w_{ia} \geq 0 \tag{6.26}$$

Constraints 6.16 and 6.17 reduce some variables because of the known information.

Constraint 6.18 ensures that task i completes all the computation time in the BP_h. At each busy period, a task can execute 0, 1 or more activations. Therefore, it has to complete its WCET as many times as activations occur plus the produced interference. Similarly, constraint 6.19 ensures that each activation of each task is executed in its activation interval. Equation 6.20 ensures that all tasks end before the arrival of their deadlines. Constraint 6.21 ensures that only one task is being executed at each point in time at each core. Equations 6.22 and 6.23 calculate the produced interference.

Constraint 6.24 calculates the response time of each activation of all tasks. Equations 6.25 and 6.26 represent the decision variable domains.

6.8.1. RHMA with warm start

A way of improving the RHMA performance is to make use of the CS scheduling plan, using it as an input of the RHMA (see Figure 6.9). The RHMA receives both the scheduling plan and the busy periods from the CS.

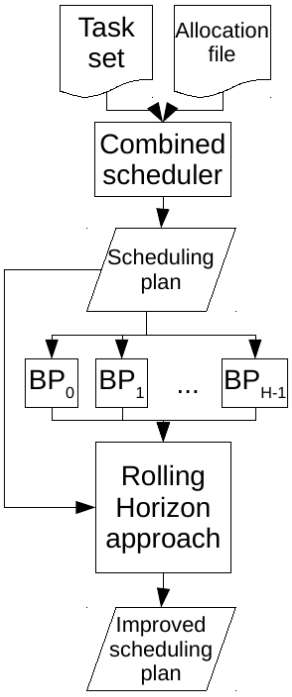


Figura 6.9: Rolling horizon with warm start schema.

Warm starting information is an additional input data that allows the algorithm to speed up solving the problem, reducing fastly the distance from the optimality. In practice, it consists on providing manually starting solution vectors of the problem to the solver.

If the MILP solver finds that the input solution is feasible, then the input solution provides an incumbent solution and a bound for the branch-and-bound algorithm. If the solution is not

feasible, the MILP solver tries to repair it. When it is difficult to find a good integer feasible solution for a problem, a warm start can significantly reduce the solution time. The effectiveness of the warm start in MILP solvers depends on many factors. Sometimes, warm starts do not help the solver to find solutions more quickly. For example, if a significant amount of time is expended proving optimality of a good solution (as is often the case), then there will be no noticeable change in run time by supplying a warm start. However, authors may consider providing feasible starting points to fix an upper bound on the objective value (in case of minimization) and thus can be used to prune nodes during the search [6.13].

The feasible starting point considered in this work is the scheduling plan provided by the CS.

6.9. Evaluation

In order to asses the scheduling method proposed, we have conducted the evaluation with synthetic task loads. The methodology is shown in Figure 6.10, which consists of four main phases: load generation, allocation, scheduling and validation.

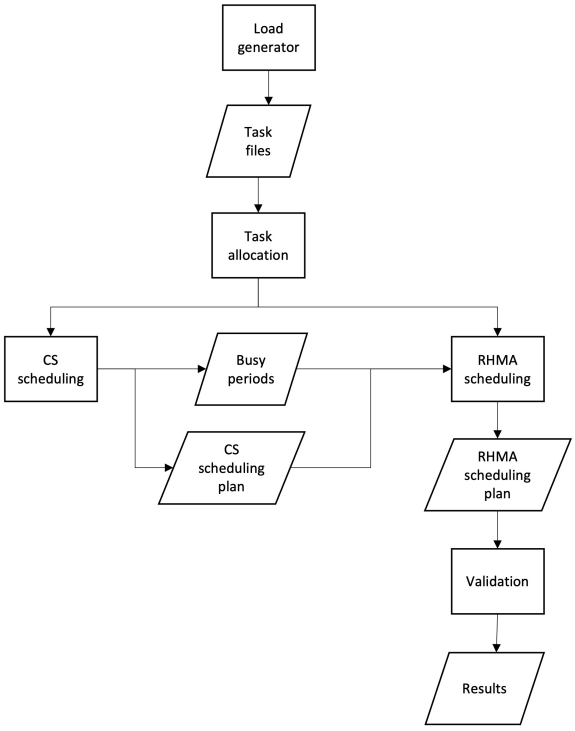


Figura 6.10: Experimental evaluation overview.

The load is obtained using a synthetic task generator. The inputs of this generator are: the number of cores M, the number of tasks n, the number of broadcasting tasks, the theoretical utilisation of the task set U_τ and the interference of broadcasting tasks as a percentage of the WCET. The specific parameters used as inputs are shown in Table 6.3. Task parameters are obtained randomly to achieve the theoretical utilization.

	2 cores	4 cores	8 cores
Theoretical utilisation	1.5	3	6
Number of tasks	4	12	20
Number of broadcasting tasks	2	3	5
Interference (% over WCET)	10 %	10 %	10 %

Tabla 6.3: Experimental parameters

The next step is the allocation of tasks to the different cores. Two different approaches are used for the allocation, Worst Fit Decreasing Utilisation (WFDU) [6.23] and Wmin [6.1]. We choose these two allocators based on the comparison made in [6.1] where Wmin showed the best results in terms of interference reduction and WFDU presents the best results results in terms of schedulability.

With the task set and the allocation, the CS schedules the task set and calculates the BP, being both an input for the RHMA algorithm.

The results obtained with the proposed RHMA will be compared with EDF scheduling. That is, we will compare two allocation algorithms with two schedulers resulting in four schedulers:

- WFDU+EDF
- Wmin+EDF
- WFDU+RHMA
- Wmin+RHMA

The comparison will be done in terms of schedulability and increased utilization. The schedulability measures the percentage of schedulable sets out of the total number of generated sets. The increased utilization is measured as:

$$1 - \sum_{k=0}^{k=m-1} \frac{U_{M_k}}{U'_{M_k}} = 1 - \frac{U_\tau}{U'_\tau} \tag{6.27}$$

that is, it is the ratio between the real utilization and the theoretical utilization.

The results are depicted for schedulability in Figure 6.11.

In this figure, it can be seen that the percentage of schedulability decreases with the number of cores, as expected. Comparing allocators, WFDU obtains better schedulability rates than Wmin as concluded in [6.1]. With respect to the scheduling algorithms, RHMA improves EDF results in the sense that RHMA can schedule task sets that EDF is not capable of. Therefore, WFDU+RHMA is the best choice. As a conclusion, regarding schedulability, choosing the right allocator is key rather than the scheduler algorithm.

Figure 6.12 depicts the increased utilisation with respect to the theoretical utilisation.

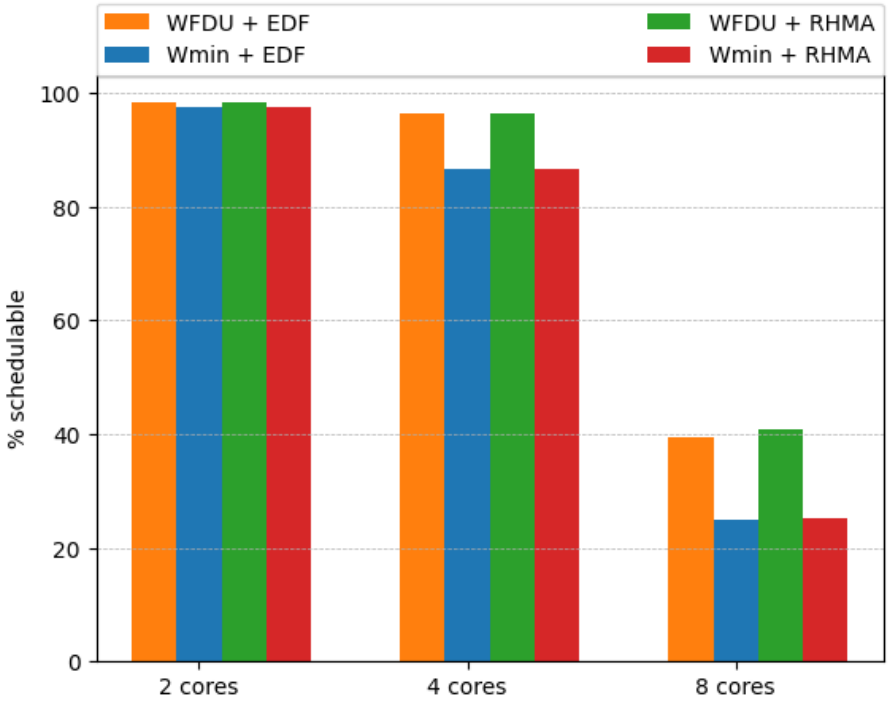


Figura 6.11: Percentage of schedulable task sets depending on the number of cores, allocators and scheduling algorithms.

As seen in this figure, to allocate tasks to cores using WFDU algorithm means a higher increased utilisation over using Wmin. This is due to the fact that Wmin tries to group tasks with interference together in the same core, in order to reduce the overall produced interference. WFDU balances the load considering only the theoretical utilisation of the tasks and not the interference. Regarding to the proposed scheduling algorithms, RHMA significantly reduces the increased utilisation, specially when tasks have been allocated using WFDU algorithm. That is because, as WFDU produces an allocator with high interference, RHMA has a wider range of action to reduce it. As Wmin already achieves an allocation with less interference, RHMA is less likely to reduce it.

In terms of figures, Figures 6.13 and 6.14 depict the average values of schedulability and increased utilisation. From Figure 6.13 it can be concluded that the combination of allocator and scheduler which offers better schedulability rates is WFDU and RHMA, 78.56 %. These results are slightly better than WFDU with EDF, i.e., there are task sets that EDF can not schedule and RHMA can. The same happens with Wmin. In general, Wmin schedulability rates are lower than WFDU (only 78 % vs 69 %) but RHMA works better than EDF. Figure 6.14 shows that Wmin and RHMA is the combination that reduces the increased utilisation the most, only 0.819 %. This is logical, as we are applying interference reduction techniques in both the allocator and the scheduler.

As it is said in previous sections, RHMA scheduler uses the CS algorithm to obtain a solution in some busy periods of large size, so it is relevant to make a comparison between RHMA and

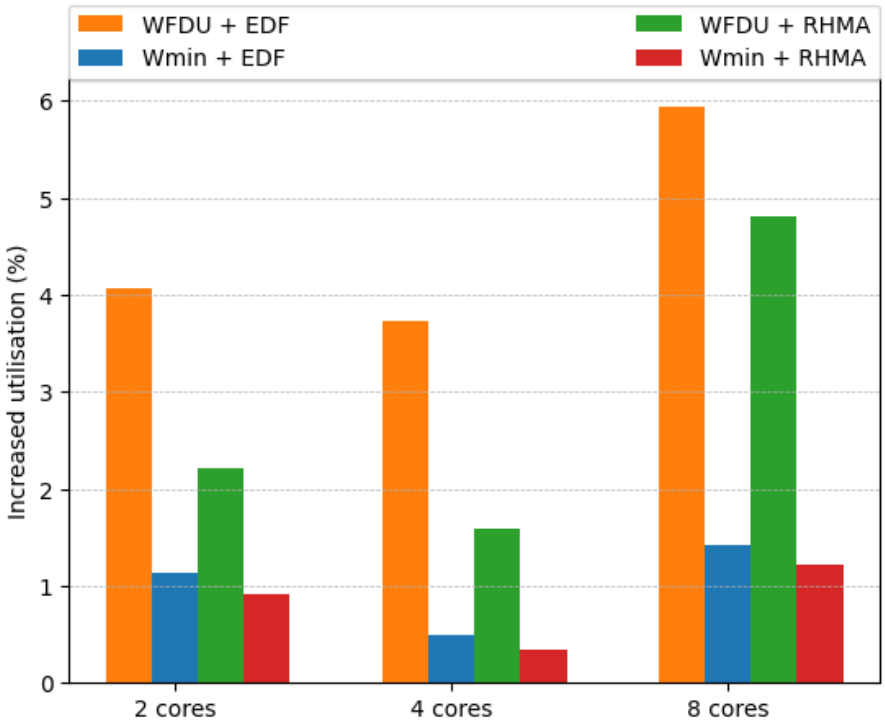


Figura 6.12: Increased utilisation of the task sets depending on the number of cores, allocators and scheduling algorithms.

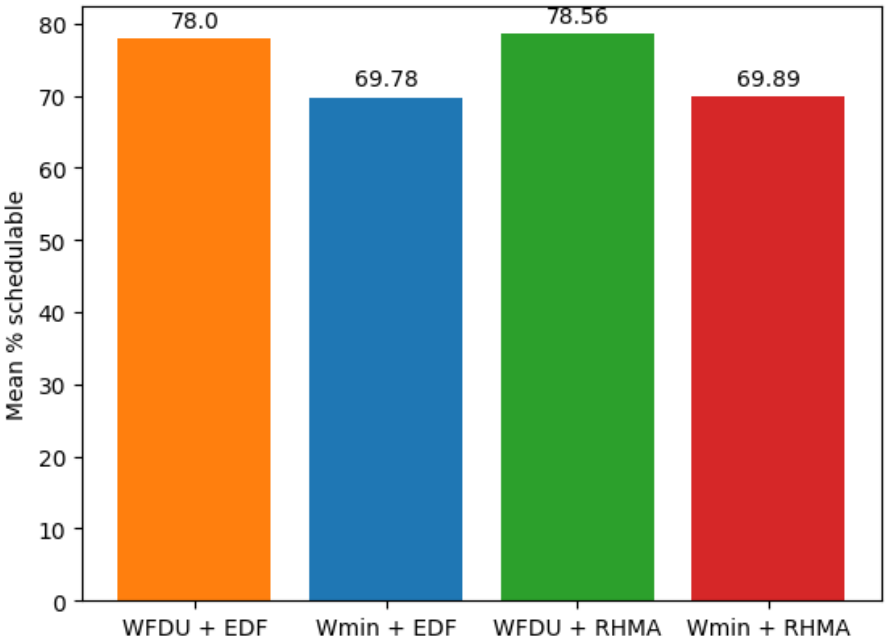


Figura 6.13: Percentage of average schedulable task sets depending on the allocators and schedulability algorithms.

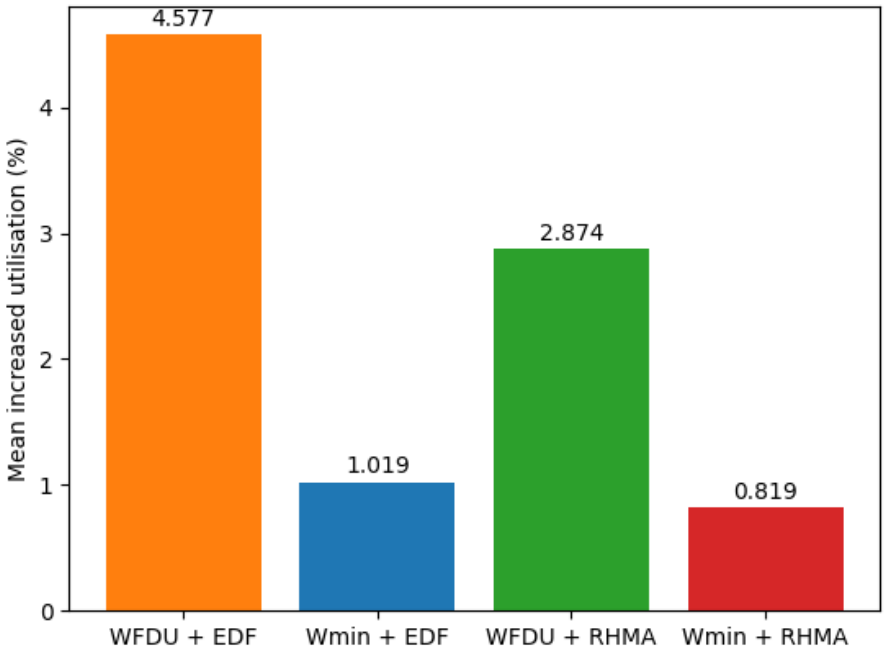


Figura 6.14: Percentage of average increased utilisation of the task sets depending on the allocators and schedulability algorithms.

CS. In terms of increased utilisation, as it can be seen in the Figure 6.15 the plans made by RHMA scheduler have a lower increased utilisation index compared to the plans strictly made by the CS. This happens regardless of which allocator is used for assigning tasks to cores, in our case with Wmin and WFDU allocators. It can be observed as well that, in the same way as it can be seen in previous graphics, with 8 cores the average of increased utilisation caused by interference in the plans is higher than the cases of 2 and 4 cores and the scheduling resulted after a Wmin allocation made less interference than the resulted from a WFDU allocation. Note that the increased utilization in 2 cores is higher than in 4 cores and this is due to the experimental parameters. Scenarios with 2 cores have a higher percentage of broadcasting tasks (50 %) than scenarios with 4 and 8 cores (25 %). The parameters have been selected in this way to make a fair comparison with [6.1].

Since the role of the CS is to assist RHMA in some busy periods, it is interesting to know in how many BPs the RHMA is not able to find a solution and then, the CS schedule is used. Figure 6.16 represents the percentage of BPs with respect to the total number of BPs in a hyperperiod in which the scheduling solution is the one provided by the CS. The graphic shows that with 2 cores RHMA does not need to use the CS in any case. This is due to the fact that the size of the problem to be solved with 2 cores is much simpler than with more cores. However, with 8 cores the complexity of the scheduling problem increases a lot for each BP, so the RHMA tends to use the CS help up to more than 25 % of the busy periods in the case of 8 cores.

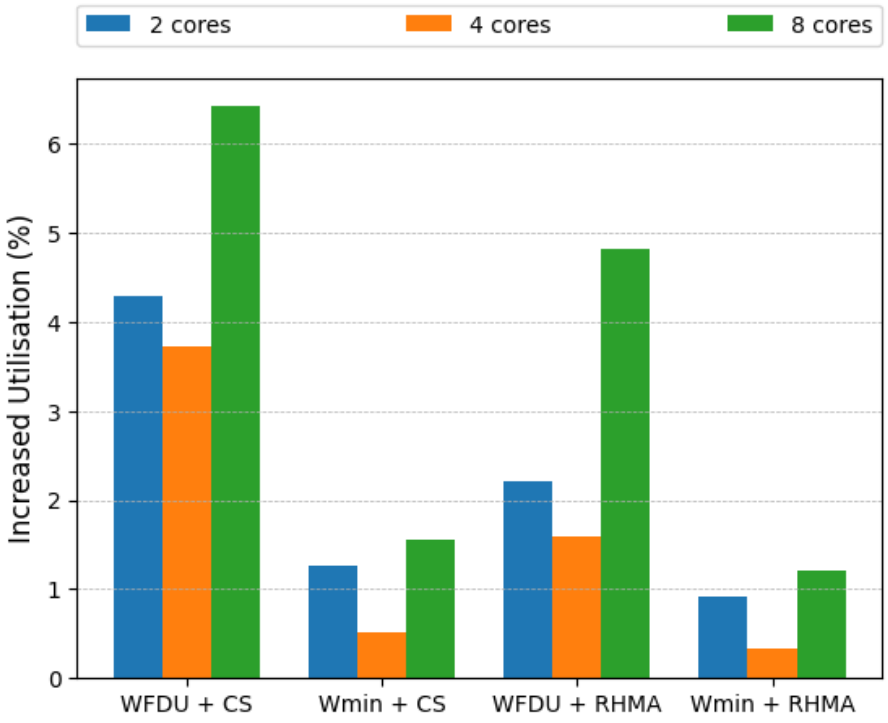


Figura 6.15: Comparison between RHMA and CS: Percentage of average increased utilisation depending on the allocators and schedulability algorithms.

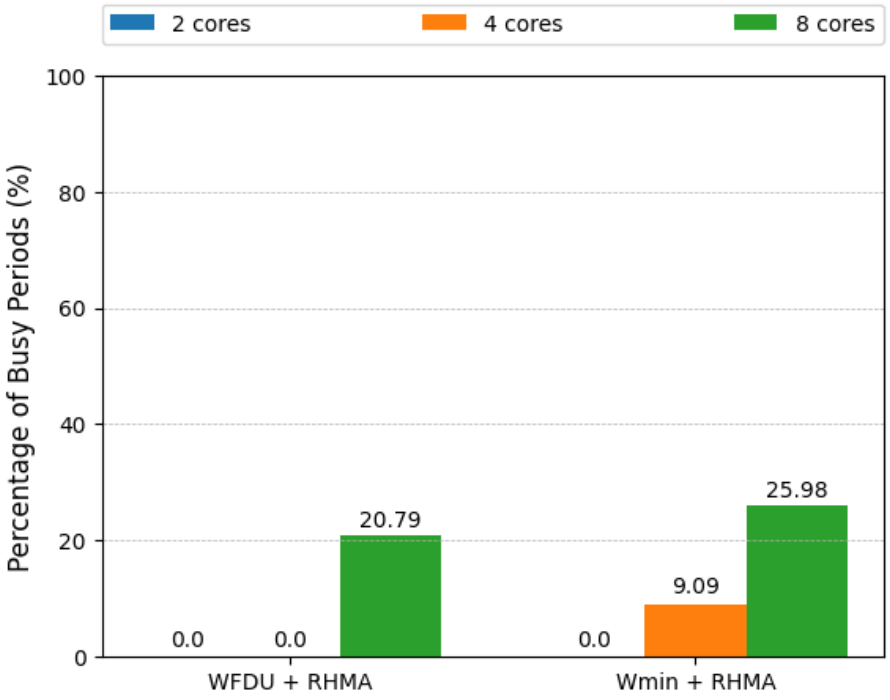


Figura 6.16: Percentage of Busy Periods where RHMA needs the CS.

6.10. Conclusions

In this paper, a new scheduling strategy is proposed with the aim of reducing interference due to the contention of shared hardware resources in the context of partitioned multicore scheduling. The first technique, the CS scheduler, is based on independently schedule each busy period in order to apply in each one the best scheduling heuristic in terms of interference reduction. The second one is based on a MILP model. In this case we have proposed the scheduling of all the hyperperiod at once but, due to its disadvantages, we propose another technique that, as the CS does, independently finds the scheduling plan that minimizes the interference. This technique based on rolling horizons (RHMA) is able to obtain solutions in a reasonable amount of time, as it divides the hyperperiod into smaller independent busy periods. The evaluation done has showed that RHMA reduces the interference produced and it is the best solution when it is combined with an allocator that also takes into account the interference when allocating tasks to cores.

As future work, the use of MILP techniques to reduce not only interference but also several parameters at the same time is proposed. We also aim to find a MILP technique that is able to plan the BPs where the CS solution is now needed.

Bibliografía

[6.1] Aceituno, J. M., Guasque, A., Balbastre, P., Simó, J., and Crespo, A. Hardware resources contention-aware scheduling of hard real-time multiprocessor systems. Journal of Systems Architecture 118 (2021).

[6.2] Altmeyer, S., Davis, R. I., Indrusiak, L., Maiza, C., Nelis, V., and Reineke, J. A generic and compositional framework for multicore response time analysis. In Proceedings of the 23rd International Conference on Real Time and Networks Systems (New York, NY, USA, 2015), RTNS '15, Association for Computing Machinery, p. 129–138.

[6.3] Baruah, S. An ilp representation of a dag scheduling problem. Real-Time Systems 58 (03 2022).

[6.4] (CAST), C. A. S. T. Multi-core Processors—Position Paper CAST-32A, November 2016. Technical Report.

[6.5] Choi, J., Kang, D., and Ha, S. Conservative modeling of shared resource contention for dependent tasks in partitioned multi-core systems. In 2016 Design, Automation Test in Europe Conference Exhibition (DATE) (2016), pp. 181–186.

[6.6] Dasari, D., Akesson, B., Nélis, V., Awan, M. A., and Petters, S. M. Identifying the sources of unpredictability in cots-based multicore systems. In 2013 8th IEEE International Symposium on Industrial Embedded Systems (SIES) (2013), pp. 39–48.

[6.7] Dasari, D., Andersson, B., Nelis, V., Petters, S. M., Easwaran, A., and Lee, J. Response time analysis of cots-based multicores considering the contention on the shared memory bus. In 2011IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications (2011), pp. 1068–1075.

[6.8] Davis, R., Altmeyer, S., Indrusiak, L., Maiza, C., Nelis, V., and Reineke, J. An extensible framework for multicore response time analysis. Real-Time Systems 54 (07 2018).

[6.9] Davis, R. I., and Burns, A. A survey of hard real-time scheduling for multiprocessor systems. ACM Comput. Surv. 43, 4 (Oct. 2011).

[6.10] Davis, R. I., Griffin, D., and Bate, I. Schedulability Analysis for Multi-Core Systems Accounting for Resource Stress and Sensitivity. In 33rd Euromicro Conference on Real-Time Systems (ECRTS 2021) (Dagstuhl, Germany, 2021), B. B. Brandenburg, Ed., vol. 196 of Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl – Leibniz-Zentrum für Informatik, pp. 7:1–7:26.

[6.11] Fernandez, G., Abella, J., Quiñones, E., Rochange, C., Vardanega, T., and Cazorla, F. Contention in multicore hardware shared resources: Understanding of the state of the art. In WCET (2014).

[6.12] Guasque, A., Aceituno, J. M., Balbastre, P., Simó, J., and Crespo, A. Schedulability analysis of dynamic priority real-time systems with contention. J. Supercomput. 78, 12 (aug 2022), 14703–14725.

[6.13] Guasque, A., Tohidi, H., Balbastre, P., Aceituno, J. M., Simó, J., and Crespo, A. Integer programming techniques for static scheduling of hard real-time systems. IEEE Access 8 (2020), 170389–170403.

[6.14] He, C., Zhu, X., Guo, H., Qiu, D., and Jiang, J. Rolling-horizon scheduling for energy constrained distributed real-time embedded systems. Journal of Systems and Software 85, 4 (2012), 780–794.

[6.15] Huang, W.-H., Chen, J.-J., and Reineke, J. Mirror: Symmetric timing analysis for real-time tasks on multicore platforms with shared resources. In Proceedings of the 53rd Annual Design Automation Conference (New York, NY, USA, 2016), DAC '16, Association for Computing Machinery.

[6.16] Kim, H., de Niz, D., Andersson, B., Klein, M., Mutlu, O., and Rajkumar, R. Bounding memory interference delay in cots-based multi-core systems. In 2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS) (2014), pp. 145–154.

[6.17] Lampka, K., Giannopoulou, G., Pellizzoni, R., Wu, Z., and Stoimenov, N. A formal approach to the wcrt analysis of multicore systems with memory contention under phase-structured task sets. Real-Time Systems 50 (11 2014), 736–773.

[6.18] Lehoczky, J. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In [1990] Proceedings 11th Real-Time Systems Symposium (1990), pp. 201–209.

[6.19] Liu, C. L., and Layland, J. W. Scheduling algorithms for multiprogramming in a hard-real-time environment. J. ACM 20, 1 (Jan. 1973), 46–61.

[6.20] Lugo, T., Lozano, S., Fernández, J., and Carretero, J. A survey of techniques for reducing interference in real-time applications on multicore platforms. IEEE Access 10 (2022), 21853–21882.

[6.21] Maiza, C., Rihani, H., Rivas, J. M., Goossens, J., Altmeyer, S., and Davis, R. I. A survey of timing verification techniques for multi-core real-time systems. ACM Comput. Surv. 52, 3 (jun 2019).

[6.22] Marquant, J. F., Evins, R., and Carmeliet, J. Reducing computation time with a rolling horizon approach applied to a milp formulation of multiple urban energy hub system. Procedia Computer Science 51 (2015), 2137–2146. International Conference On Computational Science, ICCS 2015.

[6.23] Oh, Y., and Son, S. H. Allocating fixed-priority periodic tasks on multiprocessor systems. Real-Time Syst. 9, 3 (Nov. 1995), 207–239.

[6.24] Rihani, H., Moy, M., Maiza, C., Davis, R. I., and Altmeyer, S. 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 (2016), RTNS '16, p. 67–76.

[6.25] Ripoll, I., Crespo, A., and Mok, A. K. Improvement in feasibility testing for real-time tasks. Real-Time Syst. 11, 1 (July 1996), 19–39.

[6.26] Xiao, J., Altmeyer, S., and Pimentel, A. Schedulability analysis of non-preemptive real-time scheduling for multicore processors with shared caches. In 2017 IEEE Real-Time Systems Symposium (RTSS) (2017), pp. 199–208.

[6.27] Zhang, F., and Burns, A. A worst-case pattern of task load allocation and execution for multiprocessor global real-time scheduling. International Journal of Simulation: Systems, Science Technology 17 (01 2016), 9.1–9.4.

Capítulo 7

Conclusiones

Esta tesis doctoral ha cumplido con los objetivos para los que había sido planteada. En los cinco artículos que componen esta tesis se han propuesto distintos algoritmos, metodologías, modelos y análisis que abarcan cada uno de los objetivos que previamente se habían establecido.

Las metodologías que se proponen en cada artículo se han evaluado de manera experimental y sus resultados han sido comparados con los de otros algoritmos y heurísticas clásicos en la literatura de planificación de sistemas de tiempo real estrictos. En la siguiente sección se realiza una recopilación de los resultados de los algoritmos que se proponen en los artículos. Estos resultados muestran un mejor rendimiento de los algoritmos propuestos en función de diversos parámetros y métricas usados, como pueden ser el incremento de utilización, la planificabilidad del sistema o los cambios de contexto.

En la planificación de sistemas de tiempo real estrictos, un buen rendimiento y el cumplimiento de los requisitos temporales son criterios fundamentales que se exigen en áreas como la aviación, la seguridad ferroviaria, satélites o control de procesos. En ese sentido, las metodologías propuestas en los artículos de esta tesis doctoral han mostrado con sus resultados que aseguran el cumplimiento de los requisitos temporales y que mejoran el rendimiento de diversos parámetros. Por tanto, estos algoritmos y metodologías pueden ser considerados como opciones elegibles para su implantación en sistemas de tiempo real críticos. Los resultados de esta tesis contribuyen a superar la barrera que encuentra el uso de las plataformas multinúcleo en aplicaciones de control de sistemas críticos bajo restricciones de tiempo real estricto. Con esto se contribuye a que la implementación de este tipo de sistemas sea más eficiente en términos de coste energético, peso, complejidad del hardware y, por supuesto, económico.

Por todo ello, puede concluirse que el trabajo realizado en esta tesis doctoral supone una contribución dado que propone diversos algoritmos y métodos a la literatura de planificación de sistemas de tiempo real mononúcleo y multinúcleo. Especialmente, el aporte que se ha realizado se sitúa en el ámbito del problema de la contención de recursos compartidos que se produce en los sistemas multinúcleo a causa de la interferencia entre las tareas.

7.1. Análisis de los resultados de los algoritmos propuestos

7.1.1. Resultados de los algoritmos de asignación de tareas a núcleos

Los algoritmos de asignación de tareas a núcleos que se han propuesto en los artículos de esta tesis doctoral son cuatro: *Wmin* (artículo 2), *UDmin* (artículo 2), *UDmax* (artículo 2) e *Imin* (artículo 3).

7.1.1.1. Wmin, UDmin y UDmax. Artículo 2

En este artículo se proponen tres algoritmos de asignación de tareas a núcleos y se comparan con otros tres métodos de asignación clásicos, FFDU, WFDU y BFDU. Para compararlos, se utilizan dos criterios distintos, la tasa de planificabilidad y el incremento de utilización.

Respecto a la tasa de planificabilidad, se puede observar en la gráfica de la izquierda de la Figura 7.1 que el algoritmo de asignación WFDU presenta el mayor porcentaje de planificabilidad, seguido de cerca por UDmin y Wmin.

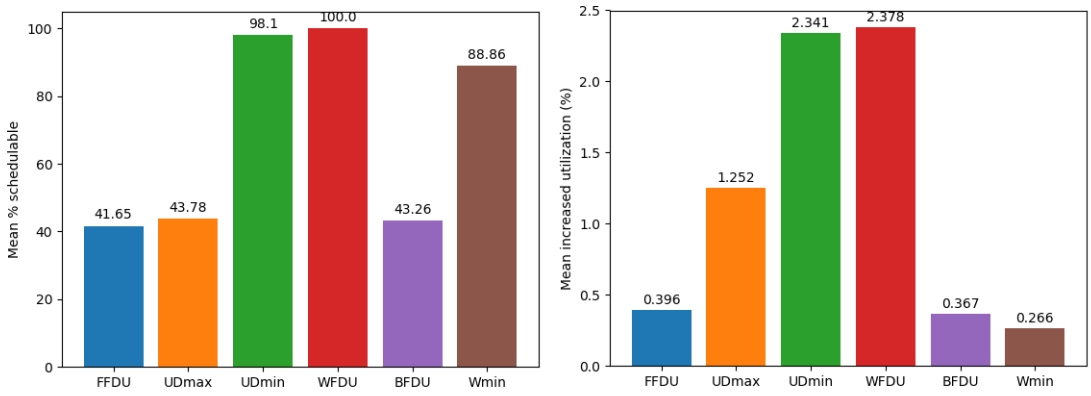


Figura 7.1: Valores experimentales de planificabilidad (izquierda) e incremento de utilización (derecha) en función a los diferentes algoritmos de asignación.

Respecto al incremento de utilización, se puede observar en la gráfica de la derecha de la Figura 7.1 que el algoritmo de asignación propuesto en el artículo 2, Wmin, es el algoritmo que menos incremento de utilización presenta de los seis algoritmos evaluados.

Si realizamos un análisis de los dos criterios (planificabilidad e incremento de utilización) podemos deducir que Wmin posee unos resultados globales mejores que el resto de algoritmos de asignación. Esto se justifica porque Wmin posee el menor incremento de utilización y la tercera mejor tasa de planificabilidad.

Por ende, se concluye que el algoritmo de asignación Wmin presentado en el artículo 2 es un algoritmo eficiente en términos de planificabilidad e incremento de utilización del sistema. Por tanto, es un algoritmo elegible para un sistema de tiempo real estricto multinúcleo, especialmente cuando se pretende optimizar la interferencia entre tareas sin apenas perder rendimiento en la planificabilidad del sistema.

7.1.1.2. Imin. Artículo 3

En el artículo 3 se propone un algoritmo de asignación de tareas a núcleos llamado Imin. El algoritmo propuesto se compara con otros tres algoritmos. Dos de ellos son los clásicos FFDU y WFDU, y el otro algoritmo con el que se compara es Wmin, el cual se propuso en el artículo 2. Tras la experimentación realizada, los resultados muestran que el algoritmo propuesto es el que obtiene mejor rendimiento de los cuatro respecto a dos métricas analizadas, la planificabilidad de los conjuntos de tareas y el incremento de la utilización del sistema.

Respecto a la planificabilidad del sistema, el algoritmo de asignación Imin es el que presenta un mejor rendimiento dado que obtiene el mayor porcentaje promedio de planificabilidad. Aunque el resultado de Wmin y WFDU se acercan al resultado de Imin. Esto se muestra en la Figura 7.2.

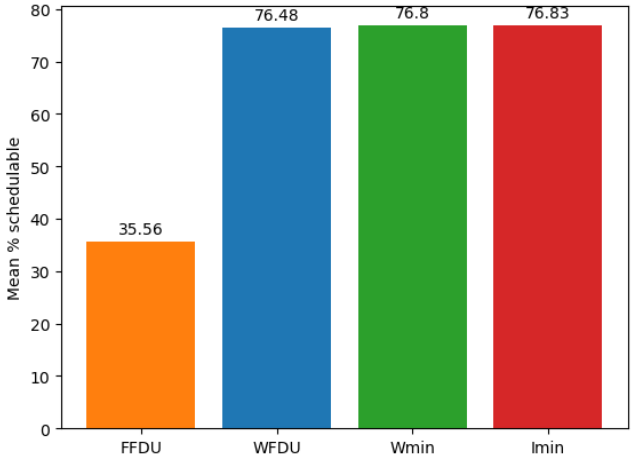


Figura 7.2: Porcentajes promedios de la planificabilidad de los conjuntos de tareas en función a los cuatro algoritmos de asignación.

Respecto al incremento de utilización, el algoritmo Imin es el que presenta el mejor rendimiento porque es el que obtiene, en promedio, el menor incremento de utilización. En este caso, el único algoritmo cuyo resultado se aproxima al de Imin es el algoritmo Wmin. Esto se muestra en la Figura 7.3.

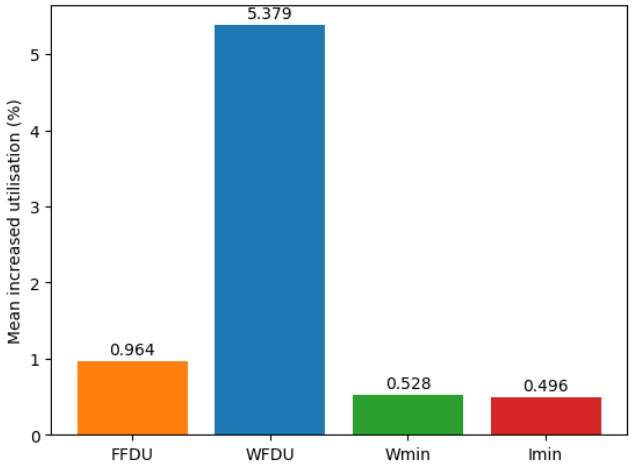


Figura 7.3: Incremento de utilización promedio en función de los algoritmos de asignación.

Por tanto, se concluye que el algoritmo de asignación de tareas a núcleos que el artículo 3 aporta a la literatura, llamado Imin, es un algoritmo elegible para un sistema de tiempo real estricto de plataformas multinúcleo. Especialmente, Imin será adecuado cuando el objetivo sea minimizar la interferencia entre las tareas.

7.1.2. Resultados de los algoritmos de planificación

Los algoritmos de planificación que se proponen en los artículos de esta tesis doctoral son cuatro: Algoritmo de metodología ILP para mononúcleo (artículo 1), algoritmo que computa la interferencia (artículo 2), algoritmo de metodología ILP para multinúcleo (artículo 5) y el planificador combinado (artículo 5).

No obstante, el algoritmo de planificación propuesto en el artículo 2 no se compara con otros preexistentes. Este algoritmo se utiliza en la experimentación como planificador para computar la interferencia y de tal modo, se utiliza para comparar los distintos algoritmos de asignación de tareas a núcleos. Por tanto en esta sección se realiza un análisis técnico de este algoritmo.

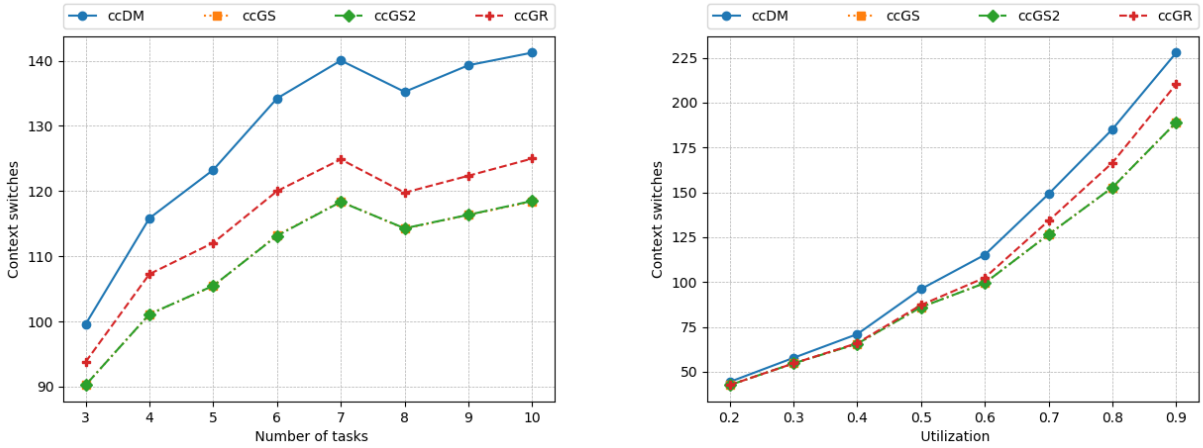
En el artículo 5 se propone un método de planificación que internamente está formado por dos algoritmos de planificación que se detallan en el propio artículo. Por tanto, en esta sección se realiza un análisis de ambos algoritmos por separado, así como un análisis de los resultados que muestra el método de planificación completo.

7.1.2.1. Algoritmo ILP. Artículo 1

En el artículo 1 se proponen tres algoritmos de planificación que utilizan técnicas ILP. El rendimiento de estos algoritmos se compara con una heurística clásica en planificación, DM. A continuación se presenta el resultado de la evaluación de los cuatro métodos de planificación

según dos métricas distintas, los cambios de contexto y los mejores tiempos de respuesta.

Respecto al número de cambios de contexto que presentan las planificaciones generadas por cada algoritmo, en la Figura 7.4 se muestra como los tres algoritmos propuestos obtienen menor cantidad de cambios de contexto que DM, por tanto, presentan mejor rendimiento. Esta mejora se observa desde dos perspectivas, evaluando los cambios de contexto en función al número de tareas y evaluando los cambios de contexto en función a la utilización del sistema. De los tres algoritmos propuestos, el que mejor resultados ofrece en este aspecto es el algoritmo GS2.



(a) Number of tasks

(b) Utilization

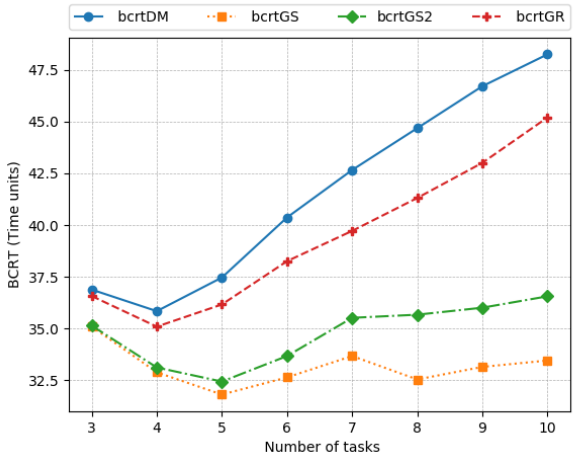
Figura 7.4: Impacto de la utilización del sistema y el número de tareas en los cambios de contexto.

Respecto a los mejores tiempos de respuesta (BCRT), los tres algoritmos propuestos en el artículo consiguen, en todos los escenarios, obtener menores valores de BCRT que los valores conseguidos por el algoritmo DM. Por tanto, los tres algoritmos de planificación propuestos en el artículo 1 presentan una mejora respecto al rendimiento de los BCRT. En la Figura 7.5 puede observarse este resultado.

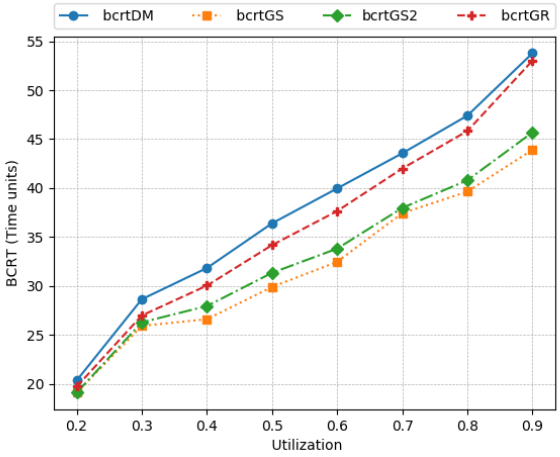
7.1.2.2. Algoritmo que computa la interferencia. Artículo 2

El artículo 2 se sitúa en el ámbito de los sistemas multinúcleo. En este ámbito se halla el problema de la interferencia de tareas, la cual causa contención de los recursos compartidos. Para afrontar el problema de la interferencia, en el artículo 2 se propone un planificador que compute o cuantifique dicha interferencia.

Es importante aclarar que este algoritmo de planificación no trata de reducir la interferencia sino de computarla. Gracias a esta cuantificación, en este artículo se realiza una comparativa entre los distintos algoritmos de asignación y puede evaluarse cuánta interferencia provoca cada



(a) Number of tasks



(b) Utilization

Figura 7.5: Influencia de la utilización del sistema y el número de tareas en el BCRT.

uno. Este algoritmo se aplica bajo el modelo temporal de tareas que se propone en el mismo artículo.

El algoritmo computa interferencia solo cuando dos o más tareas se ejecutan en el mismo instante y siempre en núcleos distintos. Para su funcionamiento, utiliza internamente una matriz binaria de tres dimensiones $n \times n \times H$, siendo n el número de tareas del conjunto y H el hiperperiodo. Para cada instante del hiperperiodo el algoritmo computa la interferencia que cada tarea provoca en cualquiera de las otras tareas.

Hasta el momento de presentar este algoritmo, el problema de la computación de la interferencia se resolvía añadiendo más tiempo de cómputo al WCET (*Worst Case Execution Time*) para considerar el peor de los casos. Sin embargo, este enfoque es demasiado pesimista. El algo-

ritmo de planificación propuesto realiza una cuantificación más realista de la interferencia que el anterior método.

Además de ser usado en la experimentación del artículo 2, el algoritmo propuesto se utiliza también en la experimentación realizada en el artículo 3.

7.1.2.3. Planificador Combinado. Artículo 5

En el artículo 5 se propone un método de planificación que está formado por dos técnicas distintas que se complementan entre sí. Una de ellas se llama planificador combinado (PC) y la otra ILP-RHMA, o simplemente RHMA (Rolling Horizon MILP Algorithm).

El planificador combinado trabaja internamente con distintas heurísticas o algoritmos. En la experimentación realizada en el artículo 5, el PC utilizó las heurísticas de planificación de EDF, DM y variantes de las mismas. En cualquier caso, el PC podría usarse con otras heurísticas o algoritmos distintos.

El PC opera sobre cada intervalo de trabajo que se va generando durante el proceso de la planificación. Dichos intervalos de trabajo son aprovechados por el algoritmo de planificación RHMA para realizar su propia planificación, intervalo por intervalo. Además, cuando RHMA no obtiene solución en un intervalo de trabajo, entonces utiliza la que previamente había generado el PC.

En el artículo 5 se realiza una comparación entre el método conjunto y otros algoritmos típicos en la planificación. Dichos resultados se muestran en la sección de RHMA. No obstante, también se ha realizado una comparativa entre RHMA y el PC para evaluar cuánto incremento de utilización provoca cada uno, y cuántas veces RHMA recurre a las planificaciones parciales del PC.

En la Figura 7.6 podemos observar una comparativa respecto al incremento de utilización entre el planificador combinado y RHMA. Como se puede observar, el método RHMA produce menos incremento de utilización para los distintos escenarios.

En la Figura 7.7 se muestra la necesidad que tiene el algoritmo RHMA del planificador combinado. Se puede observar que cuanto mayor es el número de núcleos en el sistema, mayor es la necesidad del algoritmo RHMA para usar el planificador combinado.

Por tanto, el planificador combinado demuestra ser un método de planificación que complementa útilmente al algoritmo RHMA y por ello, es imprescindible para el método de planificación conjunto que forman ambos.

7.1.2.4. Algoritmo RHMA-Planificador Combinado. Artículo 5

En esta sección se muestran los resultados que presenta el método de planificación conformado por el algoritmo basado en una técnica ILP, que llamamos RHMA y el planificador

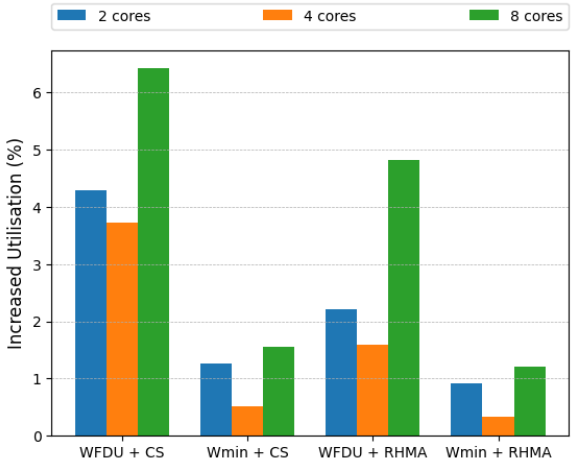


Figura 7.6: Comparación entre RHMA y el PC. Porcentaje de incremento de utilización usando dos algoritmos de asignación distintos en cada uno

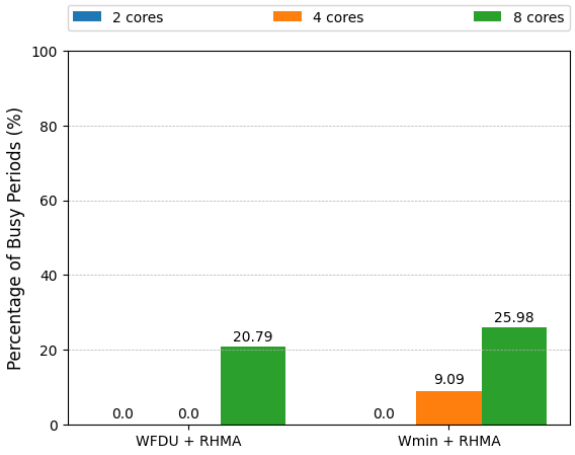


Figura 7.7: Porcentaje de intervalos de trabajo donde RHMA necesita al planificador combinado.

combinado (PC).

El rendimiento de este método se comparó con el de una heurística clásica en la planificación, EDF. Además, para mayor integridad de los resultados, se realizó la evaluación experimentando con dos asignadores de tareas distintos, WFDU y Wmin. Tras la experimentación realizada, se puede comprobar que el método de planificación propuesto obtiene mejor rendimiento que EDF. Esto se puede observar bajo dos métricas distintas, la planificabilidad del sistema y el incremento de utilización.

Respecto a la planificabilidad del sistema, en la Figura 7.8 se muestra como el método propuesto obtiene una ligera mayor planificabilidad en promedio que EDF. Esto ocurre tanto usando como algoritmo de asignación WFDU como usando Wmin.

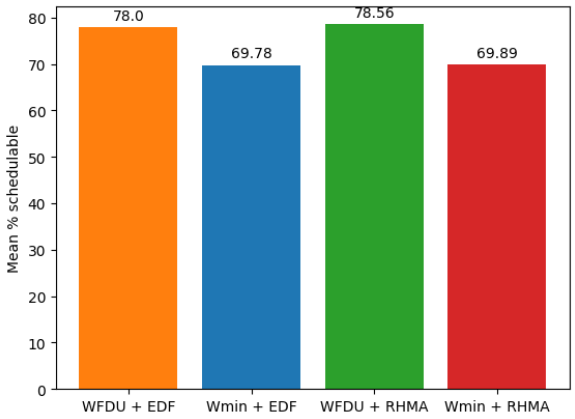


Figura 7.8: Porcentaje medio de planificabilidad de los conjuntos de tareas en función de los algoritmos de asignación y planificación.

Respecto al incremento de utilización, en la Figura 7.9 se muestra como el algoritmo propuesto obtiene mejor rendimiento respecto al incremento de utilización, tanto usando el algoritmo de asignación WFDU como usando Wmin.

Por tanto, el método de planificación de sistemas de tiempo real multinúcleo propuesto en este artículo obtiene una mejora frente a un algoritmo clásico de planificación en la literatura como es EDF. Es importante subrayar que la elección del algoritmo de asignación que se utilice conjuntamente con este método es relevante para obtener un buen rendimiento respecto a la reducción de la interferencia.

7.1.3. Resultados de los análisis de planificabilidad

En esta tesis doctoral se proponen dos metodologías para realizar análisis de planificabilidad. Por un lado se propone un método para analizar la planificabilidad de un conjunto de tareas

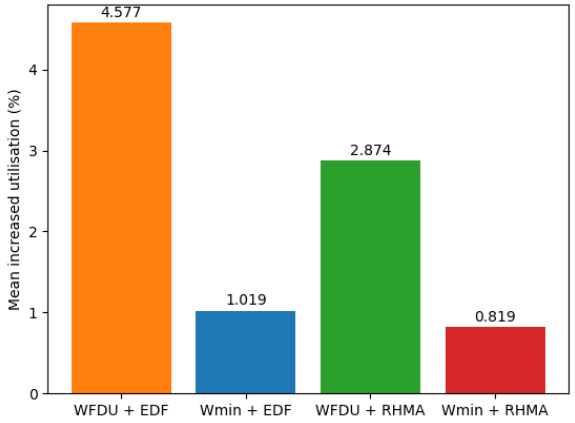


Figura 7.9: Porcentaje medio de incremento de utilización de los conjuntos de tareas en función de los algoritmos de asignación y planificación

estimando el máximo de interferencia que pueden producirse entre si las tareas de un conjunto. Este análisis se realiza considerando los plazos iguales a los periodos (artículo 3).

Y por otro lado, se propone otro análisis de planificabilidad consistente en dos métodos para estimar el límite de demanda de uso del núcleo para un conjunto de tareas. Esto se realiza considerando los plazos menores o iguales a los periodos (artículo 4).

7.1.3.1. Análisis de planificabilidad basado en la utilización con plazos iguales a periodos. Artículo 3

En esta sección se presenta los resultados obtenidos del análisis de planificabilidad que se propone en el artículo 3. Este análisis de planificabilidad presenta un método que estima el máximo de interferencia que cada tarea podría recibir de las otras tareas de un conjunto durante la planificación. En este análisis se trabaja con un modelo de tareas que considera los periodos iguales a los plazos.

En dicho artículo se realiza una experimentación para comparar varios algoritmos de asignación, entre ellos Wmin e Imin. Además, por cada conjunto de tareas usado en la experimentación, se utiliza el método propuesto para calcular la cantidad máxima de interferencia que podría producirse en la planificación.

Esta experimentación se realiza en 18 escenarios distintos. Los escenarios varían en función del número de núcleos, del número de tareas, del número de tareas que emiten interferencia, de la utilización teórica inicial y del porcentaje de interferencia que tiene cada tarea respecto a su tiempo de cómputo.

Respecto a los resultados de este método, en las Figuras 7.10 y 7.11 puede observarse la comparación entre el resultado de la utilización real y el máximo de utilización promedio que calcula el método. Esto se presenta para cada escenario y con valores promedios. A todos los conjuntos de tareas se les realizó la asignación de tareas a núcleos con el algoritmo Imin y con el algoritmo Wmin, entre otros. Concretamente, en la Figura 7.10 se compara los valores que calcula el método con la utilización real de las planificaciones cuyas tareas han sido previamente

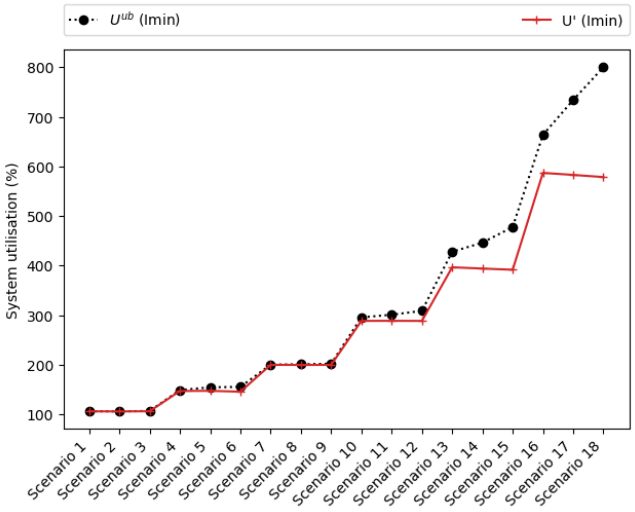


Figura 7.10: Algoritmo Imin: valores del límite máximo de utilización teórica comparado con los valores de las utilizaciones reales tras la planificación.

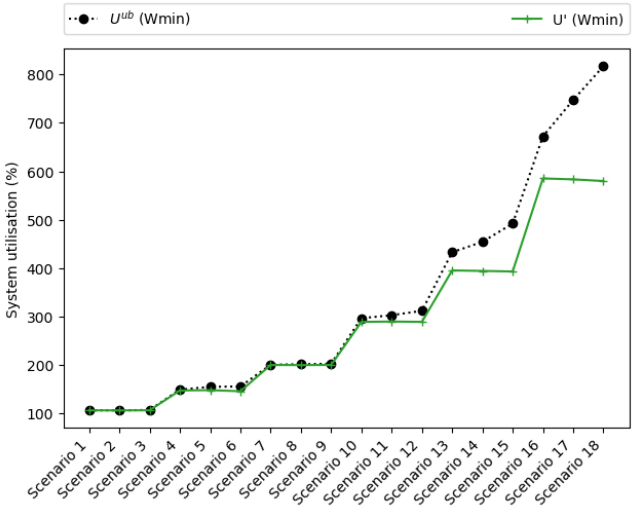


Figura 7.11: Algoritmo Wmin: valores del límite máximo de utilización teórica comparado con los valores de las utilizaciones reales tras la planificación.

asignadas por Imin. En la Figura 7.11 se realiza lo mismo respecto a Wmin. Los resultados se expresan como porcentaje de utilización del sistema para cada escenario.

En ambas Figuras puede observarse que el límite máximo de utilización que proporciona este método de análisis de planificabilidad es, como se esperaba, siempre igual o superior a la utilización real que presentan las planificaciones. Por tanto, los resultados de la experimentación indican que este método de análisis de planificabilidad es fiable.

7.1.3.2. Análisis de planificabilidad basado en el límite de la demanda con plazos inferiores o iguales a los periodos. Artículo 4

Este artículo propone dos métodos para estimar el límite en la demanda de uso del núcleo para un conjunto de tareas. El primer método se sitúa en el peor de los casos y por tanto es muy pesimista. El segundo tiene un cálculo menos pesimista.

Para evaluar ambos límites se realiza una experimentación configurando 24 escenarios distintos. Los escenarios difieren en el número de núcleos, en la carga de utilización inicial de los conjuntos de tareas, en el número de tareas, en el número de tareas que emiten interferencia y en la interferencia que cada tarea emite respecto a su tiempo de cómputo. Además, para la fase de asignación de tareas a núcleos se utilizan hasta 3 algoritmos de asignación distintos: FFDU, WFDU y Wmin.

El objetivo de la experimentación es confirmar que la utilización real de las planificaciones resultantes siempre es igual o inferior al límite de la demanda que estiman los dos métodos propuestos. Para ello, por cada método se genera un indicador (α' y α'') que representa la diferencia entre la utilización real y la estimada por cada método.

De ese modo, α' es el indicador del método más estricto y α'' es el indicador del método más relajado. En las Figuras 7.12, 7.13 y 7.14 puede observarse el porcentaje de diferencia promedio entre α' y α'' para todos los escenarios.

En las tres gráficas puede observarse, como se esperaba, que el valor de α' siempre es superior al de α''. Además se muestra que para algunos escenarios el valor es muy bajo, e incluso 0. Esto se debe a dos causas. En el caso de FFDU ocurre porque la tasa de planificabilidad es muy baja, especialmente en ciertos escenarios con un número de núcleos elevado, como es el caso del escenario 13 y en adelante, los cuales corresponden a una cantidad de 8 y 10 núcleos. Por tanto, en el caso de FFDU los valores de α' y α'' son 0 porque no se puede planificar los conjuntos de tareas en dichos escenarios. En el caso de Wmin, sencillamente se produce poca o ninguna interferencia en los escenarios donde el sistema tiene pocos núcleos.

Tras la evaluación, se concluye que los límites propuestos en este análisis de planificabilidad, $dbf'_{\tau_{M_k}}$ y $dbf''_{\tau_{M_k}}$, son válidos para asegurar la planificabilidad aunque sean pesimistas, dado que no en todas las activaciones se alcanza el máximo de interferencia. Además se observa que en el caso de $dbf''_{\tau_{M_k}}$ el límite es menos pesimista y más preciso.

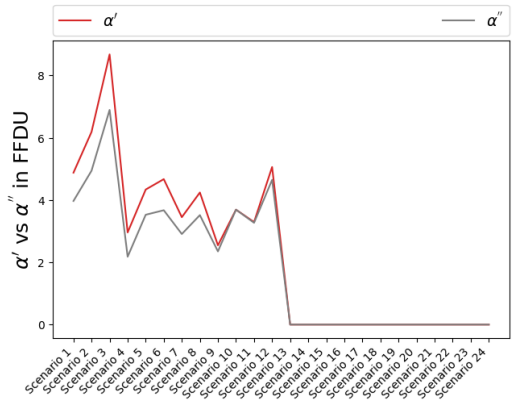


Figura 7.12: Diferencia de porcentaje entre α' y α'' para cada escenario con el algoritmo de asignación FFDU.

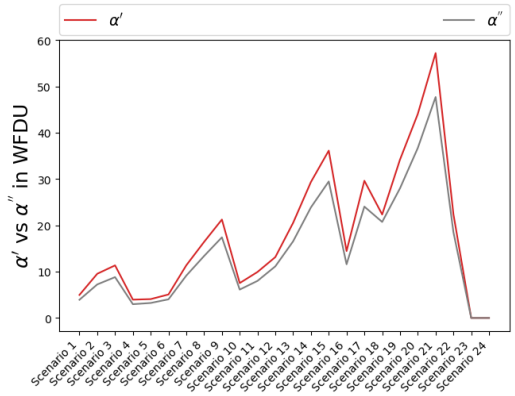


Figura 7.13: Diferencia de porcentaje entre α' y α'' para cada escenario con el algoritmo de asignación WFDU.

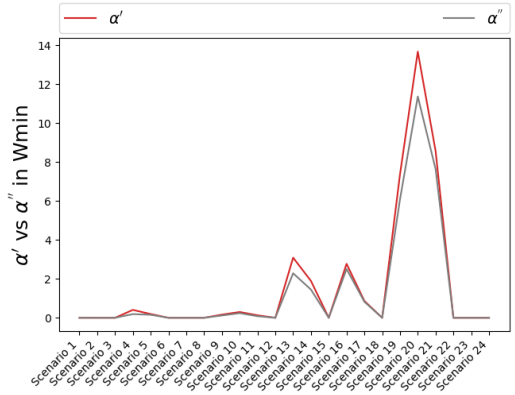


Figura 7.14: Diferencia de porcentaje entre α' y α'' para cada escenario con el algoritmo de asignación Wmin.

7.2. Trabajos futuros

En esta tesis doctoral se han propuesto diferentes metodologías para analizar y mejorar la planificación de los sistemas de tiempo real estrictos tanto mononúcleo como multinúcleo. Especialmente se ha aportado más en el ámbito de los sistemas multinúcleo. Precisamente, es en los sistemas multinúcleo donde se produce el problema de la contención de recursos compartidos. En esta tesis se ha afrontado este problema desde distintas perspectivas, desde la fase de planificación de tareas, desde la de asignación, con la propuesta de nuevos modelos de tareas o con análisis de planificabilidad.

Los algoritmos propuestos en esta tesis doctoral han sido evaluados a través de varios parámetros de rendimiento. Entre ellos se destaca la mejora de la reducción de la interferencia. Se obtiene un buen rendimiento en comparación con algoritmos tradicionales en este ámbito, como son FFDU, WFDU y BFDU en la fase de asignación, o como son EDF y DM en la fase de planificación. No obstante, se estima que el rendimiento de los algoritmos y metodologías presentadas en esta tesis doctoral aún puede mejorarse más.

Para conseguir mejor rendimiento, pueden usarse diferentes vías. Por un lado, se propone mejorar el rendimiento del algoritmo de asignación Imin respecto al incremento de la utilización, por ejemplo considerando límites de utilización menos pesimistas. Por otro lado, se plantea modificar el algoritmo de planificación propuesto RHMA para mejorar el rendimiento de otros parámetros de la planificación, como la reducción del número de cambios de contexto o acortar los tiempos de respuesta de las tareas.

Además de las técnicas usadas en los artículos de esta tesis doctoral, se plantea la opción de utilizar metodologías propias de la inteligencia artificial como son los algoritmos genéticos, el aprendizaje por refuerzo o el aprendizaje automático. Específicamente se propone el uso de una red neuronal para determinar el algoritmo de asignación más adecuado para un conjunto de tareas. El objetivo de esta metodología puede ser reducir la interferencia entre tareas. Para este reto se puede recurrir al uso de redes neuronales de tipo perceptrón o incluso recurrentes. Las redes neuronales recurrentes son otro tipo de redes neuronales artificiales que permiten analizar datos secuenciales de manera eficiente. Sin embargo, estas redes son más complejas y tienen un mayor consumo de recursos computacionales. De ese modo, se plantea utilizar ambos tipos de redes para predecir los algoritmos de asignación más adecuados para cada conjunto de tareas y comparar sus respectivos resultados.

Por otro lado, en esta tesis doctoral se ha propuesto un algoritmo de planificación llamado planificador combinado que actúa sobre los intervalos de trabajo y utiliza un conjunto de clásicos algoritmos de planificación de tareas. El planificador combinado planifica cada intervalo de trabajo con cada uno de los algoritmo candidatos. Cada algoritmo obtiene un resultado en cada intervalo de trabajo. El planificador combinado elige el algoritmo que presenta mejor resultado para cada intervalo de trabajo. Esta metodología tiene un coste respecto al tiempo empleado para la obtención del plan dado que hay que planificar cada intervalo tantas veces como algo-

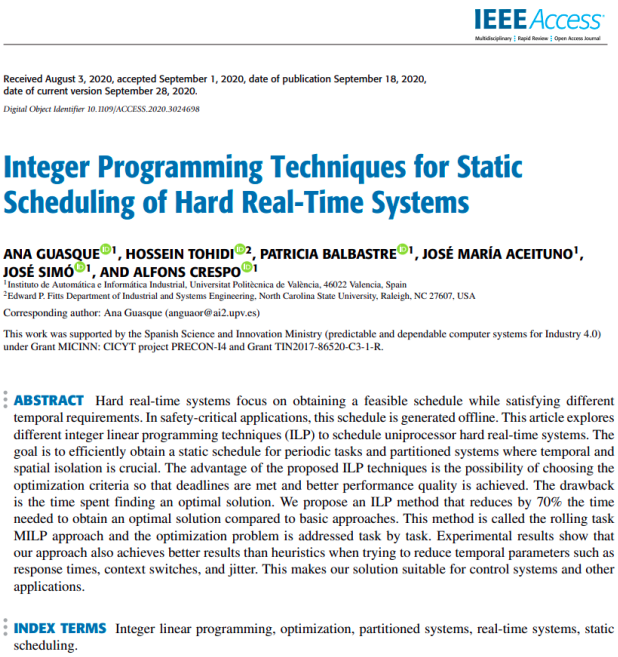
ritmos hayan. Esta técnica podría ser mejorada utilizando redes neuronales artificiales con la misma metodología que se sugirió en el párrafo anterior, de modo que la red neuronal prediga cuál es el algoritmo más adecuado para cada intervalo de trabajo. De ese modo no existiría la necesidad de tener que planificar cada intervalo con cada uno de los algoritmos candidatos.

Además, otra linea de trabajo para mejorar el rendimiento del planificador combinado puede ser la limitación de los intervalos. Actualmente el planificador combinado utiliza los momentos ociosos en todos los núcleos para establecer la delimitación de los intervalos. Estos intervalos en ocasiones son demasiado largos. Una opción podría ser establecer un límite no estricto para realizar estas planificaciones parciales. Un establecimiento aleatorio como delimitador de un intervalo puede provocar la no planificabilidad del siguiente intervalo de trabajo. Por tanto, el algoritmo que se encargase de esta tarea debería comprobar la viabilidad al establecer el fin de un intervalo en un instante dado. De este modo se evitaría el problema de establecer el fin de un intervalo en un instante que imposibilite la planificabilidad del siguiente intervalo.

7.3. Informe de los indicadores de calidad de los artículos publicados

A continuación se presenta el factor de impacto y otros datos relacionados con la calidad de cada uno de los artículos que conforman esta tesis por compendio. Los datos que se expresan por cada artículo son recogidos de las publicaciones de *Journal Citation Reports* por parte de la empresa *Clarivate*. Las métricas y puntuaciones que otorga esta publicación anual son una de las más reconocidas en el ámbito científico y académico.

7.3.1. Indicadores de calidad del Artículo 1



IEEE Access

Multidisciplinary | Rapid Review | Open Access Journal

Received August 3, 2020, accepted September 1, 2020, date of publication September 18, 2020,
date of current version September 28, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3024698

Integer Programming Techniques for Static Scheduling of Hard Real-Time Systems

ANA GUASQUE[1], HOSSEIN TOHIDI[2], PATRICIA BALBASTRE[1], JOSÉ MARÍA ACEITUNO[1], JOSÉ SIMÓ[1], AND ALFONS CRESPO[1]

[1]Instituto de Automática e Informática Industrial, Universitat Politècnica de València, 46022 Valencia, Spain
[2]Edward P. Fitts Department of Industrial and Systems Engineering, North Carolina State University, Raleigh, NC 27607, USA

Corresponding author: Ana Guasque (anguaor@ai2.upv.es)

This work was supported by the Spanish Science and Innovation Ministry (predictable and dependable computer systems for Industry 4.0) under Grant MICINN: CICYT project PRECON-I4 and Grant TIN2017-86520-C3-1-R.

ABSTRACT Hard real-time systems focus on obtaining a feasible schedule while satisfying different temporal requirements. In safety-critical applications, this schedule is generated offline. This article explores different integer linear programming techniques (ILP) to schedule uniprocessor hard real-time systems. The goal is to efficiently obtain a static schedule for periodic tasks and partitioned systems where temporal and spatial isolation is crucial. The advantage of the proposed ILP techniques is the possibility of choosing the optimization criteria so that deadlines are met and better performance quality is achieved. The drawback is the time spent finding an optimal solution. We propose an ILP method that reduces by 70% the time needed to obtain an optimal solution compared to basic approaches. This method is called the rolling task MILP approach and the optimization problem is addressed task by task. Experimental results show that our approach also achieves better results than heuristics when trying to reduce temporal parameters such as response times, context switches, and jitter. This makes our solution suitable for control systems and other applications.

INDEX TERMS Integer linear programming, optimization, partitioned systems, real-time systems, static scheduling.

Figura 7.15: primera página del artículo 1 en IEEE Access

El primer artículo fue publicado en la prestigiosa revista IEEE Access en el año 2020.
Esta revista está en tres categorías: *Telecommunications* con un cuartil de **Q2** y una posición en el ranking de **36 de 91**. También está en la categoría de *Computer Science, Information System* con un cuartil de **Q2** y una posición en el ranking de **65 de 161**. Y por último también está en la categoría de *Material Science* con un cuartil de **Q2** y una posición en el ranking de **94 de 273**.
Respecto al factor de impacto, la revista IEEE Access obtuvo en 2020 una puntuación de **3.367**.

7.3.2. Indicadores de calidad del Artículo 2

Journal of Systems Architecture 118 (2021) 102223

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

Hardware resources contention-aware scheduling of hard real-time multiprocessor systems☆

José María Aceituno, Ana Guasque *, Patricia Balbastre, José Simó, Alfons Crespo

Universitat Politècnica de València, Camino de Vera s/n, Valencia, 46022, Valencia, Spain

ARTICLE INFO

Keywords:
Multicore
Real-time
Scheduling
Memory contention
MILP

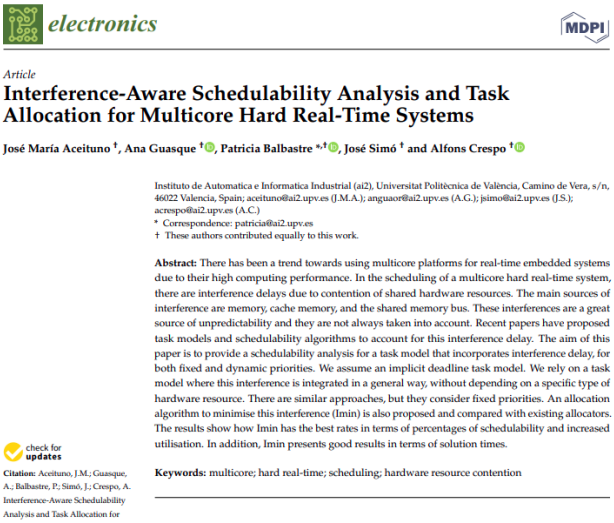
ABSTRACT

In hard real-time embedded systems, switching to multicores is a step that most application domains delay as much as possible. This is mainly due to the number of sources of indeterminism, which mainly involve shared hardware resources, such as buses, caches, and memories. In this paper, a new task model that considers the interference that task execution causes in other tasks running on other cores due to memory contention is proposed. We propose a scheduling algorithm that calculates the exact interference. We also analyse and compare existing partitioning algorithms and propose three strategies to allocate tasks to cores to schedule as many tasks as possible and minimise total interference.

Figura 7.16: primera página del artículo 2 en Journal of System Architecture

El segundo artículo fue publicado en la revista Journal of Systems Architecture en el año 2021.
Esta revista está en dos categorías: *Computer Science, Hardware and Architecture* con un cuartil de **Q1** y una posición en el ranking de **8 de 54**. Y también está en la categoría de *Computer Science, Software Engineering* con un cuartil de **Q1** y una posición en el ranking de **12 de 110**. Respecto al factor de impacto, la revista Journal of Systems Architecture obtuvo en 2021 una puntuación de **5.836**.

7.3.3. Indicadores de calidad del Artículo 3



electronics

MDPI

Article

Interference-Aware Schedulability Analysis and Task Allocation for Multicore Hard Real-Time Systems

José María Aceituno †, Ana Guasque †, Patricia Balbastre *,†, José Simó † and Alfons Crespo †

Instituto de Automatica e Informatica Industrial (ai2), Universitat Politècnica de València, Camino de Vera, s/n, 46022 Valencia, Spain; aceituno@ai2.upv.es (J.M.A.); anguaor@ai2.upv.es (A.G.); jsimo@ai2.upv.es (J.S.); acrespo@ai2.upv.es (A.C.)

* Correspondence: patricia@ai2.upv.es

† These authors contributed equally to this work.

Abstract: There has been a trend towards using multicore platforms for real-time embedded systems due to their high computing performance. In the scheduling of a multicore hard real-time system, there are interference delays due to contention of shared hardware resources. The main sources of interference are memory, cache memory, and the shared memory bus. These interferences are a great source of unpredictability and they are not always taken into account. Recent papers have proposed task models and schedulability algorithms to account for this interference delay. The aim of this paper is to provide a schedulability analysis for a task model that incorporates interference delay, for both fixed and dynamic priorities. We assume an implicit deadline task model. 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. An allocation algorithm to minimise this interference (Imin) is also proposed and compared with existing allocators. The results show how Imin has the best rates in terms of percentages of schedulability and increased utilisation. In addition, Imin presents good results in terms of solution times.

Keywords: multicore; hard real-time; scheduling; hardware resource contention

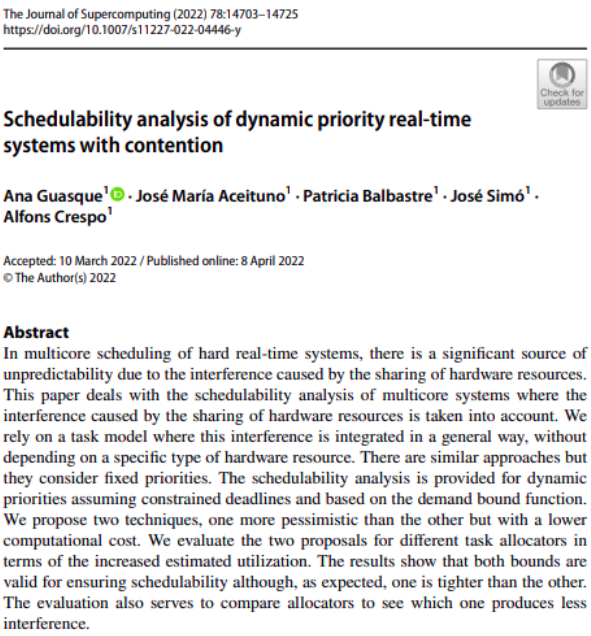
check for updates

Citation: Aceituno, J.M.; Guasque, A.; Balbastre, P.; Simó, J.; Crespo, A. Interference-Aware Schedulability Analysis and Task Allocation for

Figura 7.17: primera página del artículo 3 en Electronics

El tercer artículo fue publicado en la revista Electronics (Suiza) en el año 2022.
Esta revista está en tres categorías: *Computer Science, Information Systems* con un cuartil de **Q3** y una posición en el ranking de **99 de 158**. También está en la categoría de *Engineering, Electrical and Electronic* con un cuartil de **Q2** y una posición en el ranking de **131 de 275**. Y por último también está en la categoría de *Physics Applied* con un cuartil de **Q2** y una posición en el ranking de **71 de 159**.
Respecto al factor de impacto, la revista Electronics obtuvo en 2022 una puntuación de **2.9**.

7.3.4. Indicadores de calidad del Artículo 4



The Journal of Supercomputing (2022) 78:14703–14725
https://doi.org/10.1007/s11227-022-04446-y

Check for updates

Schedulability analysis of dynamic priority real-time systems with contention

Ana Guasque[1] · José María Aceituno[1] · Patricia Balbastre[1] · José Simó[1] · Alfons Crespo[1]

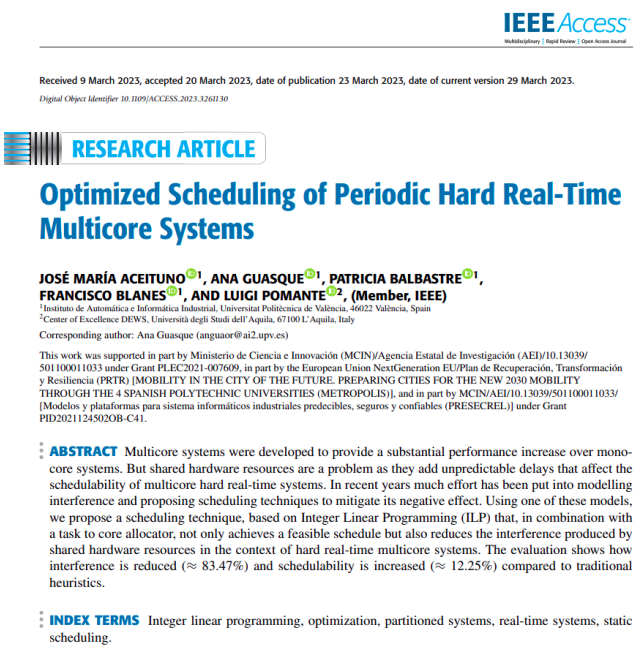
Accepted: 10 March 2022 / Published online: 8 April 2022

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.

Figura 7.18: primera página del artículo 4 en Journal of Supercomputing

El cuarto artículo fue publicado en la prestigiosa revista Journal of Supercomputing en el año 2022.
Esta revista está en tres categorías: *Computer Science, Hardware and Architecture* con un cuartil de **Q2** y una posición en el ranking de **22 de 54**. También está en la categoría de *Computer Science, Theory and Methods* con un cuartil de **Q2** y una posición en el ranking de **37 de 111**. Y por último también está en la categoría de *Engineering, Electrical and Electronic* con un cuartil de **Q2** y una posición en el ranking de **114 de 275**.
Respecto al factor de impacto, la revista Journal of Supercomputing obtuvo en 2022 una puntuación de **3.3**.

7.3.5. Indicadores de calidad del Artículo 5



IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 9 March 2023, accepted 20 March 2023, date of publication 23 March 2023, date of current version 29 March 2023.
Digital Object Identifier 10.1109/ACCESS.2023.3261130

RESEARCH ARTICLE

Optimized Scheduling of Periodic Hard Real-Time Multicore Systems

JOSÉ MARÍA ACEITUNO[1], ANA GUASQUE[1], PATRICIA BALBASTRE[1], FRANCISCO BLANES[1], AND LUIGI POMANTE[2], (Member, IEEE)
[1]Instituto de Automática e Informática Industrial, Universitat Politècnica de València, 46022 València, Spain
[2]Center of Excellence DEWS, Università degli Studi dell'Aquila, 67100 L'Aquila, Italy
Corresponding author: Ana Guasque (anguaor@ai2.upv.es)

This work was supported in part by Ministerio de Ciencia e Innovación (MCIN)/Agencia Estatal de Investigación (AEI)/10.13039/501100011033 under Grant PLEC2021-007609, in part by the European Union NextGeneration EU/Plan de Recuperación, Transformación y Resiliencia (PRTR) [MOBILITY IN THE CITY OF THE FUTURE. PREPARING CITIES FOR THE NEW 2030 MOBILITY THROUGH THE 4 SPANISH POLYTECHNIC UNIVERSITIES (METROPOLIS)], and in part by MCIN/AEI/10.13039/501100011033/ [Modelos y plataformas para sistema informáticos industriales predecibles, seguros y confiables (PRESECREL)] under Grant PID2021124502OB-C41.

ABSTRACT Multicore systems were developed to provide a substantial performance increase over mono-core systems. But shared hardware resources are a problem as they add unpredictable delays that affect the schedulability of multicore hard real-time systems. In recent years much effort has been put into modelling interference and proposing scheduling techniques to mitigate its negative effect. Using one of these models, we propose a scheduling technique, based on Integer Linear Programming (ILP) that, in combination with a task to core allocator, not only achieves a feasible schedule but also reduces the interference produced by shared hardware resources in the context of hard real-time multicore systems. The evaluation shows how interference is reduced (≈ 83.47%) and schedulability is increased (≈ 12.25%) compared to traditional heuristics.

INDEX TERMS Integer linear programming, optimization, partitioned systems, real-time systems, static scheduling.

Figura 7.19: primera página del artículo 5 en IEEE Access

El quinto artículo fue publicado en la prestigiosa revista IEEE Access en el año 2023.
A fecha del momento en el que se redacta esto, aún no existe una publicación anual de JCR para el año 2023. En su lugar, mostramos los indicadores para esta revista para el año 2022.

Esta revista está en tres categorías: *Telecommunications* con un cuartil de **Q2** y una posición en el ranking de **41 de 88**. También está en la categoría de *Computer Science, Information System* con un cuartil de **Q2** y una posición en el ranking de **72 de 158**. Y por último también está en la categoría de *Engineering, Electrical and Electronic* con un cuartil de **Q2** y una posición en el ranking de **100 de 275**.
Respecto al factor de impacto, la revista IEEE Access obtuvo en 2022 una puntuación de **3.9**.

En la Tabla 7.1 se muestran recopilados todos los datos de los indicadores de calidad que se han detallado anteriormente.

Revista	Artículo	año	Factor Imp.	Categoría	Quartil	Ranking
IEEE Access	1	2020	3.367	Telecommunications	Q2	36 de 91
				Co. Sc., Information System	Q2	65 de 161
				Material Science	Q2	94 de 173
Journal of system Architecture	2	2021	5.836	Co. Sc. Hardw. and Arch.	Q1	8 de 54
				Co.Sc. Software Eng.	Q1	12 de 110
Electronics	3	2022	2.9	Co. Sc., Information System	Q3	99 de 158
				Eng. Electrical & Electronic	Q2	131 de 275
				Physics Applied	Q2	71 de 159
Journal of supercomputing	4	2022	3.3	Co.Sc. Hardw. and Arch.	Q2	22 de 54
				Co. Sc. Theory & Methods	Q2	37 de 111
				Eng. Electrical & Electronic	Q2	114 de 275
IEEE Access	5	2023	3.9	Telecommunications	Q2	41 de 88
				Co. Sc., Information System	Q2	72 de 158
				Eng. Electrical & Electronic	Q2	100 de 275

Tabla 7.1: Tabla que sitúa los indicadores de cada una de las revistas de los artículos de esta tesis doctoral.

7.4. Coautores de los artículos

Lista de coautores de todos los artículos presentados en esta tesis:

- Ana Guasque
  Instituto Universitario de Automática e Informática Industrial (ai2)
  Universitat Politècnica de València
  Artículos 1, 2, 3, 4 y 5

- Patricia Balbastre
  Instituto Universitario de Automática e Informática Industrial (ai2)
  Universitat Politècnica de València
  Artículos 1, 2, 3, 4 y 5

- Jose Simó
  Instituto Universitario de Automática e Informática Industrial (ai2)
  Universitat Politècnica de València
  Artículos 1, 2, 3 y 4

- Alfons Crespo
  Instituto Universitario de Automática e Informática Industrial (ai2)
  Universitat Politècnica de València
  Artículos 1, 2, 3 y 4

- Francisco Blanes
  Instituto Universitario de Automática e Informática Industrial (ai2)
  Universitat Politècnica de València
  Artículo 5

- Luigi Pomante
  Center of Excellence DEWS
  Università degli Studi dell'Aquila, Italia
  Artículo 5

- Hossein Tohidi
  Department of Industrial and Systems Engineering
  North Carolina State University
  Artículo 1

Bibliografía

[1] Multicore Timing Analysis for DO-178C. `https://www.rapitasystems.com/downloads/multicore-timing-analysis-do-178c`. Rapita Systems.

[2] Avionics Application Software Standard Interface (ARINC-653). PART 1 – REQUIRED SERVICES, March 2006 2006. Airlines Electronic Eng. Committee.

[3] Gurobi optimizer reference manual. Inc. Gurobi Optimization, 2019.

[4] Aceituno, J. M., Guasque, A., Balbastre, P., Simó, J., and Crespo, A. Hardware resources contention-aware scheduling of hard real-time multiprocessor systems. Journal of Systems Architecture 118 (2021).

[5] Altmeyer, S., Davis, R. I., Indrusiak, L., Maiza, C., Nelis, V., and Reineke, J. A generic and compositional framework for multicore response time analysis. In Proceedings of the 23rd International Conference on Real Time and Networks Systems (New York, NY, USA, 2015), RTNS '15, Association for Computing Machinery, p. 129–138.

[6] Andersson, B., Kim, H., Niz, D. D., Klein, M., Rajkumar, R. R., and Lehoczky, J. Schedulability analysis of tasks with corunner-dependent execution times. ACM Trans. Embed. Comput. Syst. 17, 3 (may 2018).

[7] Baker, T. P., and Shaw, A. The cyclic executive model and ada. In Proceedings. Real-Time Systems Symposium (Dec 1988), pp. 120–129.

[8] Baruah, S. Feasibility analysis of preemptive real-time systems upon heterogeneous multiprocessor platforms. In 25th IEEE International Real-Time Systems Symposium (2004), pp. 37–46.

[9] Baruah, S. An ilp representation of a dag scheduling problem. Real-Time Systems 58 (03 2022).

[10] Baruah, S., and Fisher, N. The partitioned multiprocessor scheduling of sporadic task systems. In 26th IEEE International Real-Time Systems Symposium (RTSS'05) (Dec 2005), pp. 9 pp.–329.

[11] Baruah, S. K., Howell, R. R., and Rosier, L. E. Feasibility problems for recurring tasks on one processor. Theoretical Computer Science 118, 1 (1993), 3 – 20.

[12] Baruah, S. K., Mok, A. K., and Rosier, L. E. Preemptively scheduling hard-real-time sporadic tasks on one processor. In [1990] Proceedings 11th Real-Time Systems Symposium (Dec 1990), pp. 182–190.

[13] Baruah, S. K., Rosier, L. E., and Howell, R. R. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. Real-Time Syst. 2, 4 (Oct. 1990), 301–324.

[14] Boyd-Wickizer, S., Morris, R., and Kaashoek, M. F. Reinventing scheduling for multicore systems. In Proceedings of the 12th Conference on Hot Topics in Operating Systems (2009), HotOS'09, USENIX Association, p. 21.

[15] Burns, A. Scheduling hard real-time systems: A review. Software Engineering Journal (1991), 116–128.

[16] Burns, A., and Davis, R. I. A survey of research into mixed criticality systems. ACM Compututing Surveys 50, 6 (2017).

[17] Buttazzo, G. C. Hard real-time computing systems : predictable scheduling algorithms and applications, 3rd ed. ed. Springer, New York, 2011.

[18] Casini, D., Biondi, A., Nelissen, G., and Buttazzo, G. A holistic memory contention analysis for parallel real-time tasks under partitioned scheduling. In 2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS) (2020), pp. 239–252.

[19] (CAST), C. A. S. T. Multi-core Processors—Position Paper CAST-32A, November 2016. Technical Report.

[20] Cervin, A. Improved scheduling of control tasks. In Proceedings of 11th Euromicro Conference on Real-Time Systems. Euromicro RTS'99 (1999), pp. 4–10.

[21] Cheng, S., Stankovic, J. A., and Ramamritham, K. Scheduling algorithms for hard real-time systems–a brief survey.

[22] Choi, J., Kang, D., and Ha, S. Conservative modeling of shared resource contention for dependent tasks in partitioned multi-core systems. In 2016 Design, Automation Test in Europe Conference Exhibition (DATE) (2016), pp. 181–186.

[23] Cobham. Multi-core software considerations. Available at `https://www.gaisler.com/doc/antn/GRLIB-AN-0005.pdf` (2015/10/28), 2015.

[24] Coffman, E. G., Garey, M. R., and Johnson, D. S. Approximation Algorithms for Bin Packing: A Survey. PWS Publishing Co., USA, 1996, p. 46–93.

[25] Coffman Jr., E. G., Csirik, J., Galambos, G., Martello, S., and Vigo, D. Bin Packing Approximation Algorithms: Survey and Classification. Springer New York, New York, NY, 2013, pp. 455–531.

[26] Crespo, A., Alonso, A., Marcos, M., de la Puente, J. A., and Balbastre, P. Mixed criticality in control systems. IFAC Proceedings Volumes 47, 3 (2014), 12261 – 12271. 19th IFAC World Congress.

[27] Crespo, A., Balbastre, P., Simo, J., and Albertos, P. Static scheduling generation for multicore partitioned systems. In Information Science and Applications (ICISA) 2016 (Singapore, 2016), K. J. Kim and N. Joukov, Eds., Springer Singapore, pp. 511–522.

[28] Crespo, A., Balbastre, P., Simó, J., Coronel, J., Gracia Pérez, D., and Bonnot, P. Hypervisor-based multicore feedback control of mixed-criticality systems. IEEE Access 6 (2018), 50627–50640.

[29] Crespo, A., Ripoll, I., and Albertos, P. Reducing delays in rt control: The control action interval. IFAC Proceedings Volumes 32, 2 (1999), 8527 – 8532. 14th IFAC World Congress 1999, Beijing, Chia, 5-9 July.

[30] Dasari, D., Akesson, B., Nélis, V., Awan, M. A., and Petters, S. M. Identifying the sources of unpredictability in cots-based multicore systems. In 2013 8th IEEE International Symposium on Industrial Embedded Systems (SIES) (2013), pp. 39–48.

[31] Dasari, D., Andersson, B., Nelis, V., Petters, S. M., Easwaran, A., and Lee, J. Response time analysis of cots-based multicores considering the contention on the shared memory bus. In 2011IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications (2011), pp. 1068–1075.

[32] Davis, R., Altmeyer, S., Indrusiak, L., Maiza, C., Nelis, V., and Reineke, J. An extensible framework for multicore response time analysis. Real-Time Systems 54 (07 2018).

[33] Davis, R. I., and Burns, A. Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. In 2009 30th IEEE Real-Time Systems Symposium (Dec 2009), pp. 398–409.

[34] Davis, R. I., and Burns, A. A survey of hard real-time scheduling for multiprocessor systems. ACM Comput. Surv. 43, 4 (Oct. 2011).

[35] Davis, R. I., Griffin, D., and Bate, I. Schedulability Analysis for Multi-Core Systems Accounting for Resource Stress and Sensitivity. In 33rd Euromicro Conference on Real-Time Systems (ECRTS 2021) (Dagstuhl, Germany, 2021), B. B. Brandenburg, Ed., vol. 196 of Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl – Leibniz-Zentrum für Informatik, pp. 7:1–7:26.

[36] Devaraj, R., Sarkar, A., and Biswas, S. Real-time scheduling of non-preemptive sporadic tasks on uniprocessor systems using supervisory control of timed des. In 2017 American Control Conference (ACC) (2017), pp. 3212–3217.

[37] Devaraj, R., Sarkar, A., and Biswas, S. Exact task completion time aware real-time scheduling based on supervisory control theory of timed des. In 2018 European Control Conference (ECC) (2018), pp. 1908–1913.

[38] Di Natale, M., and Zeng, H. An efficient formulation of the real-time feasibility region for design optimization. IEEE Transactions on Computers 62 (04 2013), 644–661.

[39] Easwaran, A., Anand, M., and Lee, I. Compositional analysis framework using edp resource models. In 28th IEEE International Real-Time Systems Symposium (RTSS 2007) (2007), pp. 129–138.

[40] Easwaran, A., Lee, I., Sokolsky, O., and Vestal, S. A compositional framework for avionics (arinc-653) systems. Tech Report MS- CIS-09-04, University of Pennsylvania. (01 2009).

[41] Escoffier, B., Bonifaci, V., and Ausiello, G. Complexity and approximation in reoptimization. Computability in Context: Computation and Logic in the Real World (02 2011).

[42] Fernandez, G., Abella, J., Quiñones, E., Rochange, C., Vardanega, T., and Cazorla, F. Contention in multicore hardware shared resources: Understanding of the state of the art. In WCET (2014).

[43] Fleming, T., and Burns, A. Investigating mixed criticality cyclic executive schedule generation. In Proc. Workshop on Mixed Criticality (WMC) (2015).

[44] Galizzi, J., Vigeant, F., Perraud, L., Crespo, A., Masmano, M., Carrascosa, E., Brocal, V., Balbastre, P., Quartier, F., and Milhorat, F. Wcet and multicores with tsp. In DASIA 2014 DAta Systems In Aerospace (2014).

[45] Garey, M. R., and Johnson, D. S. Computers and Intractability; A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York, NY, USA, 1990.

[46] Gracioli, G., Alhammad, A., Mancuso, R., Fröhlich, A. A., and Pellizzoni, R. A survey on cache management mechanisms for real-time embedded systems. ACM Comput. Surv. 48, 2 (nov 2015).

[47] Guasque, A., Aceituno, J. M., Balbastre, P., Simó, J., and Crespo, A. Schedulability analysis of dynamic priority real-time systems with contention. J. Supercomput. 78, 12 (aug 2022), 14703–14725.

[48] Guasque, A., Tohidi, H., Balbastre, P., Aceituno, J. M., Simó, J., and Crespo, A. Integer programming techniques for static scheduling of hard real-time systems. IEEE Access 8 (2020), 170389–170403.

[49] Guo, Z., Yang, K., Yao, F., and Awad, A. Inter-task cache interference aware partitioned real-time scheduling. In Proceedings of the 35th Annual ACM Symposium on Applied Computing (New York, NY, USA, 2020), SAC '20, Association for Computing Machinery, p. 218–226.

[50] Hantom, W., Aldweesh, A., Alzaher, R., and Rahman, A. A survey on scheduling algorithms in real-time systems. 686–690.

[51] Harter, Jr., P. K. Response times in level-structured systems. ACM Trans. Comput. Syst. 5, 3 (Aug. 1987), 232–248.

[52] Hassan, M., and Pellizzoni, R. Analysis of Memory-Contention in Heterogeneous COTS MPSoCs. In 32nd Euromicro Conference on Real-Time Systems (ECRTS 2020) (2020), M. Völp, Ed., vol. 165 of Leibniz International Proceedings in Informatics (LIPIcs), pp. 23:1–23:24.

[53] He, C., Zhu, X., Guo, H., Qiu, D., and Jiang, J. Rolling-horizon scheduling for energy constrained distributed real-time embedded systems. Journal of Systems and Software 85, 4 (2012), 780–794.

[54] Henties, T., Ag, S., Hunt, J., Locke, D., Nilsen, K., Na, A., Schoeberl, M., and Vitek, J. Java for safety-critical applications. Electronic Notes in Theoretical Computer Science - ENTCS (01 2009).

[55] Hong, I., Kirovski, D., Gang Qu, Potkonjak, M., and Srivastava, M. B. Power optimization of variable-voltage core-based systems. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 18, 12 (1999), 1702–1714.

[56] Huang, W.-H., Chen, J.-J., and Reineke, J. Mirror: Symmetric timing analysis for real-time tasks on multicore platforms with shared resources. In Proceedings of the 53rd Annual Design Automation Conference (New York, NY, USA, 2016), DAC '16, Association for Computing Machinery.

[57] Igarashi, S., Ishigooka, T., Horiguchi, T., Koike, R., and Azumi, T. Heuristic contention-free scheduling algorithm for multi-core processor using let model. In 2020 IEEE/ACM 24th International Symposium on Distributed Simulation and Real Time Applications (DS-RT) (2020), pp. 1–10.

[58] Insik Shin, and Insup Lee. Periodic resource model for compositional real-time guarantees. In RTSS 2003. 24th IEEE Real-Time Systems Symposium, 2003 (2003), pp. 2–13.

[59] Jeffay, K. Scheduling sporadic tasks with shared resources in hard-real-time systems. In [1992] Proceedings Real-Time Systems Symposium (1992), pp. 89–99.

[60] Johnson, D. Near-Optimal Bin Packing Algorithms. PhD thesis, Massachusetts Institute of Technology, Dept. of Math., 1973.

[61] Joseph, M., and Pandya, P. Finding response times in a real-time system. The Computer Journal 29, 5 (1986), 390–395.

[62] Karuppiah, N. The impact of interference due to resource contention in multicore platform for safety-critical avionics systems. International Journal for Research in Engineering Application Management (IJREAM) 02 (01 2016), 39–48.

[63] Karuppiah, N. The impact of interference due to resource contention in multicore platform for safety-critical avionics systems. International Journal for Research in Engineering Application Management (IJREAM) 02 (01 2016), 39–48.

[64] Kim, H., de Niz, D., Andersson, B., Klein, M., Mutlu, O., and Rajkumar, R. Bounding memory interference delay in cots-based multi-core systems. In 2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS) (2014), pp. 145–154.

[65] Kim, H., De Niz, D., Andersson, B., Klein, M., Mutlu, O., and Rajkumar, R. Bounding and reducing memory interference in cots-based multi-core systems. Real-Time Syst. 52, 3 (May 2016), 356–395.

[66] Kim, J., Oh, H., Ha, H., Kang, S.-H., Choi, J., and Ha, S. An ilp-based worst-case performance analysis technique for distributed real-time embedded systems. pp. 363–372.

[67] Lampka, K., Giannopoulou, G., Pellizzoni, R., Wu, Z., and Stoimenov, N. A formal approach to the wcrt analysis of multicore systems with memory contention under phase-structured task sets. Real-Time Systems 50 (11 2014), 736–773.

[68] Lehoczky, J. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In [1990] Proceedings 11th Real-Time Systems Symposium (1990), pp. 201–209.

[69] Leung, J. Y.-T., and Whitehead, J. On the complexity of fixed-priority scheduling of periodic, real-time tasks. Performance Evaluation 2, 4 (1982), 237 – 250.

[70] Lisper, B., and Mellgren, P. Response-time calculation and priority assignment with integer programming methods.

[71] Liu, C. L., and Layland, J. W. Scheduling algorithms for multiprogramming in a hard-real-time environment. J. ACM 20, 1 (jan 1973), 46–61.

[72] Liu, C. L., and Layland, J. W. Scheduling algorithms for multiprogramming in a hard-real-time environment. J. ACM 20, 1 (Jan. 1973), 46–61.

[73] Locke, C. D. Software architecture for hard real-time applications: Cyclic executives vs. fixed priority executives. Real-Time Syst. 4, 1 (Mar. 1992), 37–53.

[74] Lugo, T., Lozano, S., Fernández, J., and Carretero, J. A survey of techniques for reducing interference in real-time applications on multicore platforms. IEEE Access 10 (2022), 21853–21882.

[75] Maiza, C., Rihani, H., Rivas, J. M., Goossens, J., Altmeyer, S., and Davis, R. I. A survey of timing verification techniques for multi-core real-time systems. ACM Comput. Surv. 52, 3 (jun 2019).

[76] Mangeruca, L., Baleani, M., Ferrari, A., and Sangiovanni-Vincentelli, A. Uniprocessor scheduling under precedence constraints for embedded systems design. ACM Trans. Embed. Comput. Syst. 7, 1 (Dec. 2007).

[77] Marquant, J. F., Evins, R., and Carmeliet, J. Reducing computation time with a rolling horizon approach applied to a milp formulation of multiple urban energy hub system. Procedia Computer Science 51 (2015), 2137–2146. International Conference On Computational Science, ICCS 2015.

[78] Mitra, T., Teich, J., and Thiele, L. Time-critical systems design: A survey. IEEE Design Test 35, 2 (2018), 8–26.

[79] Mohammadi, A., and Akl, S. Scheduling algorithms for real-time systems.

[80] Mok, A. Fundamental design problems of distributed systems for the hard-real-time environment.

[81] Mok, A. K., and Xiang Alex. Towards compositionality in real-time resource partitioning based on regularity bounds. In Proceedings 22nd IEEE Real-Time Systems Symposium (RTSS 2001) (Cat. No.01PR1420) (2001), pp. 129–138.

[82] Nguyen, V. A., Hardy, D., and Puaut, I. Cache-conscious off-line real-time scheduling for multi-core platforms: algorithms and implementation. Real-Time Systems 55, 4 (2019), 810–849.

[83] Nowotsch, J. Interference-sensitive Worst-case Execution Time Analysis for Multi-core Processors. PhD thesis, November 2014.

[84] Nuth, P. R., and Dally, W. J. A mechanism for efficient context switching. In [1991 Proceedings] IEEE International Conference on Computer Design: VLSI in Computers and Processors (1991), pp. 301–304.

[85] Oh, S.-H., and Yang, S.-M. 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) (1998), pp. 31–36.

[86] Oh, Y., and Son, S. H. Allocating fixed-priority periodic tasks on multiprocessor systems. Real-Time Syst. 9, 3 (Nov. 1995), 207–239.

[87] Paul, A. A., and S. Pillai, B. A. Reducing the number of context switches in real time systems. In 2011 International Conference on Process Automation, Control and Computing (2011).

[88] Pellizzoni, R., Betti, E., Bak, S., Yao, G., Criswell, J., Caccamo, M., and Kegley, R. A predictable execution model for cots-based embedded systems. In 2011 17th IEEE Real-Time and Embedded Technology and Applications Symposium (2011), pp. 269–279.

[89] Pellizzoni, R., and Lipari, G. Feasibility analysis of real-time periodic tasks with offsets. Real-Time Systems 30 (05 2005), 105–128.

[90] Pellizzoni, R., Schranzhofer, A., Jian-Jia Chen, Caccamo, M., and Thiele, L. Worst case delay analysis for memory interference in multicore systems. In 2010 Design, Automation Test in Europe Conference Exhibition (DATE 2010) (2010), pp. 741–746.

[91] Ramamritham, K., Stankovic, J. A., and Shiah, P. F. Efficient scheduling algorithms for real-time multiprocessor systems. IEEE Trans. Parallel Distrib. Syst. 1, 2 (apr 1990), 184–194.

[92] Reder, S., and Becker, J. Interference-aware memory allocation for real-time multi-core systems. 2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS) (2020), 148–159.

[93] Rihani, H., Moy, M., Maiza, C., Davis, R. I., and Altmeyer, S. 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 (2016), RTNS '16, p. 67–76.

[94] Ripoll, I., Crespo, A., and Mok, A. K. Improvement in feasibility testing for real-time tasks. Real-Time Syst. 11, 1 (July 1996), 19–39.

[95] Rivera-Verduzco, H. J., and Bril, R. J. Best-case response times of real-time tasks under fixed-priority scheduling with preemption thresholds. In Proceedings of the 25th International Conference on Real-Time Networks and Systems (New York, NY, USA, 2017), RTNS '17, Association for Computing Machinery, p. 307–346.

[96] Rouxel, B., Derrien, S., and Puaut, I. Tightening contention delays while scheduling parallel applications on multi-core architectures. ACM Trans. Embed. Comput. Syst. 16, 5s (Sept. 2017).

[97] Salmani, V., Taghavi Zargar, S., and Naghibzadeh, M. A modified maximum urgency first scheduling algorithm for real-time tasks. Proc. Seventh World Enformatika Conference (01 2005).

[98] Skalistis, S., and Kritikakou, A. Dynamic Interference-Sensitive Run-time Adaptation of Time-Triggered Schedules. In ECRTS 2020 - 32nd Euromicro Conference on Real-Time Systems (July 2020), pp. 1–22.

[99] Spuri, M. Analysis of deadline scheduled real-time systems. Tech. rep., 1996.

[100] Stankovic, J. A., and Ramamritham, K. Hard Real-Time Systems. IEEE Computer Society Press, Washington, DC, USA, 1988.

[101] Sun, Y., and Natale, M. D. Weakly hard schedulability analysis for fixed priority scheduling of periodic real-time tasks. ACM Trans. Embed. Comput. Syst. 16, 5s (Sept. 2017).

[102] Tompkins, M. F. Optimization techniques for task allocation and scheduling in distributed multi-agent operations. PhD thesis, Massachusetts Institute of Technology, Dept. of Electrical Engineering and Computer Science, 2003.

[103] Xiao, J., Altmeyer, S., and Pimentel, A. Schedulability analysis of non-preemptive real-time scheduling for multicore processors with shared caches. In 2017 IEEE Real-Time Systems Symposium (RTSS) (2017), pp. 199–208.

[104] Zamorano, J., Alonso, A., and [de la Puente], J. Building safety-critical real-time systems with reusable cyclic executives. Control Engineering Practice 5, 7 (1997), 999 – 1005.

[105] Zhang, F., and Burns, A. A worst-case pattern of task load allocation and execution for multiprocessor global real-time scheduling. International Journal of Simulation: Systems, Science Technology 17 (01 2016), 9.1–9.4.