Universitat Politècnica de València

Departament d'Informàtica de Sistemes i Computadors

Tesis Doctoral

Utilización de Memorias Cache con Bloqueo en Sistemas de Tiempo Real

Presentada por:
D. Antonio Martí Campoy

Dirigida por:
Dr. D. J osé Vicente Busquets Mataix

Julio 2003

A mis padres,
por enseñarme el
camino y ayudarme
a recorrerlo

Agradecimientos:

Al Doctor D. José Vicente Busquets Mataix, director de esta tesis, por su orientación, constante apoyo y ayuda durante todo el trabajo realizado.

A D. Alberto Pérez y a D. Angel Perles, compañeros del Departamento de Informática de Sistemas y Computadores, por su constante disponibilidad y pronto socorro en momentos de dificultad.

A todos los miembros del grupo de investigación de Sistemas Tolerantes a Fallos, en especial a los Doctores D. Juan José Serrano, D. Rafael Ors y D. Pedro Gil, directores del mismo, y sin cuya ayuda esta tesis no hubiera sido posible. Así mismo, al Doctor D. José Carlos Campelo y a D. Francisco Rodríguez por su inestimable ayuda y consejo.

Especial agradecimiento merecen mi mujer y mi hijo, por el tiempo prestado para la realización de esta tesis.

Resum

Els processadors actuals ofereixen una relació preu prestacions molt interessant, a més d'altres qualitats com la garantia de funcionament o la gran disponibilitat d'eines de desenvolupament. Aquest conjunt de virtuts els fa molt atractius per al desenvolupament de qualsevol sistema informàtic, inclosos els sistemes de temps real (STR).

No obstant això, els sistemes de temps real necessiten verificar no sols la correcció dels càlculs i operacions que realitzen, sinó que també és necessari garantir que les tasques que ha de realitzar el sistema es duran a terme dins dels límits temporals establerts. I aquesta garantia ha d'obtindre's en qualsevol circumstància i condició.

En la validació de la correcció temporal d'un STR, tasca que rep el nom d'*Anàlisi de Planificabilitat*, és on sorgeixen els problemes amb els processadors actuals. Aquestos processadors aconsegueixen alts nivells de prestacions gràcies als avanços en la tecnologia, però també gràcies a la inclusió de millores en la seua estructura i arquitectura que permeten aprofitar els recursos disponibles de la millor manera possible. Però aquest bon ús dels recursos no es produeix de forma constant, sinó que dependrà de l'estructura i les dades del programa que s'execute. D'aquesta manera, les prestacions oferides per un processador variaran per als diferents programes que execute i fins i tot per al mateix programa en funció de les seues dades d'entrada. Aquesta falta de determinisme en la resposta temporal del processador complica de manera important la realització de l'anàlisi de planificabilitat.
Un cas concret d'aquestes millores estructurals que presenten una seriosa falta de determinisme és la memòria *cache*. La seua inclusió en la jerarquia de memòria dels computadors ha permés aconseguir unes prestacions molt elevades, per la qual cosa s'han convertit en un element comú en la majoria dels sistemes informàtics. No obstant això, la gran variabilitat que introdueix en els temps d'execució ha generat dos efectes: d'una banda, han sigut tradicionalment evitades pels dissenyadors de sistemes de temps real; d'altra, s'han publicat un important nombre de treballs per a permetre el seu ús en sistemes de temps real. La gran quantitat de treballs relacionats amb l'ús de memòries *cache* en els STR ofereix una idea de la importància del problema.

En aquesta tesi es descriuen les principals solucions al problema generat per l'ús de les memòries *cache* en els STR, i s'identifiquen les mancances i inconvenients de les propostes existents fins al moment. Per a superar aquestos inconvenients, aquest treball proposa una nova estructura de memòria *cache* que rep el nom de *cache* amb bloqueig (*locking cache*), i que ofereix un comportament totalment determinista i predicible.

Per a la utilització de la memòria *cache* amb bloqueig en STR es proposen dos modes diferents, anomenats *ús estàtic* i *ús dinàmic*. Encara que l'objectiu principal d'aquest treball és l'obtenció de determinisme, l'ús dinàmic presenta un menor grau de predicibilitat, però per contra ofereix, de forma general, millors prestacions que l'ús estàtic. Per a aquestos dos usos es presenten els algoritmes i mètodes que permeten realitzar l'anàlisi de planificabilitat d'un STR, utilitzant dues polítiques de planificació distintes. Aquestos algorismes tenen, com a gran virtut, una gran senzillesa comparada amb els proposats anteriorment.

El determinisme que ofereix la *cache* amb bloqueig se sustenta en la precàrrega i bloqueig de la mateixa en determinats instants del funcionament del sistema. La selecció dels continguts que es bloquejaran en *cache* és un problema crucial des del punt de vista de les prestacions. Amb l'objectiu d'aconseguir simultàniament el determinisme perseguit i unes prestacions semblants a les oferides per les *caches* convencionals, s'ha desenvolupat un algorisme genètic que permet, en un temps de còmput acceptable, trobar el conjunt d'instruccions subòptim que ha de ser precarregat en la *cache* amb bloqueig per a un determinat sistema.

Per a verificar que els dos objectius perseguits –determinisme i prestacions- s'aconsegueixen amb les propostes detallades en aquesta tesi, s'ha realitzat un conjunt d'anàlisis estadístiques que permeten avaluar el guany o pèrdua de prestacions que s'obté al utilitzar una *cache* amb bloqueig enfront d'una *cache* convencional. Aquesta informació s'obté en funció d'un conjunt nombrós de característiques del sistema, tant del programari com del maquinari, per la qual cosa els seus resultats són un eina útil per al dissenyador de sistemes de temps real, ja que li permetran avaluar el cost, des del punt de vista de les prestacions, d'utilitzar un sistema determinista i predicible, i per tant, d'utilitzar unes eines senzilles i practicables per a la realització de l'anàlisi de planificabilitat.

Paraules clau: memòries *cache*, sistemes de temps real, algorismes genètics, temps d'execució, temps de resposta, anàlisi de planificabilitat, avalaució de prestacions, disseny d'experiments, regressió lineal.

Resumen

Los procesadores actuales ofrecen una relación precio prestaciones muy interesante, además de otras cualidades como la garantía de funcionamiento o la gran disponibilidad de herramientas de desarrollo. Este conjunto de virtudes los hace muy atractivos para el desarrollo de cualquier sistema informático, incluidos los sistemas de tiempo real (STR).

Sin embargo, los sistemas de tiempo real necesitan verificar no sólo la corrección de los cálculos y operaciones que realizan, sino que también es necesario garantizar que las tareas que debe realizar el sistema se llevarán a cabo dentro de los límites temporales establecidos. Y esta garantía debe obtenerse bajo cualquier circunstancia y condición.

En la validación de la corrección temporal de un STR, tarea que recibe el nombre de *Análisis de Planificabilidad*, es donde surgen los problemas con los procesadores actuales. Dichos procesadores alcanzan altos niveles de prestaciones gracias a los avances en la tecnología, pero también gracias a la inclusión de mejoras en su estructura y arquitectura que permiten aprovechar los recursos disponibles de la mejor manera posible. Pero este buen uso de los recursos no se produce de forma constante, sino que dependerá de la estructura y los datos del programa que se ejecute. De este modo, las prestaciones ofrecidas por un procesador variarán para los diferentes programas que ejecute e incluso para el mismo programa en función de sus datos de entrada. Esta falta de determinismo en la respuesta temporal del procesador complica de manera importante la realización del análisis de planificabilidad.

Un caso concreto de estas mejoras estructurales que presentan una seria falta de determinismo es la memoria *cache*. Su inclusión en la jerarquía de memoria de los computadores ha permitido alcanzar unas prestaciones muy elevadas, por lo que se han convertido en un elemento común en la mayoría de los sistemas informáticos. Sin embargo, la gran variabilidad que introduce en los tiempos de ejecución ha generado dos efectos: por un lado, han sido tradicionalmente evitadas por los diseñadores de sistemas de tiempo real; por otro lado, se han publicado un importante número de trabajos para permitir su uso en sistemas de tiempo real. La gran cantidad de trabajos relacionados con el uso de memorias *cache* en los STR ofrece una idea de la importancia del problema.

En esta tesis se describen las principales soluciones al problema generado por el uso de las memorias *cache* en los STR, y se identifican las carencias e inconvenientes de las propuestas existentes hasta el momento. Para superar dichos inconvenientes, este trabajo propone una nueva estructura de memoria *cache* que recibe el nombre de *cache* con bloqueo (*locking cache*), y que ofrece un comportamiento totalmente determinista y predecible.

Para la utilización de la memoria *cache* con bloqueo en STR se proponen dos modos diferentes, llamados *uso estático* y *uso dinámico*. Aunque el objetivo principal de este trabajo es la obtención de determinismo, el uso dinámico presenta un menor grado de predecibilidad, pero por contra ofrece, de forma general, mejores prestaciones que el uso estático. Para ambos usos se presentan los algoritmos y métodos que permiten realizar el análisis de planificabilidad de un STR, utilizando dos políticas de planificación distintas. Estos algoritmos tienen, como gran virtud, una gran sencillez comparada con los propuestos anteriormente.

El determinismo que ofrece la *cache* con bloqueo se sustenta en la precarga y bloqueo de la misma en determinados instantes del funcionamiento del sistema. La selección de los contenidos que se bloquearán en *cache* es un problema crucial desde el punto de vista de las prestaciones. Con el objetivo de alcanzar simultáneamente el determinismo perseguido y unas prestaciones similares a las ofrecidas por las *cache*s convencionales, se ha desarrollado un algoritmo genético que permite, en un tiempo de cómputo aceptable, encontrar el conjunto de instrucciones subóptimo que debe ser precargado en la *cache* con bloqueo para un determinado sistema.

Para verificar que los dos objetivos perseguidos –determinismo y prestaciones- se alcanzan con las propuestas detalladas en esta tesis, se ha realizado un conjunto de análisis estadísticos que permiten evaluar la ganancia o pérdida de prestaciones que se obtienen al utilizar una *cache* con bloqueo frente a una *cache* convencional. Esta información se obtiene en función de un numeroso conjunto de características del sistema, tanto *software* como *hardware*, por lo que sus resultados son una herramienta útil para el diseñador de sistemas de tiempo real, ya que le permitirán evaluar el coste, desde el punto de vista de las prestaciones, de utilizar un sistema determinista y predecible, y por tanto, de utilizar unas herramientas sencillas y practicables para la realización del análisis de planificabilidad.

Palabras clave: memorias *cache*, sistemas de tiempo real, algoritmos genéticos, tiempo de ejecución, tiempo de respuesta, análisis de planificabilidad, evaluación de prestaciones, diseño de experimentos, regresión lineal.

Abstract

Today processors offer a very interesting price performance relationship, in addition to other advantages like reliability or the great availability of development tools. This set of virtues makes them very attractive for the development of any computer system, including real-time systems (RTS).

Nevertheless, real-time systems need to verify not only the correctness of the calculations and operations that make, but also it is necessary to guarantee that the tasks will finish within the established temporary limits. And this guarantee must be obtained under any circumstance and condition.

In the validation of the temporary correctness of a RTS, task that receives the name of *Schedulability Analyses*, it is where the problems with the present processors arise. These processors reach high levels of performance thanks to the advances in the technology, but also due to the inclusion of improvements in their structure and architecture, using the best way possible the resources available. But this good use of the resources is not constant, and it depends on the structure and the data of the task in execution. This way, the performance offered by a processor will vary for the different programs that it executes and even for the same program based on their input data. This lack of determinism in its temporal behaviour presents serious issues in the accomplishment of the schedulability analysis.

An example of these structural improvements that present a serious lack of determinism is the cache memory. Its inclusion in the memory hierarchy of computers has allowed reaching very high performance, so they have become a common element in most of the computer systems. However, the great variability that introduces in the execution times has generated two effects: on the one hand, they have been traditionally avoided by the designers of real-time systems; on the other hand, an important number of works have been published to allow its use in real-time systems. The great amount of works related to the use of cache memories in RTS offers an idea of the importance of the problem.

In this thesis the main solutions to the problem generated by the use of cache memories in RTS are described, and the deficiencies and disadvantages of the existing proposals are identified. In order to surpass these disadvantages, this work proposes a new structure of cache memory, called *locking cache*, which offers a totally determinist and predictable behaviour.

Two ways of using locking cache in RTS is proposed, calling them *static use* and *dynamic use*. Although the major objective of this work is to reach determinism, the dynamic use presents a smaller degree of predictability, but it offers better performance than the static use. For both uses are presented the algorithms and methods necessaries to make the schedulability analysis of a RTS, using two different scheduling policies. These algorithms have a great simplicity compared with the previously proposed ones.

Locking cache obtains determinism by pre-loading and locking its contents at certain moments of the system operation. The selection of the contents that will be locked in cache is a crucial problem from the performance point of view. With the objective to reach both determinism and performance, the thesis presents a genetic algorithm that finds, within acceptable computational cost, the sub-optimal set of instructions that must be preloaded in cache.

In order to verify that both goals -determinism and performance- are reached with the proposals detailed in this thesis, a set of statistical analyses has been made that allow to evaluate the gain or lost of performance that are obtained when using locking cache, versus the use of conventional one. Results have been obtained from a numerous set of systems characteristics, software and hardware. The real-time engineer may use this information to a priori evaluate the appropriateness of the locking cache for a given system. Using locking cache, the system is very predictable and allows simple and practicable tools for the accomplishment of the schedulability analysis.

Key words: cache memories, real-time systems, genetic algorithms, execution time, response time, schedulability analysis, performance evaluation, design of experiments, linear regression.

Índice general

Índice de figuras

Índice de tablas

Motivación y objetivos

A. Motivación

Tradicionalmente los sistemas informáticos de tiempo real se han diseñado *ad-hoc*, tanto en su parte *hardware* como *software*. Esto es debido, por una parte, a la especificidad de los problemas y aplicaciones en las que se necesitan este tipo de sistemas, y por otra parte, a la idiosincrasia propia de los sistemas de tiempo de real. La obligación de que estos tipos de sistemas proporcionen los resultados deseados dentro de un margen de tiempo es un requisito que supedita, en muchos casos, cualquier otra consideración o decisión a la hora de diseñar un sistema informático de tiempo real.

Algunas de las áreas y disciplinas donde se requieren sistemas de tiempo real han sido, y siguen siendo, la aeronáutica, la exploración espacial y los sistemas de mantenimiento de vida, entre otras. Sin embargo, otros campos como la automoción, los sistemas multimedia o las telecomunicaciones han incorporado, en los últimos tiempos, requisitos de tiempo real. Estos nuevos ámbitos de aplicación de los sistemas de tiempo de real corresponden con productos de consumo, donde el factor coste se convierte en un parámetro determinante a la hora de diseñar el sistema final. Además, otras necesidades, como una elevada potencia de cálculo, fiabilidad, disponibilidad de los componentes y herramientas o la facilidad de programación, modificación e integración cobran una importancia capital en el diseño del sistema.

Estos últimos requisitos han hecho que los procesadores de propósito general aumenten su atractivo de cara a los diseñadores de sistemas de tiempo real, e incluso se conviertan en la única posibilidad para obtener un producto competitivo. La fabricación y venta masiva de estos microprocesadores permite disponer de muy altas prestaciones a un precio extremadamente interesante. Además, son probados de forma exhaustiva antes y después de su puesta en circulación, por lo que se trata de un producto altamente fiable. Y por supuesto, disponen de un número elevado de herramientas de desarrollo. En resumen, se trata de dispositivos baratos, potentes, fáciles de utilizar y muy fiables.

Sin embargo, aunque en las nuevas áreas de aplicación de los sistemas de tiempo real cobren mayor importancia las características descritas anteriormente, sigue tratándose de sistemas de tiempo real, es decir, sigue existiendo la necesidad de garantizar unos tiempos de respuesta o de reacción acotados. Y es en este punto donde los microprocesadores de propósito general presentan la mayor desventaja.

El increíble aumento de prestaciones, con un coste limitado, que presentan los procesadores actuales se debe a avances en la tecnología y especialmente a mejoras en su estructura y arquitectura. Estas mejoras se basan en estudios estadísticos de la carga que ejecuta el procesador, identificando el caso más frecuente y organizando la estructura interna del procesador para que responda de la forma más rápida posible ante esta situación. Este aumento de la velocidad en el caso medio o más común puede llevar aparejado un empeoramiento de las prestaciones o una reducción de la velocidad de ejecución en el resto de situaciones, pero que se verá ampliamente compensada por la aceleración obtenida para el caso más frecuente.

Uno de los ejemplos más claros de estas mejoras estructurales son las memorias *cache*. Los principios de localidad temporal y espacial, que en mayor o menor grado cumplen todos los programas, producen que la mayoría de los accesos a memoria que genera un programa "convencional" sean aciertos en *cache*. Estos accesos a memoria que producen acierto en *cache* pueden ser servidos en un tiempo de hasta dos ordenes de magnitud menor que si fueran servidos directamente por la memoria principal. El problema que presentan las memorias *cache* es que no todos los programas, convencionales o no, presentan los mismos parámetros de localidad, y por tanto la tasa de aciertos y fallos en los accesos a *cache* varía para cada programa. Es más, en función de las rutas que ejecute un programa, el porcentaje de aciertos y fallos en *cache* puede variar, dependiendo del grado de localidad de cada sección del código

ejecutado. Esto provoca que el incremento de velocidad proporcionado por la memoria *cache* no sea constante, no ya para dos programas distintos, sino para el mismo programa en función de los datos de entrada. De este modo, encontrar el máximo tiempo de ejecución de una tarea, o lo que es lo mismo, calcular el tiempo de ejecución en el peor caso para verificar que los plazos temporales se cumplen, se convierte en un problema complejo, ya que no es sencillo modelar y predecir el número de aciertos y fallos que se producirán al acceder a *cache*.

Y las memorias *cache* aun presentan dos problemas más. Cuando se trata de un sistema de tiempo real multitarea, es necesario considerar las posibles interacciones entre las distintas tareas al acceder a la memoria *cache*, ya que estas interacciones pueden modificar el número de aciertos que produce cada tarea. Y del mismo modo, la memoria *cache* puede interactuar con otras mejoras estructurales, como la segmentación o la predicción de saltos, técnicas de aumento de prestaciones que pueden modificar, en determinadas situaciones, el aumento de velocidad ofrecido por la *cache*.

Parte de la motivación de esta tesis proviene de los trabajos desarrollados en el marco de los proyectos CICYT TAP990443-C05-02 y CICYT DPI2000-0618, en los que se desarrollaron sistemas distribuidos tolerantes a fallos y de tiempo real para el control de robots móviles y para su integración en sistemas de automoción. En estos sistemas se observó que la utilización de memorias *cache* introducía una variabilidad extremadamente alta en los tiempos de respuesta de las tareas, y que era necesario emprender alguna acción para poder obtener una estimación del tiempo de respuesta de cada tarea en el peor de los casos. Tres líneas de actuación son posibles:

- Eliminar, o desconectar, la memoria *cache*. El principal inconveniente de esta técnica es que las prestaciones obtenidas son considerablemente bajas, por lo que es una opción no aceptable para muchos de los problemas de tiempo real actuales, aunque ha sido y sigue siendo una práctica habitual.
- Modelar el comportamiento de la *cache*, incorporando dicho modelo a las técnicas de análisis de planificabilidad utilizadas actualmente o desarrollando nuevas técnicas. De este modo, se mantienen las prestaciones ofrecidas por la memoria *cache* y es factible obtener una estimación del tiempo de ejecución y de respuesta de las tareas.
- Modificar o rediseñar la arquitectura de la memoria *cache*, manteniendo el aumento de prestaciones pero con un comportamiento más determinista y predecible, facilitando de este modo su incorporación en los algoritmos de análisis e incluso permitiendo la utilización de las técnicas de análisis existentes.

B. Objetivos

Los objetivos de esta tesis son los siguientes:

1. Identificar los problemas e inconvenientes que presenta la utilización de memorias *cache* en sistemas de tiempo real.
2. Estudiar las soluciones y propuestas existentes hasta el momento que permiten de forma útil y eficaz la utilización de memorias *cache* en sistemas de tiempo real.
3. Proponer una modificación a la arquitectura de memoria *cache* convencional, que ofrezca un total determinismo, basado en la precarga y bloqueo de contenidos en memoria *cache*, permitiendo de este modo el modelado de su comportamiento de una forma sencilla.
4. Desarrollar dos técnicas de utilización del esquema de *cache* propuesto, aprovechando su determinismo y manteniendo el aumento de prestaciones al mismo nivel que el ofrecido por las memorias *cache* convencionales.
5. Inclusión del comportamiento del nuevo esquema de *cache* en las técnicas existentes de análisis de planificabilidad y de cálculo del tiempo de ejecución, desarrollando las herramientas necesarias para la estimación de los tiempos de ejecución y de respuesta de las tareas. Verificación experimental de la consecución de determinismo.

6. Valoración de la perdida o ganancia de prestaciones, mediante la evaluación experimental y en referencia a las memorias *cache* convencionales, derivada del empleo de las dos técnicas de precarga y bloqueo de contenidos en *cache*. Comparación de ambas técnicas desde el punto de vista de las prestaciones.
7. Identificación de la características *software* y *hardware* del sistema de tiempo real que determinan el comportamiento, en cuanto a prestaciones, de las técnicas de precarga y bloqueo de *cache* propuestas. Modelado de dicho comportamiento.

C. Desarrollo de la tesis.

Con el propósito de alcanzar los objetivos descritos anteriormente, la presente tesis se organiza en los siguientes capítulos:

- El capítulo 1 describe los sistemas de tiempo real y la problemática que generan las diferentes mejoras estructurales incluidas en los procesadores actuales. En este capítulo se comentan las principales soluciones propuestas hasta el momento en relación con dicha problemática. Con especial detalle se tratan los temas relacionados con la utilización de memorias *cache* en sistemas de tiempo real.
- El capítulo 2 describe la propuesta de arquitectura de *cache* para obtener un comportamiento determinista. Esta nueva estructura se deriva de las *cache*s con bloqueo (*locking cache*s). Para la arquitectura propuesta se presentan los siguientes trabajos:
    - Descripción detallada de los requisitos, en cuanto a estructura, de la *cache* con bloqueo para obtener conjuntamente determinismo con la mínima perdida de prestaciones.
    - Descripción de la técnica llamada "uso estático" que permite utilizar la *cache* con bloqueo y modelar su comportamiento. Detalle de los algoritmos necesarios para la obtención del tiempo de ejecución y del tiempo de respuesta de las tareas, así como para la realización del análisis de planificabilidad.
    - Propuesta de un algoritmo genético para la selección de los contenidos que se precargarán y bloquearán en *cache*.
    - Verificación experimental de la obtención de determinismo, así como evaluación, de forma general, de la ganancia o pérdida de prestaciones sufrida al utilizar de forma estática la *cache* con bloqueo.
- El capítulo 3 presenta una nueva forma de utilizar la *cache* con bloqueo, llamada "uso dinámico", cuyo objetivo es aumentar las prestaciones del sistema frente al "uso estático". Para esta nueva técnica se presentan las modificaciones necesarias a la arquitectura de *cache* y al algoritmo genético, así como los métodos para la obtención de los tiempos de ejecución y de respuesta de las tareas. La utilización del mismo corpus de experimentos en este capítulo y en el anterior permite comparar los dos usos de *cache* con bloqueo tanto desde el punto de vista del determinismo alcanzado como desde la ganancia o pérdida de prestaciones.
- Los capítulos 4, 5 y 6 presentan un conjunto de análisis estadísticos que permiten identificar los factores y características *software* y *hardware* que definen el comportamiento de las prestaciones de la *cache* con bloqueo. Estos análisis permiten, además, comparar con detalle los tres esquemas de *cache* estudiados en esta tesis: *cache*s convencionales, uso estático de *cache* con bloqueo y uso dinámico de *cache* con bloqueo, permitiendo definir los escenarios y situaciones en los que la utilización de cada uno de estos esquemas es más adecuado.
- Finalmente, el capítulo 7, cierra la presente tesis con las conclusiones y las posibles líneas de investigación abiertas.

Capítulo 1

Introducción y fundamentos

El objetivo de este primer capítulo es proporcionar una descripción de las principales características y particularidades de los sistemas de tiempo real. La descripción de este tipo de sistema informático se realizará desde dos puntos de vista, la arquitectura *software* y la arquitectura *hardware*. Para cada uno de ellos se describirán las posibles alternativas y variantes que el diseñador de sistemas de tiempo real puede utilizar, y cómo la elección de una u otra alternativa influye en el análisis de planificabilidad, es decir, en la verificación de que el sistema cumple los requerimientos temporales.

Especial énfasis se realizará en las implicaciones que conlleva la utilización de memorias *cache* en sistemas de tiempo real, y cómo los análisis de planificabilidad genéricos deben ser adaptados, modificados o ampliados para considerar, de forma correcta y segura, la existencia de estas memorias en la arquitectura *hardware* del sistema. Las propuestas realizadas hasta el momento para incluir las memorias *cache* en el análisis de planificabilidad se estudiarán con detalle, identificando sus principales limitaciones, carencias e inconvenientes.

1.1 Sistemas de tiempo real

En la bibliografía existente pueden encontrase múltiples definiciones sobre qué es un sistema de tiempo real (STR). Por citar dos de ellas:

Un sistema computador de tiempo real puede ser definido como aquél que realiza sus funciones y responde a eventos externos asíncronos en un tiempo acotado (predecible o determinado) [Furht91]

En los sistemas de tiempo real, la corrección de los resultados no sólo depende de los cálculos, sino también del tiempo empleado en obtenerlo. [Stankovic88]

Resumiendo estas dos definiciones y otras muchas, a un STR se le exige no sólo que realice de forma correcta el trabajo para el que se ha diseñado, sino que lo realice dentro de un límite temporal. Dependiendo de si este límite temporal no debe ser superado nunca, o de si es tolerable que en algunos casos los resultados se obtengan vencido el tiempo, los STR se clasifican en sistemas estrictos (*hard real-time systems)* o en sistemas no estrictos (*soft real-time systems)*.

Los STR estrictos dan respuesta a problemas planteados por sistemas críticos, donde el retraso en la obtención de los resultados puede provocar severos daños materiales, la destrucción del propio sistema e incluso la perdida de vidas humanas. Este es el caso de sistemas de soporte de vida, aviónica, exploración espacial o sistemas de seguridad en entornos de trabajo peligrosos. Pero también es el caso de nuevos sistemas, más cotidianos, y de gran consumo, como los sistemas de seguridad activa en automóviles (ABS, control de tracción, distribución de la frenada) [Davis95b].

En el caso de los sistemas de tiempo real no estricto, la obtención de los resultados más allá del tiempo límite produce una degradación en la calidad del servicio, pero no pérdidas importantes de ninguna índole. Este es el caso de sistemas de reconocimiento de voz, de identificación de formas, sistemas multimedia, redes de computadores y otros [Nilsen95a]. En estos casos, es deseable que el sistema realice los cálculos y tareas dentro de los plazos temporales, pero es tolerable que esto no suceda en contadas ocasiones.

En ambos casos, el diseñador de un STR deberá tomar las medidas y realizar las acciones necesarias para encontrar respuesta a la pregunta: ¿responde mi sistema dentro de los plazos temporales establecidos? Para resolver esta incógnita, el diseñador deberá considerar y analizar

los elementos *software* y *hardware* de su sistema con el objetivo de obtener el tiempo de respuesta (tiempo que tarda el sistema en proporcionar los resultados) y compararlo con los tiempos máximos establecidos. Este análisis responde al nombre de análisis de planificabilidad.

A continuación se describen las principales características de los elementos *software* y *hardware* que pueden integrar un STR y la influencia de éstas en la realización del análisis de planificabilidad. El resto del capítulo, especialmente las secciones dedicadas al análisis de planificabilidad, se refieren a sistemas de tiempo de real estricto, aunque algunos de los conceptos presentados puedan también aplicarse a sistemas no estrictos.

1.1.1 Arquitectura software

Desde el punto de vista de la arquitectura *software*, un sistema de tiempo real está formado por programas que deben ejecutarse en el sistema, haciendo uso compartido de los diferentes recursos, como el procesador, la memoria o los periféricos. Estas ejecuciones de los programas reciben el nombre de tareas, y cada tarea está descrita por una terna de tres valores (C,D,T). El primer valor, C, es el tiempo de cómputo de la tarea, es decir, el tiempo que tardan en ejecutarse las instrucciones sobre un procesador concreto. El segundo valor, D, es el valor del plazo (*deadline*), que indica el tiempo máximo que la tarea puede tardar en obtener los resultados. Es el máximo tiempo de respuesta[1] permitido para la tarea. Aunque el tiempo de respuesta de una tarea depende del sistema informático en el que se ejecuta y de sus características *software* y *hardware*, el valor del *deadline* es una característica del sistema físico que se desea controlar o gobernar, es decir, del problema que se trata de solucionar, independientemente del sistema computador que se utilice para resolver el problema. Por último, el valor T es el periodo entre las activaciones de la tarea. Es la frecuencia con la que una tarea debe ejecutarse en el sistema, y al igual que el *deadline*, es una característica intrínseca del problema o sistema físico.

En función de este último parámetro T, las tareas de un sistema de tiempo real pueden dividirse en tres categorías:

- Tareas periódicas: la activación de la tarea se produce exactamente cada T unidades de tiempo.
- Tareas esporádicas: aunque no se conoce el instante exacto de activación de la tarea, sí se conoce el periodo mínimo, es decir, la frecuencia máxima con que se producirán las activaciones de la tarea.
- Tareas aperiódicas: en este caso no se conoce el instante de activación ni el tiempo mínimo entre sucesivas activaciones. Normalmente están asociadas a eventos externos, y su ejecución se condiciona a pruebas de aceptación en tiempo de ejecución para no poner en peligro los requisitos temporales del resto del sistema.

Las tareas que forman un sistema de tiempo real deben ejecutarse en el sistema compartiendo todos los recursos del mismo. La planificación del sistema es el algoritmo o protocolo que indicará cómo las tareas entran en ejecución, haciendo uso de los diferentes recursos disponibles. En función de la política de planificación utilizada deberán emplearse diferentes algoritmos para realizar el análisis de planificabilidad. Se dice que un sistema es planificable si para una determinada política de planificación, todas las tareas cumplen sus requisitos temporales, es decir, su tiempo de respuesta es menor o igual que el plazo o *deadline* para cada tarea.

A continuación se presenta una descripción de las políticas más frecuentemente utilizadas, tanto en aplicaciones reales como en trabajos de investigación. Para cada una de ellas se indica el

[1] En los sistemas informáticos multitarea, el tiempo de cómputo de una tarea no tiene porqué coincidir con su tiempo de respuesta, sino que habitualmente este último será mayor. El tiempo de respuesta se define como el tiempo que transcurre desde que se activa la tarea hasta que esta finaliza su ejecución, sin ninguna consideración a los eventos (ejecución de otra tarea, suspensión de la ejecución por espera de un evento) que sucedan entre el inicio y la conclusión de la ejecución.

algoritmo o algoritmos de análisis de planificabilidad que pueden utilizarse. Es importante remarcar que los algoritmos descritos no tienen en consideración características avanzadas del *hardware*, como la existencia de memorias *cache*, que se trata en el último punto de este capítulo.

A. Planificación estática

La política de planificación estática se basa en la construcción de una tabla de tiempos, con tamaño mínimo igual al mínimo común múltiplo (MCM o hiperperiodo) de los periodos de todas las tareas del sistema. En esta tabla se especifica el instante exacto en que cada tarea iniciará su ejecución y hará uso de los recursos del sistema. Durante la ejecución del sistema, el planificador sólo debe seguir las indicaciones de la tabla para lanzar a ejecución la tarea indicada en cada instante de tiempo. Ninguna tarea puede entrar en ejecución fuera de los instantes de tiempo que tenga asignados en la tabla. Puesto que esta ejecución se repite para cada MCM, esta planificación recibe el nombre de cíclica.

La principal ventaja de esta política de planificación es que todo se encuentra predeterminado antes de lanzar el sistema a ejecución, por lo que si no hay ningún tipo de sobrecarga no contemplada durante la etapa de diseño, el sistema es planificable, sin necesidad de realizar o ejecutar ningún análisis de planificabilidad.

Sin embargo, la planificación estática presenta básicamente dos inconvenientes. El primero de ellos es la complejidad de su diseño, y más concretamente, la complejidad de la construcción de la tabla de ejecución, que requiere un laborioso trabajo de ajuste. En [Burns95a] se describe el diseño de sistemas cíclicos que cumplan los requisitos temporales. El segundo de ellos está relacionado con el mantenimiento y modificación del sistema. Cualquier variación en alguna de las tareas puede invalidar completamente la tabla de ejecuciones, obligando a construirla completamente de nuevo.

La falta de flexibilidad y el alto coste de diseño y mantenimiento hace inviable la utilización de esta política de planificación en los sistemas de tiempo real actuales, enmarcados cada vez con mayor frecuencia en entornos dinámicos, limitándose su uso a sistemas muy sencillos, especialmente con un número muy reducido de tareas.

B. Planificación dinámica con prioridades fijas en sistemas con expulsión

Estos planificadores reciben el nombre de dinámicos porque el orden en que se ejecutan las tareas no se conoce a priori, sino que se establece durante la ejecución del sistema en función de los periodos y la prioridad de las tareas. El adjetivo *prioridades fijas* es debido a que la prioridad de las tares, es decir, la importancia o valor utilizado para decidir qué tarea entra en ejecución en cada instante de tiempo se asigna durante el diseño del sistema, y permanece invariable durante todo el funcionamiento. Sólo en el caso de que se utilice un algoritmo de herencia de prioridades para resolver el acceso a determinados recursos [Sha90b] [Sha87] [Baker90] [Rajkumar88] [Rajkumar89], una tarea puede, momentáneamente, modificar su prioridad.

El tercer adjetivo, *expulsivo*, indica que la tarea en ejecución puede ser desplazada del procesador para lanzar a ejecución otra tarea, sin que la primera haya finalizado su ejecución, debiendo esperar la liberación del procesador para continuar, que no reiniciar, sus cálculos. Esta expulsión o adelantamiento (*premption*) se realiza manteniendo la corrección computacional, pero, por supuesto, aumentando su tiempo de respuesta respecto al que obtendría si fuera la única tarea en el sistema. De este modo, siempre se ejecuta la tarea disponible con mayor prioridad. Su utilización se remonta a las primeras misiones *APOLLO* [Ramamritham94], aunque la primera publicación en que se describe formalmente este tipo de planificación y su correspondiente análisis de planificabilidad es [Liu73].

La principal ventaja de esta planificación es su sencillez y flexibilidad, ya que el diseñador no necesita explicitar con todo detalle el comportamiento temporal del sistema, sino solamente

asignar las prioridades a las tareas, con arreglo a algún criterio. Además, si se modifica alguna tarea, no es necesario rediseñar todo el sistema. Otra ventaja añadida es la estabilidad, entendida como que, en el caso de sobrecarga, las tareas con menor prioridad serán las que perderán los plazos de entrega, mientras que las más prioritarias mantendrán su corrección temporal. Sin embargo, la prioridad de las tareas no tiene porqué coincidir con la importancia de la misma, por lo que tareas importantes pueden perder su plazo si se les asigna una baja prioridad. Por importancia de una tarea se entiende cuan crítica es una tarea para la integridad física del sistema o de los usuarios.

Aunque la asignación de las prioridades a las tareas puede realizarse en función de varios factores, como por ejemplo la importancia de la tarea o el tiempo necesario para su ejecución, existen dos políticas de planificación basadas en los periodos y los plazos de las tareas que consiguen una planificación óptima. Se dice que una política de planificación es óptima si es capaz de planificar -los tiempos de respuesta son menores que los plazos- cualquier conjunto de tareas que es planificable por alguna otra política.

- Política de planificación *Rate Monotonic* (RM). En esta política las prioridades se asignan en orden inverso a los periodos de las tareas. De este modo, la tarea con menor periodo será la de mayor prioridad. Esta planificación es óptima si los plazos de entrega coinciden con los periodos. El principal inconveniente es que obliga a que las tareas tengan el mismo plazo de entrega y periodo. Además de la poca probabilidad de que un sistema real cumpla esta característica, esta particularidad impide la existencia de tareas aperiódicas y esporádicas, ya que su periodo es desconocido
- Política de planificación *Deadline* Monotonic (DM). En esta política las prioridades se asignan en orden inverso a los plazos. De este modo, la tarea con el plazo de entrega más corto será la más prioritaria. Esta planificación es óptima si los plazos son menores o iguales que los periodos. La planificación DM, al permitir que el plazo y el periodo sean distintos, no presenta las limitaciones de RM, y se muestra mucho más útil, especialmente frente a los planificadores cíclicos [Locke92]

En cuanto a la corrección temporal de un sistema basado en prioridades fijas y expulsión, la verificación de que todas las tareas terminarán su ejecución, bajo cualquier condición o circunstancia, antes del plazo, es más complejo que en los planificadores cíclicos, ya que el comportamiento temporal de las tareas se desconoce con exactitud. Así, mientras en los planificadores cíclicos la corrección temporal se garantizaba durante la etapa de diseño del sistema, en los planificadores dinámicos es necesario verificar esta corrección temporal una vez finalizado el diseño del sistema.

Si la política de planificación es RM, las tareas se ejecutan en un único procesador, son todas periódicas e independientes entre sí, y conociendo el tiempo de ejecución de cada tarea τ_i, que es menor que el plazo y el periodo ($C_i < D_i = T_i$), el sistema será planificable si se cumple la siguiente desigualdad:

$$\sum_{i=1}^{N} U_i \leq N(2^{1/N} - 1)$$

Donde U_i es la utilización de la tarea τ_i, y N es el número de tareas en el sistema. Este análisis recibe el nombre de RMA (Rate Monotonic Analysis). La parte derecha de la desigualdad tiende de forma monótona decreciente a ln(2), cuyo valor es aproximadamente 0,69. Por lo tanto, un sistema con política RM es planificable si su utilización total está por debajo de 0,69, sin necesidad de evaluar la desigualdad, lo que hace que el análisis de planificabilidad sea sencillo de llevar a cabo. El principal inconveniente de este análisis es su excesivo conservadurismo, ya que se puede dar el caso de que un sistema que no cumpla la desigualdad siga siendo planificable. Así pues, este análisis es suficiente, pero no necesario.

Bajo las mismas condiciones, pero utilizando la planificación DM, en [Audsley91a] se presentan dos análisis para determinar si el sistema es planificable. El primero de los análisis es necesario y suficiente. Si dicho análisis concluye que el sistema no es planificable, el sistema no será planificable por ningún otro análisis. El principal inconveniente es su elevado coste computacional. El segundo análisis sólo es suficiente, es decir, si el análisis concluye que el sistema es planificable, efectivamente éste lo será, pero al igual que el análisis para RM, es posible que el análisis de planificabilidad indique que un sistema no es planificable cuando este sí que lo es. La principal ventaja de este segundo análisis es un coste computacional mucho menor.

Un inconveniente común a las políticas de planificación RM y DM es la asignación de prioridades basada en los periodos y los plazos, cuando en muchos sistemas reales, la prioridad de cada tarea viene definida por su funcionalidad y por su relación con el problema que el sistema trata de resolver o controlar. Por ello, en [Harter84] y [Joseph86] se presenta un análisis de planificabilidad que es independiente de la asignación de prioridades a las tareas. Este análisis obtiene el tiempo de respuesta, en el peor caso, de todas las tareas del sistema. Una vez obtenido el tiempo de respuesta de las tareas, la planificabilidad o no del sistema se comprueba comparando dichos tiempos de respuesta con los plazos de las tareas. Este análisis es exacto - necesario y suficiente- y recibe el nombre de RTA (Response Time Analysis) de las siglas en ingles de Análisis del Tiempo de Respuesta. La expresión para la obtención del tiempo de respuesta de una tarea se presenta en la siguiente ecuación:

$$w_i^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^n}{T_j} \right\rceil x C_j$$

donde C_x es el tiempo de cómputo de la tarea τ_x, hp(i) es el conjunto de tareas más prioritarias que la tarea τ_i y T_j es el periodo de la tarea τ_j.

La ecuación se resuelve de forma iterativa, considerando que en la primera iteración el tiempo de respuesta de la tarea bajo análisis τ_i coincide con su tiempo de ejecución C_i. En cada iteración, la ventana o tiempo de respuesta de la tarea bajo análisis se ve incrementada por la ejecución de tareas más prioritarias, finalizando el análisis cuando $w_i^{n+1} = w_i^n$ que corresponde con el tiempo de respuesta R_i de la tarea τ_i. Si el valor calculado de R_i es mayor que el plazo D_i para cualquier tarea τ_i del sistema, este no será planificable.

La gran flexibilidad que ofrece RTA, cuya única exigencia es que la prioridad de las tareas se mantenga invariable durante el funcionamiento del sistema, ha permitido el desarrollo de mejoras y ampliaciones de este análisis para incluir situaciones o características del sistema más particulares, como la utilización de protocolos para la sincronización entre tareas y el compartimiento de recursos, la existencia de desplazamientos en el lanzamiento de las tareas, plazos de entrega arbitrarios e incluso superiores al periodo o la existencia de tareas aperiódicas y esporádicas. [Tindell92c][Tindell94a][Audsley93a][Audsley93b][Tindell92a][Tindell94b]

C. Planificación dinámica con prioridades dinámicas en sistemas con expulsión

En esta política de planificación, al igual que en la de prioridades fijas, el orden en que se ejecutan las tareas no es conocido a priori, sino que se establece en tiempo de ejecución en función de los periodos de las tareas y de sus prioridades. Asimismo, una tarea puede ser expulsada del procesador si existe otra tarea con mayor prioridad que espera entrar en ejecución. La diferencia entre esta política y la descrita anteriormente se encuentra en la asignación de prioridades. En este caso, no es el diseñador, atendiendo a razones funcionales o de diseño quien asigna la prioridad a cada tarea, sino que esta prioridad se asigna dinámicamente durante el tiempo de ejecución. De este modo, las tareas no tienen una prioridad fija y constante, sino que en función de las condiciones del sistema se le asignará una prioridad. Existen dos políticas básicas de asignación dinámica de prioridades, que se describen a continuación.

- *Earlest Deadline First* (EDF): En esta política de planificación, se asignan las prioridades en orden inverso al tiempo que falta para que venza el plazo. De esta forma, la tarea que tiene el plazo más próximo recibe la máxima prioridad. Las prioridades se reevaluan cada vez que una nueva tarea se activa.
- *Least Laxity First* (LLF): Se define la laxitud (*laxity*) de una tarea como la diferencia entre el tiempo que queda para que venza el plazo y el que falta para que la tarea finalice su ejecución. En esta política de planificación, la prioridad de las tareas se asigna en orden inverso a la laxitud de cada tarea. De esta forma, la tarea con menor laxitud, y por tanto la que tiene más comprometido finalizar antes del plazo, recibe la máxima prioridad. Las prioridades deben reevaluarse de forma periódica.

Ambas políticas son óptimas cuando las tareas son independientes entre sí. La planificación EDF es privilegiada en el sentido que produce el mínimo número de cambios de contexto, frente a la planificación LLF, que puede producir un número indeterminado de cambios de contexto si dos tareas tienen valores similares de laxitud. Por este motivo, ya que el cambio de contexto tiene un coste temporal no despreciable en la práctica, LLF tiene sólo valor teórico.

Un sistema con política EDF es planificable si la utilización del sistema no supera el 100%. Sin embargo, este análisis obliga a considerar todo el hiperperiodo, lo que convierte el análisis en un problema computacionalmente intratable, y además sólo es aplicable cuando los periodos de las tareas coinciden con sus plazos de entrega [Basumallick94] [Liu73]. Para simplificar el análisis y permitir cualquier relación entre los periodos y los plazos de entrega de las tareas, [Ripoll96] presenta un algoritmo basado en el Instante Crítico Inicial (ICI), que permite evaluar la máxima utilización del sistema sin necesidad de considerar todo el hiperperiodo. El algoritmo se basa en las siguientes ecuaciones:

$$G(t) = \sum_{i=1}^{n} C_i \left\lceil \frac{t}{P_i} \right\rceil$$

$$H(t) = \sum_{i=1}^{n} C_i \left\lfloor \frac{t + P_i - D_i}{P_i} \right\rfloor$$

donde C_i, P_i y D_i son, respectivamente, el tiempo de cómputo, el periodo y el plazo de entrega de la tarea τ_i, y n es el número de tareas en el sistema.

El Instante Crítico Inicial R se calcula resolviendo recursivamente la ecuación $R_{i+1}=G(R_i)$ hasta que $R_{i+1}= R_i$, y con $R_0 = 0$. Una vez obtenido el valor final de R, el sistema será planificable si y sólo si se cumple la siguiente ecuación:

$$H(t) < t : \forall t, 1 \le t \le R$$

1.1.2 Arquitectura hardware

La arquitectura *hardware* de un sistema informático de tiempo real comprende todos los circuitos y dispositivos necesarios para ejecutar las tareas que forman el sistema. En esta arquitectura se incluye las características, funcionalidades y particularidades de dichos circuitos y dispositivos, que determinarán no sólo las capacidades y prestaciones del sistema de tiempo real, sino también su comportamiento temporal.

En el apartado de arquitectura *software* se ha realizado una descripción de los análisis de planificabilidad necesarios o posibles para determinar la corrección temporal de un sistema de tiempo real. En estos análisis, los parámetros utilizados incluían la utilización del procesador, el tiempo de ejecución de una tarea, o su tiempo de respuesta. Aunque es evidente que estos parámetros dependen de la arquitectura *hardware*, aparentemente existe una independencia entre el análisis de planificabilidad y la arquitectura *hardware*, manteniendo una débil conexión

a través de los parámetros previamente nombrados. Sin embargo, esta independencia es sólo aparente, ya que existe una fuerte relación entre las características *hardware* del sistema y el análisis de planificabilidad que debe realizarse para garantizar la corrección temporal del sistema.

En este apartado se describirán las principales características de las arquitecturas *hardware* utilizadas en los sistemas informáticos actuales, y cómo estas características influyen y modifican los análisis de planificabilidad presentados anteriormente. Especial atención recibe el tratamiento de la memoria *cache*, objeto de esta tesis, dedicando de forma exclusiva el apartado siguiente a la descripción de las propuestas existentes para realizar el análisis de planificabilidad de un sistema informático de tiempo real estricto en el que se utilizan memorias *cache*.

La arquitectura *hardware* de un sistema informático puede dividirse en tres bloques [Hamacher87]: Unidad Central de Proceso, Memoria principal y Entrada/Salida. A continuación se describen cada uno de estos bloques, las mejoras incluidas en los sistemas actuales y su implicación en el análisis de planificabilidad.

A. *Unidad Central de Proceso.*

La unidad central de proceso (CPU) es la encargada de ejecutar las instrucciones que conforman las tareas. En una primera y simple aproximación, se puede considerar que el tiempo de ejecución de una instrucción es constante y conocido para todas las veces que es ejecutada. En este caso, para obtener el tiempo de ejecución de una tarea sólo es necesario sumar el tiempo de ejecución de todas las instrucciones que son ejecutadas por la tarea. Sin embargo, las tareas suelen estar formadas por bucles y bifurcaciones, permitiendo la ejecución de diferentes rutas en función de los datos sobre los que se trabaje.

Esta variación en las rutas ejecutadas por la tarea hace que el tiempo de cómputo de la tarea no sea constante. La mayoría de los análisis de planificabilidad hacen uso del tiempo de cómputo de las tareas, bien directamente como RTA o bien a partir de la utilización del sistema como el análisis para RM, por lo que es necesario especificar cual de todos los valores posibles se utilizará para determinar la planificabilidad del sistema. Puesto que el objetivo es determinar si las tareas finalizan su ejecución antes del plazo de entrega bajo cualquier condición o situación, se define el tiempo de cómputo o tiempo de ejecución en el peor caso (WCET *Worst Case Execution Time*) como el mayor tiempo de ejecución posible de una tarea. La utilización del WCET en el análisis de planificabilidad ofrece garantías sobre la corrección temporal del sistema, ya que en ningún caso, una tarea utilizará más tiempo de procesador para su ejecución que el indicado por el WCET.

En el caso de tareas con bucles y bifurcaciones, la obtención del WCET de la tarea se realiza evaluando todas las posibles rutas que la tarea es capaz de ejecutar, escogiendo aquella que presente un mayor coste temporal, es decir, aquélla cuya suma del tiempo de ejecución de las instrucciones sea mayor. Esta evaluación recibe el nombre de Tiempo de Ejecución de la Peor Ruta (*Worst Path Execution Time* WPET), y se puede realizar a partir de un lenguaje de alto nivel [Shaw89] [Park91] [Park93] o desde código máquina. Varios trabajos han sido desarrollados para mejorar este análisis, considerando la existencia de rutas imposibles [Kountouris96] o la estimación del número máximo de iteraciones de un bucle [Healy00], eliminando situaciones que debido a la funcionalidad de la tarea o las características de los datos de entrada no pueden suceder, y por tanto proporcionando valores del WCET más bajos y ajustados a la realidad.

Con el objetivo de mejorar las prestaciones de los procesadores, e independientemente de las mejoras puramente tecnológicas, aunque apoyándose en ellas, se han realizado mejoras en la estructura y organización de la CPU que permiten aumentar el paralelismo en la ejecución de instrucciones. Este paralelismo se consigue replicando unidades funcionales del procesador -procesadores escalares- y dividiendo la ejecución de las instrucciones en fases y ejecutando simultáneamente diferentes fases de diferentes instrucciones -procesadores segmentados-.

Mediante la segmentación se pretende mejorar la productividad del procesador ejecutando simultáneamente varias instrucciones. La ejecución de una instrucción puede descomponerse en cinco fases:

1. Búsqueda de la Instrucción (BI)
2. Decodificación (DE)
3. Búsqueda de Operandos (BO)
4. Ejecución (EJ)
5. Almacenamiento del resultado (ER)

Un procesador no segmentado ni escalar ejecuta una instrucción detrás de otra, de manera secuencial, esperando la finalización de una instrucción para iniciar la ejecución de la siguiente. En la

se muestra como, en cada ciclo de reloj, se ejecuta una etapa de la instrucción, continuando con la primera etapa de la segunda instrucción una vez finalizada la última etapa de la primera instrucción.

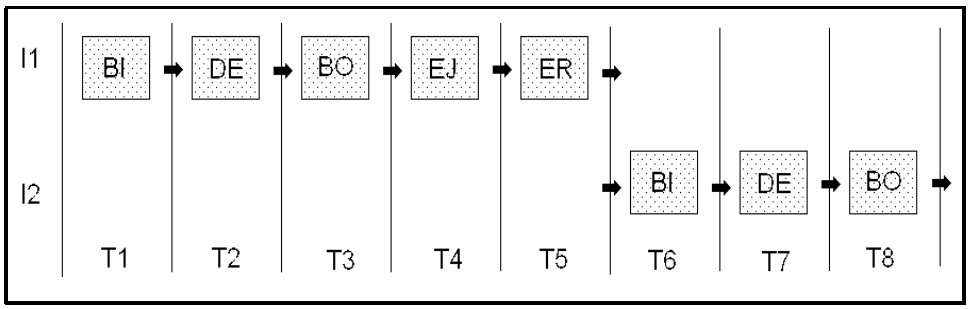


Figura 1. Ejemplo de ejecución en un procesador no segmentado.

El objetivo de la segmentación es solapar la ejecución de varias instrucciones en una única CPU. Las instrucciones se ejecutan en paralelo, cada una de ellas realizando una de sus fases en unos circuitos concretos del procesador. Cada uno de estos circuitos es capaz de ejecutar de forma independiente y autónoma una de las fases en las que se divide la ejecución de la instrucción. La Figura 2 muestra como, en cada ciclo de reloj, el procesador segmentado ejecuta simultáneamente diferentes etapas de diferentes instrucciones. Comparando con la

se puede apreciar que el procesador segmentado finaliza cuatro instrucciones en los mismos ciclos en que el procesador no segmentado no llega a finalizar ni siquiera dos.

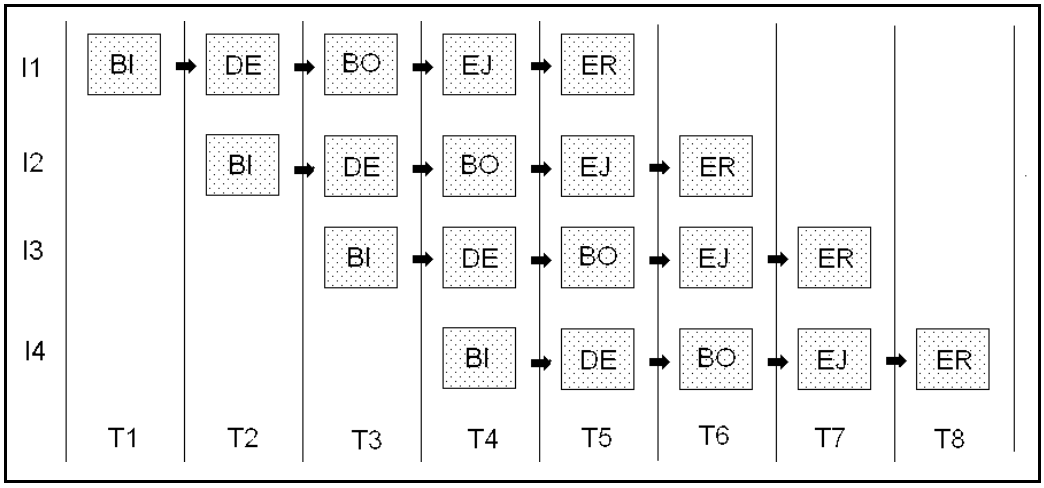


Figura 2. Ejemplo de ejecución en un procesador segmentado.

En los procesadores escalares el solapamiento en la ejecución de instrucciones se consigue mediante la replica de unidades funcionales, como unidades aritmético-lógicas u operadores de coma flotante. De esta forma, el procesador puede realizar simultáneamente la misma operación sobre diferentes instrucciones, ejecutando varias instrucciones por ciclo. En 1989, el Intel 80960CA [Intel89] se anunciaba como el primer procesador capaz de ejecutar dos instrucciones por ciclo. El procesador HP PA-RISC 8000 dispone de nueve unidades de ejecución, alcanzando las cuatro instrucciones ejecutadas por ciclo. Otros procesadores de gran consumo, como la familia Pentium disponen también de unidades replicadas y segmentadas. Actualmente la mayoría de procesadores comerciales combinan unidades escalares y segmentadas, intentado conseguir el mayor aumento posible de productividad.

Sin embargo, estas dos técnicas pueden presentar problemas durante su funcionamiento normal que impidan alcanzar las máximas prestaciones teóricas. Estos problemas, habitualmente llamados riesgos en las unidades segmentadas, obligan a detener la ejecución de instrucciones durante uno o varios ciclos, por lo que la productividad (número de instrucciones ejecutadas por ciclo) desciende bruscamente. Básicamente, existen tres tipos de riesgos [Patterson85] que afectan tanto a las unidades segmentadas como escalares:

1. Riesgos estructurales. En este caso, la propia construcción del procesador impide la ejecución simultanea de dos instrucciones, al existir un conflicto en los recursos *hardware* (circuitos y operadores) necesarios para llevar a cabo las operaciones requeridas.
2. Riesgos por dependencia de datos. Este riesgo surge cuando una instrucción necesita los resultados obtenidos por otra instrucción previa, que sin embargo no ha finalizado su ejecución y por tanto no ha calculado dichos resultados.
3. Riesgos de control de flujo. Este riesgo es debido a la existencia de instrucciones de salto condicional, en las que no se conoce la dirección de la siguiente instrucción hasta que finaliza la ejecución del salto.

Estos riesgos no sólo afectan a las prestaciones del procesador, sino que aumentan la complejidad de los análisis de planificabilidad, ya que el tiempo de ejecución de una instrucción no es constante, sino que depende de las instrucciones que le precedan, es decir, del estado del procesador cuando se ejecuta dicha instrucción. La existencia de rutas alternativas en el código que ejecuta el procesador complica aún más el análisis, ya que el tiempo de ejecución de una de estas rutas no depende exclusivamente de las instrucciones que lo forman, sino de la ruta que ha sido ejecutada anteriormente, y al existir rutas alternativas y bucles, todas las posibles combinaciones entre ellas deben ser consideradas para identificar cuál será la que presente un mayor tiempo de ejecución. En [Zhang93] [Rhee94a] se presentan análisis para calcular el WCET de una tarea ejecutada en un procesador segmentado. En ambos casos se analiza la secuencia en que las instrucciones llegan al procesador, identificando, de entre todos las posibles secuencias, la que proporciona el peor tiempo de ejecución.

Con el objetivo de mejorar las prestaciones obtenidas por la segmentación, evitando las detenciones ocasionadas por los riesgos, los procesadores modernos incluyen en su estructura circuitería que permite reordenar dinámicamente las instrucciones y predecir los saltos [Bursky96c]. Con la reordenación de código se evitan los riesgos por dependencia de datos, ya que la instrucción que provoca la detención es adelantada por otras instrucciones que pueden ejecutarse sin demora. Cuando los datos necesarios estén disponibles, la instrucción retardada será ejecutada, sin ninguna detención, en el procesador. La predicción de saltos permite eliminar los riesgos por control de flujo, ya que el procesador conoce, antes de ejecutar la instrucción de salto, cuál es la siguiente instrucción que deberá ejecutarse, por lo que ésta puede entrar a ejecución sin demora. Sin embargo, ninguna de estas técnicas garantiza que se eliminarán todos los riesgos, y por tanto todas las detenciones. Ni siquiera se garantiza la eliminación de todas las detenciones, y por tanto la variabilidad en el tiempo de ejecución de una instrucción, cuando estas técnicas *hardware* se combinan con el uso de compiladores optimizados [Jouppi89] [Bacon94] [Jourdan95a] [Lilja94] [Rymarczyk82]. En sistemas de propósito general la utilización de todos estos recursos *hardware* y *software* son plenamente aprovechados, ya que

mejoran la productividad media del computador. Sin embargo, en sistemas de tiempo real estricto, donde se necesita una garantía completa del peor tiempo de respuesta, estas técnicas requieren métodos de análisis sofisticados para poder considerar la mejora de prestaciones que ofrecen con las garantías temporales exigidas. En [Colin00] [Caironi96] se presentan diferentes algoritmos y técnicas para obtener el WCET de una tarea que se ejecuta en un procesador con predicción de saltos o reordenación de código.

La utilización de las técnicas descritas anteriormente para aumentar las prestaciones del procesador afecta no sólo al cálculo del tiempo de ejecución en el peor caso, sino también al cálculo del tiempo de respuesta cuando se consideran las interferencias entre las tareas, ya que cuando se produce una expulsión o cambio de contexto por cualquier motivo, la unidad segmentada debe ser vaciada y recargada con las nuevas instrucciones [Kogge81]. El tiempo necesario para vaciar el procesador suele ser constante y definido por el fabricante, pero es necesario realizar una estimación del máximo número de veces que se producirá este evento, para incorporarlo al tiempo de respuesta de las tareas.

Además, la interacción entre las mejoras estructurales introducidas en los procesadores y otros elementos del computador, como la memoria cache, complican aún más, si cabe, la estimación de los tiempos de ejecución y de respuesta de las tareas [Healy96] [Healy99] [Lundqvist99].

B. Jerarquía de memoria.

Cualquier instrucción ejecutada por el procesador debe ser, en primer lugar, buscada en memoria principal, al igual que sucede con una parte importante de los datos. Debe haber, por tanto, un equilibrio entre las velocidades de operación de ambos dispositivos, ya que el dispositivo más lento será el que determine la máxima velocidad de funcionamiento del sistema. De nada sirve tener una memoria que proporcione una instrucción en un ciclo de reloj, si el procesador realiza una petición cada 10 ciclos, de igual modo que tiene poca utilidad un procesador que ejecute una instrucción por ciclo si sólo puede obtener las instrucciones cada 10 ciclos.

Sin embargo esta es la situación a la que ha llegado el desarrollo tecnológico [Steininger91] [Hennessy90] [Krick91] [Boland94]. Los avances en la tecnología han permitido obtener procesadores capaces de ejecutar instrucciones mucho más rápido de lo que las memorias pueden proporcionarlas, convirtiendo la fase de búsqueda de la instrucción en el cuello de botella del sistema. Para eliminar, o al menos reducir esta diferencia de velocidad se introducen en la jerarquía de memoria las memorias *cache* [Smith82]. Estas memorias se basan en el principio de localidad temporal y espacial [Ferrari76]. Según este principio, la probabilidad de volver a acceder a una dirección de memoria en un corto espacio de tiempo es muy elevada, y es inversamente proporcional al tiempo que hace que se accedió a esa dirección -localidad temporal-. Asimismo, la posibilidad de acceder a una dirección próxima a la última referencia realizada es muy elevada, decreciendo esta probabilidad cuanto mayor es la distancia entre las dos referencias -localidad espacial-.

La memoria *cache* presenta un tiempo de acceso muy bajo, y por tanto mejora el tráfico entre el procesador y memoria, al estar implementada con tecnologías mucho más rápidas que las habitualmente utilizadas para construir la memoria principal. Actualmente, la memoria *cache* se incorpora en la misma oblea de silicio que el procesador, obteniéndose un tiempo de transferencia igual o menor que el tiempo que necesita el procesador para ejecutar una instrucción, frente al tiempo de acceso de la memoria principal, que puede ser más de 10 veces mayor que el de la *cache* [Stone93]. Sin embargo, la capacidad de que se puede disponer en una memoria *cache* está limitada y suele ser más de 200 veces menor que la de la memoria principal, tanto por problemas de espacio en el *chip*, como por el elevado coste de la tecnología utilizada, por lo que la sustitución completa de la memoria principal por memoria *cache* es inviable.

El modo de obtener un aumento significativo en la velocidad de acceso a las instrucciones y datos sin casi elevar el coste del computador [Goodman83] es introducir la memoria *cache* entre

el procesador y la memoria principal, tal como muestra la Figura 3. En la memoria *cache* se almacena una copia de parte de los contenidos de la memoria principal, intentando que estos contenidos sean los que el procesador referenciará con mayor frecuencia o en un futuro próximo, atendiendo a los principios de localidad temporal y espacial. El éxito de la memoria *cache* depende del porcentaje de aciertos que se produzcan, es decir, de la tasa con la que el procesador encuentre en memoria *cache* la dirección que ha solicitado. Cuando el acceso a memoria *cache* se completa con éxito, es decir la dirección solicitada se encuentra almacenada en memoria *cache*, se produce un acierto (*hit*) y el procesador continúa la ejecución de la instrucción sin demora. Cuando el acceso no se puede realizar porque la dirección solicitada no se en encuentra en *cache*, se produce un fallo (*miss*), y la *cache* debe copiar desde memoria la instrucción o dato solicitado. La ejecución de la instrucción que ha generado el fallo se detiene hasta que la transferencia desde memoria principal a memoria *cache* finaliza.

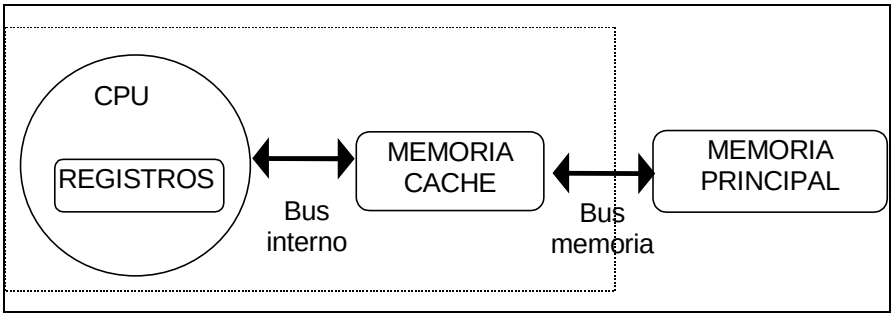


Figura 3. Representación simbólica de la jerarquía de memoria de un computador.

La memoria *cache* se organiza (Figura 4) en líneas, cada una de ellas conteniendo un bloque de memoria principal formado por una o más palabras, habitualmente 4 u 8 palabras. Para poder decidir si la dirección solicitada se encuentra en *cache*, cada línea tiene asociada una etiqueta (*tag*) con la dirección del bloque contenido en esa línea, junto con un conjunto de bits (*flags*) que indican la validez o no de los datos almacenados en dicha línea de *cache*. Para mejorar las prestaciones del sistema, la transferencia entre memoria *cache* y memoria principal se realiza por bloques de varias palabras, utilizando buses de mayor amplitud o realizando accesos en ráfaga, mientras que la transferencia entre memoria *cache* y procesador se mantiene en una palabra.

Número de línea
o dirección

0	Etiqueta	Flags	Datos (Instrucciones o datos)
1	Etiqueta	Flags	Datos (Instrucciones o datos)
2	Etiqueta	Flags	Datos (Instrucciones o datos)
3	Etiqueta	Flags	Datos (Instrucciones o datos)
	.....		
2^n	Etiqueta	Flags	Datos (Instrucciones o datos)

Figura 4. Organización de una memoria *cache* genérica.

El índice de aciertos que se produce al acceder a memoria *cache* depende del nivel de localidad de las tareas que se ejecutan en el procesador. Intuitivamente se observa que este será alto para las instrucciones, debido a la ejecución secuencial de las instrucciones y a la abundancia de

bucles. También los datos presentan niveles altos de localidad, ya que los compiladores suelen organizar las variables locales y parámetros de las funciones en la pila, utilizando posiciones contiguas de memoria, y en el caso de vectores y listas es habitual el acceso secuencial a los componentes de los mismos. La tasa de aciertos depende tanto de las características de la memoria *cache* como de la carga que se ejecuta en el procesador, pero diferentes estudios [Hennesy96][Flynn87] [Hill84] la sitúan entre el 95% y el 99%.

A continuación se describen las principales características de las memorias *cache*:

- Tamaño de *cache* y de línea: El tamaño de *cache* oscila en función de las características y potencia del procesador o computador, variando entre los pocos *Kilobytes* y el *Megabyte*. En cuanto al tamaño de línea, este se mide en palabras. En los procesadores actuales el tamaño de línea oscila entre las 4 y 8 palabras.

- Función de correspondencia: Puesto que no todo el contenido de la memoria principal puede alojarse en la memoria *cache* simultáneamente, es necesario diseñar un algoritmo de asignación de línea para los bloques de memoria principal. Básicamente existen dos algoritmos o funciones de correspondencia: directa y asociativa. En la correspondencia directa, cada bloque de memoria principal tiene asociada una sola línea de *cache* donde puede copiarse, aunque esa línea es compartida por varios bloques de memoria. En la correspondencia asociativa, un bloque de memoria puede copiarse en dos o más líneas de *cache*. Esta asociatividad puede ser completa (*full associative*), en cuyo caso el bloque puede ubicarse en cualquier línea, o puede disponer de una asociatividad de *n* conjuntos o vías (*n-way associative*) en cuyo caso, un bloque de memoria principal puede ubicarse únicamente en *n* líneas de *cache*. La principal ventaja de las memorias *cache* con función de correspondencia directa es su sencillez, tanto en cuanto a diseño, modelado y funcionamiento, lo que redunda en una mayor velocidad de acceso en caso de acierto, frente a las memorias *cache* con función de correspondencia asociativa, en las que es más costoso, tanto en *hardware* como en tiempo, decidir si la dirección referenciada se encuentra o no en *cache* [Hill88]. En los procesadores y computadores actuales la mayoría de memorias *cache* utilizan funciones de correspondencia directa o asociativa de hasta 4 vías, reservándose mayores niveles de asociatividad para *cache*s con tamaño de pocos centenares de bytes.

- Función de reemplazo: Cada vez que se produce un fallo, el bloque de memoria principal al que pertenece la referencia que ha producido el fallo debe ser copiado en memoria *cache*. En las *cache*s asociativas por conjuntos, si todas las líneas donde puede ubicarse el bloque se encuentran ocupadas, es necesario seleccionar una línea y vaciarla, reemplazando su contenido. El algoritmo óptimo seleccionaría aquella línea cuyo contenido no volverá a ser referenciado por el procesador. La imposibilidad de llevar a la práctica este algoritmo hace que la alternativa más habitual sea el algoritmo llamado *menos recientemente utilizada* (*Least Recently Used* LRU), donde la línea seleccionada para ser reescrita es aquella que hace más tiempo que fue accedida. En las *cache*s completamente asociativas, el algoritmo de reemplazo debe considerar no sólo las líneas de un conjunto, sino todas las de la *cache*, lo que implica una complejidad muy elevada que suele impedir su implementación. En el caso de la correspondencia directa, puesto que un bloque de memoria principal sólo puede ubicarse en una línea concreta, la propia correspondencia lleva implícita la función de reemplazo.

- Política de coherencia: Cuando los contenidos de una línea de *cache* son datos, éstos pueden ser modificados, por lo que en el momento de reemplazar dichos contenidos es necesario garantizar que las modificaciones se actualizan en memoria principal. Esta actualización puede hacerse de forma instantánea, modificando simultáneamente la memoria *cache* y la memoria principal (*write-trough*), o en el momento en que los contenidos de la línea son reemplazados y han sido modificados (*write-back*). En el primer caso, se mantiene la coherencia entre memoria principal y memoria *cache* de forma continua, pero se penalizan en tiempo los accesos de escritura a *cache*. En el segundo caso, sólo se accede a memoria principal si es necesario, mejorando el tiempo de acceso, pero es necesario aumentar la

complejidad de la *cache* para marcar e identificar aquellos contenidos que deben ser actualizados en memoria principal cuando se produzca el reemplazo de sus contenidos.

- Tipo de contenidos: Atendiendo al tipo de información que se almacena en memoria *cache*, estas se pueden clasificar en tres tipos: *cache* de instrucciones, en la que sólo se copia código ejecutable; *cache* de datos, en la que sólo se copian datos; y *cache* mixta, en la que coexisten instrucciones y datos.

Todas estas características, junto con las de la carga que se ejecuta, determinan las prestaciones de la memoria *cache*, es decir, la tasa de aciertos y fallos, existiendo tres razones por las que se produce el fallo en *cache*:

- Fallo obligatorio: inicialmente la memoria *cache* se encuentra vacía, por lo que cada acceso producirá un fallo. En régimen permanente, esta fuente de fallos no debe ser tenida en cuenta.
- Fallo por capacidad: cuando la tarea que se está ejecutando es mayor que la *cache*, se reemplazan bloques que luego serán nuevamente referenciados, produciendo un fallo.
- Fallo por conflicto: este fallo se produce en las *cache*s con correspondencia directa o asociativa por conjuntos, cuando existen más bloques que vías compitiendo por un determinado conjunto de líneas, lo que obliga a reemplazar bloques que posteriormente serán referenciados de nuevo.

El aumento de prestaciones proporcionado por la memoria *cache* se debe a su comportamiento dinámico y adaptativo, ya que los contenidos de la misma evolucionan y se modifican según evoluciona la ejecución del programa, tanto en la parte de código como en los datos. Sin embargo, es imposible mantener en *cache* todas las instrucciones y datos que utiliza un programa, debido a la limitación física en el tamaño de *cache*, al igual que es imposible predecir completamente cuáles serán las direcciones que referenciará el procesador en el futuro, debido a la existencia de instrucciones de salto, cambios de contexto en sistemas multitarea o acceso desordenado a datos. Esto hace que en cualquier caso existan accesos que resultan en fallo, por lo que el tiempo de ejecución de las instrucciones no es constante, ya que este depende de si la instrucción se encuentra o no en *cache*, pudiendo variar su situación entre diferentes ejecuciones de la misma instrucción.

Esta variabilidad en el tiempo de ejecución de las instrucciones complica, al igual que sucedía con las unidades segmentadas, el cálculo del WCET de las tareas, ya que el tiempo de ejecución de una ruta del programa depende del estado de la *cache* cuando se inicia esa ruta, y dicho estado depende de las instrucciones que se han ejecutado anteriormente. Además, en los sistemas multitarea, los cambios de contexto suponen un cambio brusco de los contenidos de *cache*, por lo que las tareas, cuando vuelven a ejecución tras una expulsión, se encuentran con unos contenidos diferentes a los que habían dejado, por lo que su tiempo de ejecución se ve modificado. Puesto que el objetivo de esta tesis es la utilización de una arquitectura alternativa a las memorias *cache*s descritas anteriormente, una descripción más detallada de la problemática así como de las soluciones propuestas hasta el momento, se realiza en el siguiente apartado.

C. Sistema de entrada/salida.

El sistema de entrada/salida está formado [Hamacher87] por los distintos periféricos que permiten al sistema computador enviar y recibir información hacia y desde el exterior. Normalmente, el tiempo de operación de estos periféricos es mucho mayor que el del procesador, por lo que a la hora de calcular los tiempos de ejecución de las tareas que acceden a los periféricos es necesario considerar estos dispositivos. Estos accesos a periféricos no suelen comportar grandes problemas, ya que o bien los propios periféricos se diseñan junto con el sistema, por lo que se conoce su comportamiento temporal, o bien el fabricante indica este comportamiento en las especificaciones.

Un poco más complicado es considerar la utilización de interrupciones o acceso directo a memoria, aunque su impacto en el tiempo de respuesta del sistema es muy bajo respecto a la ejecución de instrucciones o acceso a datos, debido a la poca frecuencia con la que ocurren. Por ello, es posible realizar un análisis pesimista y por tanto conservador sin alejarse mucho de los valores reales del sistema [Huang95]

1.2 Utilización de memorias cache en sistemas de tiempo real

El gran impacto en las prestaciones del computador que representa la utilización de *cache*s en la jerarquía de memoria, y la generalización de su uso en una gran mayoría de procesadores y computadores, ha llevado a los diseñadores de sistemas de tiempo real a considerar la inclusión de esta mejora en sus diseños. Pero como se ha comentado en el apartado anterior, la memoria *cache* no proporciona un aumento de prestaciones constante para todas las instrucciones, ni siquiera para todas las ejecuciones de la misma tarea ni tan sólo para todas las ejecuciones de la misma instrucción de una tarea. El comportamiento dinámico de la *cache*, que modifica sus contenidos en función de la zona de código y los datos que la tarea maneja en cada instante, hace que el tiempo de ejecución de una ruta de una tarea dependa del estado de la *cache* en cada ejecución de dicha ruta. Durante el cálculo del WCET de una tarea se hace necesario considerar la existencia de la memoria *cache* y la variabilidad que esta impone en los tiempos de ejecución de las instrucciones. Esta variabilidad recibe el nombre de interferencia intra-tarea (*intra-task*) o intrínseca, y puede definirse como el aumento en el tiempo de ejecución que la tarea sufre debido a los fallos de *cache* que se producen durante su ejecución.

Pero el efecto de la memoria *cache* no sólo se limita a la variación en el tiempo de cómputo de la tarea, sino que afecta también al tiempo de respuesta cuando se trata de un sistema multitarea con expulsiones. Cuando una tarea entra en ejecución expulsando a otra, los contenidos de *cache* se modificarán cargándose el código y los datos de la tarea en ejecución. Cuando la tarea que ha sido expulsada reanuda su ejecución, se encontrará con un estado de *cache* distinto al que tenía antes de su expulsión. Este cambio en los contenidos de la *cache* provocará una ráfaga de fallos, hasta que la *cache* vuelva a cargar las instrucciones y datos de la tarea expulsada. Estos fallos no han sido considerados durante el cálculo del WCET, ya que este cálculo se realiza habitualmente considerando que la tarea bajo análisis se ejecuta sin interrupciones, por lo que las expulsiones provocarán un aumento del tiempo de cómputo de la tarea respecto al estimado inicialmente. Así pues, la tarea expulsada verá doblemente aumentado su tiempo de respuesta: primero, por que la tarea expulsante consume tiempo para su ejecución, y segundo, por que la tarea expulsada deberá restaurar los contenidos de *cache* que tenía antes de la expulsión. Ese último efecto recibe el nombre de interferencia inter-tarea (*inter-task*) o extrínseca, y el tiempo de penalización que produce recibe el nombre de *cache refill penalty* [Busquets95a] o *cache-related preemption delay* [Lee96]

La obtención, tanto del WCET de una tarea como del análisis de planificabilidad de un sistema, cuando la jerarquía de memoria incorpora memoria *cache* es un problema complejo, como muestra la gran cantidad de trabajos y alternativas propuestas durante los últimos años. A continuación se describen los trabajos más significativos agrupados en dos conjuntos, técnicas de análisis y técnicas de aumento del determinismo. En el primer grupo las soluciones propuestas tratan de realizar un modelo, lo más preciso posible, del comportamiento de la *cache*, permitiendo realizar el análisis de planificabilidad considerando la interferencia *intra-task* y la interferencia *inter-task*. En el segundo grupo, las soluciones propuestas se basan en métodos *hardware* o *software*, que modifican el comportamiento de la *cache* o del uso que las tareas hacen de la *cache*, aumentando el determinismo de la misma y facilitando el cálculo del WCET y la realización del análisis de planificabilidad. La propuesta realizada en esta tesis doctoral, y que se describe en los capítulos siguientes se enmarca dentro de este grupo de soluciones.

1.2.1 Análisis de la memoria cache

Los trabajos incluidos en este grupo tratan de predecir, mediante simulación, modelado o análisis, los tiempos de ejecución y respuesta de las tareas que forman un sistema de tiempo real. Estos trabajos pueden clasificarse en: análisis mediante medida, análisis mediante simulación y análisis estático de la *cache*.

A. Análisis mediante medida

El objetivo de estos trabajos es obtener el tiempo de respuesta de las tareas que forman el sistema mediante su ejecución y medición. La realización de las medidas se lleva a cabo instrumentando el código [Kenny90] o aprovechando los servicios *hardware* que ofrecen los procesadores actuales para el análisis y depuración de código [Corti00]. Operando de esta manera el efecto de la *cache* no se considera de forma explícita, sino que este encuentra implícito en los valores temporales obtenidos. En [Sarkar89] se presenta un método similar pero que proporciona también un análisis estadístico de la variación que se produce en los tiempos medidos. En [Busquets93] se mide la frecuencia con que la tarea bajo estudio accede a cada línea de *cache* mediante la obtención de trazas, sin proporcionar ningún valor temporal pero caracterizando por completo el comportamiento de la memoria *cache*.

La principal ventaja de estos métodos es que no existen restricciones en cuanto a las características y estructura del código que se desea analizar, excepto, claro esta, que sea posible ejecutarlo.

Sin embargo, el análisis mediante medida presenta dos problemas. El primero de ellos es que al instrumentar el código para tomar medidas, éste es modificado, y por tanto se modifica también su comportamiento temporal, normalmente aumentado los tiempos de ejecución. Cuando además entra en juego la *cache*, esta modificación cobra mayor importancia, ya que los conflictos que se producirán en los accesos a *cache* pueden variar entre el código instrumentado y el original. El segundo problema, mucho más grave, es la imposibilidad de garantizar que los tiempos obtenidos pertenecen al peor caso. Cuando el tiempo de ejecución de las instrucciones es constate para cada ejecución, es relativamente sencillo identificar cuál es la combinación de datos de entrada que produce el peor tiempo de respuesta, ya que coincide con la ruta más larga. Cuando se introduce la memoria *cache* en el sistema computador esto deja de ser cierto debido a la variabilidad en los tiempos de ejecución, por lo que sería necesario ejecutar el sistema con todas las posibles combinaciones de datos de entrada, tarea que es imposible debido al elevado número de posibilidades y la ingente cantidad de tiempo necesario para llevarla a cabo. Así pues, al no poder dar garantía de que los tiempos de respuesta medidos no serán superados en ningún caso, este método no es valido para realizar el análisis de planificabilidad en sistemas de tiempo real estricto [Hillary02].

B. Análisis mediante simulación.

Este método se basa en la obtención de las trazas de ejecución de las tareas, describiendo con detalle los accesos a memoria que realizan. Una vez obtenido este comportamiento, los resultados temporales se obtienen mediante la simulación de los accesos que las tareas realizan a la *cache* y de la ejecución de las instrucciones. El análisis mediante simulación presenta la ventaja de que las trazas pueden obtenerse mediante mecanismos *hardware*, sin presentar por tanto intrusismo, y que permite estudiar las prestaciones de varios modelos de *cache* sin necesidad de que éstas existan realmente.

El análisis mediante simulación es un método mucho más costoso que el análisis mediante medida, ya que en primer lugar es necesario ejecutar las tareas, y luego simularlas, siendo, en muchos casos, el tiempo de simulación muy superior al tiempo necesario para ejecutarlas. Además, también presenta la imposibilidad de garantizar que los resultados obtenidos corresponden con el peor caso, y por tanto la seguridad de las conclusiones alcanzadas, por lo que se invalida el uso de este tipo de análisis en los sistemas de tiempo real estricto, al igual que sucede con el análisis mediante medida.

C. *Análisis off-line*

En el análisis off-line o estático los tiempos de ejecución y de respuesta de una tarea se obtienen mediante la creación y evaluación de un modelo del sistema, que considera, de la forma más exacta posible, las particularidades de las tareas y de la *cache*. Este análisis presenta la ventaja de que no es necesario ejecutar las tareas, por lo que no es necesario disponer del sistema físico, modificar el código para su instrumentación ni disponer de recursos *hardware* en el procesador que permitan obtener medidas.

Pero la principal ventaja de este método es que es independiente de las rutas que ejecuten las tareas, o más concretamente, todas las rutas posibles son evaluadas con un coste computacional finito. Esta evaluación completa de todas las alternativas y posibilidades proporciona, como resultado, un valor que puede considerarse seguro, ya que será siempre igual o superior al que realmente presentará el sistema en cualquier escenario y bajo cualquier condición. La importancia de obtener esta garantía, una cota superior del comportamiento temporal de las tareas, ha hecho que en los últimos años hayan aparecido diferentes aproximaciones y modelos para realizar el análisis estático. A continuación se describen los trabajos más significativos [Martí98]

En cuanto a la consideración de la interferencia intrínseca en el cálculo del WCET de una tarea, el trabajo desarrollado en [Lim94a] presenta un análisis estático para sistemas con memoria *cache* de instrucciones y función de correspondencia directa, que permite obtener el WCET de una tarea. Este mismo análisis se utiliza en [Lim94b], junto con el trabajo presentado en [Rhee94b] para combinar el análisis de unidades segmentadas y *cache*. El trabajo se basa en el análisis de tiempo presentado en [Shaw89] aplicándolo a código máquina y describiendo el estado de *cache* en puntos concretos de las rutas de la tarea. Para ello se definen dos conjuntos para cada estructura (bucles, rutas alternativas...), conteniendo cada conjunto tantos elementos como líneas tenga la memoria *cache*. En el primer conjunto (*entrada*) se almacenan los primeros bloques de código que se referenciarán al ejecutar la estructura (un bloque por cada línea). En el segundo conjunto (*salida*), se almacenan los últimos bloques ejecutados en la estructura (un bloque por cada línea).

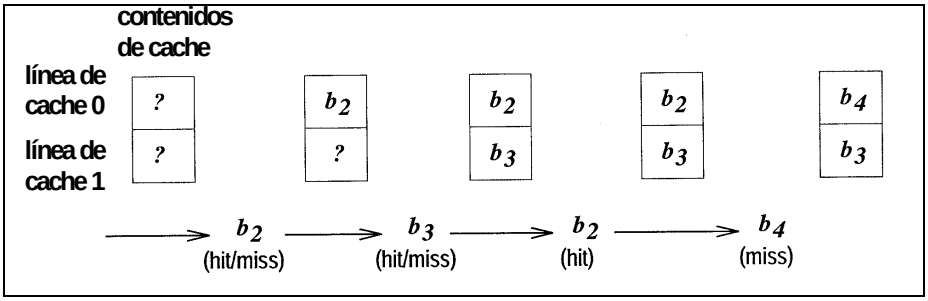


Figura 5. Ejemplo de llenado de los conjuntos *entrada* y *salida*.

Un elemento del conjunto puede estar vacío si ningún bloque de código se corresponde con la línea de *cache* que representa. Si sólo un bloque de código se corresponde con una determinada línea de *cache*, el elemento que representa a esa línea contendrá la misma información en los dos conjuntos. Finalmente, el elemento que representa a una línea de *cache* tendrá información diferente en cada conjunto si más de un bloque diferente de código se ubica en la misma línea dentro de una misma estructura. La Figura 5 muestra un ejemplo.

El cálculo del tiempo de ejecución de un bloque básico (secuencia de instrucciones sin ruptura de flujo) es sencillo, ya que cualquier referencia a un bloque que haya sido cargado en *cache* y no entre en conflicto con ningún otro bloque de código perteneciente a ese bloque básico representará acierto. Por el contrario, si existe más de un bloque ubicado en la misma línea de *cache* entre dos referencias al mismo bloque, todos serán fallos. Asimismo, se considera fallo la primera referencia a cualquier bloque de código en un bloque básico.

Sin embargo, considerar como fallo la primera referencia a un bloque de código puede introducir un exceso de pesimismo, ya que el acierto o fallo de esta primera referencia dependerá del contenido de la *cache* al finalizar la ejecución del bloque básico anterior. Por ello, en un refinamiento posterior, se reconvierte la función suma de bloques básicos en función concatenación. De esta forma, la primera referencia a un bloque de código será acierto si ese bloque se encuentra en el conjunto salida del bloque básico predecesor. Por el contrario, la referencia será un fallo. Así pues, el conjunto entrada de la concatenación de dos bloques será el conjunto entrada del primer bloque (si algún elemento es vacío, se utilizará el elemento correspondiente del conjunto entrada del segundo bloque), y el conjunto salida será el conjunto salida del segundo bloque (si algún elemento es vacío, se utilizará el elemento correspondiente del conjunto salida del primer bloque). El objetivo de esta concatenación es la construcción de nuevos bloques que se utilizarán para nuevas concatenaciones. El tiempo de ejecución de este nuevo bloque será la suma de los tiempos de ejecución de cada uno de los dos bloques que lo integran menos el tiempo de fallo considerado erróneamente para las primeras referencias del segundo bloque. Un fallo se habrá considerado erróneamente si el elemento del conjunto salida del primer bloque coincide con el del conjunto entrada del segundo bloque.

Finalmente, puesto que un bloque básico puede tener más de un predecesor, el resultado de una concatenación puede ser un conjunto de pares entrada/salida. Para limitar el crecimiento del problema, se define una función de poda que eliminará aquellos pares entrada/salida cuyo tiempo de ejecución nunca pueda ser el peor caso, considerando los bloques predecesores y sucesores (la función poda puede dejar el conjunto sin variación, ya que no siempre es posible eliminar algún par).

Esta técnica consigue, en tres pasos, un cálculo del tiempo de ejecución muy ajustado. Sin embargo, el tratamiento de bucles tal como se plantea es un problema computacionalmente intratable, ya que considera todos las posibles rutas de ejecución en todas las iteraciones. Para solucionar este inconveniente, se realiza una simplificación consistente en tratar el bucle de forma aislada al resto del código. Esta simplificación permite resolver el problema de forma eficiente pero generando un resultado superior al WCET real de la tarea, al considerar la *cache* vacía a la entrada y salida del bucle, y recordar una sola iteración en el interior del bucle.

La técnica anterior se ha descrito para memorias *cache* de instrucciones y con correspondencia directa. Para memorias *cache* asociativas de *n* conjuntos, es sencillo ampliar la técnica para manejar estas memorias en función del algoritmo de reemplazo utilizado. La modificación más importante es aumentar el tamaño de los conjuntos entrada y salida que mantienen el estado de la *cache* al iniciar y al finalizar la ejecución de un bloque de código. Para el caso de memorias *cache* de datos es aplicable la misma técnica que para las *cache*s de instrucciones, excepto para las referencias dinámicas, que no son conocidas en tiempo de compilación. Para solucionar este problema, los autores proponen una solución *hardware*, consistente en impedir que las referencias a datos cuyas direcciones no son conocidas en tiempo de compilación se copien en la *cache*. Las referencias conocidas en tiempo de compilación sí se permite que se almacenen en *cache*. De esta forma se garantiza que las referencias que no pueden ser tratadas produzcan siempre un fallo. Para implementar este mecanismo se añade un bit a cada instrucción de acceso a memoria de datos con el que se indica si el dato leído debe traerse a *cache* o no. De esta forma se consigue una estimación más ajustada del WCET de la tarea, pero a costa de empeorar las prestaciones del sistema, ya que un buen número de los datos de la tarea tendrán prohibido el acceso a *cache*.

El análisis descrito ha sido utilizado para calcular el WCET de un conjunto de tareas sobre los procesadores MIPS R3000 y R3010 en [Hur96]. El tratamiento de la *cache* de datos es un poco diferente y más conservativo. Sólo las referencias a memorias basadas en los registros *Gp* y *Sp* (*Global pointer y Stack pointer*) son analizadas para calcular el tiempo de ejecución según produzcan acierto o fallo, mientras que para cualquier otra referencia a memoria de datos se considera su tiempo de ejecución como el de dos fallos de *cache*. El primer fallo es el de la propia referencia, y el segundo es debido a que el acceso puede reemplazar en la *cache* una línea que podría haberse estimado como un acceso con acierto. Finalmente, para los accesos de

escritura su tiempo de ejecución se considera como el tiempo de escribir en memoria principal, es decir, se asume el protocolo *write trough* para mantener la coherencia de memoria. Para comprobar la efectividad de la técnica presentada se han ejecutado siete programas de prueba. Para cada programa se ha medido su tiempo de ejecución en el procesador real, analizando la tarea para utilizar como datos de entrada aquéllos que produzcan el peor tiempo de ejecución. A continuación se ha estimado el WCET considerando sólo el efecto de la segmentación (todos los accesos a *cache* se consideran fallos), considerando segmentación y *cache* de instrucciones (todas las referencias a *cache* de datos se consideran fallos), y finalmente, segmentación, *cache* de instrucciones y *cache* de datos. Los resultados son muy ajustados para la mayoría de los programas de prueba, es decir, los valores estimados del WCET son muy próximos a los observados en la ejecución real, y, lo más importante, los valores estimados son siempre superiores a los observados. Los dos casos donde las diferencias entre el WCET medido y el WCET calculado son mayores es debido a la existencia de bucles anidados dependientes de los datos de entrada. Este problema no es privativo de la *cache*, sino de la estructura de la tarea y del análisis por rutas de ejecución, y se muestra en casi todos los análisis estáticos. Una posible solución es la utilización de técnicas de estimación del máximo número de iteraciones de un bucle. Una conclusión importante que se puede extraer de los resultados experimentales es que la influencia de la memoria *cache* sobre el WCET de la tarea es muy superior a la de la segmentación, tal como se muestra en la Tabla 1.

Técnica / Programa	**Arrsum**	**Bs**	**Fib**	**FFT**	**Isort**	**MM**	**Sqrt**
Medido	242	283	683	5190	2849	9141	302
Estimado segmentación	592	574	2642	59593	17757	32857	747
Estimado segmentación y *cache* instrucciones	256	314	710	22297	6709	11653	327
Estimado segmentación, *cache* instrucciones y datos	296	346	710	27213	8077	13153	327

Tabla 1. Resultados del análisis mediante la concatenación de los conjuntos *entrada* y *salida* *(*10^3 *ciclos)*

En [Kim96] se mejora el tratamiento de la memoria *cache* de datos presentando en [Lim94a]. La mejora en el cálculo del WCET se obtiene mediante dos actuaciones: parte de las referencias dinámicas a memoria se convierten en estáticas; se mejora el tratamiento de los accesos a memoria dentro de los bucles.

Para convertir referencias dinámicas en estáticas se analiza el flujo del programa y se intenta calcular el contenido del registro base utilizado por las instrucciones de carga y almacenamiento. Este cálculo no siempre es posible, pues el valor del registro base puede depender de algún dato de entrada, pero en muchos casos se parte de un valor constante conocido que es operado varias veces. En los bucles, en muchos casos se realizan accesos consecutivos a una estructura de datos, como por ejemplo un vector. De forma similar se convierten estas referencias dinámicas en estáticas, obteniendo el número de aciertos y fallos que se producen en la ejecución del bucle. Para las referencias que no pueden ser resueltas, o en el caso de los bucles, para las referencias que generan conflictos, se considera una penalización del doble de tiempo de un fallo de *cache*. Pese a este conservadurismo, los resultados obtenidos, y que se muestran en la Tabla 2, mejoran el WCET obtenido en [Lim94a].

Los resultados obtenidos en el cálculo del WCET son bastante ajustados, pero el principal problema de este análisis sigue siendo la necesidad de analizar todos las posibles rutas de ejecución del programa, lo que puede convertir en intratable el problema para programas con un tamaño medio o grande. Aun así, los sistemas de tiempo real estrictos suelen utilizarse en sistemas de control, donde las tareas más críticas suelen tener tamaños pequeños, lo que permitiría aplicar esta técnica. Los autores también proponen utilizar técnicas de optimización

de código y de eliminación de rutas imposibles para simplificar el análisis y obtener el WCET en un tiempo razonable.

Técnica / Programa	Arrsum	Fib	Isort	MM	Sqrt
Medido	242	683	2849	9141	302
Estimado segmentación	592	2642	17757	32857	747
Estimado segmentación y *cache* instrucciones	256	710	6709	11653	327
Estimado segmentación, *cache* instrucciones y datos	256	710	6517	10453	327

Tabla 2. Resultados de la concatenación de conjuntos los conjuntos *entrada* y *salida (*10^3 ciclos)*

Para evitar el problema del crecimiento en el número de rutas a analizar, se propone en [Li95] y [Li96] un método que permite calcular el WCET de una tarea sin necesidad de considerar explícitamente todas las rutas que ésta puede seguir. Para obtener este cálculo se define un bloque básico como la máxima secuencia de instrucciones consecutivas en las que no hay ruptura en el flujo, al igual que en el primer método descrito. Sobre cada uno de los bloques básicos de la tarea se calcula el coste de ejecución de sus instrucciones y el número de veces que cada bloque se ejecuta. El coste de ejecución total de un bloque será el producto de su tiempo de ejecución por el número de veces que se ejecuta, y por lo tanto el coste de ejecución de la tarea será el sumatorio del coste total de ejecución de todos los bloques. Este sumatorio recibe el nombre de función objetivo. Sin embargo, no todos los bloques de la tarea se ejecutarán en el peor caso (sentencias *if-then-else*), por lo que el WCET se calcula maximizando mediante programación lineal la función objetivo, introduciendo en la resolución del problema un conjunto de restricciones, planteadas como ecuaciones y desigualdades. La Figura 6 presenta un ejemplo de construcción de bloques básicos a partir de lenguaje de alto nivel.

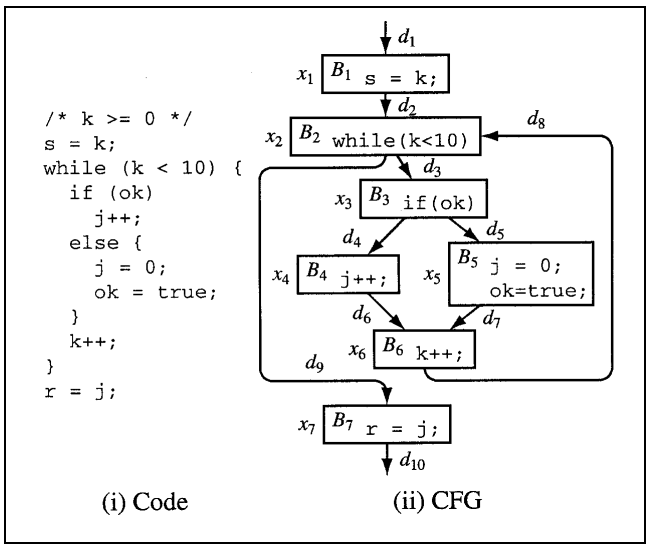


Figura 6. Creación del CFG a partir de código de alto nivel.

Inicialmente, el tiempo de ejecución de la tarea vendrá calculado por la siguiente ecuación:

$$WCET = \sum_{1}^{N} C_i * X_i$$

donde C_i es el coste de ejecución del bloque básico B_i, X_i el número de veces que se ejecuta el bloque B_i, y N el número de bloques básicos. Sin embargo, el resultado de este sumatorio es un valor del WCET extremadamente distinto al real, ya que no tiene en consideración ninguna característica de la tarea o del *hardware*. Para el ejemplo de la Figura 6, este sumatorio considera que los bloques B_4 y B_5 se ejecutan en todas las iteraciones del bucle *While*, lo cual es imposible ya que cada bloque corresponde a una ruta alternativa. Para corregir estas deficiencias se incluyen restricciones a la función objetivo, que serán consideradas durante la resolución de dicha función. El WCET de la tarea corresponderá al máximo valor de esta función objetivo que cumpla todas las restricciones definidas.

Las restricciones que pueden incorporarse a la función objetivo se dividen en dos tipos:

- Restricciones de la ruta del programa: pueden ser restricciones estructurales del programa o restricciones funcionales del programa. Las primeras se obtienen automáticamente del grafo de control de flujo, y son ecuaciones que limitan el número de veces que un bloque puede ejecutarse. Las segundas son proporcionadas por el usuario o extraídas de la semántica del programa, y se refieren normalmente a límites superiores en el número de iteraciones de un bucle.
- Restricciones del modelo de la microarquitectura: estas restricciones modelan el comportamiento del *hardware* del computador, como la segmentación del procesador o la existencia de memorias *cache*. Estas restricciones suelen referirse al tiempo de ejecución de cada uno de los bloques básicos.

El análisis de una memoria *cache* de instrucciones con función de correspondencia directa [Li96] y [Li95b] requiere redefinir la función objetivo y añadir una serie de restricciones que modelen el comportamiento de dicha *cache*. Para analizar la tarea, esta se divide en *l-bloques*, que son secuencias contiguas de instrucciones que se ubican en la misma línea de *cache*. Utilizando estos *l-bloques*, la función de coste a maximizar, y que por tanto nos proporcionará el WCET de la tarea, es el sumatorio, para todos los *l-bloques*, de su tiempo de ejecución por el número de veces que se ejecuta dicho bloque, considerando por separado los casos que son acierto en *cache* y los casos que son fallo. De esta forma, la expresión contenida en el sumatorio es el tiempo que tarda en ejecutarse un *l-bloque* en caso de acierto, multiplicado por las veces que se ejecuta y produce acierto, más el tiempo que tarda en ejecutarse en caso de fallo, multiplicado por las veces que se ejecuta y produce fallo. A esta función se le aplicarán todas las restricciones referentes a la estructura y funcionalidad del programa, junto con nuevas restricciones referidas a la memoria *cache*.

Para modelar el comportamiento de la *cache*, se necesita contar el número de aciertos y fallos que produce cada uno de los *l-bloques*. Para ello, los *l-bloques* se clasifican en dos tipos: *no-conflicto* y *conflicto*. Un *l-bloque no-conflicto* es aquel cuya ejecución no desplaza de la memoria *cache* a ningún otro *l-bloque*. En caso contrario, si la ejecución del *l-bloque* desplaza a uno o más *l-bloques* (no simultáneamente), se llama *conflicto*. Modelar el comportamiento de los *no-conflicto* es sencillo, pues su número de ejecuciones que producen fallo es como mucho 1, al no ser desplazados de *cache* una vez se han cargado. Para los bloques *conflicto* el análisis es más complicado, ya que las expulsiones que se generan entre ellos dependen del orden en que se ejecuten. Para poder tratar este problema, se construye un grafo de conflictos de *cache* (CCG) para cada línea de *cache*, donde se especifica, de igual forma que en un CFG, las transiciones entre todos los *l-bloques* que compiten por la misma línea de *cache*. Este CCG se resuelve de igual manera que el CFG de la tarea, especificando restricciones en función de las transiciones que se pueden realizar entre los diferentes nodos y de restricciones estructurales y funcionales de la propia tarea. En la Tabla 3 se presentan los resultados obtenidos mediante la herramienta *Cinderella*, que calcula el WCET considerando la existencia de una memoria *cache* de instrucciones. Una gran ventaja de este método es la velocidad con que se calcula el WCET de una tarea, ya que la resolución de las ecuaciones mediante programación lineal nunca sobrepasa los pocos minutos. El principal inconveniente es la complejidad en la especificación de las

ecuaciones. Como ejemplo, el número de restricciones utilizadas para modelar un programa de 761 líneas de código alcanza casi las 1.500 expresiones.

Programa	**Wcet medido**	**Wcet calculado**
Sort	9.99×10^6	27.8×10^6
Fast fourier	2.20×10^6	2.63×10^6
Whetstone	6.94×10^6	10.5×10^6
Dhrystone	5.76×10^5	7.57×10^5
stats	1.16×10^6	2.21×10^6

Tabla 3. Resultados obtenidos con la herramienta Cinderella (ciclos de procesador)

El tratamiento de memorias *cache* de instrucciones asociativas y de *cache* de datos mediante programación lineal es introducido en [Li96b]. El tratamiento de la *cache* de instrucciones con función de correspondencia asociativa se realiza reconvirtiendo el grafo de conflictos de *cache* en un grafo de transición de estados de *cache*, donde cada nodo del grafo representa los posibles estados de una línea de *cache*, con los diferentes *l-bloques* que puede contener. El significado es el mismo en ambos grafos, y pueden utilizarse las mismas restricciones que se definieron para la *cache* de correspondencia directa en las *cache*s de correspondencia asociativa. Para el caso de las *cache*s de datos, es necesario realizar primero un análisis del flujo de datos para obtener las direcciones de memoria a las que acceden las instrucciones de carga y almacenamiento. Sin embargo, al contrario que las direcciones de instrucciones, estas direcciones de datos se obtienen en muchos casos de forma dinámica y varían en las sucesivas ejecuciones de las instrucciones. Esto impide en muchos casos obtener una única dirección para cada instrucción de acceso a memoria, por lo que finalmente se obtiene una lista de posibles direcciones. Con esta lista de posibles direcciones se construye un grafo de conflictos de *cache* de datos, con el mismo significado que el CCG y grafo de transiciones de *cache* construido para la *cache* de instrucciones. A partir de este grafo es posible definir restricciones que permitirán resolver mediante programación lineal la función objetivo.

Un caso particular son las instrucciones que acceden a una posición de memoria o a otra, pero nunca a ambas. Para resolver estos casos se añade una restricción que indica esta situación. La función objetivo es modificada y se añade en ella el coste de los aciertos y fallos en los accesos a la *cache* de datos, así como los contadores que indican las veces que estos aciertos o fallos se producen. De esta forma, la función de coste contiene los tiempos de ejecución de cada bloque de código, considerando *cache* de instrucciones y de datos, con sus respectivos aciertos y fallos y el número de veces que estos aciertos y fallos se producen para cada bloque de código, al ejecutar repetidas veces cada bloque. Este análisis de la *cache* de datos no ha sido todavía implementado, por lo que los autores no presentan resultados. Sin embargo, teniendo en cuenta los resultados de los trabajos anteriores, el WCET obtenido será cercano al WCET real de la tarea en función de la dificultad para obtener las direcciones de datos referenciadas por cada una de las instrucciones. Esta dificultad, junto con la complejidad a la hora de especificar las restricciones que modelan la tarea y la memoria *cache* son el principal inconveniente de esta propuesta.

Una forma diferente de obtener los aciertos y fallos que se producen en una tarea es el presentado en [Mueller95b]. El objetivo de este trabajo es obtener una clasificación de todas las instrucciones de la tarea, indicando para cada instrucción si producirá acierto o fallo al acceder a la *cache* de instrucciones. Esta clasificación se obtiene en dos pasos. Primero, una simulación estática, y después una instrumentación del código de la tarea para obtener la clasificación de algunas instrucciones durante la ejecución de la tarea. El resultado es una clasificación de cada una de las instrucciones de la tarea según se muestra en la Tabla 4.

Clasificación	Significado
Always miss	Siempre que se acceda a la instrucción, se producirá un fallo.
Always hit	Siempre que se acceda a la instrucción, se producirá un acierto.
First miss	El primer acceso a la instrucción producirá un fallo, y todos los demás producirían acierto.
Conflict	No es posible asignarle alguna de las otras tres clasificaciones.

Tabla 4. Posibles clasificaciones de las instrucciones.

Para obtener esta clasificación se parte de un compilador modificado que genera, junto con el ejecutable, un grafo de control de flujo, donde cada nodo representa un *Unique Path* (UP). Un UP es un conjunto de bloques de código conectados por transiciones, donde al menos una transición es única, es decir, no pertenece a ninguna otra UP. Para cada una de estas UP, se calcula el estado de la *cache* antes de ejecutar la UP. Este cálculo se realiza partiendo de una *cache* vacía, y se recorren todas las UP de la tarea, siguiendo el flujo del programa. El estado de la *cache* antes de una UP es el estado de la *cache* después de la UP predecesora, junto con las líneas que la predecesora ha cargado en *cache* y eliminando aquellas que han sido reemplazadas. El algoritmo es muy similar al algoritmo de análisis del flujo de datos *DFA (Data Flow Analysis*) y de forma iterativa y en pocas pasadas, consigue calcular el estado de la *cache* para cada una de las UP de las tareas. Una vez obtenido este estado, es fácil clasificar cada instrucción, considerando tres factores: es la primera instrucción que se ejecuta de un bloque de código perteneciente a una línea de *cache*; la línea de *cache* a la que pertenece la instrucción se encuentra en *cache*; y existe otro bloque de código que se corresponde con la misma línea de *cache* en la UP.

Después de esta clasificación estática quedan todavía instrucciones cuya clasificación no ha podido ser determinada. Para resolver estas situaciones, se instrumenta el código para insertar contadores, que permitirán refinar la clasificación obtenida en el primer paso. En este segundo paso, las instrucciones clasificadas como conflicto modificaran su clasificación a *first miss*, o *first hit*. Esta nueva categoría indica que la primera referencia a la instrucción producirá acierto, mientras que todos los demás accesos producirán fallo [Mueller00]. El resultado es un conocimiento completo del comportamiento de la *cache* para toda la tarea.

Una vez clasificadas todas las instrucciones de la tarea, sólo resta buscar la peor ruta para encontrar el WCET de la tarea. Sin embargo, este problema no es trivial, y es necesario un análisis de tiempos, introducido en [Arnold94]. En este trabajo se construye un árbol de análisis de tiempos, donde cada nodo representa un bucle (con una o más iteraciones), y las transiciones representan el flujo del programa. Este árbol permite simplificar el cálculo del WCET. Para cada nodo del árbol, se analizan sus rutas mediante un algoritmo iterativo que trata, en cada pasada, los *first miss, first hit, always hit y always m*iss. Este algoritmo termina cuando se ha ejecutado el número de veces indicado por el límite superior del bucle o cuando no hay modificaciones en el tiempo obtenido. Con este algoritmo se obtiene el peor caso de este nodo. La concatenación de nodos se realiza procesando primero los bucles más interiores, es decir, recorriendo desde las hojas hacia la raíz el árbol de análisis de tiempos. Para cada nodo, se aplica el algoritmo anterior considerando también el tiempo de los nodos sucesores en el árbol. El resultado obtenido es bastante ajustado en algunos casos, pero la principal ventaja es el bajo coste temporal necesario para obtener el WCET de una tarea de gran tamaño considerando la existencia de memoria *cache* de instrucciones con correspondencia directa.

La utilización del DFA permite calcular los estados de la *cache* a través de las diferentes estructuras de la tarea, obteniendo un análisis para *cache*s de instrucciones asociativas en [Mueller97] de la misma forma que en el caso de *cache*s de correspondencia directa. El cálculo del WCET de la tarea se realiza utilizando el árbol de análisis de tiempo una vez obtenida la clasificación para cada una de las instrucciones. Una de las principales ventajas de este método es que la herramienta de cálculo de tiempos es totalmente independiente de la arquitectura, por

lo que puede aplicarse a cualquier configuración siempre que se disponga de la clasificación de las instrucciones. Uno de los principales inconvenientes es la necesidad de instrumentar el código para realizar la clasificación de algunas instrucciones. La instrumentación no es sencilla, ya que al realizarla debe tenerse en cuenta las instrucciones objeto del estudio. Además, esta instrumentación puede modificar el comportamiento de la tarea, tal como se comento al hablar del análisis mediante medida.

La Tabla 5 presenta la comparación entre el WCET estimado y el observado de la ejecución de un conjunto de tareas ejecutadas en un sistema con *cache* de instrucciones y función de correspondencia asociativa de cuatro conjuntos. Los resultados son muy ajustados excepto para el programa *Sort*. Esto es debido a que esta tarea está formada por dos bucles anidados. El número de iteraciones del bucle interior depende del valor del contador del bucle externo, mientras que a la herramienta de análisis de tiempo se le proporciona el número máximo de iteraciones, lo que provoca una sobreestimación del WCET. Como ya se ha comentado, este problema es común a la mayoría de métodos y técnicas que realizan un análisis estático.

Programa	**Ciclos observados**	**Ciclos estimados**	**Ratio**
Des	95877	109069	1.14
Matcnt	443754	443790	1.00
Matmult	1430538	1430538	1.00
Matsum	343628	343646	1.00
Sort	3130692	6249474	2.00
Stats	183491	192578	1.05
Average	937996	1461505	1.20

Tabla 5. Resultados obtenidos por el DFA.

En [White97] se presenta un método para el cálculo del WCET de una tarea considerando la existencia de memorias *cache* de datos basado en el método descrito anteriormente. El análisis se divide en tres partes. Primero, el compilador calcula, para cada instrucción de acceso a memoria, el rango de direcciones a los que accede. A continuación, se utiliza el método anterior para clasificar cada instrucción en una de las cuatro categorías en función de la dirección del dato al que la instrucción accede, y no en función de la dirección de la propia instrucción. Sin embargo, en el caso de instrucciones de acceso a datos y al contrario que los accesos para buscar instrucciones, la dirección puede cambiar, por lo que la clasificación para ese acceso no permanece constante. Almacenar todas las clasificaciones de todas las instrucciones de una tarea puede requerir un espacio excesivo, por ejemplo, si pensamos en un bucle recorriendo un vector. Para evitar este excesivo consumo de recursos, se añade una nueva categoría, llamada calculada, que lleva asociada el número máximo de fallos que una instrucción de acceso a datos puede generar. Si todos los accesos de la instrucción son aciertos, esta es clasificada con *always hit*, o *always miss* si todos sus accesos son fallos. Si todos los accesos de esa instrucción no pertenecen a la misma categoría, es cuando se utiliza la nueva clasificación, que aunque puede introducir cierta sobreestimación, es necesario para poder tratar el problema. Finalmente, una vez clasificadas todas las instrucciones, se utiliza el árbol de análisis de tiempos sin ninguna modificación para realizar el cálculo del WCET.

El problema de las memorias *cache* multinivel es tratado en [Mueller97b], donde se realizan modificaciones al algoritmo de clasificación de los accesos a memoria. Básicamente, el proceso de clasificación se realiza una vez por cada nivel de *cache* existente en el sistema computador. El proceso se inicia clasificando las instrucciones para los accesos al nivel más próximo al procesador, y continúa calculando las nuevas clasificaciones de cada instrucción para el siguiente nivel, hasta tener una clasificación por cada nivel para todas las instrucciones de la

tarea. El WCET de la tarea se obtiene aplicando el árbol de tiempos, pertinentemente modificado, a este conjunto de clasificaciones.

Otro trabajo similar, basado en la estimación del estado de *cache* al ejecutar una determinada instrucción y en el modelado de las rutas que el programa ejecutará se presenta en [Ferdinand99] con resultados muy similares a las técnicas descritas anteriormente.

Una aproximación completamente diferente es presentada en [Mueller98]. Una forma muy sencilla de calcular el WCET de una tarea es ejecutar dicha tarea introduciendo como datos de entrada aquellos que producen el peor caso. Sin embargo, identificar la combinación de datos de entrada que producen el peor caso de ejecución de la tarea no es sencillo, y en la mayoría de los casos ni siquiera es posible. La verificación de todas las posibles combinaciones de datos de entrada es un problema intratable. Para encontrar, en un tiempo acotado, esta combinación de datos de entrada, se propone la utilización de algoritmos genéticos. Los algoritmos genéticos, que se describirán en un capítulo posterior, permiten realizar una búsqueda pseudo-aleatoria del máximo o mínimo de una función. En este caso, y partiendo de un modelo de la tarea y de la memoria *cache*, se intenta encontrar el máximo valor de la función correspondiente a dichos modelos. Sin embargo, los algoritmos genéticos no garantizan que el resultado obtenido sea el máximo o mínimo posible, en nuestro caso el WCET de la tarea, ya que no se realiza una búsqueda exhaustiva de todos los posibles datos de entrada, por lo que no pueden ser utilizados para sistemas de tiempo real estricto, donde es necesario conocer, sino el WCET exacto de la tarea, sí una cota superior. En cambio, son utilizados en sistemas de control, como maniobras de ascensores [So97], ya que el proceso puede ser detenido antes de su finalización obteniendo una solución aproximada, o en sistemas de tiempo real no estricto, donde las tareas pueden incumplir, en mayor o menor medida, su plazo de entrega [Bernat98], o el sistema dispone de técnicas para la detección y recuperación de estas violaciones. En los experimentos presentados en [Mueller98] se muestra que en muchos casos el WCET estimado por el algoritmo genético es menor que el que realmente presenta la ejecución de la tarea, corroborando la imposibilidad de utilizarlos en sistemas de tiempo real estricto

De los métodos descritos para la obtención del WCET de una tarea, considerando la existencia de memorias *cache* en el sistema computador, ninguno de ellos puede considerarse mejor que otro, ya que cada uno de ellos se adapta mejor a un tipo de problema determinado. El método basado en la clasificación de instrucciones es adecuado para tareas de gran tamaño, mientras que la gran cantidad de información que necesita la propuesta basada en el análisis de tiempos lo hace inviable, aunque para tareas sencillas y pequeñas obtiene mejores resultados que los demás. Algo similar sucede con la obtención del WCET mediante programación lineal, que aunque presenta una mayor velocidad en la obtención del resultado, el modelado del comportamiento de la *cache* no es muy exacto, y cuando la tarea que debe analizarse es compleja, el número de restricciones y de grafos que deben resolverse puede ser un problema intratable. El principal inconveniente de la clasificación de instrucciones es la excesiva rigidez, ya que al utilizar sólo cuatro categorías puede forzar a considerar como *always miss* accesos que no siempre lo son, presentando valores del WCET con una importante sobreestimación

Aunque todos los métodos están implementados en herramientas que realizan el cálculo de forma automática, requieren en mayor o menor medida información suministrada por el usuario. La exactitud con que se proporcione esta información puede determinar la precisión de los resultados obtenidos. Un punto a favor de todos los métodos comentados es que el valor estimado del WCET es siempre menor que el tiempo de ejecución de la tarea si se desconecta la memoria *cache*. Por tanto, aunque los resultados no sean exactos o extremadamente ajustados, siempre serán mejores que la solución simple de eliminar la memoria *cache* para evitar las variaciones en los tiempos de ejecución de las instrucciones.

Respecto al tratamiento de *cache*s de datos, todos sufren el mismo problema, y es la dificultad de conocer a priori la totalidad de las referencias que se ejecutarán. Las soluciones presentadas son un compromiso entre la sencillez y la exactitud, con mayor tendencia a la sencillez que a la exactitud, debido a que el problema es realmente complejo.

Respecto a la consideración e inclusión de la interferencia extrínseca en el análisis de planificabilidad, una forma sencilla de realizar dicho análisis es ampliar RMA [Liu73] incorporando el efecto de la *cache* al cálculo de la utilización, tal como se presenta en [Basumallick94] y que recibe el nombre de CRMA (*Cache*d RMA). El análisis RMA se aplica a sistemas de prioridades fijas con expulsiones, que utilizan la política de planificación RM. Para incorporar el efecto de la *cache* en este análisis, la utilización de una tarea se calcula con las siguientes formulas:

$$U_i = \frac{C_i'}{T_i}$$

$$C_i' = C_i + \gamma$$

donde C_i es el WCET de la tarea τi considerando la interferencia intrínseca, y el factor γ es el valor de la interferencia extrínseca provocada por los cambios de contexto. El sistema será planificable si la suma de la utilización de todas las tareas no supera, aproximadamente, el 69%, tal como mostraba la ecuación de RMA.

El valor de C_i puede calcularse mediante alguno de los métodos descritos en la sección anterior. El valor de γ es más complejo de calcular, ya que depende del estado de la *cache* en el momento de la expulsión, del uso de la *cache* que haga la tarea que genera la expulsión, y del uso que haga de la *cache* la tarea expulsada al reanudar su ejecución. En [Mogul91] se proponen algunas formas de acotar este valor, pero en CRMA sólo es posible utilizar el máximo valor, que corresponde con el rellenado completo de la *cache* tras cada expulsión, ya que RMA es independiente de la secuenciación de las tareas, por lo que no puede asumirse ningún escenario en cuanto a la interacción entre las tareas.

Esta misma razón, la independencia del análisis frente a la secuenciación de las tareas, obliga a sumar la penalización por la recarga de la *cache* tras la expulsión a la tarea que produce la expulsión, y no a la tarea que sufre la expulsión. Para operar de forma contraria, es decir, incluir el tiempo de recarga de la *cache* en el tiempo de ejecución de la tarea expulsada, sería necesario saber cuantas veces es expulsada una tarea, información que no se puede obtener en RMA. Por el contrario, sí es cierto que una tarea producirá, como máximo, una sola expulsión a alguna de las tareas de menor prioridad. Cuando una tarea entra en ejecución, puede expulsar a una tarea de menor prioridad que esté ejecutándose, o no expulsará a ninguna, ni en ese momento ni en ningún otro, ya que las tareas de menor prioridad no podrán entrar a ejecución. Esta forma de operar aumenta artificialmente el tiempo de ejecución y de respuesta de las tareas al contabilizar el tiempo de recarga a la tarea expulsante en lugar de a la expulsada. Sin embargo, lo que se hace es considerar que se necesita un tiempo añadido de procesador debido a la interferencia extrínseca de la *cache*, y puesto que RMA no considera tiempos de respuesta, sino utilización total del sistema, no es relevante a quién se le adjudique ese tiempo añadido, sino que se adjudique para obtener una cota superior de la utilización global del sistema.

El principal inconveniente de CRMA se hereda de RMA, ya que este análisis considera que un sistema es planificable si su utilización se encuentra por debajo del 69%. Este valor es muy conservador y pesimista, ya que en muchos casos, sistemas con una utilización mayor siguen siendo planificables. En [Lehoczky89] se estima que los sistemas con política de planificación RM tienen una utilización planificable máxima de alrededor del 88% cuando el número de tareas es elevado.

Otros inconvenientes de CRMA se deben a su independencia respecto del secuenciamiento de las tareas. Por un lado, es imposible determinar el número exacto de expulsiones, por lo que se considera el peor caso que puede ser muy superior al real. Por la misma razón, es imposible identificar el valor real de la interferencia extrínseca, por lo que se considera el peor caso, es decir, el rellenado completo de la *cache* tras cada expulsión. La suma de estas dos aproximaciones conservadoras y pesimistas puede producir una utilización estimada muy

superior a la real. La principal ventaja de este análisis es su sencillez, junto con la importante cantidad de trabajo relacionado con RMA para considerar características particulares de los sistemas de tiempo real, como cargas aperiódicas [Leoczky87] o sincronización entre tareas [Sha90b], haciéndolo muy atractivo para los diseñadores de sistemas de tiempo real.

Con el objetivo de resolver las deficiencias presentadas por CRMA, en [Busquets95b] se presenta un análisis de planificabilidad basando en RTA (Response Time Analysis), realizando las modificaciones necesarias para incorporar el efecto de la *cache*. En [Busquets96a] y [Busquets96b] se presenta la ecuación del CRTA (*Cache*d RTA):

$$w_i^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^n}{T_j} \right\rceil x(C_j + \gamma_j)$$

donde C_i es el WCET de la tarea τ_i considerando la interferencia intrínseca, y γ_j es el valor de la interferencia extrínseca que la tarea τ_i sufre debido a la expulsión provocada por la ejecución de la tarea τ_j -*cache refill penalty* o *cache related preemption delay*-. En esta ecuación, el efecto de la interferencia extrínseca se incorpora a la tarea que debe recargar los contenidos de la *cache*, y el valor de esta interferencia puede calcularse de forma independiente para cada pareja de tareas expulsada-expulsante, permitiendo obtener valores muy ajustados de este factor. En [Busquets96b] se proponen dos métodos para calcular el valor de γ_j. El primero de ellos consiste en considerar el rellenado completo de *cache*, independientemente de cual sea la tarea expulsada y cual la tarea expulsante. El segundo método es considerar que la tarea expulsada deberá recargar sólo las líneas de *cache* que la tarea expulsante haya utilizado. En ambos casos, el valor obtenido para la interferencia extrínseca es siempre una cota superior, y aunque es posible que sea un valor pesimista, en [Busquets97b] se utiliza el primero de los métodos y se obtienen mejores resultados -mayor utilización planificable- que los proporcionados por CRMA.

Mejorar el cálculo del coste de la interferencia extrínseca de la *cache*, utilizando CRTA, es el trabajo realizado y presentado en [Lee96], [Lee97a], [Lee97b] y [Lee01]. La expresión de CRTA se modifica con el objetivo de obtener las líneas de *cache* que la tarea expulsada debe recargar por la ejecución de tareas de mayor prioridad. La expresión queda como sigue:

$$R_i^{k+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i^k}{T_j} \right\rceil C_j + PC(R_i^k)$$

donde $PC(R^k_i)$ es el tiempo que necesita la tarea τ_i para recargar todas las líneas que le son expulsadas por la ejecución de otras tareas de mayor prioridad. Esta fórmula, al igual que la anterior, se resuelve de forma iterativa, recalculando además el valor de $PC(R^k_i)$ en cada iteración. El problema es el cálculo de este valor, que se divide en dos pasos: obtención del coste de recarga de la *cache* para cada tarea; y obtención del peor caso de expulsiones para cada tarea.

El número de líneas que una tarea deberá recargar después de una expulsión dependerá del número de líneas útiles de la *cache* que la tarea tenga en el momento de la expulsión, es decir, el número de líneas cargadas en *cache* que la tarea volverá a utilizar al reanudar la ejecución tras la expulsión. Para calcular estas líneas se utiliza un método muy similar al presentado en [Lim94a] para el cálculo del WCET de una tarea en sistemas con memoria *cache*. Para cada posible punto de expulsión, se obtiene el número de líneas de *cache* cuyo contenido será nuevamente referenciado con posterioridad, y que por lo tanto, tras una expulsión, será necesario que la tarea vuelva a cargar. Finalmente se obtiene una lista de coste de expulsiones, donde aparecerá una entrada por cada punto posible de expulsión, y en aquellos puntos donde la tarea se ejecute más de una vez (bucles) se añadirán tantas entradas como veces se ejecute ese punto. Puesto que no es posible saber los puntos en que la tarea realmente sufrirá las expulsiones, y por lo tanto determinar el peor caso, esta lista se ordena de mayor a menor, utilizando estos valores en este orden y garantizando así la obtención de un límite superior. El

proceso se repite para todas las tareas, obteniendo una tabla de costes de expulsión como la mostrada en la Tabla 6, donde fa,b representa el coste de recargar la *cache* para la tarea τ_a después la expulsión número b.

Nº de expulsión **Tarea**	**1**	**2**	**3**	**....**	**k**
T1	f1,1	f1,2	f1,3	....	f1,k
T2	f2,1	f2,2	f2,3	....	f2,k
T3	f3,1	f3,2	f3,3	....	f3,k
:	:	:	:	....	:
Tn	fn,1	fn,2	fn,3	....	fn,k

Tabla 6. Tabla de costes de expulsión.

No todas las tareas tendrán el mismo número de expulsiones, por lo que para aquellas con menos de k expulsiones, su coste será cero, y para aquellas con más de k, este coste será menor o igual que el coste de la expulsión k-esima, por lo que utilizar este último coste proporcionará un límite superior.

El cálculo del coste sufrido por una tarea debido a una expulsión en una determinada ventana se obtiene con la siguiente fórmula:

$$PC_i(R_i^k) = \sum_{j=2}^{i} g_j f_j$$

donde g_j es el número de invocaciones de la tarea τ_j que son expulsadas durante una ventana R_i^k, y f_j es el coste de recarga de la *cache* para la tarea τ_j tras cada expulsión. El máximo valor posible de esta función es un límite superior del coste total de la tarea τ_i para recargar las líneas de *cache* desplazadas por las tareas de mayor prioridad. Este valor máximo se obtiene maximizando la función mediante programación lineal, utilizando dos funciones de restricción que limitan el número máximo de expulsiones que puede sufrir una tarea, y que son función de la duración de la ventana de ejecución de la tarea y del periodo de las tareas de mayor prioridad.

Dos mejoras son añadidas a este método en los últimos trabajos. La primera consiste en aumentar la tabla de coste, considerando no sólo las líneas útiles de la tarea bajo estudio, sino la intersección de estas líneas con las de las demás tareas de mayor prioridad. De esta forma, una tarea expulsada no deberá recargar todas sus líneas útiles, sino sólo aquellas que hayan sido desplazadas por otra tarea. Esto obliga a reformular ligeramente la función de coste. La segunda mejora consiste en limitar el número máximo de expulsiones que puede sufrir una tarea introduciendo nuevas restricciones al problema de programación lineal. Los resultados obtenidos indican un cálculo bastante ajustado, obteniendo resultados en pocos minutos.

Al obtener la intersección de las líneas de *cache* útiles de las tareas, se obtiene teóricamente el cálculo más exacto de la interferencia entre tareas. Sin embargo, la imposibilidad de determinar con exactitud cuantas veces, en qué momento y qué tareas producen y sufren las expulsiones, produce un resultado conservador de la interferencia. Otro problema de este método es la cantidad de pasos que deben realizarse, y la complejidad de los mismos.

Una alternativa a la aproximación separada a las dos fuentes de interferencia generadas por la memoria *cache* es la realización de un análisis combinado Todos los métodos de análisis estático comentados hasta el momento presentan dos inconvenientes, independientemente de si consideran la interferencia intrínseca o extrínseca. El primero de ellos es la complejidad de la técnica para conseguir resultados ajustados y no excesivamente conservadores. El segundo inconveniente, y quizá más importante, es que ninguno de los métodos resuelve

simultáneamente los dos problemas que presenta el uso de memorias *cache* en sistemas de tiempo real, es decir, existe un tratamiento independiente del análisis de la interferencia intrínseca respecto del análisis de la interferencia extrínseca, y viceversa.

Esta análisis separado del comportamiento de la *cache* presenta, como inconveniente inmediato, la necesidad de utilizar dos herramientas distintas, complicando el trabajo del diseñador de sistemas de tiempo real. Pero además, ésta falta de conexión en el análisis de las dos vertientes puede provocar una sobreestimación en el cálculo de la interferencia extrínseca. Intuitivamente puede verse que si el tiempo de ejecución de una instrucción es considerado, durante el cálculo del WCET, como el tiempo de ejecutarla desde memoria principal ya que no es posible garantizar que se encontrará en *cache*, su eliminación en *cache* tras una expulsión y su posterior recarga no incrementará el tiempo de ejecución de la tarea, ya que el fallo al acceder a dicha instrucción, independientemente de la razón de dicho fallo, ya ha sido considerado en el WCET de la tarea, por lo que no debe volver a contabilizarse, como interferencia extrínseca, al realizar el análisis de planificabilidad.

El problema surge porque ambas interferencias se estudian con el mismo criterio conservador y pesimista. Pero ese mismo criterio posee significados opuestos en función de si se analiza la interferencia intrínseca o la interferencia extrínseca. Al analizar la interferencia intrínseca, si una referencia a memoria no puede garantizarse que se encontrará en *cache* en todas sus ejecuciones, se considerará que producirá fallo, aumentando el WCET de la tarea en el tiempo correspondiente. Al analizar la interferencia extrínseca, si la misma referencia no puede garantizarse que siempre producirá fallo, se considera que tras una expulsión deberá recargarse, por lo que el tiempo de ejecución de la tarea se verá incrementado en el tiempo correspondiente, contabilizándose, para el mismo acceso a memoria, dos veces el tiempo de fallo. Un ejemplo sencillo de esa situación puede verse en la Figura 7, en la que se muestra el grafo de control de flujo de una porción de una tarea. Durante el cálculo del WCET, se considerará que la ejecución del código perteneciente a los vértices V2 y V3, que se ubican en la misma línea de *cache C1*, producirá siempre fallo ya que la tarea puede ejecutar alternativamente ambas rutas, siendo este el peor caso. Durante el análisis de planificabilidad, se considerará que los contenidos de la línea de *cache C1* son útiles, ya que es posible que tras una expulsión, la tarea vuelva a ejecutar la misma ruta, produciéndose un fallo por el reemplazo de sus contenidos, y siendo esta situación el peor caso. Cualquier otra consideración, realizando los dos análisis de forma independiente, podría suponer la obtención de valores inferiores a los reales. De este modo, el fallo en el acceso a los contenidos de la línea de *cache C1* se ha contabilizado dos veces.

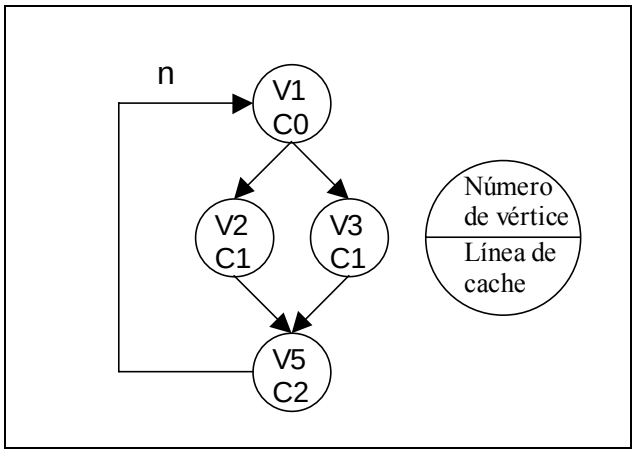


Figura 7. Grafo de control de flujo. Cada nodo representa una sección de código y la línea de cache en que se ubica.

Con el objetivo de eliminar esta duplicidad al estimar el WCET y el tiempo de respuesta de las tareas, en [Martí00] se presentan dos análisis, uno para el cálculo del WCET y otro para el análisis de planificabilidad, que comparten información sobre los bloques de memoria que se

encontrarán en *cache* en el momento de su ejecución. El trabajo se aplica a memorias *cache* de instrucciones con función de correspondencia directa.

El cálculo del WCET se realiza de forma similar al método de clasificación de instrucciones presentado en [Mueller00], con la diferencia de que son los bloques de memoria principal a los que la tarea accede los que se clasifican, en lugar de las instrucciones individuales. Estos bloques puede clasificarse en *único*, si es el único bloque que se corresponde con una línea concreta de *cache*; *compartido*, si existen dos o más bloques que compiten por la misma línea de *cache* pero su ejecución se produce de forma secuencial y sin iteraciones, es decir, una vez un bloque ha sido reemplazado por su competidor, no vuelve a ser ejecutado; y *conflicto*, cuando dos o más bloques compiten por la misma línea de *cache*, y pueden reemplazarse entre ellos en sus múltiples ejecuciones. Cada bloque recibe tantas clasificaciones como niveles de bucles a los que pertenece. Un vértice pertenece al bucle en el que se ejecuta, y a todos los bucles más exteriores que contienen el bucle en que se encuentra el vértice. Para realizar esta clasificación de forma rápida y con un coste computacional aceptable, se realiza una ampliación al grafo de control de la tarea. En este nuevo grafo, cada vértice representa una secuencia de código que se ejecuta sin ruptura de flujo y todas sus instrucciones se ubican en la misma línea de *cache*, de forma similar a como se definían los *l-bloques* en [Li96]. Además, se añade una nueva restricción, y es que un vértice puede aparecer únicamente una vez en todo el nuevo grafo de control de flujo, siendo esta restricción la que permite realizar la clasificación de forma rápida y sencilla. Con esta última restricción, la distinción entre vértices de tipo conflicto y vértices de tipo compartido puede realizarse simplemente verificando si ambos vértices pertenecen al mismo bucle. Si dos vértices pertenecen al mismo bucle, quiere decir que tras la ejecución de uno de ellos puede realizarse la ejecución del otro, y nuevamente la ejecución del primero, por lo que si ambos compiten por la misma línea de *cache*, existe la posibilidad de que se interfieran, por lo que serán clasificados como conflicto.

En función de la clasificación que recibe cada vértice en cada nivel de anidamiento, se le asigna un tiempo de ejecución constante, por lo que el cálculo del WCET puede realizarse de forma sencilla utilizando un análisis de la peor ruta (WPET). En [Martí00] se definen las diferentes combinaciones de clasificaciones que puede recibir un vértice que pertenece a varios niveles de bucles y las ecuaciones correspondientes para calcular su tiempo de ejecución.

El análisis de planificabilidad se realiza utilizando la expresión original del CRTA [Busquets96a], obteniendo el valor de la interferencia extrínseca de la clasificación realizada para el cálculo del WCET. Este valor corresponde al tiempo en que una tarea incrementará su tiempo de ejecución, respecto del estimado para el WCET, debido a la recarga de un número concreto de líneas de *cache* que deberá realizar tras cada expulsión. El número de líneas que una tarea deberá recargar se obtiene directamente de la clasificación de los vértices que se ha realizado para la obtención del WCET. Este número corresponde al número de líneas que para el WCET se espera encontrar en *cache* durante la ejecución, y que pueden producir un fallo no esperado debido a la expulsión. Sólo aquellas líneas para las que todos los vértices que se ubican en ellas han sido clasificados como *único* o *compartido*, y se encuentran al menos en un bucle, pueden producir un fallo de *cache* que no ha sido contabilizado anteriormente en el WCET.

Este análisis obtiene resultados muy ajustados en la estimación del valor de la interferencia extrínseca, pero la última restricción impuesta en el diseño de grafo de control -la aparición una sola vez de cada vértice- complica e incluso puede impedir el modelado de estructuras complejas, e introduce una falta de exactitud en la estimación del WCET, lo que supone una cota excesivamente conservadora de los tiempos de respuesta de las tareas.

1.2.2 Aumento del determinismo de la memoria cache

El principal objetivo de las soluciones presentadas en este apartado es la eliminación, o al menos reducción, de la falta de determinismo introducida por la memoria *cache*, permitiendo la utilización de técnicas de análisis estático exactas y, sobre todo, sencillas. Las soluciones

propuestas utilizan tanto mecanismos *hardware* como *software* para eliminar la interferencia producida por la *cache*.

Una de las técnicas más estudiadas y desarrolladas es la partición *hardware* de *cache*. La propuesta más evolucionada en este sentido es la realizada en [Kirk89] e implementada en [Kirk90]. Esta técnica, que recibe el nombre de SMART (*Strategic Memory Allocation for Real-Time*), se basa en la división de la *cache* en bloques o particiones, restringiendo el acceso que las tareas hacen a dichas particiones. La Figura 8 ilustra la arquitectura de SMART.

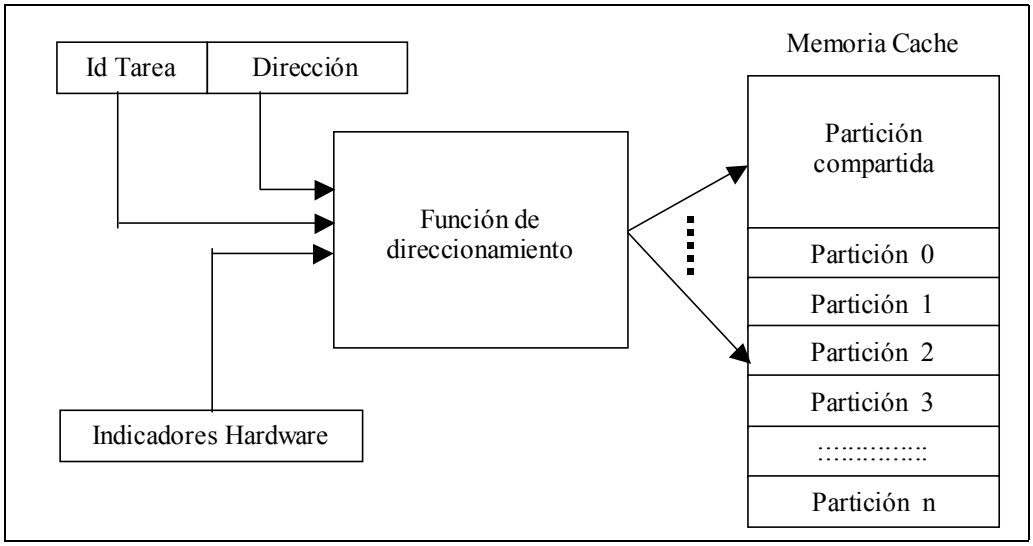


Figura 8. Diagrama de bloques de la arquitectura SMART.

La *cache* se divide en *N* particiones. A cada tarea se le asigna de forma privativa un subconjunto de estas particiones, de forma que sólo la tarea propietaria podrá acceder y modificar los contenidos de dichas particiones. Además se reserva una partición para uso general, a la que cualquier tarea puede acceder. Cada vez que una tarea genera una dirección, un circuito dedicado a tal efecto identifica la partición de *cache* a la que debe acceder para resolver la petición. Este circuito se sitúa entre el procesador y la memoria *cache*, filtrando todas las referencias generadas por el procesador. Mediante la dirección física solicitada por el procesador, el identificador de tarea, las tablas de asignación de particiones y un bit que indica si la tarea accede a sus particiones privadas o a la partición compartida, el circuito accede a la partición o particiones correspondientes, impidiendo el acceso a particiones pertenecientes a otras tareas.

La asignación de particiones a cada tarea se resuelve de forma estática, considerando parámetros como la prioridad de cada tarea, el contenido semántico o la aportación de cada tarea a la planificabilidad del sistema, existiendo diferentes algoritmos para la obtención de la asignación óptima de particiones [Kirk91a] [Kirk91b] [Sasinowski93]. Aunque estos métodos permiten asignar un número arbitrario de particiones a cada tarea, desde todas las particiones –excepto la compartida- hasta ninguna, habitualmente todas las tareas críticas reciben al menos una partición en propiedad. De esta forma se reduce la interferencia extrínseca, puesto que las particiones privadas sólo pueden ser accedidas por la tarea propietaria, y de este modo las expulsiones provocadas por la ejecución de tareas de mayor prioridad sólo reemplazarán aquéllos contenidos que se encuentren en la partición común.

Una ventaja importante de SMART es la independencia de la función de correspondencia de la memoria *cache* o del tipo de contenidos -datos, instrucciones o mixtos- que se almacenen en *cache*.

Este método presenta principalmente dos inconvenientes. El primero es la necesidad de incorporar circuitos entre el procesador y la memoria *cache*, lo que en procesadores actuales que disponen de *cache* en el mismo *chip* es bastante complejo sino se ha incorporado en el diseño

original. Además, el tiempo necesario para filtrar las direcciones generadas por el procesador y determinar la partición a la que debe accederse puede incrementar el tiempo de acceso a *cache*, empeorando las prestaciones del procesador. El segundo inconveniente es que SMART no garantiza la eliminación de la interferencia extrínseca debido a la utilización de la partición común, aunque esta interferencia puede verse reducida notablemente, sigue siendo necesario realizar un análisis complejo, como los comentados en el apartado anterior, para incorporar al análisis de planificabilidad la utilización de memorias *cache*.

En [Busquets97] se presenta la partición híbrida de *cache*, que mejora los resultados obtenidos por SMART al aumentar la flexibilidad en el número de particiones en que se divide la *cache*, pudiendo ser este menor que el número de tareas, y aplicando CRTA al nuevo escenario. Además, se proponen una batería de algoritmos que permiten obtener el número y asignación óptima de particiones a las diferentes tareas, permitiendo, además, que existan múltiples particiones compartidas entre las tareas. Aunque esta propuesta mejora significativamente a SMART sigue presentando los mismos problemas que éste.

La partición *software* surge como alternativa a la partición *hardware*, suprimiendo la necesidad de incluir *hardware* entre el procesador y la *cache*, eliminando por tanto el retardo que el filtro de direcciones de SMART incluía en los accesos a *cache* y permitiendo su utilización en procesadores comerciales en cuyo diseño no han sido incluidos mecanismos de partición *hardware*.

Básicamente se han presentado dos propuestas. La primera de ellas [Mueller95] [Tomiyama97] utiliza la reubicación del código y los datos para eliminar los conflictos que existan al ubicarse en *cache*. Para ello, es la tarea la que se divide en bloques, cada bloque con el mismo tamaño que el que se ha decidido asignar a cada partición. Una vez la tarea ha sido dividida en bloques, estos bloques se separan, desgajando la tarea y distribuyéndola por la memoria principal. Esta distribución es el punto clave de este método, ya que las direcciones en que se reubiquen los bloques de las tareas determinarán las líneas o conjuntos de *cache* donde estos se cargarán. Si se desea que dos bloques de código o datos no compitan por la misma línea de *cache*, deberán colocarse en direcciones físicas de memoria principal de tal modo que la función de correspondencia de la *cache* las ubique en líneas distintas. Con esta reorganización de las tareas pueden conseguirse los mismos resultados que con SMART o partición híbrida, ya que puede configurarse el número de particiones en que se dividirá la *cache* así como el tamaño de las mismas, el número de bloques que competirán por una misma partición, y si estos bloques pertenecerán a la misma tarea o a tareas distintas, creando las particiones compartidas necesarias o convenientes. Para conseguir todas estas configuraciones es necesario considerar las tareas en conjunto, teniendo en cuenta la ubicación de los bloques de todas las tareas a la hora de dividir y reubicar una tarea determinada, lo que puede limitar la aplicación de este método a los escenarios más básicos.

El proceso de división y reubicación de la tarea, tanto para el código como para los datos, se puede realizar en tiempo de compilación. Para el código, los bloques separados se unen mediante la inserción de instrucciones de salto. Sin embargo, la inclusión de estas instrucciones de salto puede incrementar considerablemente el tiempo de ejecución de la tarea. Para el acceso a estructuras de datos, los índices y las funciones de acceso se modifican de acuerdo a las divisiones creadas. Es posible que algunas estructuras de datos sean extremadamente complejas de manejar, y sea necesario un estudio manual para conseguir adaptar los accesos que la tarea realiza a la nueva organización de los datos [Temam94]. Estas transformaciones, que en muchos casos no son triviales, son el principal inconveniente de esta propuesta.

Para eliminar las transformaciones que debe realizar el compilador en la partición mediante la reubicación del código y los datos, en [Romer94] [Liedtke97] [Wolfe93] [Li93] y [Bershad94] se propone la utilización del sistema de memoria virtual con memorias *cache* de direcciones físicas. Aunque realmente se trata de partición *software*, la circuitería de traducción de direcciones que se exige en la partición *hardware* es sustituida por la circuitería utilizada para la traducción de direcciones virtuales a direcciones físicas del sistema de memoria virtual del

computador, por lo que podría considerarse esta propuesta como un método de partición *hardware*. La principal diferencia se encuentra en quién es el encargado de realizar el redireccionamiento a las particiones de *cache*. Mientras que en la partición *hardware* es el circuito el responsable completo de gestionar los accesos, en esta propuesta es el sistema operativo el encargado de asignar las particiones. A cada tarea se le asigna el espacio disponible de memoria virtual, de igual modo que se haría en un sistema de propósito general. Pero a diferencia de estos, la correspondencia entre páginas virtuales y páginas físicas no persigue el objetivo de reducir el tránsito entre memoria secundaria y memoria principal, sino en lograr que determinadas páginas virtuales se almacenen en páginas físicas concretas, correspondiéndose, por tanto, con determinadas líneas de *cache*. El resultado es el mismo que en la reorganización de código descrita anteriormente, lograr que las direcciones físicas de las diferentes tareas se ubiquen en bloques de memoria principal que no compitan por las mismas líneas de *cache*.

La utilización de la memoria virtual ofrece las mismas posibilidades que el resto de técnicas de partición, pero presentan mayores ventajas ya que no es necesario modificar el código, y la circuitería necesaria se encuentra incorporada en la mayoría de los procesadores comerciales, por lo que esta propuesta es, entre todas, la que mayores facilidades de uso presenta. Su principal inconveniente es la falta de flexibilidad a la hora de diseñar las particiones de *cache*, ya que estas deben tener un tamaño igual al tamaño de la página de memoria virtual, lo que puede presentar, en muchos casos, problemas para realizar una asignación óptima de particiones a las tareas del sistema de tiempo real. Otro problema, común a todos los métodos de partición, es que no existe garantía de que la interferencia extrínseca sea completamente eliminada, ya que no siempre es posible asignar una partición privada a cada tarea, por lo que sigue siendo necesario realizar algún tipo de análisis como CRMA o CRTA. Y por supuesto, nunca o casi nunca se elimina la interferencia intrínseca, siendo obligatoria la utilización de alguna de las técnicas descritas anteriormente para el cálculo del WCET de las tareas.

1.3 Conclusiones del capítulo

El diseño de sistemas de tiempo real estricto presenta un problema añadido frente al diseño de sistemas informáticos de propósito general. Este problema añadido es la necesidad de garantizar que el sistema es planificable, es decir, que las tareas críticas obtendrán sus resultados y finalizarán su ejecución antes de un plazo determinado.

La consecución del análisis de planificabilidad depende tanto de la arquitectura *software* como de la arquitectura *hardware* del sistema. Las características *software* del sistema, como son el algoritmo de planificación utilizado, la asignación de prioridades a las tareas o las relaciones entre sus características temporales determinarán qué método de análisis puede utilizarse, y la complejidad del mismo. Especial importancia cobra la política de planificación utilizada. De la descripción de las mismas realizada en este capítulo y de los métodos de análisis de planificabilidad asociados a cada política se puede deducir que cuanta mayor flexibilidad y mayores prestaciones presenta el algoritmo de planificación, más complejo es el análisis de planificabilidad asociado. Así, mientras el análisis para sistemas con planificadores cíclicos es trivial frente a la complejidad y las restricciones de dicha política de planificación, el análisis de planificabilidad de un sistema de prioridades dinámicas y expulsiones es extremadamente complejo, frente a la relativa sencillez y sobre todo flexibilidad de su diseño.

Por otro lado, las características de la arquitectura *hardware* del sistema, como la utilización de memorias *cache* o procesadores segmentados, interactúa, modificando y particularizando para cada implementación concreta, los análisis de planificabilidad diseñados para cada política. Del mismo modo se observa que la utilización de técnicas actuales para el aumento de prestaciones basadas en la modificación de la estructura del procesador, aumenta de forma considerable la complejidad del análisis de planificabilidad necesario para garantizar la corrección temporal del sistema. Este es el caso de las memorias *cache*. El elevadísimo aumento de prestaciones que proporciona la inclusión de estas memorias en la jerarquía de memoria del computador, su ampliamente extendido uso, y la complejidad del modelado de su comportamiento, han propiciado la aparición en años recientes de un gran número de métodos, algoritmos y técnicas

para incorporar sus efectos al análisis de planificabilidad. Estos efectos, resultantes de la falta de determinismo en el comportamiento de la memoria *cache*, pueden dividirse en dos: interferencia intrínseca o *intra-task*, e interferencia extrínseca o *inter-task*, cada uno de ellos con problemáticas y soluciones diferenciadas.

Dos grandes líneas se han desarrollado para considerar de forma segura y ajustada el aumento de prestaciones producido por las memorias *cache*. La primera línea, basada en el modelado del comportamiento de la memoria *cache* y en la creación de nuevos métodos de análisis. La segunda línea, orientada a redefinir el comportamiento de la *cache*, permitiendo el uso de análisis independientes del *hardware* o mediante modelos extremadamente sencillos.

Los diferentes trabajos realizados para modelar y analizar el comportamiento temporal de la *cache* presentan una ventaja importante: son capaces de modelar implementaciones comerciales, lo que permite al diseñador de sistemas de tiempo real utilizar *hardware* de gran tirada, con lo que ello conlleva de fiabilidad, precio, prestaciones y existencia de herramientas para el desarrollo de sistemas. Sin embargo, adolecen de varios inconvenientes que pueden desanimar e incluso impedir su utilización: complejidad de los algoritmos, necesidad de utilizar dos métodos distintos, uno para el análisis de la interferencia intra-task y otro para la interferencia *inter-task*, la necesidad de considerar explícitamente todas las particularidades de la memoria *cache* obligando a modificar o rediseñar el algoritmo de análisis, y la imposibilidad de obtener resultados ajustados cuando partes del análisis se simplifica o la estructura *software* del sistema es compleja.

En el caso de las soluciones orientadas a aumentar el determinismo de la *cache*, su principal ventaja es la eliminación de la falta de determinismo en el comportamiento de la *cache*, lo que permite utilizar técnicas de análisis sencillas que no necesitan considerar explícitamente la existencia de memoria *cache*. Sin embargo, ninguno de los métodos garantiza la obtención de un sistema totalmente determinista, por lo que sigue siendo necesaria la utilización de complejas técnicas de análisis. En concreto, la mayoría de los métodos de partición están orientados hacia la reducción de la interferencia extrínseca, por lo que el cálculo del WCET de las tareas deberá realizarse con algún tipo de modelo y análisis más o menos complejo. Además, las técnicas incluidas en este grupo presentan otras desventajas, como la necesidad de realizar transformaciones en el *hardware* o en el *software*, que en algunos casos no son triviales, o la perdida de prestaciones que significa particionar la *cache*, ya que cada tarea dispone de un espacio de *cache* menor, aumentando el número de fallos por capacidad y conflicto, y por tanto empeorando los tiempos de ejecución.

Aunque la utilización de memorias *cache* en sistemas de tiempo real estricto parece ser un problema resuelto, no existe una solución sencilla que al mismo tiempo permita aprovechar al máximo las prestaciones proporcionadas por esta mejora estructural. El diseñador del sistema deberá, por tanto, seleccionar aquella técnica que le sea más cómoda, accesible o bien practicable, en función de las características particulares de su sistema y sus objetivos, asumiendo los inconvenientes y limitaciones de su elección.

Capítulo 2

Uso estático de memorias cache con bloqueo

En este capítulo se propone un método nuevo y completo para realizar el análisis de planificabilidad de un sistema de tiempo real estricto, considerando de forma conjunta los efectos producidos por la memoria *cache*, es decir, tanto la interferencia intrínseca como la extrínseca. El análisis se realiza para sistemas con planificadores con expulsión, tanto con prioridades fijas como dinámicas. Todo el desarrollo realizado se sustenta en el aprovechamiento de una característica disponible en la memoria *cache* de algunos procesadores de propósito general. Esta característica consiste en la posibilidad de seleccionar una parte de los contenidos de la memoria *cache* y bloquearlos, impidiendo que sean reemplazados por otros bloques de memoria principal que se copiarían en las mismas líneas de *cache*. De esta forma se obtiene determinismo en el comportamiento de la memoria *cache*, permitiendo el desarrollo de análisis muy sencillos, rápidos y practicables tanto para el cálculo del WCET de una tarea como para la realización del análisis de planificabilidad.

La utilización de estas memorias *cache* con bloqueo (*locking caches*) recibe el nombre de estático porque los contenidos se precargan antes de lanzar a ejecución el sistema, y permanecen inalterables durante todo el funcionamiento del mismo, pudiéndose decir que su contenido es estático.

Debido a que los accesos a memoria para la búsqueda de instrucciones representan aproximadamente un 75% sobre el total de las referencias generadas por el procesador [Hennessy96], el trabajo realizado considera únicamente el diseño y análisis de la *cache* de instrucciones.

Parte del trabajo descrito en este capítulo ha sido publicado en [Martí01a] [Martí01b] [Martí01c] y [Martí03a]

2.1 Memorias cache con bloqueo (locking cache)

Una porción de memoria cache se dice que está bloqueada (locked) si el código o los datos cargados en dicha porción no podrán ser reemplazados por otras instrucciones o datos. [Idt00]

Aunque la funcionalidad y el objetivo de una *locking cache* y una *scratch pad memory* [Panda97] es similar, existen varias diferencias importantes entre ellas. Ambas se implementan con tecnología más rápida que la utilizada para la memoria principal, y ambas permiten acelerar el acceso a determinadas instrucciones y datos manteniéndolos de forma constante en su espacio de direcciones. Y es en este espacio de direcciones donde reside la principal diferencia entre ambas memorias, ya que se ubican en niveles distintos de la jerarquía de memoria. Una *scratch pad memory* forma parte de la memoria principal, encontrándose en el mismo nivel de jerarquía, y representando, realmente, la utilización de una tecnología diferente para una pequeña parte del mismo mapa de memoria. La *locking cache* es una memoria *cache* con un comportamiento particular, por lo que ocupa un nivel distinto en la jerarquía de memoria, utilizando no sólo una tecnología distinta para su construcción, sino representando un mapa de memoria distinto, o más concretamente, representando un duplicado del mismo mapa de memoria. Esta diferencia entre la ubicación de ambas memorias es fundamental en el modo en que pueden utilizarse. Si se desea utilizar una scratch pad memory, es necesario modificar el código o la organización de los datos para que éstos se ubiquen en las direcciones del mapa de memoria principal que corresponden a *la scratch pad memory*. Sin embargo, la *locking cache*, como memoria *cache* que es, permite almacenar cualquier posición de memoria principal en su espacio, sin necesidad de modificar el código o las estructuras de datos, ya que es su propia circuitería la encargada de copiar la información y reubicarla, gestionando al mismo tiempo la traducción de direcciones de memoria principal a direcciones de línea de *cache*. Esta automatización en el posicionamiento

de la información y en la traducción de las direcciones hace que la utilización de *locking cache* sea completamente transparente a las tareas, del mismo modo que lo es la memoria *cache* convencional. Esta es la principal razón por la que se propone la utilización de memorias *cache* con bloqueo para sistemas de tiempo real, ya que no será necesario modificar el código de las tareas ni ninguna característica *software* del sistema.

Son múltiples los procesadores comerciales que incorporan en su memoria *cache* la posibilidad de bloquear los contenidos, evitando que una vez han sido cargados y bloqueados éstos sean reemplazados por la nuevas direcciones que genera el procesador. Algunos de estos procesadores son el *Intel 960* [Intel89], el *CIRYX MII* [Cyrix99] de la familia del *80x86*, el *Motorola MPC7400* [Motorola99], y el *Integrated Device Technology (IDT) 79RC64574* [Idt00], por nombrar algunos de ellos. De todos ellos, los más interesantes por las características y prestaciones que ofrecen son el *MPC7400* y el *IDT79RC64574*.

El *IDT79RC64574* incluye una *cache* de instrucciones con función de correspondencia asociativa de dos conjuntos, con un tamaño total de 8 Kbytes. Mediante la ejecución del código adecuado, el primer conjunto de la *cache* puede ser bloqueado, por lo que a partir de ese momento no se cargarán nuevas instrucciones ni se modificarán las instrucciones ya almacenadas en dicho conjunto. Si el procesador referencia una instrucción que se encuentra almacenada en el primer o segundo conjunto, indistintamente, ésta instrucción será servida desde la *cache* con el tiempo correspondiente a un acierto en *cache*. Si el procesador referencia una instrucción que no se encuentra en ninguno de los dos conjuntos, esta será servida desde memoria principal y el bloque a que pertenece será copiado en el segundo conjunto, cuyo comportamiento sigue siendo el de una memoria *cache* convencional. El funcionamiento global de la instrucción de bloqueo puede interpretarse como la creación de dos particiones, una privada y otra compartida, cada una de ellas con el mismo tamaño de 4Kbytes. Sin embargo, presenta dos diferencias muy significativas frente a las técnicas de partición comentadas en el capítulo anterior. En primer lugar, la partición privada no se asigna en exclusiva a una tarea, sino que sus contenidos pueden pertenecer a cualquier porción de código de cualquier tarea del sistema. Y en segundo lugar, las instrucciones no compiten por el espacio de la partición privada, sino que sus contenidos se mantienen estáticos y sin posibilidad de ser reemplazados. Además, estos contenidos pueden ser escogidos con una granularidad de línea, ya que este procesador permite seleccionar la dirección de instrucción y el correspondiente bloque de memoria principal que se cargará en la memoria *cache*. La única limitación que presenta esta instrucción es la propia de la función de correspondencia utilizada y la división en dos de la *cache*, lo que significa que el conjunto bloqueado se comporta como una *cache* de correspondencia directa, y por tanto no podrán cargarse y bloquearse aquellos bloques que compitan por la misma línea de *cache*.

El *Motorola MPC7400* incorpora una *cache* de instrucciones de 32 Kbytes y función de correspondencia asociativa de 8 conjuntos. Este procesador también permite bloquear los contenidos de *cache*, evitando que sean reemplazados. Pero a diferencia del *IDT*, la ejecución de esta instrucción bloquea toda la memoria *cache*, por lo que una vez ejecutada esta instrucción, no se realizará ninguna nueva transferencia desde memoria principal a memoria *cache*. Para mejorar el acceso a las instrucciones que no se encuentran cargadas y bloqueadas en *cache* y aprovechar la localidad espacial, este procesador incorpora explícitamente un buffer temporal, con el mismo tamaño que una línea de *cache*. De este modo, si la *cache* ha sido bloqueada y el procesador solicita una instrucción que se encuentra en *cache* o en el buffer temporal, dicha instrucción es servida en el mismo tiempo que lo haría una *cache* convencional. Si por el contrario la instrucción no se encuentra ni en *cache* ni en el buffer temporal, la instrucción es servida por la memoria principal, y el bloque al que pertenece es copiado en el buffer temporal, reemplazando el contenido anterior. El funcionamiento es similar al del *IDT*, creándose dos particiones, pero en este caso con tamaños bien distintos: una partición que podríamos llamar privada y que tiene el tamaño de toda la memoria *cache*, y una partición compartida con el tamaño de una línea de *cache*. Respecto a los contenidos que son bloqueados en *cache*, la granularidad es también de línea, pudiendo bloquearse simultáneamente cualquier parte del

código de cualquier tarea del sistema. Sin embargo, no existe ninguna instrucción que permita, de forma selectiva, cargar determinadas instrucciones en *cache*, por lo que cuando se bloquea la *cache* permanecerán en ella aquellas instrucciones que, siguiendo el comportamiento dinámico de la *cache* y la ejecución de las tareas, se encuentren en ese momento cargadas. Esta falta de selectividad limita seriamente en la práctica la utilización del bloqueo.

Aunque la similitud con el método de partición *hardware* es elevada, existen dos diferencias fundamentales: el número de particiones sólo se encuentra limitado por el número de líneas de *cache*, ya que en el *IDT* se puede seleccionar arbitrariamente el contenido de cada línea de *cache*, y el contenido de cada partición se mantiene invariable y sin modificación una vez la *cache* ha sido bloqueada. Estas dos diferencias son muy importantes, por lo que en este trabajo se ha preferido considerar el uso de *locking caches* como una propuesta completamente distinta y no como una ampliación o variación de la partición *hardware*.

El principal objetivo del uso estático de *locking cache* perseguido en este trabajo es la obtención de un sistema determinista y predecible, permitiendo así la utilización de algoritmos de análisis sencillos, bien conocidos y practicables. Este objetivo hace que el esquema de *cache* incorporado en el Motorola sea mucho más interesante, pues sólo es necesario modelar una *cache* de una sola línea –el buffer temporal- lo cual es relativamente sencillo, frente a la necesidad de modelar toda una *cache* de 4Kbytes con correspondencia directa en el caso del *IDT*. En el caso del procesador *IDT* sigue existiendo interferencia intrínseca y extrínseca, quedando el problema de la estimación del WCET y del análisis de planificabilidad sin resolver.

Por otro lado, los sistemas de tiempo real deben aprovechar el aumento de prestaciones que proporciona la memoria *cache*. Para obtener, o al menos intentar obtener las mismas prestaciones que una memoria *cache* convencional y no determinista, es necesario seleccionar con sumo cuidado las instrucciones que se cargarán y bloquearán en *cache*, y que deben ser aquellas que proporcionen las mejores prestaciones. La inexistencia de una instrucción de precarga de *cache* en el *Motorola* impide la utilización de este procesador de forma eficiente, ya que no es posible seleccionar los contenidos que se cargarán y mantendrán en *cache*.

Para alcanzar los dos objetivos y obtener un sistema determinista con las mejores prestaciones posibles, se propone una estructura de *locking cache* con la siguientes características:

- La memoria *cache* de instrucciones puede ser completamente bloqueada, es decir, no se producirá ninguna nueva carga ni se modificará su contenido una vez la *cache* haya sido bloqueada.
- El bloqueo de la *cache* se realizará mediante la ejecución de una instrucción específica.
- Los contenidos de la *cache* podrán seleccionarse con granularidad de línea, ejecutando una instrucción específica e indicando la dirección del bloque de memoria principal que se desea cargar.
- Existirá un buffer temporal, con tamaño igual al de una línea de *cache*, y que se comportará de igual forma que una memoria *cache* de instrucciones de una sola línea.
- Si el procesador referencia una instrucción que se encuentra en la *cache*, y ésta ha sido bloqueada, está instrucción se servirá desde dicha *cache* con el mismo tiempo que si la *cache* no estuviera bloqueada.
- Si el procesador referencia una instrucción que se encuentra en el buffer temporal, ésta será servida desde dicho buffer en el mismo tiempo que se hubiera servido desde memoria *cache*.
- Si el procesador referencia una instrucción que no se encuentra en memoria *cache* ni en el buffer temporal, esta instrucción será servida desde memoria principal, con el tiempo de acceso correspondiente. Si la memoria *cache* se encuentra bloqueada, sólo el buffer temporal será rellenado con el bloque de memoria principal correspondiente a la instrucción solicitada.

- La función de correspondencia de la *cache* puede ser cualquiera.

Las características anteriores son básicamente la unión de los dos esquemas de *locking cache* incluidos en los procesadores IDT y Motorola, y su diseño y construcción no presenta ninguna complejidad ni problema.

Una memoria *cache* con las características descritas anteriormente permite obtener determinismo, ya que al precargar sus líneas y bloquearla completamente, los contenidos son conocidos y es posible determinar las instrucciones que producirán acierto o fallo durante todo el funcionamiento del sistema. La existencia del buffer temporal permite mejorar el tiempo de acceso a aquellas instrucciones que no hayan sido cargadas en *cache*, ya que sólo la referencia a la primera instrucción del bloque producirá fallo, mientras que las siguientes instrucciones producirán acierto. La función de correspondencia utilizada para asignar las líneas no influirá en la obtención de determinismo, pero puede influir en las prestaciones obtenidas. La utilización de una *cache* completamente asociativa permite cargar en *cache* cualquier conjunto de instrucciones, mientras que una *cache* directa presentaría limitaciones, ya que bloques que compitan por la misma línea no podrían ser cargados y bloqueados en *cache* simultáneamente.

Durante el arranque del sistema (*start-up*), una pequeña rutina formada básicamente por un bucle, leería desde una tabla las direcciones de las instrucciones seleccionadas y las cargará en memoria. Una vez todas las instrucciones seleccionadas hayan sido precargadas bloqueará la *cache*. Las instrucciones precargadas pueden pertenecer a cualquier segmento de cualquier tarea del sistema, y pueden ser largas secuencias contiguas de código o bloques individuales. Para evitar que la rutina de precarga interfiera en la *cache* con los bloques que se están precargando, esta rutina puede situarse en un área de memoria principal cuyos contenidos no puedan ser copiados en memoria cache.

2.2 Cálculo del WCET

Aunque las memorias *cache* con bloqueo ofrecen determinismo en su comportamiento, es necesario realizar el cálculo del WCET de las tareas considerando la existencia de esta particularidad *hardware* ya que el tiempo de ejecución de instrucción dependerá de si ésta se encuentra o no cargada en *cache*. Este cálculo del WCET depende no sólo de la arquitectura *hardware* sobre la que se ejecuta la tarea, sino de la estructura de la propia tarea, que determinará el conjunto de instrucciones que la tarea ejecutará en el peor caso y por tanto el tiempo de ejecución en dicho peor caso.

En primer lugar, y partiendo del código máquina de la tarea, se construye su Grafo de Control de Flujo (CFG, *Control Flow Graph*) teniendo en cuenta la existencia de la *cache*. Este CFG extendido recibe el nombre de *Cached Control Flow Graph* (CCFG) y consiste en un grafo dirigido, formado por vértices y arcos. Este grafo se construye siguiendo las siguientes reglas:

- Los vértices del CCFG representan conjuntos de instrucciones que se ejecutan secuencialmente, es decir, sin ruptura de flujo.
- Todas las instrucciones contenidas en un vértice se ubican en la misma línea de *cache*.
- Los vértices se encuentran unidos por arcos, que representan las distintas secuencias en que pueden ejecutarse las instrucciones contenidas en los vértices.
- Sólo puede existir un arco que una dos vértices A y B, pero un vértice puede estar conectado con un número ilimitado de vértices, por lo que un vértice puede tener múltiples arcos de salida y de entrada.
- Un mismo vértice puede aparecer repetidas veces en el CCFG. Por ejemplo, en el caso de llamadas a funciones estas pueden representarse de forma desarrollada (*in-line*).

La Figura 9 muestra un ejemplo donde se construye el CCFG para una porción de código que incluye saltos condicionales e incondicionales, y donde las instrucciones se ubican en diferentes líneas de *cache*.

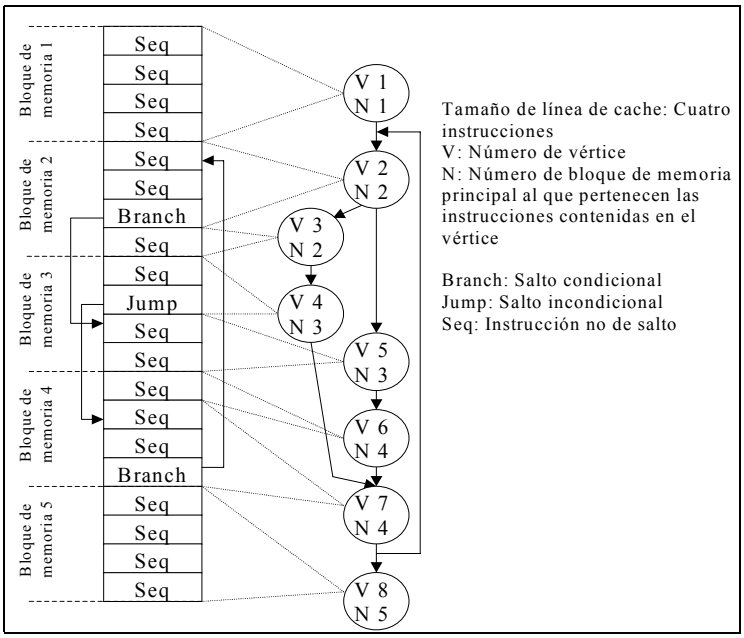


Figura 9. Ejemplo de construcción del CCFG.

La diferencia entre el CFG y el CCFG es que el primero modela únicamente las rutas que la tarea puede seguir durante su ejecución, mientras que el segundo añade a este modelo información sobre cómo estas rutas utilizan la *cache*. Esta información permitirá obtener los tiempos de ejecución de cada una de las posibles rutas considerando el efecto de la *cache* en la ejecución de los vértices.

El WCET de la tarea puede calcularse obteniendo el tiempo de ejecución de la peor ruta, es decir, el mayor de los tiempos de ejecución de todas las posibles rutas. Para calcular esta peor ruta, el CCFG puede representarse mediante una expresión aritmética, obtenida de la aplicación del análisis de tiempos [Shaw89] al CCFG. En esta expresión cada término representa el tiempo de ejecución de un vértice, y la expresión completa del CCFG se obtiene concatenando mediante diferentes operadores, en función del número de arcos que partan o lleguen a cada vértice, los tiempos de ejecución de cada vértice. Según la terminología habitual de grafos [Junghickel99], se define un vértice A como predecesor de un vértice B si existe un arco que parte de A y llega a B, y el vértice B se define como sucesor del vértice A. Las definiciones de predecesor y sucesor cumplen la propiedad transitiva, por lo que un vértice A se define como predecesor de un vértice C, si el vértice A es predecesor de un vértice B que a su vez es predecesor del vértice C. Es posible particularizar la definición de predecesor en función de si esta propiedad se adquiere de forma directa o si se adquiere a través de la propiedad transitiva. Si existe un arco que conecta el vértice A con el vértice B, diremos que A es un predecesor inmediato de B. Si por el contrario, A es predecesor de B, y B es predecesor de C, diremos que A es un predecesor no inmediato de C. Del mismo modo pueden definirse vértices sucesores inmediatos y no inmediatos. Es importante remarcar que un vértice puede ser predecesor de varios vértices al mismo tiempo, tanto de forma inmediata y no inmediata como combinando ambas posibilidades. Como ejemplo extremo, el primer vértice del CCFG es predecesor de todos los demás, ya que desde la primera instrucción de una tarea puede alcanzarse cualquier otra instrucción. Si esto no fuera cierto, estaríamos hablando de una porción de código que no es posible ejecutar.

En el CCFG pueden identificarse tres estructuras básicas, en función de la relación de precedencia y sucesión que tengan los vértices. Cada una de estas estructuras puede representarse mediante una expresión a partir de la cual obtener su tiempo de ejecución. La expresión completa del CCFG de la tarea puede obtenerse uniendo las expresiones de las distintas estructuras mediante el operador suma. Las tres expresiones básicas son:

- Estructura simple: está formada por un conjunto de vértices, cada uno de los cuales es predecesor inmediato de un único vértice. Es decir, de cada vértice sólo parte un arco, y a cada vértice sólo llega un arco. La expresión para esta estructura es la suma de los tiempos de ejecución de cada vértice.
- Estructura bifurcación-unión: está formada por un conjunto de vértices, el primero de los cuales es predecesor inmediato de al menos dos vértices. A su vez, sucesores inmediatos o no inmediatos de este vértice son predecesores inmediatos de un mismo vértice. Esta estructura se representa mediante el operador máximo, aplicándose a las expresiones que representan cada una de las distintas rutas alternativas formadas por los sucesores del primer vértice. Básicamente, esta estructura representa la sentencia compleja IF-THEN-ELSE de los lenguajes de alto nivel, cuyo peor tiempo de ejecución es el máximo de los tiempos de ejecución de todas las posibles rutas.
- Estructura bucle: está formada por un conjunto de vértices, de los cuales al menos uno de ellos es predecesor de un vértice del cual es, a su vez, sucesor, creando de esta manera el bucle. Esta estructura se representa multiplicando el número de iteraciones que se ejecuta el bucle por la expresión que representa las estructuras internas del bucle. En función del tipo de bucle de alto nivel que haya generado el código pueden existir ligeras variaciones en las expresiones resultantes.

La Figura 10 muestra las expresiones para estas tres estructuras básicas. La Figura 11 muestra el CCFG mostrado en la Figura 9 junto con su correspondiente expresión. En estas expresiones, E_i representa el tiempo de ejecución del vértice V_i. Otras estructuras más complejas, o variaciones sobre estas mismas estructuras pueden representarse de forma sencilla mediante expresiones similares o derivadas de las correspondientes básicas. Estas expresiones no requieren ningún tipo de restricción, y el tiempo de ejecución de un vértice puede aparecer varias veces, combinarse en múltiples expresiones o anidarse si por ejemplo existen bucles anidados.

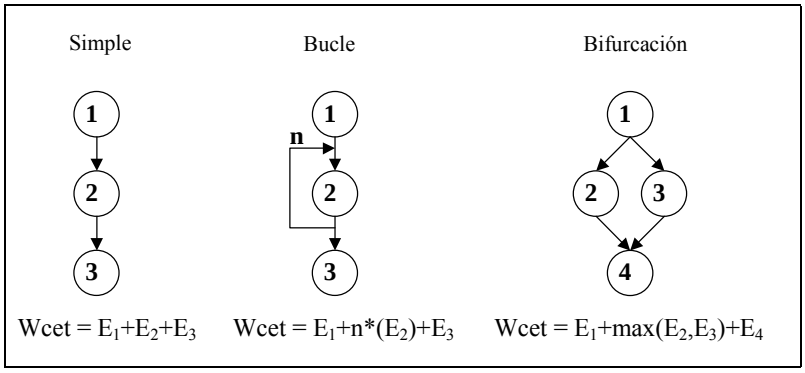


Figura 10. Tres estructuras básicas con sus correspondientes *expresiones.*

La obtención del WCET de la tarea puede obtenerse simplemente evaluando la expresión obtenida, pero para ello es necesario conocer el tiempo de ejecución de cada vértice. Éste es también el punto de partida de la mayoría de métodos de estimación del WCET de una tarea considerando la existencia de memoria *cache* [Mueller95b] [Martí00] como los comentados en el capítulo anterior, en los que el verdadero problema es calcular el tiempo de ejecución de cada vértice.

La utilización del bloqueo estático de *cache* permite resolver el problema del cálculo del tiempo de ejecución de cada vértice. Una vez el sistema inicia la ejecución en régimen permanente, los contenidos de *cache* permanecen inalterables, y además son conocidos puesto que es el diseñador quien selecciona los bloques que se precargarán en *cache*. De este modo, para todas y cada una de las tareas del sistema se conoce a priori las instrucciones que se encontrarán en *cache*, y por tanto cuales se referenciarán siempre con acierto y cuales se referenciarán siempre con fallo, por lo que sus tiempos de ejecución son estáticos y conocidos. Así pues, si se conoce

el tiempo de ejecución de cada instrucción, se conocen también los tiempos de ejecución de cada vértice, que vendrán definidos por las siguientes ecuaciones:

- Para un vértice V_i cuyas instrucciones se encuentran cargadas y bloqueadas en *cache*, su tiempo de ejecución $E(V_i)$ es: $E(V_i) = E_i$
- Para un vértice V_i cuyas instrucciones no han sido cargadas ni bloqueadas en *cache*, su tiempo de ejecución $E(V_i)$ es: $E(V_i) = T_{miss} + E_i$

En estas ecuaciones, T_{miss} es el tiempo necesario para transferir un bloque desde memoria principal hasta el buffer temporal, y E_i es el tiempo necesario para ejecutar las instrucciones incluidas en el vértice V_i desde memoria *cache* -básicamente, y dependiendo de la arquitectura del procesador, la suma de los tiempos de ejecución de las instrucciones incluidas en el vértice-.

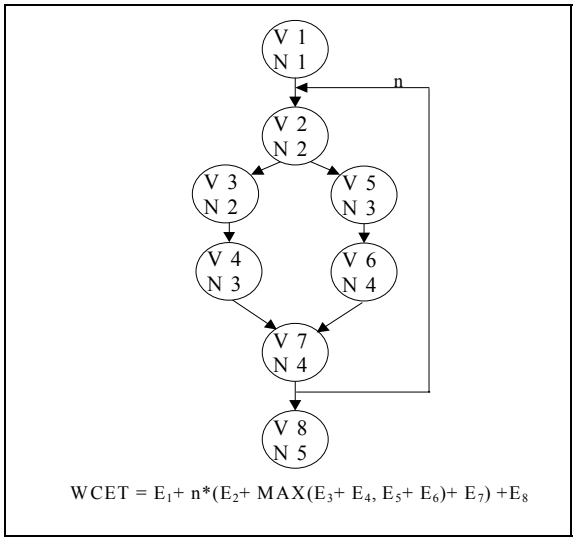


Figura 11. Expresión para el CCFG ejemplo de la Figura 9.

Debido a la existencia del buffer temporal, aquellos vértices cuyas instrucciones no hayan sido cargadas y bloqueadas en *cache* deberán ser transferidas previamente a su ejecución desde memoria principal hasta el buffer. Pero esta transferencia se realizará sólo para la primera instrucción ejecutada, ya que las siguientes, al transferirse todo el bloque, se encontrarán ya en el buffer. Por ello, sólo es necesario sumar una sola vez el tiempo de transferencia desde memoria principal al buffer temporal al tiempo de ejecución del vértice.

Sustituyendo los valores de $E(V_i)$ en la expresión del CCFG puede obtenerse una cota extremadamente ajustada del WCET de la tarea. Este WCET estimado no tiene porqué ser exacto, ya que en este modelo se considera que tras la ejecución del código cargado en el buffer temporal, éste es reemplazado por otro código, es decir, siempre se producirá un fallo cada vez que se ejecute dicho código, cuando en función de la estructura de la tarea esto no tiene porqué ser cierto. Los resultados de los experimentos mostrarán que esta *sobreestimación* es mínima, y que no representa ningún problema de importancia.

Pese a esta pequeña *sobreestimación*, el modelo descrito presenta varias ventajas. La primera, que proporciona, de forma segura, una cota siempre superior, lo que deriva en un análisis seguro. La segunda, que se trata de una análisis simple y muy estudiado, lo que proporciona una facilidad de aplicación extrema, con el valor añadido de la existencia de múltiples trabajos desarrollados para mejorar sus resultados, como la detección y eliminación de rutas imposibles [Kountouris96], o el modelado de otras mejoras estructurales [Jouppi89] [Zhang93] que pueden ser utilizados sin necesidad de considerar la existencia de memoria *cache*.

2.3 Análisis de planificabilidad.

El análisis de planificabilidad, tal como se describió en el capítulo anterior, es dependiente del algoritmo de planificación, de las características *software* del sistema como la relación entre los plazos y los periodos, y de la arquitectura *hardware*. Todos estos factores influyen en el número de expulsiones que sufre una tarea, y en el coste, en relación con la *cache*, de cada una de estas expulsiones. Estos dos valores o parámetros son los verdaderos interrogantes en la realización del análisis de planificabilidad, y a los que en este apartado se da respuesta. En este apartado se presentarán dos análisis de planificabilidad para sistemas que utilizan de forma estática memorias *cache* con bloqueo. El primer análisis se aplica a sistemas con planificador expulsivo y prioridades fijas. El segundo análisis se aplica a sistemas con planificador expulsivo y prioridades dinámicas, asignando la máxima prioridad a la tarea que tiene el plazo más próximo (EDF). En ambos casos, los plazos de las tareas pueden ser menores o iguales que los periodos, y se considera que todas las tareas se lanzan en el mismo instante de tiempo, llamado instante crítico. Aunque esta última restricción puede ser eliminada en los dos análisis que se presentan a continuación, su mantenimiento permite mantener las ecuaciones en su máximo nivel de sencillez.

2.3.1 Análisis de planificabilidad para prioridades fijas

El análisis de planificabilidad que se propone en este trabajo para sistemas expulsivos con prioridades fijas se basa en la extensión al *Response Time Analysis* (RTA) y que resulta en el *Cached Response Time Analysis* (CRTA). Este análisis de planificabilidad se realiza resolviendo, en varias iteraciones, la siguiente ecuación, que proporciona los tiempos de respuesta de cada una de las tareas del sistema. Comparando estos tiempos de respuesta con los plazos de la tarea puede comprobarse si todas las tareas finalizarán su ejecución antes del plazo, y por tanto verificar si el sistema es planificable.

$$w_i^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^n}{T_j} \right\rceil x(C_j + \gamma_j)$$

En esta ecuación se incorporan los dos efectos – intra-task e inter-task – de la *cache*. El efecto de la interferencia intrínseca se incorpora en el C_i de cada tarea τ_i, que representa el WCET de la tarea considerando la existencia de la *cache* con bloqueo. Este valor se obtiene mediante el análisis descrito en la sección anterior. La interferencia extrínseca de la *cache* se incorpora en el parámetro γ_j, que representa una cota máxima del tiempo que necesita la tarea τ_i expulsada para recargar las instrucciones que tenía en *cache* antes de la expulsión, y que fueron desplazadas de la misma debido a la ejecución de la tarea τ_j de mayor prioridad. La bondad de este análisis, es decir, su exactitud, depende de lo ajustado que sea el valor que se le asigne a este parámetro γ_j, que recibe el nombre de *cache-refill penalty* o *cache-related preemption delay*.

Dos son las causas que hacen que el cálculo de un valor ajustado del *cache-refill penalty* sea una labor compleja. La primera causa es el comportamiento dinámico de la *cache* y el desconocimiento de los puntos exactos de expulsión de una tarea. Esto hace que sea realmente complicado identificar las instrucciones que una tarea tiene en *cache* en el momento de la expulsión, y por tanto cuales entrarán en conflicto con las instrucciones de la tarea expulsante y que deberán ser recargados tras la expulsión. Por ello, los trabajos desarrollados hasta el momento [Busquets96b] [Lee01] realizan aproximaciones conservadoras, o intentan fijar los puntos en que una tarea puede ser expulsada [Luculli97] [Lee99] para reducir la complejidad del problema. Esta última solución, aunque con resultados satisfactorios, entra en conflicto con la definición de planificador expulsivo con prioridades, ya que fuerza que una tarea se mantenga en ejecución pese a la activación de tareas de mayor prioridad, asemejándose más a un planificador cíclico.

La segunda causa de complejidad en la estimación del valor de γ_j es la posibilidad de que una tarea, al ser expulsada, sufra dos tipos de interferencia extrínseca: directa e indirecta. La

interferencia directa se produce cuando una tarea ve aumentado su tiempo de ejecución, y por tanto su tiempo de respuesta, al recargar la *cache* debido a que la tarea expulsante los ha desplazado de la memoria *cache*. La interferencia indirecta se produce cuando se anidan dos o más expulsiones, es decir, la tarea τ_x es expulsada por la tarea τ_y, que a su vez es expulsada por la tarea τ_z. La tarea de menor prioridad, τ_x deberá recargar los contenidos de *cache* desplazados no sólo por la tarea τ_y, sino también los contenidos desplazados por la tarea de mayor prioridad τ_z. Además, la tarea de prioridad intermedia τ_y verá aumentado su tiempo de ejecución por la interferencia directa que le provoca la activación de la tarea de mayor prioridad. Este aumento en el tiempo de ejecución repercutirá en el tiempo de respuesta de la tarea de menor prioridad τ_x. De este modo, una tarea ve aumentado su tiempo de respuesta no sólo por las expulsiones directas que sufre, sino por el aumento en el tiempo de respuesta que sufre la tarea expulsante al ser, a su vez expulsada y sufrir interferencia extrínseca. Esta fuente de complejidad en la estimación del *cache-refill penalty* no puede ser evitada ya que es una circunstancia consustancial a la política de planificación, y sólo puede resolverse de forma conservadora, asumiendo que se produce aquella interferencia, directa o indirecta, que supone un mayor incremento en el tiempo de respuesta, ya que las expulsiones sólo están acotadas en número de sucesos, pero no en los instantes en que se producen.

La Figura 12 describe gráficamente los efectos de los dos tipos de interferencia extrínseca que puede sufrir una tarea. En esta imagen se ha supuesto que el *cache-refill penalty* que sufre la tarea 2 es de 3N ciclos, y el *cache-refill penalty* que sufre la tarea 3 es 2N ciclos. Estos valores dependen de los conflictos que se producen entre las tareas involucradas en cada expulsión, y por tanto pueden ser diferentes para cada tarea e incluso para cada expulsión que sufra una misma tarea. Cada división representa N ciclos de ejecución del procesador. En la Figura 12-a la tarea 3 sufre dos expulsiones, una provocada por la tarea 1 y otra provocada por la tarea 2. En ambos casos, la interferencia extrínseca es directa, y supone un coste del *cache refill penalty* para la tarea 3 de 4N ciclos (2N+2N). En la Figura 12-b la tarea 3 sufre también dos expulsiones, provocadas así mismo por la tarea 1 y la tarea 2, pero en este caso existe interferencia directa e indirecta, ya que la tarea 1 expulsa a la tarea 2 mientras esta última había expulsado a la tarea 3. En este caso el valor del *cache refill penalty* es de 5N ciclos (3N+2N), mayor que el del escenario anterior. Debido a que RTA y por tanto CRTA no son capaces de identificar cual de los dos escenarios sucederá realmente, sino que sólo indican que se producirán dos expulsiones, debe considerarse siempre el peor caso posible, pudiéndose incorporar una sobreestimación en el tiempo de respuesta calculado. En la Figura 12 se ha tomado la licencia de representar el rellenado de la *cache* tras una expulsión inmediatamente después de la reanudación de la ejecución de la tarea expulsada, aunque realmente este rellenado se producirá distribuido a lo largo de toda la ejecución de la tarea, en el momento que las instrucciones desplazadas de *cache* por la expulsión sean nuevamente solicitadas.

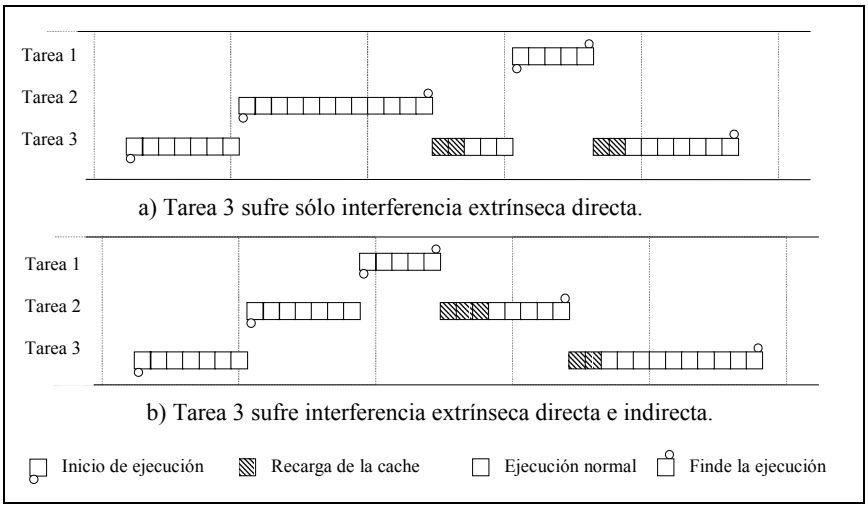


Figura 12. Ejemplos de interferencia extrínseca directa e indirecta.

La utilización estática de memorias *cache* con bloqueo simplifica enormemente el problema del cálculo de la interferencia extrínseca. En primer lugar, porque la interferencia se reduce a la recarga del buffer temporal, y en segundo lugar, porque el efecto de la interferencia directa e indirecta es el mismo, y no es necesario realizar un tratamiento detallado o conservador de las dos variantes de este efecto.

En la utilización estática de *cache*, los contenidos de ésta permanecen invariables durante toda la ejecución del sistema, por lo que la expulsión de una tarea no representa ninguna modificación en las instrucciones que se encuentran en *cache*. Cuando una tarea reanuda su ejecución tras una expulsión, encontrará en *cache* los mismos contenidos que había antes de ser expulsada, por lo que no necesitará recargar ningún bloque de memoria principal, ni sufrirá ningún fallo en *cache* que incremente su tiempo de ejecución respecto del estimado durante el cálculo del WCET. La única variación que una tarea puede encontrar al retomar la ejecución tras una expulsión puede provenir del buffer temporal, ya que su comportamiento, tal como se ha definido en la descripción de la arquitectura *hardware* propuesta, plantea la posibilidad de que la tarea expulsante modifique el contenido de dicho buffer. Si en el momento de la expulsión, la tarea de menor prioridad se encuentra ejecutando instrucciones que han sido copiadas al buffer temporal, existe la posibilidad de que estas instrucciones sean reemplazadas por el código de la tarea de mayor prioridad, por lo que, al reanudar su ejecución, la tarea expulsada deberá volver a traer esas instrucciones de memoria principal, incrementando su tiempo de ejecución respecto del estimado durante el cálculo del WCET. Por tanto, el tiempo de recargar el buffer temporal –T_{miss} o tiempo de fallo- será el valor del *cache-refill penalty*, ya que no existe ninguna otra fuente de variación en los aciertos o fallos que una tarea sufra al acceder a memoria *cache* respecto de los considerados durante el cálculo del WCET.

En cuanto a la interferencia directa o indirecta, el problema es eliminado, ya que el coste de una expulsión, en lo relativo a la *cache*, es el mismo para todas las tareas del sistema, independientemente del punto de expulsión, por lo que el coste de rellenar la *cache* será siempre el mismo para cualquier expulsión de cualquier tipo, por lo que no es necesario realizar distinción alguna entre interferencia directa e indirecta. De este modo γ_j es igual a T_{miss} para todos los casos.

Finalmente, la ecuación de CRTA para la utilización estática de *cache* queda de la siguiente manera,

$$w_i^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^n}{T_j} \right\rceil x(C_j + T_{miss})$$

donde T_{miss}, es decir, el coste de recargar el buffer temporal, se añade al tiempo de respuesta de la tarea bajo análisis por cada expulsión que sufra, independientemente de si es directa o indirecta.

La forma de proceder descrita anteriormente supone cierto pesimismo y conservadurismo, ya que se asume que cuando una tarea es expulsada siempre está ejecutando código ubicado en el buffer temporal, y que por tanto deberá recargarlo al reanudar la ejecución. Esto no siempre es cierto, pero al ser extremadamente bajo el valor del *cache-refill penalty*, en comparación con los tiempos de ejecución de las tareas, este pesimismo no es significativo, tal como se verá en el apartado de resultados experimentales.

Dos son las principales ventajas que ofrece el uso estático de *caches* con bloqueo al análisis de planificabilidad. Por una parte su sencillez, ya que se trata de una aplicación directa del análisis del tiempo de respuesta original, sin necesidad de realizar ningún tipo de estimación ni cálculo añadido. Por otra parte, lo ajustado de los resultados que se obtienen, propiedad derivada directamente del análisis original, ya que el valor del *cache-refill penalty* es extremadamente bajo como para hacerse patente una posible sobreestimación. La única fuente de error surge del análisis original, ya que es posible que se produzca un exceso en la estimación del número de expulsiones que una tarea puede sufrir, pero no es el efecto de la *cache* quién desviará los

resultados de la realidad, sino la consideración de que otra tarea, con un tiempo de ejecución extremadamente superior al necesario para recargar el buffer temporal, ocupará la CPU.

2.3.2 Análisis de planificabilidad para prioridades dinámicas

En este apartado se describe el análisis de planificabilidad para el uso estático de *cache* cuando el sistema utiliza un planificador expulsivo de prioridad dinámicas, y las prioridades se asignan de forma inversa al tiempo que la tarea dispone antes de que venza su plazo. Esta política de planificación recibe el nombre de EDF (*Earliest Deadline First*), tal como se describió en el capítulo anterior. El análisis utilizado permite que las tareas tengan plazos de entrega menores que el periodo.

En este tipo de sistema, el análisis de planificabilidad puede realizarse durante el diseño verificando la planificabilidad del sistema durante un determinado intervalo de tiempo, llamado Instante Crítico Inicial (ICI) [Ripoll96]. Este análisis se basa en la resolución de dos funciones, llamadas G(t) y H(t).

La primera función, G(t), calcula el total del tiempo de cómputo, es decir, de procesador, solicitado por todas las activaciones de las tareas del sistema entre el instante inicial y un determinado instante de tiempo t. La ecuación que permite obtener esta suma del tiempo de cómputo es:

$$G(t) = \sum_{i=1}^{n} C_i \left\lceil \frac{t}{P_i} \right\rceil$$

La segunda función, H(t), calcula el total de tiempo de cómputo solicitado por todas las activaciones de las tareas del sistema cuyo plazo de entrega es menor o igual a un determinado tiempo t. En otras palabras, H(t) representa el total de tiempo de cómputo que el planificador deber haber servido hasta el instante de tiempo t para que todos los plazos se hayan cumplido. La ecuación que permite calcular el valor de H(t) es:

$$H(t) = \sum_{i=1}^{n} C_i \left\lfloor \frac{t + P_i - D_i}{P_i} \right\rfloor$$

Utilizando las funciones G(t) y H(t) se puede obtener el valor del instante crítico inicial resolviendo la siguiente ecuación recursiva: $R_{i+1}=G(R_i)$ hasta que $R_{i+1}= R_i$, y con $R_0 = 0$

El último valor de R_i es el valor del ICI, que representa el primer instante de tiempo en el que todas las peticiones han sido servidas, es decir, todas las tareas que se han activado hasta el momento han terminado su ejecución, y no hay ninguna petición pendiente, es decir, no se ha producido ninguna nueva activación de ninguna tarea. Una vez conocido el valor del ICI, la planificabilidad del sistema está asegurada si y sólo si se cumple la siguiente desigualdad:

$$H(t) < t : \forall t, 1 \le t \le R$$

El análisis de planificabilidad presentado no considera en ningún momento la existencia de memoria *cache*, por lo que no hay ninguna referencia a la interferencia extrínseca ni ninguna indicación de cómo introducir el valor del *cache-refill penalty*. Es más, la inclusión de este efecto en este análisis de planificabilidad es compleja, ya que el *cache-refill penalty* está directamente relacionado con las expulsiones que se producen durante la ejecución del sistema, mientras que el análisis de planificabilidad basado en el ICI no tiene en consideración el secuenciamiento de las tareas, no permite calcular el número de expulsiones que se producirán en el sistema, y mucho menos identificar la tarea expulsante ni la expulsada. Este problema se deriva directamente de la política de asignación de prioridades, que al ser dinámica, convierte en un problema casi irresoluble el conocimiento de la secuenciación de las tareas, es decir, la identificación del orden en que las tareas entran en ejecución. Sin embargo, la posibilidad de

analizar sistemas donde los plazos de las tareas sean menores que los periodos hace muy interesante la utilización de este análisis frente al empleo de *CUA (Cache-aware Utilization-based Analysis*) [Basumallick94], donde es necesario que los plazos sean idénticos a los periodos.

La utilización estática de *cache* con bloqueo hace sencillo incorporar el efecto de la *cache* al análisis de planificabilidad. En primer lugar, en las ecuaciones de G(t) y H(t) el valor de C_i que representa el WCET de la tarea τ_i debe ser calculado teniendo en cuenta el efecto de la interferencia intrínseca. Este cálculo puede realizarse mediante el procedimiento descrito en el apartado correspondiente. En segundo lugar, el efecto de la interferencia extrínseca puede incorporarse al tiempo de cómputo de la tarea expulsante y no de la tarea expulsada, de forma contraria a como se realizaba en el análisis propuesto en el apartado anterior para planificador de prioridades fijas. Esta forma de proceder es válida por las siguientes razones:

- Bajo la política de planificación EDF, la activación de una tarea producirá una expulsión en el mismo instante de la activación, o no la producirá nunca. De forma trivial, si dos tareas A y B se han activado, y la tarea B se encuentra en ejecución, es debido a que su plazo está más próximo que el de la tarea A, y puesto que el tiempo avanza por igual para las dos tareas, la distancia a los plazos de ambas tareas se reducirá de manera idéntica. Así pues, una tarea no puede producir más expulsiones durante un periodo de tiempo T que el número de veces que se active durante ese periodo de tiempo T.
- El análisis de planificabilidad basado en el ICI contabiliza la utilización global del procesador, y no los tiempos de respuesta de cada tarea tal como hacia CRTA. Por tanto, es independiente a qué tarea se le asigne el tiempo de recarga de la *cache*, ya que este tiempo se sumará a la utilización global del procesador, y no al tiempo de respuesta de cada tarea.

Tal como se mencionó en análisis para prioridades fijas, el tiempo que una tarea debe utilizar, en el peor caso, para recargar los contenidos desplazados por una expulsión, corresponde con el tiempo que se necesita para recargar el buffer temporal, siendo este tiempo T_{miss}, el tiempo de fallo en *cache*. Ninguna otra fuente de interferencia extrínseca existe en el sistema, y no es necesario considerar explícitamente las interferencias directa e indirecta, ya que, al ser constante el coste de recarga de ambas interferencias, sólo es necesario considerar el número de expulsiones que suceden en el sistema, pero no el tipo de expulsión.

Para incluir el valor del *cache-refill penalty* se modifican las ecuaciones de G(t) y H(t) de la siguiente forma:

$$G(t) = \sum_{i=1}^{n} (C_i + T_{miss}) \left\lceil \frac{t}{P_i} \right\rceil$$

$$H(t) = \sum_{i=1}^{n} (C_i + T_{miss}) \left\lfloor \frac{t + P_i - D_i}{P_i} \right\rfloor$$

En este análisis se introducen dos fuentes de pesimismo y sobreestimación. Por un lado, la consideración de que todas las tareas, en todas sus activaciones, producen una expulsión. Esto no tiene porque ser cierto, ya que una tarea puede activarse con una distancia a su plazo mucho mayor que la distancia al plazo de la tarea en ejecución. Por otro lado, y al igual que en el análisis de sistemas con prioridades fijas, la asunción de que en todas las expulsiones, la tarea expulsada se encontrará ejecutando código desde el buffer temporal, y por tanto deberá recargarlo al reanudar su ejecución. Esta sobreestimación, tal como muestran los resultados experimentales, es mínima para la mayoría de los casos, y no representa una pérdida de exactitud significativa.

2.4 Selección de contenidos para la locking cache

Tal como se ha descrito en los apartados anteriores, la utilización estática de memorias *cache* con bloqueo ofrece un sistema de memoria con un comportamiento determinista y predecible, permitiendo la utilización de métodos y algoritmos para el cálculo del WCET y la realización del análisis de planificabilidad de forma sencilla.

Sin embargo, la utilización estática de *locking cache* restringe el número de instrucciones –bloques de memoria principal- que se cargarán en *cache*, ya que habitualmente el tamaño de la *cache* será mucho menor que el tamaño de las tareas del sistema en conjunto, y por tanto se reduce el número de instrucciones que se verán beneficiadas por el incremento de velocidad proporcionado por esta. La carga y bloqueo en *cache* de bloques de memoria principal, seleccionados de forma aleatoria, proporcionará determinismo, pero puede representar una pérdida importante de prestaciones, ya que puede darse el caso que bloques que se referencien de forma constante por el procesador queden fuera de la *cache*, generando un número elevado de fallos, mientras que bloques cuya ejecución sólo se produce al inicio de la tarea o en muy contadas ocasiones sean cargados en *cache*, mejorando su tiempo de ejecución, pero no aportando realmente mejora al tiempo total de ejecución de la tarea. Por tanto, se hace imprescindible una selección cuidadosa de las instrucciones que se cargarán y bloquearán en *cache*, con el objetivo de obtener las mejores prestaciones posibles, al tiempo que se obtiene determinismo.

Esta selección de contenidos no es una tarea sencilla debido a múltiples razones.

- El número de posibles soluciones es desmesurado. Por ejemplo, si el conjunto de tareas que forman el sistema ocupa un total de n bloques de memoria principal, y la memoria *cache* está formada por m líneas, cada una con capacidad de un bloque, existen n sobre m combinaciones posibles. Para un sistema que ocupe 500 bloques de memoria principal y utilizando una *cache* de 250 líneas, existen más de 10^{149} posibles soluciones. La siguiente ecuación muestra cómo calcular el número de posibles soluciones:

$$\binom{n}{m} = \frac{n!}{m!(n-m)!}$$

- La carga y bloqueo de una instrucción en *cache* afecta al WCET de la tarea a la que pertenece dicha instrucción, pero también afecta al tiempo de respuesta del resto de tareas debido a la existencia de expulsiones, ya que el tiempo de respuesta de una tarea depende del tiempo de ejecución de las tareas más prioritarias si éstas le generan alguna expulsión.
- Las limitaciones en el nivel de asociatividad hacen que no todos los subconjuntos posibles de instrucciones sean soluciones válidas al problema, ya que puede resultar imposible cargar dos bloques de memoria principal debido a que la función de correspondencia de la *locking cache* asigne a ambos la misma línea de *cache*, produciendo un conflicto.
- La inexistencia de una métrica aceptada universalmente para comparar la bondad de dos sistemas de tiempo real impide decidir, de forma clara, cual de dos posibles soluciones es la mejor. En principio se ha de lograr, o al menos es el objetivo prioritario, que el sistema sea planificable. Pero ante dos soluciones distintas que generen sistemas planificables, es difícil decidir en abstracto y de forma genérica cual de los dos presenta mejores prestaciones. Esta dificultad a la hora de evaluar la bondad de un sistema se traslada a la selección de la solución óptima.
- El problema no presenta un comportamiento monótono ni lineal, ya que ni siquiera al aumentar el número de bloques cargados y bloqueados en *cache* se garantiza una mejora en las prestaciones del sistema.

El elevadísimo número de posibles soluciones hace inviable la utilización de algoritmos de búsqueda exhaustiva, incluyendo ramificación y poda. Las interacciones entre las tareas que

forman el sistema, y la complejidad de modelar tanto la arquitectura *hardware* como las prestaciones del sistema impiden el diseño de algoritmos específicos para realizar una búsqueda directa. La no linealidad del problema impide la utilización de algoritmos como la programación lineal o *hill climbing* basados en la búsqueda de gradientes.

Los algoritmos genéticos [Goldberg89] imitan el proceso evolutivo de los seres vivos, y para ello utilizan una búsqueda pseudoaleatoria para encontrar, en un tiempo de cómputo aceptable, una solución subóptima. Estos algoritmos presentan múltiples ventajas, ya que permiten encontrar solución, aunque no sea la mejor, a problemas con una complejidad elevada de representación o modelado sin necesidad de desarrollar sofisticados algoritmos. Son idóneos cuando el espacio de búsqueda tiene un tamaño desmesurado, ya que su diseño hace que el algoritmo presente una solución desde el primer momento de ejecución. Esta solución se mejora, aunque no de forma constante, según avanza el tiempo de ejecución del algoritmo, por lo que puede determinarse el tiempo máximo de ejecución del algoritmo, obteniendo una solución al final de dicho tiempo. Esta solución será más o menos óptima en función del tiempo asignado al algoritmo genético y de los ajustes realizados a los parámetros de ejecución. Esta posibilidad de acotar el tiempo de respuesta para obtener una solución aceptable es una de las principales ventajas frente a otros algoritmos, como ramificación y poda o búsqueda puramente aleatoria, métodos donde el tiempo de respuesta no está ni puede acotarse.

Un algoritmo genético basa su funcionamiento en los siguientes elementos y operadores:

- Representación de la solución, llamada codificación.
- Creación de las posibles soluciones iniciales, que reciben el nombre de población inicial.
- Una función para evaluar la bondad de las soluciones manejadas por el algoritmo. Esta función recibe el nombre de función de *fitness*.
- Selección de las mejores soluciones. Estas soluciones se utilizarán para crear nuevas soluciones, es decir, para construir una nueva población. Este proceso de selección asigna una probabilidad de participar en la generación de la nueva población a cada individuo.
- Operaciones de cruce y mutación, que combinarán las soluciones escogidas en el proceso anterior para producir una nueva población.
- Sintonización de los parámetros que afectarán a los diferentes operadores del algoritmo genético.

El algoritmo genético trabaja de forma iterativa, aplicando los operadores a un conjunto de soluciones en cada una de sus iteraciones, obteniendo como resultado de estos operadores un nuevo conjunto de soluciones, sobre el que vuelven a aplicarse los operadores, y así durante un número prefijado de iteraciones, llamadas generaciones, o durante un tiempo delimitado de tiempo. La Figura 13 muestra el diagrama de flujo de un algoritmo genético. En cada iteración se obtiene la mejor solución de entre todas las que forman la población que el algoritmo maneja en esa iteración.

El algoritmo genético desarrollado en este trabajo proporciona, como resultados, no sólo los bloques de memoria principal que deben cargarse en la *locking cache* para obtener las mejores prestaciones posibles, sino que ofrece también el tiempo de respuesta de cada tarea –si se trata de prioridades fijas- o la utilización global del sistema –si se trata de prioridades dinámicas- por lo que se obtiene, al mismo tiempo que el conjunto de bloques de memoria principal a cargar y bloquear, el análisis de planificabilidad del sistema. De este modo, el diseñador del sistema, al utilizar conjuntamente la *locking cache* y el algoritmo genético, resuelve en un único paso dos problemas: la selección de contenidos de *cache*, y el análisis de planificabilidad del sistema para el escenario propuesto por el algoritmo genético.

A continuación se describen con detalle los elementos y operadores enumerados anteriormente, especificando su definición y modo de operación para el problema que se pretende resolver, esto es, la selección de instrucciones que se precargarán y bloquearán en la *locking cache*. La

función de *fitness* y la selección de soluciones se han definido dependiendo del tipo de planificador –prioridades fijas o EDF- utilizado, por lo que se realizará una descripción separada.

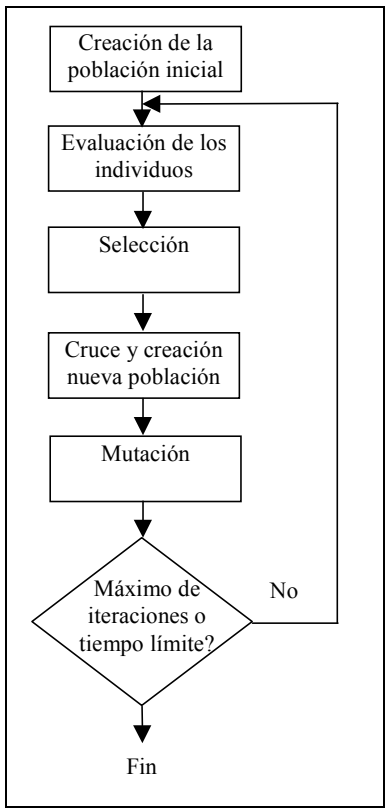


Figura 13. Diagrama de flujo del algoritmo genético.

2.4.1 Codificación

Básicamente la codificación es la estructura utilizada para la representación de la solución del problema. Cada posible solución recibe el nombre de individuo, y el conjunto de individuos sobre el que se trabaja en cada iteración recibe el nombre de población. Cada individuo –solución- está compuesto de genes, que representan los diferentes factores o coeficientes que determinan la solución. Estos genes pueden ser homogéneos, todos del mismo tipo y rango de valores, o heterogéneos, dependiendo directamente del problema que se está tratando.

El problema que nos ocupa es la selección de un subconjunto de instrucciones –realmente bloques de memoria principal- del total de instrucciones de todas las tareas del sistema, que deberán cargarse en la *locking cache*. De este modo un individuo será un vector de *n* elementos, donde n es el número de bloques de memoria que ocupan las tareas del sistema. El individuo tendrá tantos genes como elementos tenga el vector, y cada gen representa el estado de cada bloque de memoria, es decir, si el bloque se encuentra cargado y bloqueado en *cache* o por el contrario deberá ser buscado en memoria principal y transferido al buffer temporal. El valor de 1 de un gen indicará que el bloque correspondiente se encuentra en la *locking cache*, mientras que un valor de 0 indicará que no.

2.4.2 Creación de la población inicial

Aunque el algoritmo genético puede explorar todo el espacio de búsqueda a través de los operadores de cruce y mutación, una buena inicialización de la población inicial, es decir, de las soluciones de partida, puede ayudar al algoritmo a encontrar una solución subóptima con un número mínimo de iteraciones. Los individuos iniciales deben estar lo más cerca posible de la

solución del problema, o al menos tener unas características próximas a las que se intuye que tendrá la solución final.

En el caso del uso estático de *cache*, una buena solución será un conjunto de bloques cargados en memoria *cache*, en número igual al tamaño de la *cache* con el objetivo de aprovechar al máximo el aumento de prestaciones proporcionado por esta. Además, y debido a la estructura habitual de las tareas, estos bloques serán consecutivos en memoria, representando los bucles con mayor nivel de anidamiento de las tareas. De este modo, la población inicial se crea con secuencias de 1's consecutivos, con longitud máxima igual al tamaño de la *cache*, y posicionando estas secuencias de forma aleatoria en el vector que representa la solución. Al mismo tiempo, y para evitar que la evolución del algoritmo se restrinja a un mínimo local, la población inicial debe permitir que el algoritmo genere soluciones muy dispares como resultado del cruce y la mutación. Por ello, se añaden dos soluciones extremo a la población inicial, la primera de ellas sin ningún bloque seleccionado, y la segunda con todos los bloques seleccionados, aunque el número de bloques exceda el tamaño de la *cache*.

Es importante remarcar que la población inicial influye en la velocidad de convergencia, es decir, en el tiempo que se tarda en encontrar una solución subóptima que el algoritmo no es capaz de mejorar, pero no en la bondad de la solución encontrada. Es por ello que aunque esta inicialización es importante, no llega a ser determinante y son admisibles en su creación suposiciones cuya veracidad no está garantizada.

2.4.3 Función de Fitness

La función de *fitness* evalúa la bondad de cada individuo, permitiendo compararlos entre ellos para aplicar con éxito las operaciones posteriores. Esta función de *fitness* puede ser la misma función que el algoritmo intenta resolver, aunque habitualmente suele utilizarse otra función que no sólo permita identificar la mejor solución de cada generación, sino que permita que el algoritmo genético evolucione de forma rápida hacia la solución subóptima. En función del planificador que se utilice –prioridades fijas o EDF- se ha utilizado una función de *fitness* diferente:

- Planificador con prioridades fijas: la función de *fitness* consiste en la media ponderada del tiempo de respuesta de cada tarea, donde se asigna más peso a las tareas de menor prioridad. Esta función se basa en las interacciones existentes entre las tareas. Dadas dos tareas τ_1 y τ_2, con τ_1 más prioritaria que τ_2, el WCET de τ_1 influirá en su propio tiempo de respuesta, pero también, en menor medida y debido a las posibles expulsiones que τ_1 provoque a τ_2, en el tiempo de respuesta de τ_2, mientras que el WCET de τ_2 influirá únicamente en su propio tiempo de respuesta. De forma trivial, la asignación de líneas de *cache* a bloques pertenecientes a τ_1 mejorará los tiempos de respuesta de ambas tareas. Para evitar que el algoritmo, de forma viciada, asigne las líneas de *cache* a las tareas de mayor prioridad sin explorar otras posibilidades, se ha aumentado el peso de los tiempos de respuesta de las tareas de menor prioridad en la función de *fitness*. El tiempo de respuesta de cada tarea se obtiene estimando, en primer lugar, el WCET de la tarea aplicando el algoritmo descrito en el apartado 2.2 teniendo en cuenta los bloques que la solución propone que se precarguen y bloqueen en *cache*, y en segundo lugar, resolviendo la ecuación de CRTA propuesta en el apartado 2.3.1 y que considera como única interferencia extrínseca el tiempo de llenado del buffer temporal. Este tiempo de respuesta es una cota superior del tiempo de respuesta real de la tarea, tal como se describió en el apartado de análisis de planificabilidad para *locking cache*, por lo que el diseñador del sistema puede utilizar este valor para verificar la planificabilidad del sistema.
- Planificador EDF: en este caso la función de *fitness* escogida es la utilización global del sistema, obtenida directamente de la aplicación del análisis de planificabilidad propuesto en el apartado 2.3.2. La propuesta de esta función de *fitness* se basa fundamentalmente en que es la única información disponible, de forma relativamente sencilla, sobre el comportamiento del sistema, ya que es el único resultado que el análisis de planificabilidad

para EDF ofrece, no disponiéndose, como en el caso de prioridades fijas, de los tiempos de respuesta de las tareas. Además, el desconocimiento del secuenciamiento de las tareas en este tipo de planificadores impide el diseño de funciones de *fitness* como la propuesta para prioridades fijas. El WCET de las tareas, necesario para resolver el análisis de planificabilidad, se obtienen del modo descrito en el apartado 2.2. El resultado de esta función de *fitness* indica también si el sistema es planificable –utilización menor del 100%- por lo que el diseñador conoce, al mismo tiempo que el conjunto de bloques a cargar en *cache*, si el sistema es o no planificable con dicho conjunto de bloques.

Uno de los principales inconvenientes de las funciones de *fitness* utilizadas es que su tiempo de resolución no es constante, ya que se tratan de algoritmos iterativos, donde el número de iteraciones es desconocido ya que depende de la relación entre los periodos y los plazos de las diferentes tareas. Esto impide acotar el tiempo de ejecución del algoritmo genético para un número concreto de iteraciones, y hace que este tiempo de cómputo sea variable y dependiente de las características temporales de las tareas que forman el sistema.

2.4.4 Selección

El operador de selección es el encargado de escoger las mejores soluciones de cada generación para construir, a través de los operadores de cruce y mutación, una nueva población de soluciones. El proceso de selección se basa en los resultados obtenidos de la aplicación de la función de *fitness*, y en el número de bloques cargados en *cache*, ordenando los diferentes individuos y asignándoles una probabilidad de ser utilizados por el operador cruce. Según la función de *fitness* utilizada, el operador de selección se aplica del siguiente modo:

- Selección para prioridades fijas: Combinando el tiempo de respuesta medio ponderado de una solución y el número de bloques que dicha solución ha cargado en *cache*, cada individuo puede clasificarse en cuatro grupos:

  A. Tiempo de respuesta (medio ponderado) finito, lo que indica que el sistema es planificable, y número de bloques cargados en *cache* menor o igual que el número de líneas de la *cache*. Este es un individuo válido.

  B. Tiempo de respuesta (medio ponderado) finito -sistema planificable- con número de bloques cargados en *cache* superior al número de líneas de la *cache*. Este es un individuo no válido, ya que al utilizar más espacio de *cache* del disponible, la solución que representa no es utilizable.

  C. Tiempo de respuesta (medio ponderado) infinito, lo que indica que el sistema no es planificable, con un número de bloques cargados en *cache* menor o igual que el número de líneas de *cache*. Este es un individuo válido, pero una muy mala solución.

  D. Tiempo de respuesta (medio ponderado) infinito -sistema no planificable- con número de bloques cargados en *cache* superior al número de líneas de la *cache*. Además de ser un individuo no válido, se trata de una mal solución.

  La existencia de individuos no validos impide la utilización de un esquema de selección basado directamente en el resultado de la función de *fitness*. En lugar de este esquema, los individuos se ordenan en función de su validez y de su valor de *fitness*, asignándoles una probabilidad en función de la posición que ocupan tras esta ordenación. La ordenación se realiza del siguiente modo: posiciones más altas para los individuos válidos, utilizando el resultado de la función de *fitness* para ordenarlos entre ellos. Los individuos para los cuales la función de *fitness* retorna un valor infinito, se ordenan entre ellos utilizando la media ponderada de los tiempos de respuesta de las tareas que sí cumplen con sus plazos. Posiciones más bajas para los individuos no válidos, ordenándolos entre ellos en función del número de bloques que han cargado en *cache*, asignando las posiciones más altas a los individuos que han cargado un número menor de bloques, es decir, aquellos que están más próximos de ser válidos. Una vez ordenados, la probabilidad se asignará de forma decreciente, dando, por tanto, mayor probabilidad de ser escogidos a los individuos que

ocupan las posiciones más altas. La inclusión de individuos no válidos en este proceso de selección, cuando podrían haber sido directamente descartados, ayuda a mantener la variabilidad del algoritmo genético, permitiendo ampliar el espacio explorado.

- Selección para prioridades dinámicas, EDF: la función de *fitness* devuelve, para cada individuo, la utilización global del sistema si este es planificable, o la distancia relativa al ICI donde se superó el 100% de utilización si el sistema no es planificable. Combinando estos resultados con el número de bloques cargados en *cache* se pueden clasificar los individuos en cuatro tipos:

  A. Utilización menor del 100%, es decir, sistema planificable, con número de bloques cargados en *cache* menor o igual al número de líneas de la memoria *cache*. Este es un individuo válido.

  B. Utilización menor del 100% -sistema planificable- con número de bloques cargados en *cache* superior al número de líneas de la memoria *cache*. Este es un individuo no válido, ya que al utilizar más espacio de *cache* del disponible, la solución que representa no es utilizable.

  C. Utilización superior al 100%, es decir, sistema no planificable, con número de bloques cargados en *cache* menor o igual al número de líneas de *cache*. Este individuo representa una mala solución, pero válida.

  D. Utilización superior al 100% -sistema no planificable- con número de bloques cargados en *cache* superior al número de líneas de la memoria *cache*. Además de ser un individuo no válido, se trata de una solución mala.

  Al igual que en el esquema de selección para planificadores con prioridades fijas, la existencia de individuos no válidos impide la utilización directa del valor de la función de *fitness* para la asignación de probabilidades, pero sigue siendo interesante, con el objetivo mantener la variabilidad del algoritmo genético, incorporar estas soluciones con una probabilidad baja. Por ello, los individuos se ordenan en función de su validez y planificabilidad, asignando la probabilidad mayor a los que ocupan las posiciones más altas. La ordenación se divide en tres segmentos. En el primer segmento, con las probabilidades más altas, los individuos válidos que además son planificables. En el segmento intermedio, los individuos válidos que no son planificables. Y en el último segmento, con las probabilidades más bajas, los individuos no válidos. Dentro del primer segmento, los individuos se ordenan de forma inversa a la utilización global del sistema, asignando, por tanto, la probabilidad mayor al individuo con una utilización menor. En el segundo segmento, los individuos se ordenan de forma inversa a la distancia al ICI, dando mayor prioridad a los individuos más próximos a la planificabilidad. Por último, los individuos no válidos se ordenan en forma inversa al número de bloques de memoria que cargan en *cache*, por lo que, dentro de este segmento, los individuos con menor número de bloques cargados serán los que tendrán asignada una mayor probabilidad.

En ambos operadores de selección, la suma de las probabilidades de todos los individuos debe sumar 1, por lo que las probabilidades se asignan de forma decreciente y no constante, reduciéndose la distancia entre las probabilidades según se avanza en el vector. Además, para facilitar la operación de cruce, se calcula la probabilidad acumulada de cada individuo como la suma de la probabilidad acumulada del individuo precedente y la probabilidad asignada a dicho individuo. Las siguientes ecuaciones muestran la asignación de probabilidades a cada individuo. El individuo 0 representa la mejor solución encontrada y que ha sido colocado en la cima del vector de ordenación. En estas ecuaciones, *MP* es la probabilidad que se asignará al mejor de todos los individuos e *i* indica la posición que un individuo ocupa en el vector de ordenación.

El individuo que ocupa la última posición, y que representa la peor de todas las soluciones de la generación, tendrá la máxima probabilidad acumulada, mientras que el mejor individuo de todos tendrá la menor probabilidad acumulada. Sin embargo, cuanto mayor sea la probabilidad acumulada, menor será la distancia entre las probabilidades acumuladas de dos individuos. Esta

forma de proceder es equivalente a asignar la misma probabilidad a todos los individuos, pero replicando los mejores individuos varias veces, aumentando por tanto la posibilidad real de que sean escogidos. La Tabla 7 muestra un ejemplo para siete individuos, donde la probabilidad para el mejor individuo se establece en 0.1. Cuando el número de individuos es suficientemente grande, la probabilidad tiende a 0, y la probabilidad acumulada tiende a 1. Aun así, las probabilidades del último individuo se establecen manualmente para garantizar que la convergencia se alcanza.

$$Probabilidad\ (0) = MP$$

$$Probabilidad\ (i) = MP(1-MP)^i$$

$$Probabilidad\_acumulada\ (0) = MP$$

$$Probabilidad\_acumulada\ (i) = Probabilidad\_acumulada\ (i-1) + Probabilidad\ (i)$$

Individuo	Probabilidad	Probabilidad acumulada
0	0,1000000000	0,100000000
1	0,0900000000	0,190000000
2	0,0810000000	0,271000000
3	0,0729000000	0,343900000
4	0,0656100000	0,409510000
5	0,0590490000	0,468559000
149	0,0000000152	0,999999860

Tabla 7. Probabilidad asignada y acumulada de cada individuo en función de su posición.

2.4.5 Cruce

El operador cruce es el encargado de escoger dos individuos y combinarlos para formar dos nuevos individuos, que en teoría obtendrán las mejores características de sus "progenitores". Para escoger los dos individuos que darán origen a las dos nuevas soluciones se utiliza la asignación de probabilidad acumulada realizada durante el proceso de selección. Para ello, se generan dos números aleatorios entre 0 y 1, y se escogen los dos individuos cuya probabilidad acumulada es inmediatamente mayor que el número generado. La utilización de la probabilidad acumulada hace que los mejores individuos, que son los que tienen una probabilidad acumulada menor, tengan más posibilidades de ser escogidos para crear los nuevos individuos ya que la distancia entre sus probabilidades acumuladas es mayor.

Una vez escogidos dos individuos se decide, aleatoriamente, si estos individuos se recombinarán para generar dos nuevas soluciones, o si ellos mismos pasarán, sin modificación alguna, a la siguiente generación. El objetivo de este paso previo es permitir que las mejores soluciones de una generación se mantengan durante un número no excesivo de generaciones posteriores, evitando de este modo que el algoritmo genético abandone o pierda demasiado rápido buenas soluciones. El objetivo es imitar a la naturaleza, donde el individuo más fuerte puede serlo no sólo de su generación, sino de unas pocas generaciones posteriores, pudiendo proporcionar a la población una buena cantidad de hijos con unas características óptimas. Como ejemplo contrario, y aun a riesgo de desviar el tema de esta tesis, la naturaleza presenta el caso del macho de "mantis religiosa", que tras su primera y única reproducción, muere. Para evitar que las mejores soluciones se perpetúen durante todas las generaciones, la probabilidad de que dos individuos pasen a la nueva generación sin recombinarse es ligeramente inferior (en un 10%) a la probabilidad de que se recombinen.

Si la suerte decide que los dos individuos deben cruzarse para generar las dos nuevas soluciones, este cruce se realiza escogiendo al azar un bit que divide en dos partes desiguales a cada individuo, e intercambiando los dos segmentos, tal como muestra la Figura 14.

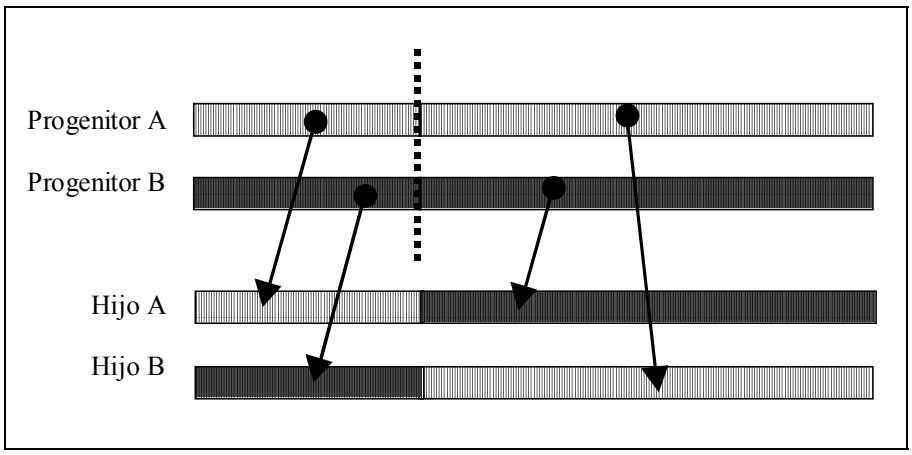


Figura 14. Ilustración del proceso de cruce de dos individuos. La línea discontinua indica el gen seleccionado para dividir los individuos.

El proceso de cruce se repite el número de veces necesario para generar una nueva población completa. Como en la naturaleza, cada nueva población tendrá un número elevado de individuos resultado de cruzar los mejores individuos, un número menor de replicas de los mejores individuos de la generación anterior, y un número muy bajo, pero que ayuda a mantener la variabilidad del algoritmo genético, de individuos resultado de cruzar una buena solución con una muy mala, e incluso no válida solución

2.4.6 Mutación

El objetivo de la mutación es introducir cierta variabilidad aleatoria en algunos de los individuos creados tras la aplicación del operador cruce. Básicamente, se trata de cambiar al azar algún o algunos bits de un número reducido de individuos, aumentando el espacio de exploración del algoritmo genético. Sin embargo, esta mutación puede aplicarse de forma dirigida si se conocen las características de los individuos que forman la nueva población, como es el caso que nos ocupa.

El resultado del cruce genera un número elevado de individuos con un número de bloques cargados y bloqueados en *cache* superior al permitido, es decir, al número de líneas de la memoria *cache*. Esto tiene una explicación sencilla. Cuantas más instrucciones se encuentren cargadas en *cache*, menor será el WCET de las tareas, y por tanto menor serán los tiempos de respuesta, no sólo debido al menor coste de ejecución, sino también a que cada tarea sufrirá un número menor de expulsiones, ya que se encuentra durante menos tiempo expuesta a la activación de tareas de mayor prioridad. Así pues, los mejores individuos serán aquellos que tengan un número mayor, dentro del límite admisible, de bloques cargados en *cache*. Al combinarlos entre ellos de la forma descrita en el operador cruce, ambos progenitores aportarán lo mejor de cada uno, sumándose, sino de forma completa, si de forma importante, el número de bloques que cada uno carga en *cache*. Para evitar que la nueva generación esté *plagada* de individuos no válidos, y por tanto no contenga una solución al problema, el operador mutación se aplica de forma diferente para tres situaciones o casos distintos:

A. Si el número de bloques cargados en *cache* es superior al número de líneas, el individuo será no válido, por lo que su aportación a la solución del problema será mínima. Para intentar mejorar su situación, se escogerán aleatoriamente algunos de los bits que tienen valor 1 y se mutarán al valor 0, reduciendo por tanto el número de bloques cargados en *cache*. Esta operación no afecta a todos los bits, ni siquiera afecta a los bits que exceden el número de líneas disponibles. El objetivo es mejorar su situación, y no reconvertir el

individuo en una solución válida. Si el exceso en el número de bloques cargados en *cache* es muy bajo, es posible que el individuo se convierta en una solución válida, pero si el exceso es muy elevado, éste sólo se verá mínimamente reducido.

B. Si el número de bloques cargados en *cache* es igual al número de líneas de *cache*, se trata de individuo completamente válido que realiza un aprovechamiento máximo del espacio disponible de memoria *cache*, por lo que, con muy baja probabilidad, se intercambiará alguna pareja de genes con valores distintos, es decir, se sustituirá alguno de los bloques cargados en *cache* por otro, pero sin variar el número total de bloques cargados.

C. Si el número de bloques cargados en memoria *cache* es menor que el número de líneas de la *cache*, se trata de un individuo válido, pero que no aprovecha completamente la capacidad de la *cache*. Presumiblemente, aumentando el número de bloques cargados mejorará el tiempo de respuesta de alguna tarea, por lo que en este caso, la mutación escogerá algún gen que tenga valor 0, y lo mutará al valor 1. Al igual que en el caso A, esta mutación no persigue igualar el número de bloques cargados con el número de líneas de *cache*, sino que intenta mejorar, de forma aleatoria y con una probabilidad de mutación muy baja, alguno de los individuos de este tipo.

Para los tres operadores de mutación definidos es importante remarcar que éstos se aplican con una probabilidad muy baja, por lo que la mayoría de los individuos no se verán afectados, y los que sí resulten afectados son escogidos de forma aleatoria. Una mutación excesiva provocaría en el algoritmo genético un comportamiento errático, ya que las características heredadas de la generación anterior se verían diluidas por los cambios aleatorios introducidos por la mutación. Sin embargo, una mutación demasiado escasa o inexistente podría hacer que el algoritmo genético se quedará atascado en un mínimo local. El objetivo de la mutación es ampliar el espacio explorado, pero sin alejarse demasiado de las soluciones encontradas hasta el momento, por lo que su correcta aplicación es fundamental.

Una vez realizada la mutación, y aunque no es estrictamente parte de la mutación, se incluye un operador adicional sobre los nuevos individuos de la generación recién creada. Durante todo el proceso de evaluación, cruce y mutación realizado en cada generación, no se ha considerado en ningún momento el nivel de asociatividad de la *locking cache*. Así pues, si estamos hablando de una *locking cache* con función de correspondencia directa, es posible que los individuos válidos no lo sean, debido a que carguen en memoria *cache* bloques que compiten entre ellos, es decir, que se ubican en la misma línea de *cache*, siendo por tanto imposible utilizar dicha solución. La incorporación de esta restricción durante la generación de nuevos individuos supondría una complejidad y un coste computacional excesivo, por lo que se ha decido "reparar" los individuos válidos que presenten algún conflicto a la hora de cargar y bloquear los bloques seleccionados. Para resolver estos conflictos, manteniendo las características de cada individuo, los bloques que presenten conflicto son sustituidos, de forma aleatoria, por otros que no presenten conflicto, manteniendo el número total de bloques cargados en *cache*. Es importante remarcar que, al contrario que la mutación, este proceso garantiza que ningún individuo válido presentará problemas debidos a la función de correspondencia, es decir, se modificarán todos aquellos genes que sea necesario.

2.4.7 Sintonización de parámetros

Existe un gran número de parámetros que afectan o modifican el comportamiento de los operadores del algoritmo genético, e incluso que determinan el comportamiento global del mismo. Aunque los algoritmos genéticos son muy robustos, presentando un comportamiento similar ante variaciones significativas de estos parámetros, se han realizado una serie de experimentos para determinar los valores óptimos de dichos parámetros. Para algunos de estos experimentos se conocía de antemano la solución óptima que el algoritmo genético debía encontrar, por lo que se podía evaluar, de forma bastante ajustada, la bondad de los valores asignados a los parámetros. Sin embargo, el algoritmo genético también es dependiente de los datos de entrada, por lo que es necesario escoger para los parámetros valores de compromiso,

que aunque no presenten un comportamiento óptimo en ningún caso, sean los que mejor se adaptan a los diferentes problemas planteados. Los resultados de estos experimentos han concluido en los siguientes valores para los parámetros, que se encuentran, todos ellos, dentro de los rangos habituales comentados en la bibliografía relacionada [Mitchell96].

- Tamaño de la población: 200 individuos. Una población excesivamente grande hace que el algoritmo genético evolucione demasiado despacio, ya que el coste computacional de evaluar todos los individuos sería enorme. Por el contrario, una población demasiado reducida limitaría el espacio explorado por cada generación, por lo que las generaciones se producirían a gran velocidad, pero con muy poca o ninguna mejora en la solución encontrada en cada una de ellas.
- Número de generaciones: 5000 generaciones o iteraciones. Uno de los problemas que presentan los algoritmos genéticos se deriva de la imposibilidad de obtener, en un sistema informático, números realmente aleatorios. La utilización de números aleatorios para definir el comportamiento de los operadores de cruce y mutación hace que la evolución del algoritmo genético sea dependiente de la semilla inicial utilizada para generar dichos números aleatorios. Para eliminar el efecto que pueda producir la utilización de números pseudoaleatorios, se ha determinado un número de iteraciones muy superior al necesario. De esta forma, aunque la utilización de una semilla u otra determine la evolución del algoritmo durante las primeras generaciones, y por tanto la bondad de la solución encontrada, el gran número de generaciones que se manejan hace que el resultado final no sea dependiente de la semilla utilizada. El problema de la solución escogida es el incremento en el tiempo de respuesta del algoritmo, que en un computador personal de mediana potencia puede llegar, en función de los datos del problema, a cerca de una hora.
- Probabilidad asignada al individuo situado en la cima del vector de ordenación o Mejor Probabilidad: 0,1. Esta probabilidad se asigna durante el proceso de selección al mejor individuo, y determina la probabilidad asignada a los siguientes individuos según la ecuaciones mostradas. El valor de 0,1 indica que la probabilidad de que este individuo sea escogido para cruzarse con otro es del 10%, y es el individuo con mayor probabilidad de ser escogido.
- Posibilidad de cruce: 0,6. Esta probabilidad determina los individuos que, una vez escogidos, se recombinarán entre ellos para formar dos nuevos individuos, o por el contrario, pasarán a la nueva generación sin modificación. Con una generación completamente ideal de números aleatorios, 60 de cada 100 parejas se recombinarán para formar los nuevos individuos, y tan solo el 40% pasará sin modificación a la siguiente generación.
- Probabilidad de mutación: 0,01 para los individuos con número de bloques cargados igual al número de líneas de *cache*, y 0,001 para el resto de individuos. Estos valores indican que, durante el proceso de mutación y de forma ideal, 1 de cada 100 genes y uno de cada 1000 genes, respectivamente, se verán alterados, cambiando su valor de 0 a 1 o viceversa. El primer valor se ha ajustado para forzar al algoritmo genético a ampliar el espacio explorado. El segundo valor se ha ajustado para que los individuos no válidos o que no aprovechan la capacidad de la *cache* tiendan hacia el, teóricamente, óptimo número de bloques cargados en *cache*, pero sin ser altamente modificados, manteniendo de esta forma la diversidad de la población.

2.5 Experimentos

A continuación se describen los experimentos realizados y los resultados obtenidos al emplear el uso estático de *locking cache* sobre un conjunto sintético y heterogéneo de sistemas de tiempo real. Los objetivos perseguidos con estos experimentos son dos.

El primer objetivo es evaluar la exactitud de los resultados obtenidos al analizar el comportamiento temporal de las tareas mediante la estimación del WCET y el análisis de

planificabilidad propuestos en los apartados anteriores. No hay que olvidar que tanto durante la estimación del WCET como en el análisis de planificabilidad para prioridades fijas y dinámicas, se ha introducido una *sobreestimación*, derivada fundamentalmente del comportamiento dinámico del buffer temporal incluido en la arquitectura *hardware* de la *locking cache*. Así pues, el primer objetivo es la medición o valoración de esta *sobreestimación* o pesimismo. Este objetivo se alcanza comparando los tiempos de respuesta y la utilización global del sistema calculada por el algoritmo genético frente a los valores obtenidos de la simulación de las tareas sobre una *locking cache*.

El segundo objetivo es la valoración del coste, en prestaciones, que se debe pagar por obtener determinismo. El altísimo aumento de prestaciones proporcionado por las memorias *cache* se debe fundamentalmente a su comportamiento dinámico, que permite adaptar los contenidos cargados en *cache* a las necesidades de las tareas en cada instante de tiempo. Por contra, el uso estático de *locking cache*, tal como índica su nombre, obtiene determinismo mediante la eliminación de la componente dinámica de la *cache*, por lo que el aumento de prestaciones ofrecido por la memoria *cache* puede verse comprometido. El segundo objetivo perseguido con los experimentos es determinar si existe una pérdida significativa de prestaciones al utilizar, de forma estática, las memorias *cache* con bloqueo. Este objetivo se alcanza comparando los tiempos de respuesta y la utilización global del sistema calculada por el algoritmo genético frente a los valores obtenidos de la simulación de las tareas sobre diferentes arquitecturas con memorias *cache* convencionales.

2.5.1 Herramientas utilizadas

La primera herramienta utilizada es la implementación del algoritmo genético tal como se ha descrito en el apartado 2.4. Se han realizado dos versiones, una para cada tipo de planificador –prioridades fijas y prioridades dinámicas-, diferenciadas únicamente en la función de *fitness* y el esquema de selección. Ambas versiones –a partir ahora referidas como "el algoritmo"- se han desarrollado en C y ejecutado sobre computadores personales utilizando el sistema operativo *Linux*. El algoritmo recibe como parámetros de entrada los siguientes datos:

- Número de tareas en el sistema, prioridad, periodos y plazos de entrega de cada una de ellas.
- Expresión que representa el c-cfg de cada una de las tareas que forman el sistema, y que se utilizará para calcular el WCET de cada una de ellas.
- Tabla con la correspondencia entre vértices y bloques de memoria principal utilizados por cada tarea, incluyendo el número de instrucciones que contiene cada vértice. Esta tabla es dependiente de las características de la memoria *cache* utilizada, en concreto del tamaño de la línea de *cache*. La correspondencia entre vértice y bloque de memoria principal es necesaria para resolver la expresión del c-cfg, ya que permite calcular el tiempo de ejecución de cada vértice, al identificar los bloques que se encuentran cargados y bloqueados en *cache*.
- Características *hardware* de la *locking cache*, incluyendo tiempo de acierto y tiempo de fallo, tamaño total de la *cache* sin considerar el buffer temporal, y tamaño de la línea de *cache*.
- Parámetros propios del algoritmo genético: probabilidad de cruce y mutación, número de generaciones, tamaño de la población, etc. ajustados a los valores anteriormente descritos.

Como resultado de la ejecución del algoritmo genético se obtienen los siguientes resultados:

- Direcciones de los bloques de memoria principal que deben cargarse y bloquearse en la *locking cache*. Este es el principal resultado y objetivo del algoritmo.
- En la versión para prioridades fijas, tiempo de respuesta y WCET de cada una de las tareas que forman el sistema, representando la violación del deadline mediante un tiempo de respuesta infinito. Los tiempos de respuesta permiten al diseñador verificar que el algoritmo

ha encontrado una solución planificable al problema, y disponer de una estimación segura -cota superior- de los tiempos de respuesta que presentará el sistema en el peor caso.

- Utilización global del sistema. Este valor permite comparar las prestaciones obtenidas al utilizar estáticamente la *locking cache* frente al uso de memorias *cache* convencionales.
- Número de bloques seleccionados para cada tarea. Un estudio detallado de este resultado puede ayudar a mejorar la sintonización del sistema, aunque básicamente se utiliza para verificar la corrección de los resultados proporcionados por el algoritmo.

La segunda herramienta utilizada es un simulador de la arquitectura MIPS R2000/R3000, mediante el que se simulará la ejecución de las tareas utilizando diferentes modelos de *cache*, tanto convencionales como con bloqueo. Los resultados de las simulaciones permiten evaluar la exactitud de los tiempos de respuesta estimados por el algoritmo genético, y por tanto, la exactitud de los métodos de análisis presentados en esta tesis y el grado de determinismo proporcionado por el uso estático de memorias *cache* con bloqueo. Además, la simulación de memorias convencionales permitirá comparar las prestaciones obtenidas por el uso estático de *locking cache* frente a las memorias *cache* convencionales, evaluando el coste, en términos de prestaciones, de la obtención de determinismo.

El simulador utilizado ha sido el SPIM [Patterson00]. Este *software*, de libre distribución y con código fuente disponible, permite simular la arquitectura de los procesadores MIPS R2000 y R3000, y no incluye ninguna mejora estructural -segmentación, predicción de saltos o ejecución especulativa- lo que permite estudiar el efecto de la utilización de memorias *cache* sin ninguna interferencia. Sin embargo, este simulador tampoco incluye memoria *cache*, multitarea ni contabilidad de los tiempos de ejecución, prestaciones necesarias para la consecución de los objetivos perseguidos por los experimentos. La herramienta ha sido modificada, por tanto, para incorporar diferentes modelos y esquemas de memoria *cache* de instrucciones incluyendo *locking cache*, así como contabilidad de tiempos de ejecución, tiempos de respuesta y número de instrucción ejecutadas. También se ha incluido, en el propio simulador, un gestor de multitarea. Al estar gestionada dicha multitarea por el simulador y no por el sistema operativo se evitan posibles interferencias generadas por el planificador y otras funciones necesarias para la multitarea.

Los parámetros de entrada al simulador son los siguientes:

- Número de tareas, prioridades, periodos y plazos de entrega de cada tarea que forma el sistema. El simulador calcula el hiperperiodo correspondiente al sistema como el máximo común múltiplo de los periodos de las tareas, finalizando la simulación cuando se alcanza este punto, o alguna de las tareas pierde un plazo de entrega.
- Tipo de planificador que debe utilizarse: prioridades fijas o EDF.
- Código ensamblador de cada una de las tareas.
- Modelo y esquema de *cache* utilizado, incluyendo, función de correspondencia -directa, asociativa de 2 y 4 conjuntos y completamente asociativa- tiempo de acierto y tiempo de fallo, tamaño de la *cache* y tamaño de la línea de *cache*.
- Para el modelo de *cache* correspondiente a *locking cache*, identificadores de los bloques de memoria principal que deber cargarse y bloquearse en *cache* antes de lanzar la ejecución del sistema.

El resultado de la simulación de un sistema es una lista de eventos con el siguiente formato:

Evento	Número de tarea	Marca de tiempo	Instrucciones ejecutadas

El significado de cada campo es el siguiente:

- Evento: indica qué evento se ha producido en el sistema. Se definen tres eventos: activación de una tarea -cumplimiento del periodo-; entrada en ejecución de una tarea; y finalización de la ejecución de una tarea. El evento "expulsión" no se considera explícitamente, pero puede detectarse fácilmente por la entrada en ejecución de una tarea antes de que otra tarea finalice su ejecución.
- Número de tarea: indica el número de tarea, según el orden en que se introdujeron en el simulador, que ha generado el evento.
- Marca de tiempo: instante de tiempo en que se produce el evento, referenciado al instante inicial que corresponde al inicio de la simulación.
- Instrucciones ejecutadas: este campo indica el número de instrucciones ejecutadas por una tarea entre dos eventos, permitiendo realizar un seguimiento detallado de la simulación.

Un programa realizado en C procesa la lista de eventos, ofreciendo información sobre el número de activaciones de cada tarea, su tiempo máximo, mínimo y medio de respuesta, número máximo y mínimo de expulsiones sufridas, activación en la que soportó el mayor número de expulsiones, tiempo máximo y mínimo de ejecución, número de instrucciones ejecutadas, tanto máximo como mínimo, y la utilización global del sistema. Toda esta información permite comparar los tiempos de respuesta y las prestaciones obtenidas por el algoritmo genético frente a los resultados de la simulación, además realizar un control de la validez de los resultados obtenidos.

La tercera herramienta utilizada es un generador automático de tareas. Con el objetivo de utilizar el mayor número posible de tareas distintas, y con un alto grado de variabilidad entre sus características, se ha desarrollado una sencilla herramienta que permite, a partir de unos parámetros básicos, generar código ensamblador del procesador MIPS R2000. Como parámetros de entrada a esta herramienta se define el número de bucles anidados, el número de bucles dentro de cada nivel así como el tamaño del bucle y el número de iteraciones que ejecutará cada bucle. Dentro de cada bucle se indica el número de estructuras *if-then-else* que contendrá, especificando el tamaño de cada ruta alternativa y la condición que se evaluará para decidir qué ruta ejecutar. Todos estos parámetros pueden especificarse con un valor concreto o mediante un máximo y un mínimo, dejando al azar el valor definitivo. La herramienta permite incluir en el código de las tareas otras estructuras más complejas, aunque con la intervención del usuario para garantizar la corrección semántica del código generado. Como resultado de la ejecución del generador de código se obtiene, además del propio código de la tarea, la expresión del c-cfg necesaria para estimar el WCET de la tarea, así como la tabla de correspondencia entre los vértices y los bloques de memoria principal utilizados por la tarea, y las instrucciones contenidas en cada vértice, información que como ya se ha comentado es necesaria para realizar el cálculo del WCET y el análisis de planificabilidad. Para poder obtener toda esta información es necesario proporcionar, junto con las características de las tareas, el tamaño de línea de *cache*. De esta forma, toda la información necesaria para realizar una estimación del WCET de la tarea es conocida.

Una de las principales ventajas de utilizar código generado sintéticamente es que se conoce con exactitud cuál es la ruta y los datos de entrada que producen el peor tiempo de ejecución, por lo que es posible simular la ejecución de las tareas para obtener dicho tiempo.

2.5.2 Descripción de los experimentos

Un experimento se define por los siguientes elementos:

- Un conjunto de tareas, entre 3 y 8, generadas sintéticamente.
- Parámetros de las tareas: prioridades, periodos y plazos de entrega. Todas las tareas se han considerado periódicas. Estos valores, junto con el código de las tareas, forman lo que a partir de ahora se llamará *carga*.

- Características *hardware* de la *cache*: tamaño total, tamaño de línea de *cache*, tiempo de acierto y tiempo de fallo en *cache*. La *carga* junto con las características *hardware* de la *cache* recibe el nombre de sistema o experimento.

Para realizar los experimentos se han generado cerca de 60 tareas distintas, que se han replicado y agrupado para formar un total de 15 conjuntos. Para cada conjunto de tareas se han definido dos grupos de periodos y plazos. En el primer grupo, los periodos de todas las tareas que forman una *carga* se han definido con el mismo valor, de forma que no exista interferencia alguna entre las tareas. En el segundo grupo, los periodos se han definido manualmente para conseguir el máximo grado de interferencia entre las tareas, pero garantizando que todas las tareas cumplen sus plazos de entrega y que el sistema es planificable. Los plazos de entrega, en ambos casos, se han definido entre un 10 y 30 por ciento menores a los periodos, y en el caso del planificador con prioridades fijas, se ha asignado la prioridad de cada tarea según la política *Rate Monotonic* -periodo más bajo, prioridad mayor-.

La definición de los dos grupos de periodos y plazos proporciona un total de 30 *cargas* distintas. Cada una de las 30 *cargas* -terna tareas, periodos, plazos- se ha evaluado sobre siete tamaños de *cache* distintos, comprendidos entre 1 kilobyte y 64 kilobytes, ambos inclusive y en tamaños correspondientes a potencias de dos, arrojando un total de 210 experimentos o sistemas evaluados.

La utilización de siete tamaños distintos de *cache* para cada *carga* ha permitido considerar los dos casos extremos en cuanto a la relación entre los tamaños del código total ejecutado y el tamaño de la *cache*, presentándose el caso de una *cache* con tamaño superior al del código -64 KB de *cache*- y una *cache* con un tamaño muy inferior al del código ejecutado -1KB de *cache* para código total superior a 32KB-. En todos los casos, el resto de características *hardware* de la *cache* se han mantenido invariables: tiempo de acierto igual a 1 ciclo del procesador, tiempo de fallo en *cache* igual a 10 ciclos del procesador, tamaño de línea de *cache* igual a 4 palabras de 32 bits, lo que equivale a 4 instrucciones. Estos parámetros de la cache son los utilizados en gran parte de los trabajos de investigación relacionados con esta tesis [Arnold94] [Hill88] [Hennessy90]. La Tabla 8 muestra un resumen de los experimentos definidos junto con las características más importantes.

Con el objeto de evaluar el impacto de las características de la arquitectura de la *cache*, se han definido 112 nuevos experimentos, en los que se han utilizado 4 de las *cargas* anteriores y se ha modificado el tiempo de fallo en *cache* y el tamaño de línea, estudiando el comportamiento de dichos factores a dos niveles. Las características, resultados y conclusiones extraídas de estos 112 experimentos se presentan en el capítulo 4, ciñéndose los resultados y conclusiones mostradas en los siguientes apartados de este capítulo a los 210 primeros experimentos, en los que los parámetros *hardware* de la *cache* se han mantenido constantes, excepto el tamaño de *cache*.

Cada experimento se ha llevado a cabo por duplicado, utilizando en el primer caso un planificador de prioridades fijas, y en el segundo caso el planificador EDF. Aunque se ha intentado mantener las características de los experimentos sin variación para ambos planificadores, la naturaleza de los planificadores ha forzado la modificación de los valores de los periodos para algunos experimentos, y en especial de los plazos de entrega de algunos sistemas en función del planificador utilizado.

Característica	**Valor**
Número de *cargas*	30
Número de experimentos	210 (x2 políticas de planificación)
Número de tareas en cada sistema	Mínimo 3 Máximo 8
Tamaño de *cache*	Mínimo 1KB Máximo 64KB
Tamaño de tarea	Mínimo 2KB Máximo 32KB
Tamaño del sistema (suma de las tareas del sistema)	Mínimo 8KB Máximo 64KB
Instrucciones ejecutadas por cada tarea	Mínimo 50,000 Máximo 8,000,000
Instrucciones ejecutadas por cada sistema	Mínimo 200,000 Máximo 10,000,000
Tamaño de línea de *cache*	16 bytes (4 instrucciones)
Tiempo de acierto (T_{hit})	1 ciclo de procesador
Tiempo de fallo (T_{miss})	10 ciclos de procesador
Modelo de *cache* convencional (algoritmo LRU de reemplazo)	Correspondencia directa, asociativa de 2 y 4 vías, y completamente asociativa
Modelo de *locking cache*	Correspondencia directa
Políticas de planificación	Prioridades fijas y EDF
Ejecuciones y simulaciones para cada experimento	4 simulaciones *cache* convencional, una simulación *locking cache* y una ejecución algoritmo genético

Tabla 8. Resumen de las principales características de los experimentos desarrollados.

2.5.3 Resultados para planificador de prioridades fijas

Para evaluar la exactitud en la estimación del tiempo de respuesta realizada por los métodos presentados en este trabajo se ha ejecutado el algoritmo genético para cada uno de los 210 experimentos, considerando una *locking cache* con correspondencia directa. Una vez obtenido el conjunto de bloques de memoria principal que el algoritmo ha seleccionado para su carga y bloqueo en *cache*, se han simulado los experimentos utilizando una *locking cache* precargada con dichos contenidos. La exactitud en la estimación se evalúa comparando los tiempos de respuesta estimados por el algoritmo para cada tarea frente a los obtenidos de la simulación, y que teóricamente corresponden con los tiempos de respuesta reales. Para cada experimento se ha simulado el hiperperiodo completo, definido como el mínimo común múltiplo de los periodos de las tareas que forman la carga, y que permite considerar todas las interacciones que las tareas pueden producir entre sí, ya que este hiperperiodo comprende todas las activaciones necesarias para alcanzar, nuevamente, la situación inicial desde la que partió la simulación. De esta forma se garantiza que se obtendrá el peor tiempo de respuesta de cada tarea, independientemente de en que activación se produzca. El error en dichos tiempos de respuesta se define según la siguiente ecuación. Puesto que teóricamente se ha verificado que este error será siempre por exceso, es decir, el tiempo de respuesta estimado será siempre superior al real, este error recibe el nombre de *sobreestimación* (S). Este valor se ha calculado en porcentaje, mediante la siguiente formula:

$$S = \frac{TRE}{TRS} - 1$$

donde TRE es el tiempo de respuesta estimado por el algoritmo genético y TRS es el tiempo de respuesta obtenido por la simulación del sistema sobre la *locking cache*. Cuanto mayor es el valor de S mayor es el error cometido al estimar el tiempo de respuesta. Valores de S negativos indicarían que el algoritmo genético ha estimado un tiempo de respuesta inferior al real, situación que, en principio, y debido al análisis conservador realizado, no debería suceder.

La Figura 15 muestra el histograma de frecuencias para diferentes intervalos de *sobreestimación*. El eje Y indica el numero de tareas cuya *sobreestimación* se encuentra en el intervalo correspondiente del eje X. Así, se puede observar que sólo para 1 tarea se ha cometido un error entre el 5% y el 1% al estimar su tiempo de respuesta. Los resultados de esta figura confirman que en ningún caso el tiempo de respuesta estimado es inferior al tiempo de respuesta real, puesto que los resultados obtenidos del algoritmo genético son una cota superior, y por tanto un valor seguro para realizar el análisis de planificabilidad. También, en dicha figura, se puede observar que para todas las tareas, el error cometido al estimar su tiempo de respuesta es menor del 5%.

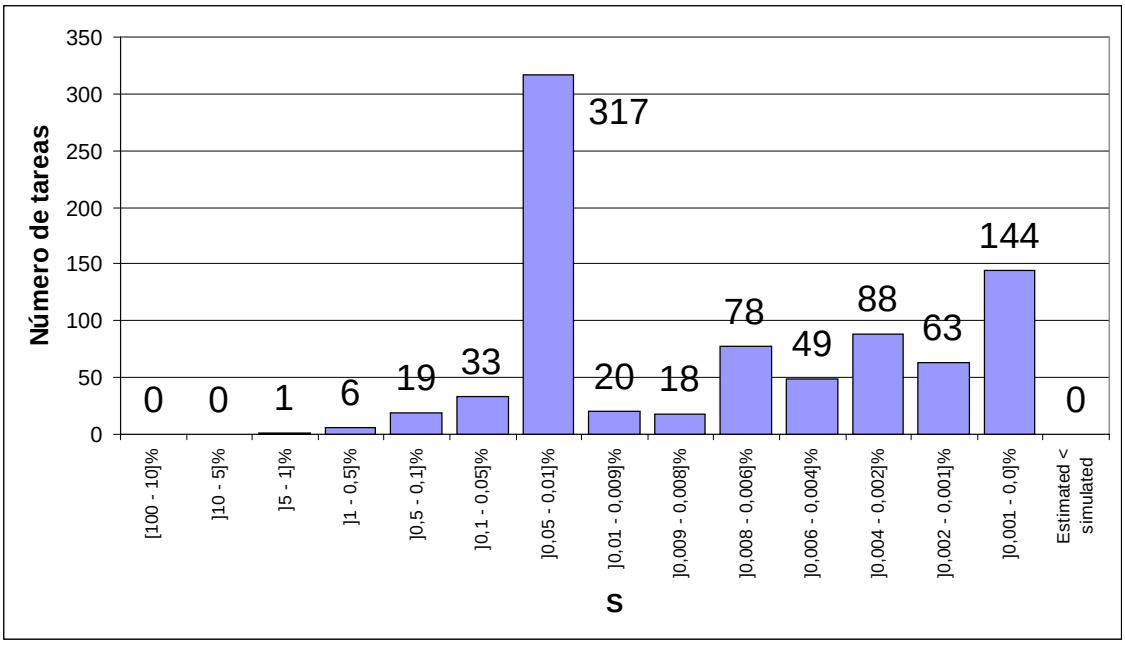


Figura 15. Histograma de frecuencias para la *sobreestimación*. Cada columna indica el número de tareas con un valor de S comprendido en el intervalo correspondiente del eje X. Prioridades fijas.

La Figura 16 presenta el histograma de frecuencia acumulada, en porcentaje, para diferentes valores de sobreestimación. El valor del eje X indica el porcentaje de tareas para los que se ha obtenido una sobreestimación mayor o igual al indicado por el valor correspondiente del eje Y. En esta figura se puede observar que para algo menos del 10% de las tareas, el error cometido es superior o igual al 0,05%. Sin embargo, es más útil la lectura inversa (100-Frecuencia), de forma que en la figura se observa que para más del 90% de las tareas (100 - 10) el error cometido es menor de un 0,05%, y que para más del 50% de las tareas el error es menor del 0,01%, mostrando el alto grado de exactitud de los métodos de análisis presentados y el gran determinismo proporcionado por el uso estático de *locking cache*. Los errores introducidos en el cálculo del WCET y en el análisis de planificabilidad debidos a la existencia del buffer temporal son, como muestran los resultados, mínimos.

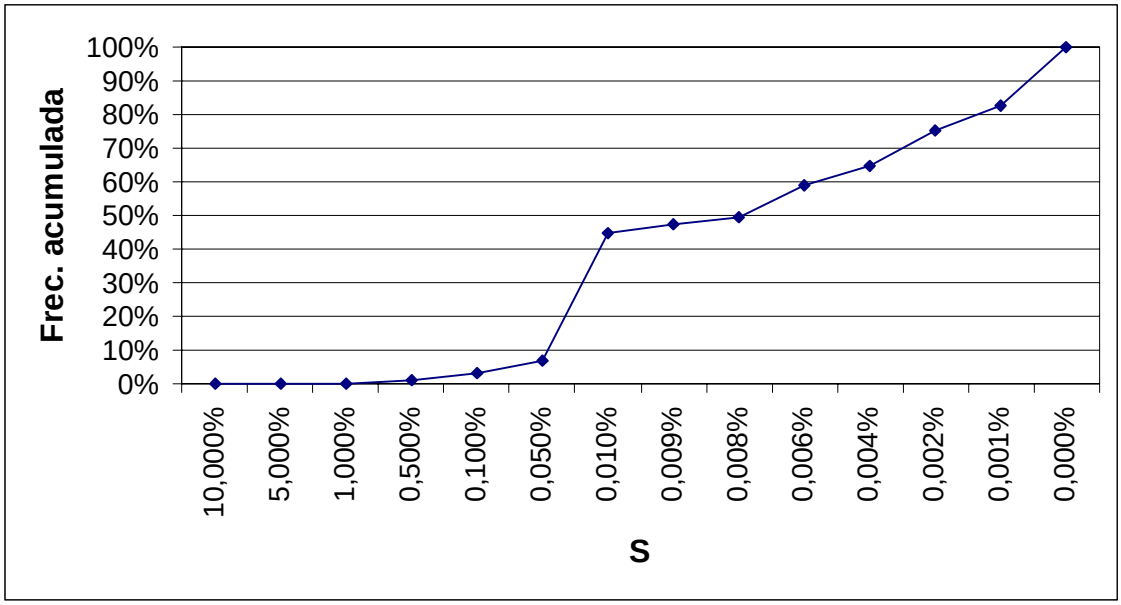


Figura 16. Histograma de frecuencias acumuladas para la *sobreestimación*. El eje Y indica el porcentaje de tareas con un valor de S igual o mayor que el valor indicado en el eje X. Prioridades fijas.

Para evaluar la posible pérdida de prestaciones al utilizar de forma estática la *locking cache* frente a la utilización de *caches* convencionales, se ha comparado el tiempo de respuesta de cada tarea estimado por el algoritmo genético frente al obtenido de la simulación del sistema, comprendiendo el hiperperiodo, y utilizando *caches* convencionales. Cada experimento se ha simulado utilizando *caches* convencionales con cuatro funciones de correspondencia: directa, asociativa de dos conjuntos, asociativa de cuatro conjuntos y completamente asociativa. Para realizar la comparación se ha escogido, en cada experimento, los resultados de aquella función de correspondencia que presentaba el menor tiempo de respuesta, presentando de esta forma el caso menos ventajoso para la *locking cache*. Estos tiempos de respuesta se han comparado con los obtenidos por el algoritmo genético, y no con los obtenidos de la simulación de la *locking cache*, que como se ha visto en las figuras anteriores, son un poco menores. Esto es debido a que el diseñador de sistemas de tiempo de real sólo puede confiar en los valores estimados, que se tratan de una cota segura, y no de los simulados, que debido a la dificultad de encontrar la peor ruta de ejecución o los datos de entrada que producen el peor tiempo de respuesta, podrían no corresponder con el peor caso.

Para comparar las prestaciones se ha utilizado el ratio de prestaciones o *speed-up* al utilizar una *locking cache* frente a la mejor función de correspondencia de *cache* convencional. El ratio de prestaciones se ha calculado como el cociente entre el tiempo de respuesta simulado para *cache* convencional y el tiempo de respuesta estimado por el algoritmo genético. Valores superiores a 1 indican que el uso estático de *cache* proporciona mejores prestaciones al presentar un tiempo de respuesta menor. Por el contrario, valores inferiores a 1 indican que el uso estático de *cache* conlleva una pérdida de prestaciones, al ser su tiempo de respuesta mayor que el obtenido al utilizar una *cache* convencional.

La Figura 17 presenta el histograma de frecuencias para diferentes intervalos del ratio de prestaciones. El eje Y indica el número de tareas que presentan un *speed*-up que se encuentra en el intervalo indicado por el eje X. En la figura se puede apreciar que existe una gran variabilidad en los resultados obtenidos, con un número importante de tareas con un valor de *speed-up* tanto menor que 1 como mayor que 1. Pese a esta variabilidad es notable el elevado número de tareas con un *speed-up* comprendido en el rango]1 - 1,1], es decir con una leve mejora de prestaciones a favor del uso estático de *locking* cache.

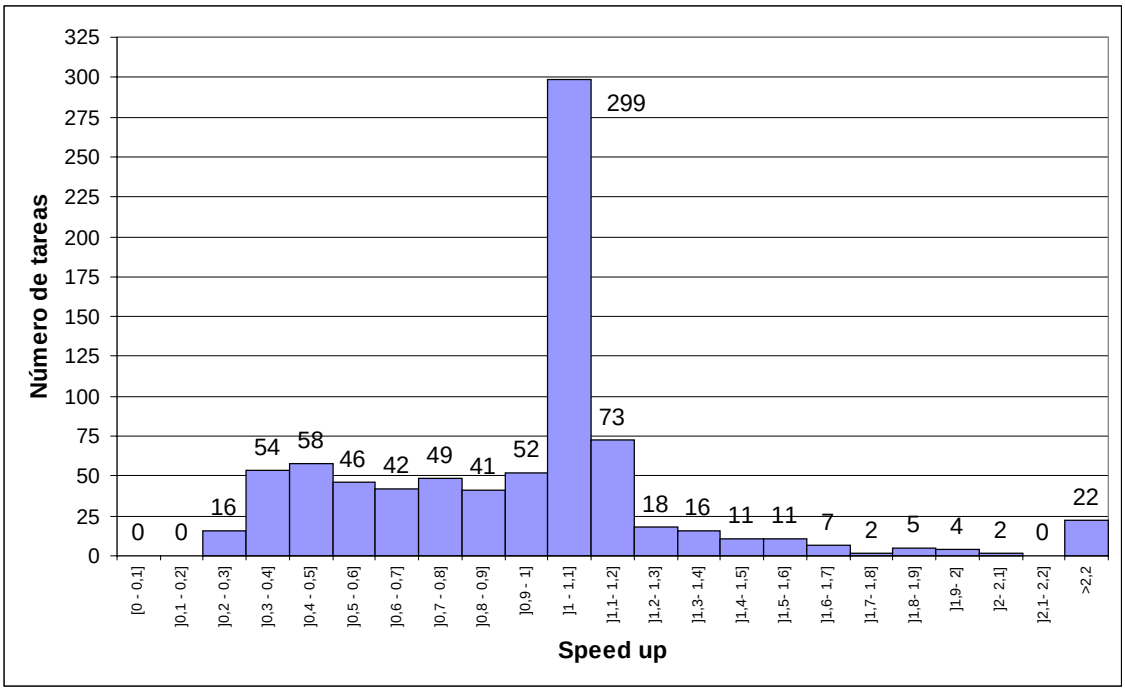


Figura 17. Histograma de frecuencias para la variación de prestaciones. Cada columna indica el número de tareas con un *speed-up* comprendido en el intervalo indicado por el eje X. Prioridades fijas.

La Figura 18 presenta el histograma de frecuencia acumulada, en porcentaje, para diferentes intervalos de *speed-up*. El valor del eje X indica el porcentaje de tareas que presentan un *speed-up* inferior o igual al correspondiente valor del eje X. En la figura se puede observar que para poco más del 40% de las tareas, el *speed-up* es menor o igual a 1, o lo que es lo mismo, que para casi el 60% de las tareas el *speed-up* es superior a 1, lo que significa que el uso estático de *locking cache* proporciona determinismo sin pérdida de prestaciones frente al uso de *caches* convencionales para la mayoría de las tareas.

La utilización del tiempo de respuesta individual de cada tarea para comparar las prestaciones obtenidas al utilizar la *locking cache* no ofrece una visión global del comportamiento de dicha propuesta, ya que un estudio más detallado de los resultados muestra que, en un mismo experimento, el *speed-up* obtenido por cada tarea es diferente, dándose, en múltiples ocasiones, situaciones contrarias, en las que unas tareas obtienen un *speed-up* mayor que 1, es decir, reducen su tiempo de respuesta, y otras tareas obtienen un *speed-up* menor que 1, es decir, ven aumentado su tiempo de respuesta. Para presentar una visión más completa de los resultados es necesario utilizar otra métrica que aglutine en un solo valor la ganancia o pérdida de prestaciones de todas las tareas que forman un experimento. Aunque existen múltiples métricas [Hennesy96] como la media aritmética, media ponderada o la media geométrica de los tiempos de respuesta, la métrica más utilizada habitualmente y que permite ser interpretada con mayor facilidad es la utilización total del procesador. Esta métrica no incorpora información sobre la planificabilidad del sistema o lo próximo que se encuentra el tiempo de respuesta de cada tarea a su plazo de entrega, por lo que no es una métrica indicada para evaluar un sistema de tiempo real, pero sí es apropiada para comparar las prestaciones obtenidas al utilizar dos arquitecturas *hardware* para un sistema de tiempo real una vez se conoce que en ambos casos el sistema es planificable. Además, esta métrica ofrece información interesante, ya que permite conocer de forma aproximada cuanto tiempo de procesador queda disponible para la ejecución de tareas esporádicas no críticas como gestión, reconfiguración u obtención de estadísticas del funcionamiento del sistema.

Aun más apropiada que la utilización del sistema para comparar dos sistemas de tiempo real es la utilización planificable. Se define la utilización planificable como la máxima utilización del

procesador que se puede alcanzar manteniendo la planificabilidad del sistema. La utilización planificable es una cota superior de la utilización real del sistema, ya que esta utilización planificable se calcula considerando que cada ejecución de cada tarea en el sistema se producirá con el máximo y peor posible tiempo de respuesta, mientras que en la realidad, y debido a las relaciones entre los periodos de las tareas, los tiempos de respuesta en cada activación de una tarea pueden variar.

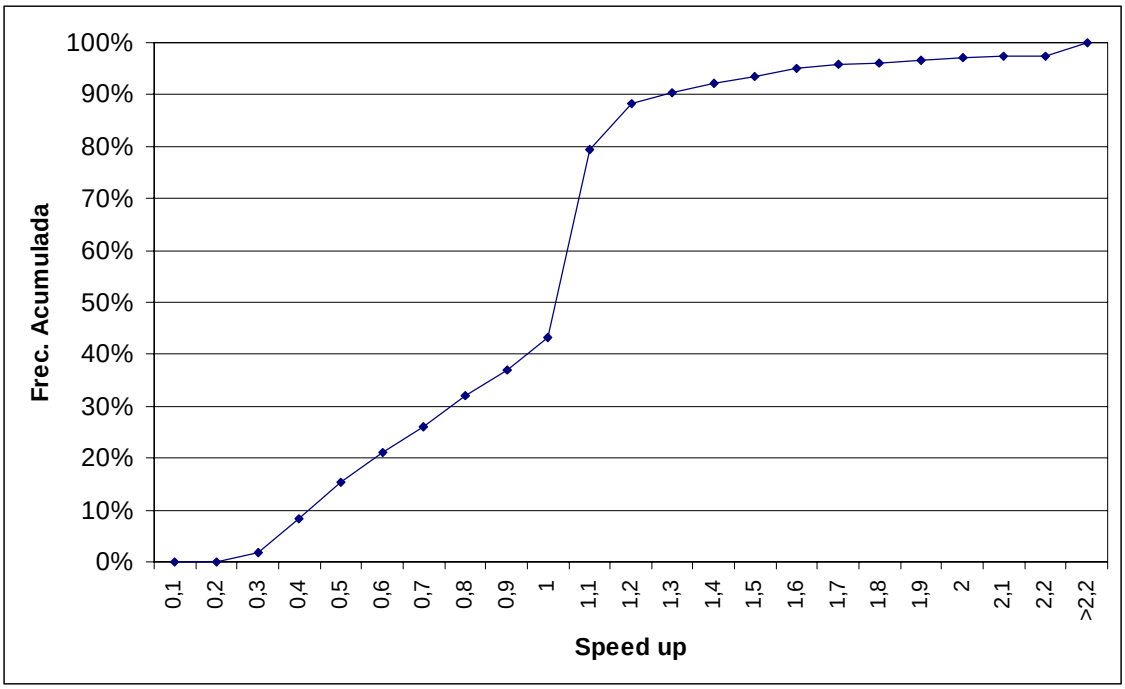


Figura 18. Histograma de frecuencias acumuladas para la variación de prestaciones. El eje Y indica el porcentaje de tareas con un *speed-up* menor o igual que el valor indicado por el eje X. Prioridades fijas.

La utilización planificable proporcionada por el uso estático de la *locking cache* (U_l) se obtiene de los tiempos de respuesta estimados por el algoritmo genético. De esta forma, y puesto que los tiempos de respuesta incorporan los efectos de la interferencia intrínseca y extrínseca, estos efectos serán también incluidos en la utilización planificable resultante. La utilización del sistema es calculada habitualmente como el sumatorio de C_i/T_i, donde C_i es el tiempo de ejecución de la tarea τ_i, y T_i es el periodo de la tarea τ_i. Sin embargo, C_i, que representa el WCET de la tarea τ_i incorpora tan solo el efecto de la interferencia intrínseca, ya que se calcula considerando que la tarea se ejecuta aislada y sin expulsiones. Para incluir el efecto de la interferencia extrínseca, se define la utilización planificable del sistema con *locking cache* mediante la siguiente ecuación:

$$U_l = \sum_{1}^{n} \frac{C_i'}{T_i}$$

donde *n* es el número de tareas en el sistema, T_i es el periodo de la tarea τ_i y C'_i es el tiempo de ejecución de la tarea τ_i considerando tanto la interferencia intrínseca como la extrínseca, es decir, el tiempo de procesador que la tarea τ_i consume en recargar la *cache* tras una expulsión.

El valor de C'_i se obtiene a partir del tiempo de respuesta de τ_i y de los tiempos de respuesta y ejecución de las tareas más prioritarias. Es importante remarcar que en el tiempo de respuesta R_i se encuentra incluido el efecto de la interferencia extrínseca, ya que este tiempo de respuesta ha sido estimado por el algoritmo genético utilizando la expresión modificada de CRTA y considerando el valor del *cache-refill penalty*. Sin embargo, no puede utilizarse de forma

inmediata el valor de R_i como tiempo de ejecución C'_i de la tarea τ_i ya que en este tiempo de respuesta se encuentran incluidas las ejecuciones de las tareas más prioritarias. Para calcular correctamente el valor de C'_i se parte de la ecuación de RTA:

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil x C_j$$

e intercambiando los términos y utilizando el valor estimado por el algoritmo genético para R_i, se obtiene el valor de C'_i:

$$C'_1 = R_1$$

$$C'_i = R_i - \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil x C_j$$

esto es, el tiempo de ejecución de la tarea τ_i es su tiempo de respuesta menos el tiempo que otras tareas de mayor prioridad han estado ejecutándose desde la activación hasta la finalización de la tarea τ_i. Como el valor de R_i utilizado es el estimado por el algoritmo genético, y este valor incorpora el *cache-refill penalty*, también lo incorpora C'_i.

Para obtener la utilización planificable del sistema cuando se emplean *caches* convencionales (U_c) se procede de igual manera que la descrita para la obtención de U_l, pero en ese caso, los valores de R_i empleados en las ecuaciones anteriores corresponden al peor tiempo de respuesta de cada tarea obtenido de la simulación de la ejecución del sistema sobre la *cache* convencional. Al igual que en el caso anterior, el valor de R_i incluye tanto el efecto de la interferencia intrínseca como el de la interferencia extrínseca. El empleo de la utilización obtenida directamente de la simulación representaría la utilización real del sistema, pero no sería el valor de la utilización planificable, por lo que no sería posible comparar dicha utilización con la estimada para el uso de *locking cache*. Es por esta razón que dicho resultado de la simulación se desecha. La siguiente ecuación indica cómo calcular el valor de U_c, pero en este caso C'_i se calculado a partir de los tiempos de respuesta de la simulación del sistema (siempre simulando el hiperperiodo completo).

Para comparar las prestaciones del uso estático de *locking cache* frente a las *caches* convencionales se define el valor Prestaciones (π) como el cociente de la utilización planificable para *cache* convencional entre la utilización planificable para *locking cache* ($\pi = U_c/U_l$). Valores de π mayores que 1 indican que el uso estático de *locking cache* presenta mejores prestaciones que la *cache* convencional al ofrecer una utilización planificable menor.

La Figura 19 muestra el histograma de frecuencias para diferentes intervalos de π. El eje Y indica el número de experimentos que presentan un valor de π que se encuentra en el intervalo indicado por el eje X.

La Figura 20 presenta el histograma de frecuencia acumulada, en porcentaje, para diferentes intervalos de π. El valor del eje X indica el porcentaje de tareas que presentan un valor de π inferior o igual al correspondiente valor del eje X. En la figura se puede observar que para poco menos del 40% de los experimentos existe una pérdida de prestaciones ($\pi \leq 0{,}9$) al utilizar *locking cache*, o lo que es lo mismo, que para algo más del 60% de los experimentos, el uso estático de *locking cache* proporciona determinismo sin pérdida de prestaciones, y que para algo más del 50% de los experimentos el determinismo se consigue al mismo tiempo que se mejoran las prestaciones ($\pi > 1{,}0$). La curva es muy similar a la presentada en la Figura 18 donde se comparaban las prestaciones considerando las tareas de forma individual, pero en este caso existe menos variabilidad, y los valores extremos de π son menores.

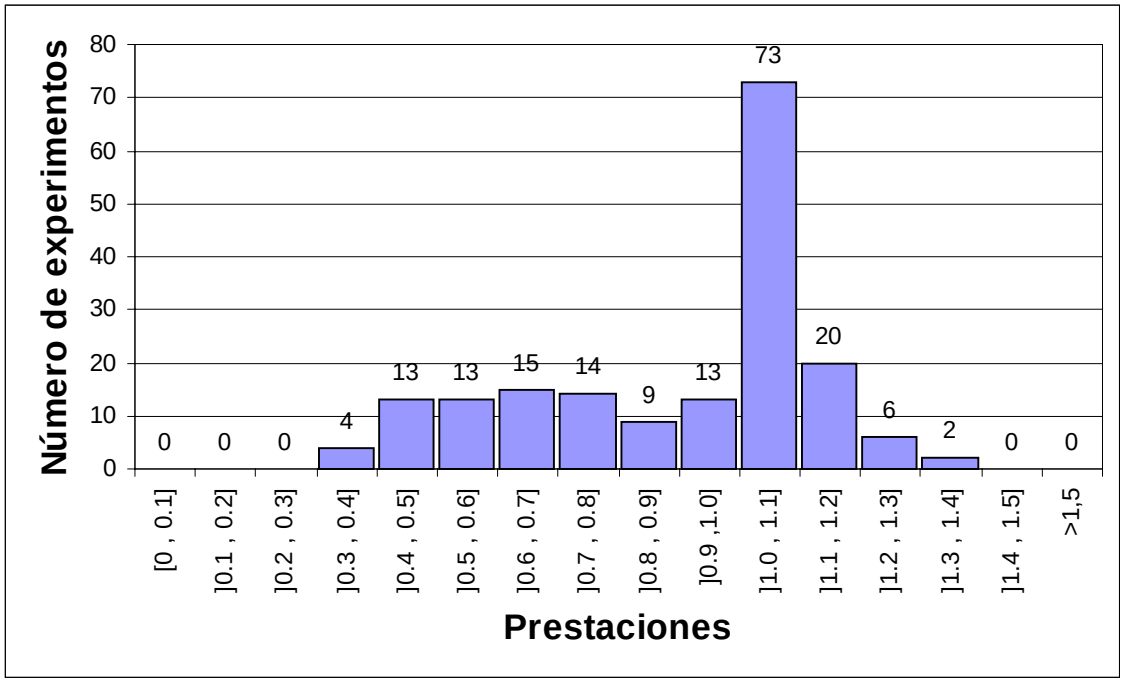


Figura 19. Histograma de frecuencias para U_c/U_l. Cada columna indica el número de sistemas con un valor comprendido en el intervalo indicado por el eje X. Prioridades fijas.

Figura 20. Histograma de frecuencias acumuladas para U_c/U_l. El eje Y indica el porcentaje de sistemas con un valor menor o igual que el valor indicado por el eje X. Prioridades fijas.

La comparación de las prestaciones utilizando valores simulados para las *caches* convencionales representa el caso más desfavorable para la *locking cache*, ya que estos valores son los mínimos posibles, mientras que para la *locking cache* se utilizan valores estimados, que como se ha visto en las figuras anteriores, representan una cota superior y no las prestaciones reales que se obtienen con dicha arquitectura. La otra alternativa, realizar una estimación de los tiempos de respuesta o de la utilización obtenida con una *cache* convencional, a través de alguno de los métodos descritos en el capítulo anterior, inclinaría la balanza a favor de la *locking cache*, ya

que todas las técnicas de análisis estático de *cache* cometen un error, que aunque en la mayoría de los casos es mínimo, empeoraría las prestaciones proporcionadas por las *caches* convencionales.

2.5.4 Resultados para planificador de prioridades dinámicas, EDF

Los mismos experimentos que se han realizado para evaluar la exactitud de las estimaciones y la pérdida o ganancia de prestaciones del uso estático de *locking cache* con un planificador de prioridades fijas, se han llevado a cabo para sistemas con planificador de prioridades dinámicas EDF. Tanto el algoritmo genético como el simulador se han adaptado a esta nueva política de planificación, tal como se ha descrito a lo largo del presente capítulo. La principal diferencia en los experimentos realizados para EDF radica en el tipo de resultado obtenido. Mientras que para los experimentos con prioridades fijas se ha obtenido el tiempo de respuesta de cada tarea y la utilización planificable del procesador, para los experimentos realizados con planificador EDF sólo se ha obtenido la utilización global del procesador. Esto es debido a que el método utilizado para realizar el análisis de planificabilidad, basado en el ICI, realiza dicho análisis en función de la utilización del sistema, y no de los tiempos de respuesta como hace CRTA.

De este modo, para comparar la exactitud de los resultados estimados por los métodos de análisis propuestos y que calcula el algoritmo genético, se ha comparado la utilización total indicada por el algoritmo genético frente a la utilización total resultante de simular la ejecución del experimento sobre una *locking cache*, en la que se han precargado y bloqueado los bloques de memoria principal indicados por el algoritmo genético. En este caso, y al contrario que en los experimentos realizados para la política de planificación por prioridades fijas, no se trata de la utilización planificable, es decir, no hablamos de la máxima utilización para la cual el sistema sigue siendo planificable, sino de la utilización que realmente se hace del procesador en el peor escenario posible. La utilización estimada, U_e, calculada por el algoritmo genético, corresponde con una cota superior de la utilización real que se obtendría al utilizar una *locking cache* con los bloques escogidos cargados en *cache*, ya que los métodos de análisis se han diseñado con este objetivo. La utilización simulada, U_{sl}, corresponde con la utilización real que se obtendría al ejecutar realmente el código sobre una *locking cache*, ya que se ha simulado completamente el hiperperiodo, y los datos de entrada a cada tarea son aquéllos que proporcionan el peor tiempo de ejecución. La utilización de código sintético permite conocer este conjunto de datos de entrada.

La Figura 21 presenta el histograma de frecuencias de la *sobreestimación* cometida en la utilización calculada por el algoritmo genético U_e frente a la utilización obtenida de la simulación de la *locking cache* U_{sl}. Esta *sobreestimación* se ha definido de igual manera como se hizo para prioridades fijas: $S = (U_e/U_{sl}) - 1$. El valor del eje X indica el número de experimentos cuya *sobreestimación* se encuentra en el intervalo indicado por el eje Y. Cuanto mayor es el valor de S mayor es el error cometido al estimar el tiempo de respuesta. Valores de S negativos indicarían que el algoritmo genético ha estimado un tiempo de respuesta inferior al real, situación que, en principio, y debido al análisis conservador realizado, no debería suceder, como muestra el histograma. En la figura se puede observar que el mayor error cometido no supera el 0,5%, y que sólo para 4 experimentos la *sobreestimación* es igual o superior al 0,1%.

La Figura 22 presenta el histograma para la frecuencia acumulada, en porcentaje, para diferentes valores de S. El valor del eje X indica el porcentaje de experimentos para los que la *sobreestimación* al estimar la utilización del sistema es mayor o igual que el valor correspondiente del eje Y. En esta figura se puede observar que para menos del 10% de los experimentos el error cometido es mayor del 0,05%, o lo que es lo mismo, para más del 90% de los casos la *sobreestimación* no alcanza el 0,05%, lo que muestra el alto grado de exactitud de los métodos de análisis presentados y el gran determinismo proporcionado por el uso estático de *locking cache*. Los errores introducidos en el cálculo del WCET y en el análisis de planificabilidad debidos a la existencia del buffer temporal son, como muestran los resultados, mínimos.

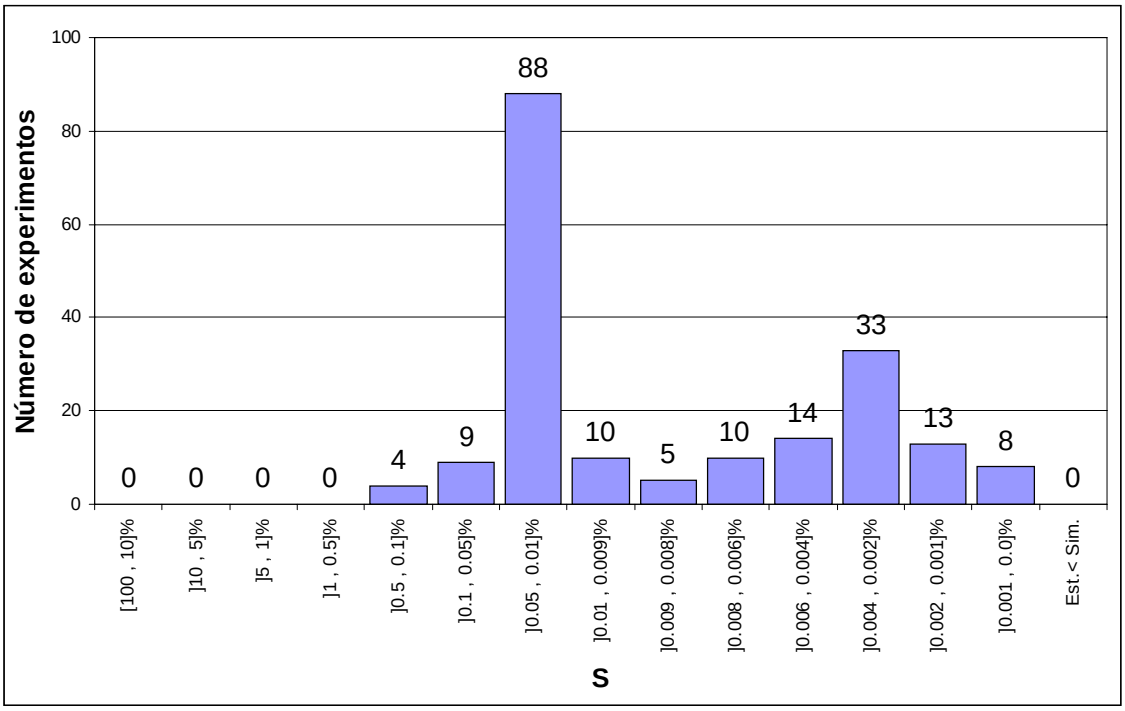


Figura 21. Histograma de frecuencias para la *sobreestimación*. Cada columna indica el número de sistemas con un valor de S comprendido en el intervalo correspondiente del eje X. EDF.

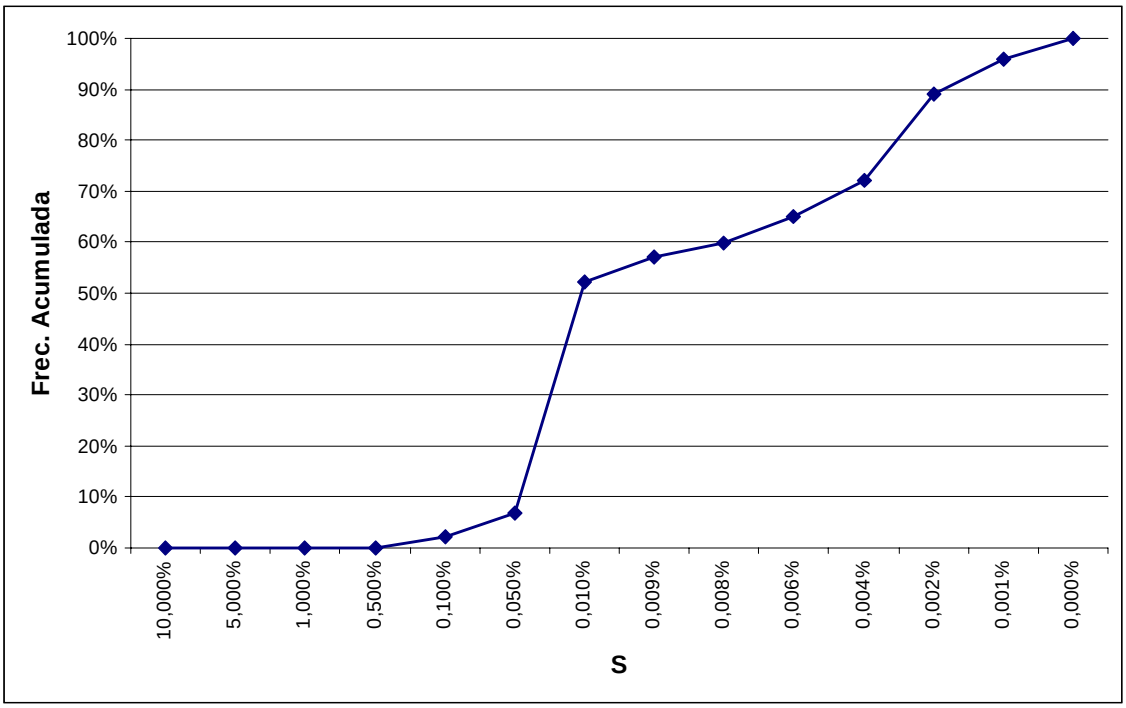


Figura 22. Histograma de frecuencias acumuladas para la *sobreestimación*. El eje Y indica el porcentaje de sistemas con un valor de S igual o mayor que el valor indicado en el eje X. EDF.

La evaluación de la posible pérdida de prestaciones al utilizar *locking cache* frente al empleo de *caches* convencionales se ha realizado comparando la utilización proporcionada por el algoritmo genético, y que corresponde con una cota superior y segura de dicho valor real, frente a la utilización proporcionada por la simulación de la ejecución de cada experimento sobre una *cache* convencional. Al igual que en el caso de prioridades fijas, se han utilizado cuatro

funciones de correspondencia -directa, asociativa de dos conjuntos, asociativa de cuatro conjuntos y completamente asociativa- escogiendo aquella función que proporcionaba, en cada caso, la menor utilización, es decir, el caso más desfavorable para la *locking cache*. Asimismo, se ha simulado el hiperperiodo completo utilizando como datos de entrada para cada tarea aquéllos que proporcionaban el peor tiempo de ejecución. La comparación se ha realizado utilizando el ratio de prestaciones (π), calculado como el cociente entre la utilización resultante de la simulación de *cache* convencional y la utilización estimada por el algoritmo genético (U_{sc}/U_e). Valores superiores a 1 indican que el uso estático de *cache* proporciona mejores prestaciones al presentar una utilización menor. Por el contrario, valores inferiores a 1 indican que el uso estático de *cache* conlleva una pérdida de prestaciones, al ser su utilización mayor que la obtenida al utilizar una *cache* convencional. La Figura 23 presenta el histograma de frecuencias para diferentes intervalos del ratio de prestaciones. El eje Y indica el número de experimentos cuyo valor de π se encuentra en el intervalo indicado por el eje X. En la figura se puede apreciar que existe una gran variabilidad, con experimentos cuyo valor de π es menor que 0,4 -pérdida de prestaciones importante al utilizar *locking cache*- como experimentos con valores de π superiores a 1,3 -ganancia de prestaciones al utilizar *locking cache*-. Significativo es el gran número de experimentos que se sitúan en el intervalo]0,9;1,1], es decir, no hay ni pérdida ni ganancia de prestaciones.

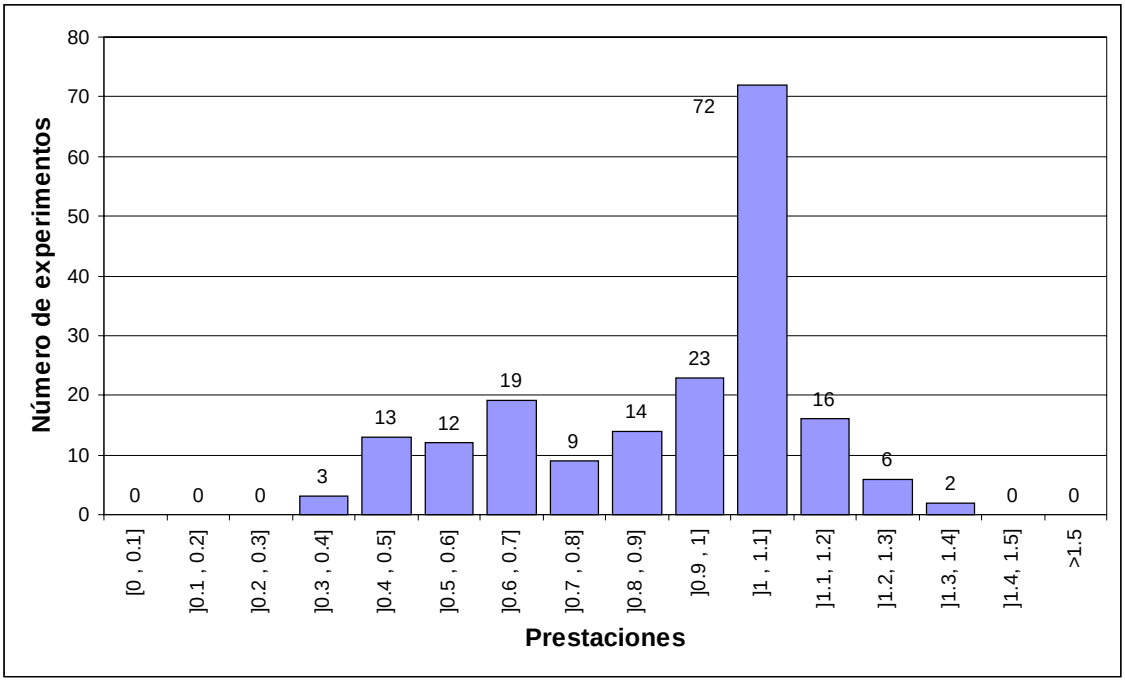


Figura 23. Histograma de frecuencias para U_{sc}/U_e. Cada columna indica el número de sistemas con un valor comprendido en el intervalo indicado por el eje X. EDF.

La Figura 24 presenta el histograma de frecuencia acumulada, en porcentaje, para diferentes intervalos de π. El valor del eje X indica el porcentaje de tareas con un valor de π menor o igual al correspondiente valor del eje X. En la figura se puede observar que sólo para un 10% de los experimentos las prestaciones están por debajo de 0,5, y que para más del 60% de los casos, no existe pérdida de prestaciones o ésta es mínima ($\pi > 0,9$).

El comportamiento del uso estático de *locking cache*, en cuanto a prestaciones, es bastante similar para los dos tipos de planificadores utilizados. En el capítulo 4 se realiza una comparación más detallada de los resultados obtenidos para ambos planificadores.

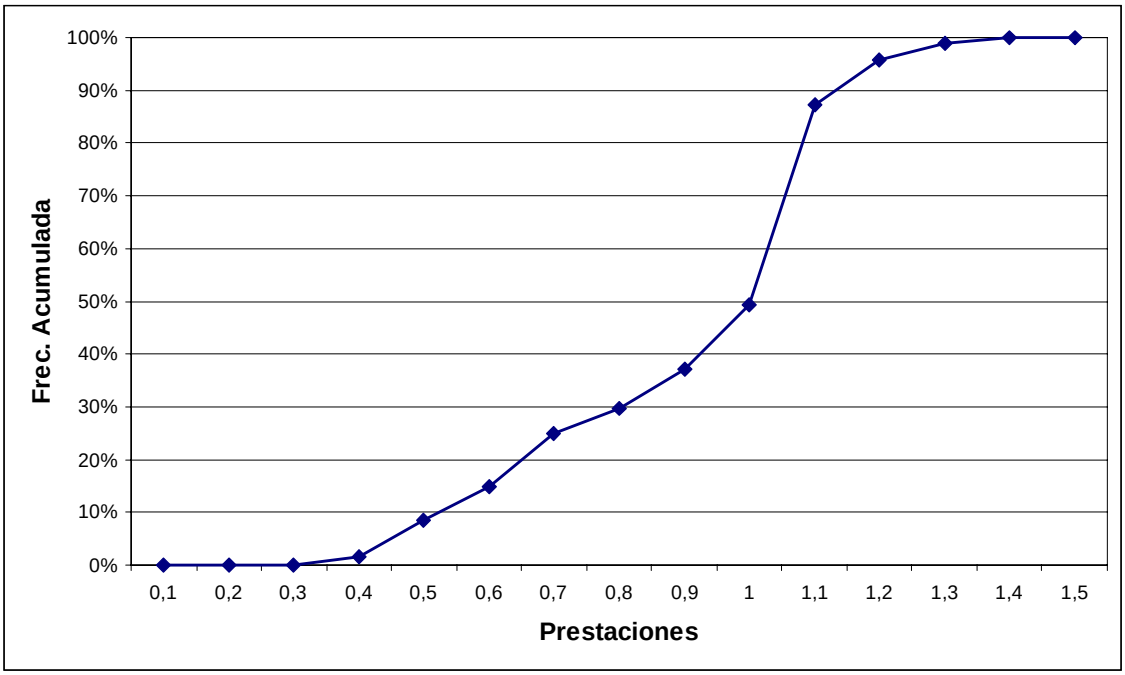


Figura 24. Histograma de frecuencias acumuladas para U_{sc}/U_e. El eje Y indica el porcentaje de sistemas con un valor menor o igual que el valor indicado por el eje X. EDF.

2.6 Conclusiones del capítulo

En este capítulo se ha presentando una arquitectura de *cache*, llamada *cache* con bloqueo o *locking cache*, y un modo de utilizarla, que recibe el nombre de *uso estático*, que permite convertir la memoria *cache* de los computadores modernos en un sistema completamente determinista. La arquitectura y su uso propuesto guardan semejanza con otras propuestas, como son las *scratch pad memory* y la *partición hardware*, pero presenta ventajas importantes respecto a ellas:

- La utilización de una scratch pad memory permite obtener el mismo resultado final que el uso estático de una *locking cache*, pero presenta mayores inconvenientes a la hora de diseñar el sistema. La *scratch pad memory* forma parte del espacio de memoria principal del computador, por lo que para acceder a sus contenidos el procesador debe generar una serie de direcciones concretas. Así, si se desea cargar en esta *scratch pad memory* un conjunto concreto de instrucciones, están deben ser reubicadas, habitualmente por el compilador, dentro del espacio global de memoria para que se sitúen, físicamente, en el espacio correspondiente a la scratch pad memory. Por el contrario, las memorias *cache* disponen de una circuitería *hardware* que, a través de la función de correspondencia, traduce las direcciones de memoria principal a direcciones de *cache*, por lo que el código puede ser generado sin necesidad de considerar la existencia de la memoria *cache*, ya que la traducción de direcciones se realiza en tiempo de ejecución y de forma transparente al programador y al compilador.
- La partición *hardware* de *cache* presenta una desventaja fundamental frente la utilización estática de *caches* con bloqueo, y es la falta de determinismo para el cálculo del WCET de las tareas. Puesto que cada tarea utiliza libremente la partición o particiones que tiene asignadas, se mantiene e incluso aumenta la interferencia intrínseca, por lo que es necesario recurrir a complejos métodos de análisis para estimar un valor seguro y ajustado de dicho WCET. Por el contrario, el uso estático de *cache* con bloqueo elimina esta interferencia, permitiendo la obtención del WCET de las tareas, necesario para realizar el análisis de planificabilidad, de una forma casi trivial.

Para la propuesta realizada se han presentado los correspondientes métodos para la estimación del WCET de las tareas y la realización del análisis de planificabilidad, tanto para políticas de planificación basadas en prioridades fijas como en el caso de prioridades dinámicas basadas en EDF. El análisis se circunscribe a memoria *cache* de instrucciones y sistemas formados por tareas periódicas, sin considerar otras mejoras estructurales como la predicción de saltos o la ejecución especulativa, con el objeto de evaluar los resultados proporcionados por el uso estático de *locking cache* sin interferencias o efectos derivados de otras estructuras distintas a las memorias *cache*.

Los experimentos realizados permiten comprobar los dos grandes logros alcanzados a través de la arquitectura y su utilización propuesta en este capítulo: la obtención de un sistema determinista, y el mantenimiento de las prestaciones frente a la utilización de memorias *cache* convencionales no deterministas.

El máximo error cometido en la estimación de los tiempos de respuesta de las tareas o de la utilización del procesador se sitúa alrededor del 1 por cien, y para más del 50 por cien de los casos, la *sobreestimación* es inferior al 0,05%. Además, los valores obtenidos son seguros, es decir, suponen una cota superior a los valores reales, por lo que pueden utilizarse con total garantía para identificar el peor caso posible en la ejecución del sistema mediante la utilización de algoritmos muy sencillos.

En cuanto a las prestaciones, en más del 60 por cien de los experimentos la pérdida de prestaciones no supera el 10 por cien, por lo que se puede considerar que, en general, no existe pérdida de prestaciones al emplear el uso estático de *locking cache*. Sólo en un porcentaje muy pequeño de los casos, la pérdida de prestaciones es significativa respecto a la utilización de *caches* convencionales.

Capítulo 3

Uso dinámico de memorias cache con bloqueo

La propuesta descrita en este capítulo recibe el nombre de uso dinámico en contraposición a la propuesta realizada en el capítulo anterior. Al igual que en el uso estático de la *locking cache*, se aprovecha la capacidad de estas memorias *cache* de precargar sus contenidos y mantenerlos sin cambios, evitando que se produzcan reemplazos. Sin embargo, esta situación de carga estática de los contenidos de la *cache* no se mantiene durante todo el funcionamiento del sistema tal como sucede en el uso estático, sino que, en determinados instantes de la ejecución del sistema, los contenidos de *cache* son reemplazados por otros nuevos, manteniéndose sin variación hasta que se alcanza un nuevo instante de cambio de contenidos. El principal objetivo perseguido con este uso dinámico es mejorar las prestaciones obtenidas al utilizar *caches* con bloqueo, manteniendo al mismo tiempo un nivel alto de determinismo que permita utilizar técnicas sencillas para realizar el análisis de planificabilidad.

La mejora de prestaciones se obtiene al permitir que los contenidos de *cache* se modifiquen, adaptándose a las necesidades de la tarea en ejecución de forma similar a cómo se comportan las memorias *cache* convencionales. El determinismo del sistema se logra definiendo y determinando los instantes de tiempo en que dichos contenidos se modificarán, y también en la especificación estricta de cuáles serán los contenidos que se cargarán en la *cache* en cada instante. Los dos objetivos son antagónicos, ya que cuanto mayor sea la frecuencia con que se producen los reemplazos en la memoria *cache,* mejores prestaciones se obtendrán, pero también será más complejo determinar el comportamiento del sistema, y por tanto estimar de forma ajustada los tiempos de ejecución y de respuesta de las tareas. Las memorias *cache* convencionales son uno de los extremos de este binomio, mientras que el uso estático de *locking cache* es el otro extremo.

Con el uso dinámico de la *locking cache* se pretende diseñar un sistema que se encuentre en el punto de equilibrio, ofreciendo las ventajas en cuanto a prestaciones de las *caches* convencionales y el determinismo proporcionado por la *locking cache*. Para alcanzar este equilibrio, los contenidos involucrados en cada reemplazo serán estáticos y especificados durante la etapa de diseño, y los instantes de tiempo en que se cargará la memoria *cache* y se bloqueará para mantenerla sin variación corresponden con el inicio de la ejecución de una tarea –no con su activación- y con la reanudación de la ejecución de una tarea tras una expulsión. De esta forma, cuando la tarea entra en ejecución, los contenidos de la *cache* son conocidos, ya que han sido explícitamente precargados, y se mantendrán sin variación hasta que la tarea abandone el procesador, bien por finalizar su ejecución, bien por ser expulsada por otra tarea de mayor prioridad. Con este modo de operar se logra que cada tarea disponga de forma exclusiva de toda la *locking cache*, frente al uso estático, donde el espacio de *cache* debía ser compartido por todas las tareas. El resultado es que se aumenta el tamaño de *cache* disponible para cada tarea, por lo que las prestaciones obtenidas serán mejores.

Al igual que para el uso estático, la propuesta presentada en este capítulo considera únicamente el diseño y análisis de *cache* de instrucciones.

El uso dinámico de la *locking cache* elimina completamente la interferencia intrínseca, ya que una vez cargadas y bloqueadas las instrucciones de una tarea, éstas no podrán ser desplazadas de la *cache* por la ejecución de otras instrucciones de la misma tarea. Sin embargo sí existe interferencia extrínseca, ya que la entrada en ejecución de una tarea conlleva una modificación de los contenidos de la memoria *cache*. La subsistencia de esta interferencia extrínseca incluirá cierta complejidad en el análisis de planificabilidad, y como mostrarán los resultados de los experimentos, introducirá cierta inexactitud en la estimación de los tiempos de respuesta de las

tareas. Otra consecuencia de la existencia de esta interferencia extrínseca es la inconveniencia de la aplicación del uso dinámico de *locking cache* en planificadores dinámicos como es EDF, por los motivos que se exponen en el apartado 3.6. Por ello, los métodos de análisis que se describirán, así como los experimentos realizados y los resultados obtenidos se circunscriben a sistemas de tiempo real con planificador de prioridades fijas.

Parte del trabajo presentado en este capítulo ha sido publicado en [Martí02]

3.1 Memorias cache con bloqueo (locking cache)

Aunque el principal objetivo del uso dinámico de *locking cache* es la mejora de prestaciones frente al uso estático, la obtención de determinismo sigue siendo un requisito imprescindible. Es por ello que las características de la *locking cache* siguen siendo las mismas que se describieron en el capítulo anterior, y que vuelven a detallarse a continuación:

- La memoria *cache* de instrucciones puede ser completamente bloqueada, es decir, no se producirá ninguna nueva carga ni se modificará su contenido una vez la *cache* haya sido bloqueada.
- El bloqueo de la *cache* se realizará mediante la ejecución de una instrucción específica.
- Los contenidos de la *cache* podrán seleccionarse con granularidad de línea, ejecutando una instrucción específica e indicando la dirección del bloque de memoria principal que se desea cargar.
- Existirá un buffer temporal, con tamaño igual al de una línea de *cache*, y que se comportará de igual forma que una memoria *cache* convencional de instrucciones de una sola línea, reemplazando sus contenidos de forma dinámica.
- Si el procesador referencia una instrucción que se encuentra en la *cache*, y ésta ha sido bloqueada, esta instrucción se servirá desde *cache* con el mismo tiempo que si la *cache* no estuviera bloqueada.
- Si el procesador solicita una instrucción que se encuentra en el buffer temporal, ésta será servida desde dicho buffer en el mismo tiempo que se hubiera servido desde memoria *cache*.
- Si el procesador solicita una instrucción que no se encuentra en memoria *cache* ni en el buffer temporal, esta instrucción será servida desde memoria principal, con el tiempo de acceso correspondiente. Además, el buffer temporal será rellenado con el bloque de memoria principal correspondiente a la instrucción solicitada.
- La función de correspondencia de la *cache* puede ser cualquiera.

Para permitir la utilización dinámica de este esquema de *cache* es necesario realizar una pequeña ampliación a dicho esquema, así como aumentar la funcionalidad del planificador, que puede involucrar, ligeramente, al sistema operativo utilizado, aunque este último punto es sumamente dependiente de la implementación final que se realice del método propuesto. A continuación se describen las ampliaciones al esquema de *locking cache* y las funcionalidades requeridas al planificador:

- La memoria *cache* puede ser desbloqueada y vaciada –invalidando sus contenidos- mediante la ejecución de instrucciones *software*.
- Cada vez que el planificador inicie la ejecución de una nueva tarea, precargará la *cache* con un conjunto de bloques de memoria principal predefinido para dicha tarea, bloqueando la *cache* una vez precargada.
- Cada vez que el planificador reanude la ejecución de una tarea tras una expulsión, precargará la *cache* con un conjunto de bloques de memoria principal predefinido para la tarea expulsada, bloqueando la *cache* una vez precargada.

- Los bloques que deben precargarse para cada tarea se identificarán mediante las correspondientes direcciones de memoria principal, y se almacenarán en una lista en memoria. Esta lista recibe el nombre de Tabla de Bloqueo de Tarea (TBT).
- El planificador mantendrá, en las estructuras de datos necesarios, la dirección de inicio de la TBT de cada tarea y el número de bloques que componen dicha TBT.
- La precarga de la *cache* se realizará mediante la ejecución de un sencillo bucle, que básicamente leerá las direcciones contenidas en la TBT y ejecutará, para cada dirección, la instrucción de llenado de la *cache*. Una vez cargados en *cache* todos los bloques especificados en la TBT, la *cache* será bloqueada.
- Para evitar interferencias en el uso de la *cache* entre el bucle de precarga y los contenidos que deben cargarse en *cache*, el primero de ellos puede ubicarse en un espacio de memoria que no pueda copiarse a memoria *cache*, aunque puede hacer uso del buffer temporal.

La principal modificación radica en la necesidad de poder desbloquear la *cache* para rellenarla con nuevos contenidos, permitiendo así que cada tarea disponga de su conjunto de instrucciones en *cache*, que permanecerá invariable, cada vez que inicie o reanude su ejecución, eliminando de este modo la interferencia intrínseca. En cuanto a la interferencia extrínseca, no es eliminada, pero sí es conocida ya que todas las tareas deberán, tras una expulsión, recargar la *cache* con las instrucciones indicadas en la TBT.

3.2 Cálculo del WCET.

Aunque el uso dinámico de *cache* con bloqueo elimina por completo la interferencia intrínseca, es necesario realizar el cálculo del WCET de las tareas considerando la existencia de esta particularidad *hardware*. El cálculo el WCET de cada tarea se realiza suponiendo que la tarea se ejecuta aislada del resto de las tareas y por tanto sin expulsiones. Puesto que la tarea, siempre que entra en ejecución, se encuentra el mismo estado de *cache*, es decir, las mismas instrucciones, la situación es equivalente a la descrita en el uso estático. Por ello, el cálculo del WCET de cada tarea, considerando un uso dinámico de *locking cache*, se realiza de forma idéntica al descrito en el apartado 2 del capítulo anterior. Esta equivalencia es completa, incluyendo las técnicas de modelado, la estimación de tiempos, la sobreestimación cometida y las posibles mejoras y ampliaciones, presentando una sola modificación. Esta modificación es debida a la variación del instante en que la *cache* es precargada.

Tal como se ha descrito en el modo de operación del uso dinámico de *locking cache*, cada tarea, al iniciar su ejecución, debe cargar en la *cache* sus propios contenidos. Esta carga se realiza al menos una vez por cada activación de la tarea, y aunque es realizada por el planificador, el número de bloques cargados, y por tanto el tiempo necesario para llevar a cabo la operación depende de cada tarea. A todos los efectos, se puede considerar que este proceso de carga pertenece al start-up de la tarea, es decir, a la ejecución de código necesaria para preparar la ejecución de la tarea. Es por ello que el tiempo necesario para cargar la *cache* debe considerarse como tiempo de ejecución de la tarea, ya que el proceso se realizará cada vez que se produzca una nueva instanciación de la tarea, independientemente del tipo de planificador, prioridad de la tarea o número de tareas que formen el sistema. Este tiempo, que recibe el nombre de Tiempo de Carga (T_{carga}) debe sumarse al resultado de la evaluación de la expresión de la tarea para obtener el correcto WCET de la tarea. Si llamamos T_{ejec} al tiempo de ejecución en el peor de los casos, obtenido de la expresión del CCFG tal como se describió en el capítulo anterior, el WCET de la tarea será la suma de T_{carga} más T_{ejec}. La Figura 25 representa visualmente el WCET de una tarea cuando se emplea el uso dinámico de *locking cache* frente al WCET de la tarea cuando se emplea el uso estático de *locking cache*.

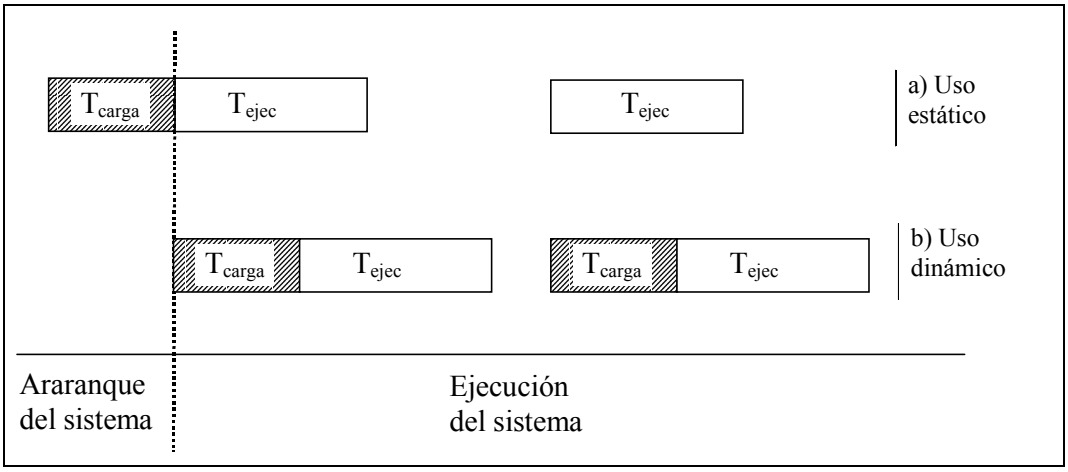


Figura 25. Instante de carga de la *locking cache* para uso estático y uso dinámico

3.3 Análisis de planificabilidad.

En este apartado se presenta el análisis de planificabilidad para sistemas con política de planificación basada en prioridades fijas, y que utilizan de forma dinámica una *locking cache*.

Al igual que para el uso estático de *locking cache*, el análisis propuesto se basa en la utilización de CRTA, cuya ecuación ya ha sido detallada anteriormente.

$$w_i^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^n}{T_j} \right\rceil x(C_j + \gamma_j)$$

El efecto de la *cache* se incorpora en los parámetros C_i y γ_j. El primero de ellos incorpora a la ecuación el efecto de la interferencia intrínseca, y corresponde al WCET de la tarea suponiendo que ésta se ejecuta aislada del resto de las tareas pero considerando la existencia de la *locking cache* y su uso dinámico. Este valor se obtienen mediante el CCFG y la expresión que lo representa, tal como se indica en el apartado anterior. El segundo parámetro, γ_j, incorpora a la ecuación el efecto de la interferencia extrínseca, y corresponde con el aumento en el tiempo de respuesta que una tarea τ_i sufre por la modificación de los contenidos de la memoria *cache*, modificación realizada por la ejecución de una tarea τ_j de mayor prioridad. Este tiempo recibe el nombre de *cache-refill penalty* o *cache*-related preemption delay.

Aunque ya se han descrito a lo largo de esta tesis los dos tipos de expulsiones o interferencia extrínseca que puede sufrir una tarea, debido a la importancia que estos conceptos tienen a la hora de realizar el análisis de planificabilidad cuando se realiza un uso dinámico de la *locking cache*, se volverán a describir con detalle.

Una tarea puede sufrir durante su ejecución dos tipos de expulsión, directa o indirecta, que producirán a su vez dos tipos de interferencia extrínseca, llamadas también directa e indirecta. Por expulsión directa se entiende que, durante la ejecución de la tarea τ_i, se produce la activación de otra tarea τ_j, de mayor prioridad, y que por tanto desplaza del uso del procesador a la tarea τ_i. Estas expulsiones directas producen una interferencia que se denomina interferencia directa. Por expulsión indirecta se entiende que, durante la ejecución de una tarea τ_i se produce la activación de otra tarea τ_k de mayor prioridad que desplaza a la primera del uso del procesador, pero que a su vez, esta tarea τ_k es expulsada y desplazada del procesador por otra tarea τ_j. De este modo, la tarea inicial τ_i parece sufrir dos expulsiones, pero sólo ha sido desplazada del procesador una sola vez, por lo que sólo ha sufrido en realidad una expulsión efectiva. La segunda activación recibe el nombre de expulsión indirecta, y produce lo que se denomina interferencia indirecta.

Aunque CRTA contabiliza de forma muy precisa el número de expulsiones que sufre una tarea, no indica cuál es el tipo de expulsión ni el tipo de interferencia que produce cada una de ellas. Por ello, para realizar una estimación ajustada, pero sobre todo segura, es decir, para obtener una cota superior del tiempo de respuesta de la tarea τ_i, es necesario calcular el valor de γ_j considerando los dos tipos de interferencia, optando por el peor de los casos, es decir, por el mayor de ellos en cada expulsión.

Cuando se pone en práctica el uso dinámico de *locking cache*, el valor de γ_j es, en función del tipo de interferencia, el siguiente:

- Interferencia directa: tras una expulsión directa, la tarea expulsada reanudará su ejecución, recargando, previamente, los contenidos de *cache*. Por tanto, el tiempo de penalización sufrido por la tarea τ_i, expulsada por la ejecución de la tarea τ_j, depende de los contenidos que deba recargar la tarea expulsada, y se define como γ_j igual a T_{carga_i}. Es decir, el *cache-refill penalty* sufrido por la tarea τ_i debido a la ejecución de la tarea τ_j es igual al tiempo necesario para recargar los contenidos de la *cache* de la tarea τ_i.
- Interferencia indirecta: si una tarea τ_i sufre una expulsión indirecta, quiere decir que ha sido realmente expulsada por otra tarea de mayor prioridad, τ_k, que a su vez sufrirá una expulsión directa por la ejecución de otra tarea τ_j de mayor prioridad todavía. La tarea de menor prioridad τ_i verá incrementado su tiempo de respuesta por la ejecución de la tarea τ_k y también por el tiempo que la tarea τ_k necesita para recargar la *cache* tras la expulsión, es decir, por la propia interferencia directa de la tarea τ_k. Por tanto, la interferencia indirecta γ_j sufrida por la tarea τ_i, suponiendo que la tarea τ_j expulsa de forma directa a una tarea τ_k de prioridad intermedia es el tiempo de carga de la tarea τ_k, es decir, T_{carga_k}.

Para obtener el valor final del *cache-refill penalty* γ_j sufrido por la tarea τ_i debido a la ejecución de la tarea τ_j, es necesario considerar, primero, que se desconoce el tipo de interferencia que la tarea de mayor prioridad produce, y segundo, que una tarea sólo puede realizar una expulsión efectiva, aunque a cualquier tarea de menor prioridad. La primera consideración forzará a estimar el peor caso, buscando el máximo valor posible de γ_j, que corresponderá al tiempo de recarga de la tarea expulsada, o al tiempo de recarga de alguna de las tareas de mayor prioridad que pueden ser a su vez expulsadas. La segunda consideración delimita los casos a considerar. Puesto que sólo se produce una expulsión, sólo deberá considerarse una vez el peor caso posible. Este peor caso corresponde con la expulsión de la tarea de prioridad menor que la expulsante y mayor o igual que la tarea para la que se intenta estimar su tiempo de respuesta que presente el mayor tiempo de recarga de la *cache*.

Es importante explicar con detalle el significado de la segunda consideración. Si la tarea τ_i sufre una expulsión o interferencia extrínseca indirecta quiere decir que se han producido al menos dos activaciones de tareas más prioritarias. Habrá sufrido una expulsión directa y efectiva por la tarea τ_k, y otra expulsión indirecta y no efectiva por la tarea τ_j que expulsa a la anterior. Esto puede llevar a la conclusión de que el valor final de la interferencia para la tarea τ_i es la suma de todas las interferencias de las tareas de mayor prioridad. Efectivamente esto es así, pero esta suma se realiza mediante el sumatorio de la ecuación del CRTA, y no es necesario, es más, representaría una doble contabilización, incluirlo en el cálculo del *cache-refill penalty* de la tarea τ_i.

La siguiente ecuación muestra el cálculo de la interferencia sufrida por la tarea τ_i debido a la activación de la tarea τ_j. El valor de esta interferencia se define como γ_j^i donde el índice j es la tarea que provoca la expulsión y el índice i es la tarea que sufre la expulsión. Se especifican los dos índices ya que el valor del *cache-refill penalty* depende no sólo de la tarea que genera la expulsión sino también de la tarea que es expulsada, es decir, la interferencia extrínseca que produce una tarea depende de la tarea a la que se le produce dicha interferencia.

$$\gamma_j^i = \underset{j<z\leq i}{MAX}(T_{carga\_z}) + T_{miss}$$

donde i, z, j indican el nivel de prioridad de las tareas, con τ_j mas prioritaria que τ_i si j<i. En esta ecuación se ha añadido, de forma constante, el tiempo de recargar el buffer temporal, ya que siempre existe la posibilidad de que la tarea expulsada se encontrara ejecutando código ubicado en este buffer en el momento de la expulsión, por lo que sería necesario recargar sus contenidos tras la expulsión.

El valor de γ_j^i se incorpora a la ecuación del CRTA del modo que se muestra en la siguiente ecuación:

$$w_i^{n+1} = C_i + \sum_{j\in hp(i)} \left\lceil \frac{w_i^n}{T_j} \right\rceil x(C_j + \gamma_j^i)$$

La Figura 26 muestra los dos tipos de expulsión que puede sufrir una tarea y el coste temporal de cada una de los tipos de interferencia. Cada división representa un tiempo de ejecución de K ciclos de procesador. Según lo mostrado en dicha figura, al estimar el tiempo de respuesta de la tarea τ_3 será necesario calcular el *cache-refill penalty* γ_1^3 provocado por la tarea τ_1 y el *cache-refill penalty* γ_2^3 provocado por la tarea τ_2. En el caso de γ_2^3 su cálculo es inmediato, ya que tanto para el escenario a) como para el escenario b), la ejecución de la tarea τ_2 obliga a la tarea τ_3 a recargar sus contenidos, es decir γ_2^3 será igual a T_{carga_3}. Sin embargo, al calcular el valor de γ_1^3 es necesario diferenciar entre el escenario a) y el escenario b), ya que la penalización que provoca la ejecución de la tarea τ_1 a la tarea τ_3 es diferente en función de a qué tarea expulsa realmente del procesador. Puesto que los dos escenarios son posibles, y se desea una cota superior, el valor de γ_1^3 corresponderá con el peor caso, es decir, aquél que represente un mayor coste temporal. Según lo mostrado en la Figura 26, el valor de γ_1^3 será igual a T_{carga_2} ya que es mayor que T_{carga_3} (3K ciclos y 2K ciclos de procesador, respectivamente). La siguiente ecuación muestra una iteración del cálculo del tiempo de respuesta de la tarea τ_3 según lo descrito anteriormente y el escenario presentado en la Figura 26. Esta ecuación sólo pretende mostrar el cálculo de γ_j^3 integrado en el proceso completo, por lo que se trata simplemente de un ejemplo.

$$w_3^n = C_3 + 1x(C_1 + \gamma_1^3) + 1x(C_2 + \gamma_2^3)$$

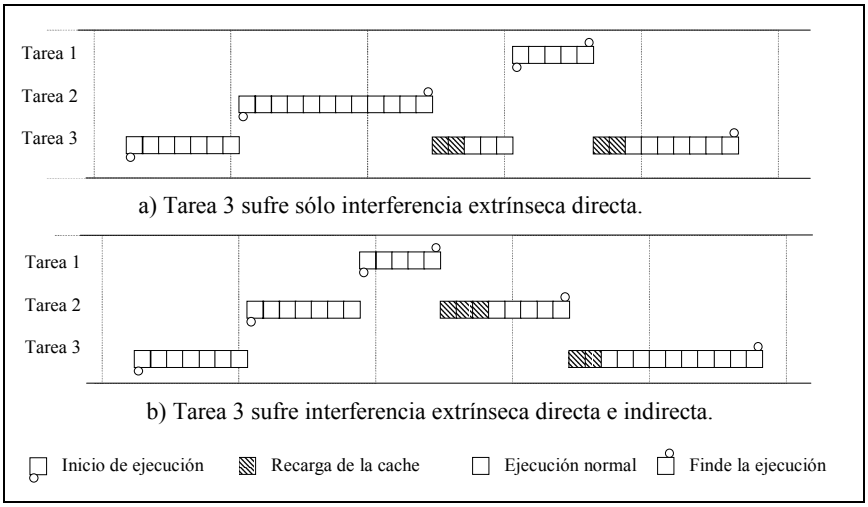


Figura 26. Ejemplos de interferencia extrínseca directa e indirecta.

En el análisis de planificabilidad propuesto existen tres posibles fuentes de error. La primera de ellas se deriva del cómputo del número de expulsiones que puede sufrir una tarea. Aunque CRTA estima este valor de forma muy ajustada, en función de la relación entre los periodos de las tareas puede estimar un número mayor de expulsiones del que realmente sucede, ya que debe considerar la peor secuenciación de las tareas.

La segunda fuente de error es la consideración de que la tarea expulsada siempre se encuentra ejecutando código desde el buffer temporal, lo que no es necesariamente cierto. Sin embargo, al ser tan bajo el tiempo necesario para recargar el buffer temporal, este error es casi inapreciable.

La tercera fuente de error surge de la imposibilidad de diferenciar si una expulsión produce interferencia directa o indirecta. El número de bloques que se recargan en *cache* tras una expulsión puede variar de forma significativa entre las tareas, por lo que considerar siempre el peor caso puede inducir una estimación del *cache-refill penalty* significativamente mayor del que sería su valor real, si este pudiera medirse con seguridad.

Ninguna de las tres fuentes de error descritas tiene relevancia de forma aislada. El error que se muestra en el apartado de experimentos se deriva de la combinación, fundamentalmente, de la primera y la tercera fuente de error. Al estimar un número de expulsiones superior al real, el error en la estimación del *cache-refill penalty* se ve amplificado, por lo que la desviación en el tiempo de respuesta final puede ser importante para las tareas de menor prioridad, en las que el valor estimado para el *cache-refill penalty* puede estar más alejado del real, al existir más tareas de mayor prioridad y por tanto aumentar la posibilidad de cometer un error significativo. Pese a este posible error, el análisis de planificabilidad de *locking cache* presenta como ventajas la sencillez de su realización y la obtención de un valor seguro -una cota superior del tiempo de respuesta de cada tarea- ya que los errores cometidos son siempre por exceso.

3.4 Selección de contenidos para la locking cache.

Al igual que para el uso estático de *locking cache*, en el uso dinámico la selección de las instrucciones que se cargarán y bloquearán en *cache* es fundamental para obtener unas prestaciones similares, o lo más próximas posibles, a las proporcionadas por las *caches* convencionales, al mismo tiempo que se obtiene determinismo. En este caso, el problema es ligeramente diferente, pero no menos complejo que el descrito para el uso estático. En el uso dinámico de *locking cache*, cada tarea puede bloquear tantos bloques de memoria principal como líneas de *cache* existan, sin necesidad de compartir el espacio disponible con otras tareas. Aparentemente, esto simplifica el problema, ya que no aparecen interacciones entre las tareas. Esta falta de interacciones se circunscribe a la asignación del espacio de *cache*, pero se mantiene una fuerte relación entre el número de bloques que una tarea carga y bloquea en *cache* y el tiempo de respuesta del resto de las tareas, en concreto de las tareas de menor prioridad. La existencia de interferencia extrínseca indirecta hace que el tiempo de carga que necesita una tarea influya sobre el tiempo de respuesta de las tareas de menor prioridad. Si una tarea tiene un tiempo de carga de *cache* elevado es porque necesita cargar un gran número de bloques, lo que posiblemente reducirá su tiempo de ejecución, reduciendo al mismo tiempo su tiempo de respuesta. Pero este beneficio puede perderse si existe una tarea de menor prioridad que es expulsada en numerosas ocasiones, ya que tras cada expulsión verá aumentado su tiempo de respuesta de forma exagerada al considerar que sufre el peor caso, es decir, que sufre un elevado número de expulsiones para las cuales la interferencia, tanto indirecta como directa, representa un elevado coste temporal.

Para lograr el equilibrio entre una reducción significativa del WCET de la tarea y un aumento mínimo en los tiempos de respuesta debido a las expulsiones, se utilizan nuevamente los algoritmos genéticos para encontrar una solución subóptima con un coste computacional aceptable. El algoritmo genético desarrollado para el uso dinámico de *locking cache* es extremadamente similar al desarrollado para el uso estático, por lo que a continuación se describen de forma resumida los elementos y operadores del algoritmo, detallando solamente

aquéllos que presentan diferencias importantes respecto a la versión descrita en el capítulo anterior.

La nueva versión del algoritmo genético proporciona como resultados, al igual que la versión anterior, el WCET de cada tarea, el tiempo de respuesta de cada tarea y el conjunto de bloques de memoria principal que cada tarea debe cargar y bloquear en *cache*.

3.4.1 Codificación

La codificación utilizada para representar la solución del problema es la misma que en la versión del algoritmo desarrollada para el uso estático. Un individuo se representa mediante un vector de elementos binarios, donde cada elemento -gen- representa el estado -bloqueado o no- de los diferentes bloques de memoria principal ocupados por todas las tareas del sistema. El modo en que se accederá a esta estructura genera la aparición de un nuevo elemento, llamado cromosoma, que no es más que un conjunto de genes cuyo tratamiento en grupo tiene un significado particular. En la codificación utilizada, un cromosoma es el conjunto de genes que representan el estado de los bloques de memoria de una tarea determinada, por lo que existirán tantos cromosomas como tareas en el sistema. La división en cromosomas se trata de una partición lógica, que no modifica la estructura de datos utilizada para codificar las soluciones, sino que se utilizarán un conjunto de índices auxiliares para delimitar los cromosomas. Por ello, el individuo puede ser accedido considerando la existencia de estos cromosomas, o de forma genérica a cualquier gen.

3.4.2 Creación de la población inicial

Tal como se ha comentado al introducir el problema, la mejor solución no tiene porqué ser aquélla que haga el máximo aprovechamiento del espacio de *cache* disponible, ya que esta política podría introducir costes muy elevados para el *cache-refill penalty*. Por ello, la inicialización se realizará mediante secuencias de unos, es decir, secuencias consecutivas de bloques cargados en *cache*, con una longitud máxima igual al tamaño de *cache*, pero con una longitud efectiva aleatoria. Además, estas secuencias se generarán para cada cromosoma, es decir, para cada tarea, ya que en el uso dinámico, cada tarea puede utilizar todo el espacio de *cache*. Esto significa que el número total de bloques que un individuo carga y bloquea en *cache* puede ser claramente superior al número de líneas de *cache*, pero representando una solución completamente válida siempre y cuando el número de bloques para cada cromosoma no exceda el tamaño de la *cache*.

Con los mismos objetivos perseguidos en la resolución del problema para uso estático, se añaden dos soluciones extremo a la población inicial, la primera de ellas sin ningún bloque seleccionado para ninguna tarea, y la segunda con todos los bloques de todas las tareas seleccionados, aumentando de esta forma la variabilidad del algoritmo genético al explorar el espacio de búsqueda.

3.4.3 Función de Fitness

La función de *fitness* empleada es la misma que se definió para el algoritmo genético utilizado en el uso estático de *locking cache*, con la única salvedad de los algoritmos utilizados para la estimación del WCET y el tiempo de respuesta de cada tarea, aplicando los algoritmos descritos en el apartado 3.2 y apartado 3.3 de este capítulo.

Esta función de *fitness* corresponde con la media ponderada de los tiempos de respuesta de cada tarea, asignado un peso creciente según se reduce la prioridad de la tarea, intentando de este modo que la función de *fitness* contemple las interacciones existentes entre las tareas.

Esta función de *fitness* presenta como principal inconveniente la variabilidad en su tiempo de cómputo, siendo estas variaciones debidas a la iteratividad de CRTA. Esto impide acotar el tiempo de respuesta del algoritmo genético, ya que dependerá de los parámetros de entrada del problema.

3.4.4 Selección

El operador de selección, encargado de escoger las mejores soluciones de cada iteración para, a partir de ellas, generar una nueva población de soluciones mediante la aplicación de los operadores de cruce y mutación, se mantiene sin variaciones, definiéndose los cuatro tipos posibles de individuos en función de su tiempo de respuesta medio ponderado y el número de bloques cargados en *cache*, en este caso, número de bloques cargados para cada tarea:

A. Tiempo de respuesta (medio ponderado) finito, lo que indica que el sistema es planificable, y número de bloques cargados en *cache* menor o igual, para todas las tareas, que el número de líneas de la *cache*. Este es un individuo válido.

B. Tiempo de respuesta (medio ponderado) finito –sistema planificable- con número de bloques cargados en *cache* superior al número de líneas de la *cache* para, al menos, una tarea o cromosoma. Este es un individuo no válido, ya que al utilizar más espacio de *cache* del disponible, la solución que representa no es utilizable.

C. Tiempo de respuesta (medio ponderado) infinito, lo que significa que el sistema no es planificable, con un número de bloques cargados en *cache* menor o igual, para todas las tareas, que el número de líneas de *cache*. Este es un individuo válido, aunque es una solución sumamente mala.

D. Tiempo de respuesta (medio ponderado) infinito –sistema no planificable- con número de bloques cargados en *cache* superior al número de líneas de la *cache* para, al menos, una tarea. Además de ser un individuo no válido, se trata de una muy mala solución.

Al igual que en el uso estático, la existencia de individuos no válidos impide la utilización de un esquema de selección basado directamente en el resultado de la función de *fitness*. En lugar de este esquema, los individuos se ordenan en función de su validez y de su valor de *fitness*, asignándoles una probabilidad en función de la posición que ocupan tras esta ordenación. Los individuos válidos ocuparán las posiciones más altas, por lo que se les asignará una mayor probabilidad, mientras que los individuos no válidos ocuparán las posiciones más bajas, obteniendo una probabilidad menor. Los individuos válidos se ordenan entre ellos en función del valor de la función de *fitness*, mientras que para los individuos no válidos se emplea el exceso en el número de líneas de *cache* utilizadas, ocupando las posiciones más levadas los individuos con un exceso menor. La Tabla 9 muestra un ejemplo del resultado de la ordenación para un conjunto de 5 individuos, asumiendo que la *locking cache* dispone de un total de 64 líneas. La columna *número de líneas ocupadas* corresponde con el máximo valor de los diferentes cromosomas que forman el individuo. Las dos últimas columnas muestran la probabilidad asignada a cada individuo en función de su posición y según las siguientes ecuaciones:

$$Probabilidad\ (0) = MP$$

$$Probabilidad\ (i) = MP(1-MP)^i$$

$$Probabilidad\_acumulada\ (0) = MP$$

$$Probabilidad\_acumulada\ (i) = Probabilidad\_acumulada\ (i-1) + Probabilidad\ (i)$$

La probabilidad acumulada del último individuo debe ser 1, valor que se obtiene de forma directa de las ecuaciones presentadas cuando el número de individuos es superior a 150.

Posición	Resultado función de *fitness*	Número de líneas ocupadas	Probabilidad	Probabilidad acumulada
0	100,3	63	0,1	0,1
1	132,5	61	0,09	0,19
2	∞	43	0,081	0,271
3	98,7	65	0,0729	0,3439
4	∞	73	0,06561	0,40951
5	56,2	82	0,059049	0,468559

Tabla 9. Posición y probabilidad asignada a cada individuo en función del valor de la función de *fitness* y del número de líneas de *cache* utilizadas.

3.4.5 Cruce

El operador de cruce, encargado de generar dos nuevos individuos a partir de dos existentes, opera en este caso de igual forma que en el caso del uso estático. Cada "progenitor" se obtiene generando un número aleatorio, y seleccionando aquel individuo cuya probabilidad acumulada es la menor de todas las probabilidades mayores que el valor aleatorio. Una vez seleccionados los dos individuos "padre", se determina aleatoriamente un bit o gen y se divide en dos partes cada individuo. Los nuevos individuos o "hijos" se generan a partir de la combinación cruzada de las dos partes de cada "padre". Este cruce no considera la existencia de cromosomas, es decir, no opera sobre las tareas que forman el sistema, sino sobre la solución completa. Aunque es posible definir un operador de cruce más sofisticado, que tenga en consideración los cromosomas que representan a cada tarea, en la bibliografía asociada [Mitchell96] se presentan múltiples ejemplos que llevan a la conclusión de que un operador de cruce complejo no aporta ninguna mejora a la evolución del algoritmo genético. Intuitivamente, esto es debido a que el objetivo del operador cruce es introducir variabilidad en la generación de nuevas poblaciones, variabilidad que será evaluada por la función de *fitness* y por el operador de selección. Por ello, se ha optado por un operador de cruce sencillo y rápido de implementar, manteniendo de este modo el tiempo de ejecución del algoritmo lo más bajo posible.

3.4.6 Mutación

El operador mutación definido para el uso dinámico de *locking cache* difiere ligeramente del aplicado en el uso estático. Se mantienen los tres casos particulares en función del número de bloques cargados en *cache*, pero considerando la existencia de cromosomas, es decir, la utilización de todo el espacio de *cache* por cada una de las tareas. La mutación se aplica, por tanto, a cada cromosoma del individuo. La razón de operar de este modo es la posibilidad de que, dentro de una misma solución, existan cromosomas válidos y cromosomas inválidos, representando a tareas que bloquean un número de bloques igual o menor que el número de líneas de *cache*, o un número de bloques mayor que el número de líneas de *cache*, respectivamente. Para dirigir la evolución del algoritmo genético hacia individuos completamente válidos es necesario aplicar la mutación considerando las distintas situaciones que se producen dentro de un mismo individuo.

La aplicación de la mutación a cada cromosoma se realiza de una de estas tres posibles formas:

A. Si el número de bloques que una tarea del individuo carga en *cache* es superior al número de líneas, el cromosoma será no válido, por lo que su aportación a la población será mínima. Para intentar mejorar su situación, se escogerán aleatoriamente algunos de los bits que tienen valor 1 y se mutarán al valor 0, reduciendo por tanto el número de bloques cargados en *cache*. Esta operación no afecta a todos los bits ya que el objetivo es mejorar su situación, y no reconvertir el cromosoma en una solución válida.

B. Si el número de bloques cargados en *cache* por la tarea es igual al número de líneas de *cache*, se trata de un cromosoma completamente válido que realiza un aprovechamiento máximo del espacio disponible de memoria *cache*, por lo que, con muy baja probabilidad, se intercambiará alguna pareja de genes con valores distintos, es decir, se sustituirá alguno de los bloques cargados en *cache* por otro, pero sin variar el número total de bloques que la tarea carga en *cache*.

C. Si el número de bloques cargados en memoria *cache* es menor que el número de líneas de la *cache*, se trata de un cromosoma válido, pero que no aprovecha completamente la capacidad de *cache*. Presumiblemente, aumentando el número de bloques cargados mejorará el tiempo de respuesta de alguna tarea, por lo que en este caso la mutación escogerá algún gen que tenga valor cero, y lo mutará al valor uno. Al igual que en el caso A, esta mutación no afectará a todos los cromosomas de este tipo, ni tampoco persigue igualar el número de bloques cargados con el número de líneas de *cache*, sino que intenta mejorar, de forma aleatoria y con una probabilidad de mutación muy baja, alguno de los cromosomas de este tipo.

Una vez realizada la mutación, y aunque no es estrictamente parte de la mutación, se incluye un operador más sobre los nuevos individuos de la población recién creada. Este operador verificará en todos los individuos válidos, es decir, aquéllos para los que todos los cromosomas son válidos, los bloques seleccionados puedan ser cargados en la *locking cache* sin violar la función de correspondencia. En el supuesto de utilizar una *locking cache* con correspondencia directa, este nuevo operador verificará que en ningún caso dos bloques seleccionados en el mismo cromosoma o tarea compiten por la misma línea de *cache*. En caso negativo, uno de los bloques será sustituido, de forma aleatoria, por otro bloque que no presente este inconveniente.

3.4.7 Sintonización de parámetros

Los valores asignados a los diferentes parámetros del algoritmo genético son los mismos que los descritos en el apartado 2.4.7 del capítulo anterior y que se resumen en la Tabla 10.

Parámetro	**Valor**
Tamaño de la población	200 individuos
Número de generaciones	5000
Probabilidad del mejor individuo	0,1
Probabilidad de cruce	0,6
Probabilidad de mutación casos A y C	0,001
Probabilidad de mutación caso B	0,01

Tabla 10. Parámetros del algoritmo genético para la selección de contenidos en uso dinámico de la *locking cache*.

3.5 Experimentos

Tres son los objetivos perseguidos con la realización de los experimentos. Primero, evaluar el grado de predecibilidad obtenido en el uso dinámico de *locking cache*. Segundo, cuantificar la pérdida de prestaciones sufrida al utilizar dinámicamente la *locking cache* frente al uso de *caches* convencionales no deterministas. Y tercero, comparar las prestaciones obtenidas por el uso dinámico frente al uso estático de *locking cache*.

El tercer objetivo ha motivado la utilización del mismo corpus de tareas, sistemas, experimentos y parámetros que el diseñado y utilizado en el uso estático de *locking cache*, facilitando de este modo la comparación de los resultados obtenidos para las dos propuestas de uso de *locking cache*. Aunque existen herramientas estadísticas que permiten comparar muestras distintas,

siempre y cuando los elementos de ambas muestras sean aleatorios, independientes y representativos de la población, la comparación de muestras idénticas es mucho más sencilla, permitiendo el uso del análisis de datos pareados que ofrece resultados muy fiables.

Para la realización de los experimentos es necesario conocer el tiempo de precarga de la *locking cache*, tanto para incorporarlo al WCET de cada tarea como para calcular el *cache-refill penalty* de cada expulsión. Además, y debido a la realización de simulaciones, es necesario diseñar un código funcional que realice esta carga de contenidos de forma correcta, para poder medir su tiempo de ejecución e incluirlo en los tiempos de respuesta de cada tarea simulada. La Figura 27 presenta el código del algoritmo desarrollado para el simulador SPIM.

```
    flush           #invalida y desbloquea la cache
ini: addi $1,$1,-1  #se asume que el registro 1 contiene el número
                    de bloques que deben cargarse en cache
    lw $3,0($2)     #se asume que el registro 2 contiene la
                    dirección inicial de la TBT
    lab $3          #carga el bloque de memoria indicado por el
                    registro $3
    addi $2,$2,4    #siguiente entrada de la TBT
    bne $0,$1,ini
    nop
    lock                #bloquea la cache
    jr $31              #retorno desde la rutina
```

Figura 27. Código ensamblador MIPS R2000 de la rutina de carga de la locking *cache*

La rutina de precarga es ejecutada directamente por el simulador cada vez que una tarea entra en ejecución, bien después de una expulsión, o por la activación de la tarea en cada periodo. Aunque la rutina de carga de la *cache* se ejecuta de forma independiente a la tarea, su tiempo de ejecución se suma a la tarea para la cual es ejecutada.

El tiempo de ejecución de esta rutina se estima en 46 ciclos por cada bloque que se debe cargar más 23 ciclos constantes, asumiendo que el tamaño de línea de *cache* es de 4 instrucciones, el tiempo de fallo es de 10 ciclos y el tiempo ejecución de cualquier instrucción desde *cache* o el buffer temporal es de 1 ciclo. Este tiempo total se desglosa de la siguiente manera:

- 11 ciclos de ejecución del primer segmento, de los cuales 10 ciclos corresponden a la transferencia del bloque desde memoria principal a *cache* o al buffer temporal, y 1 ciclo para ejecutar la instrucción de vaciado y desbloqueo de la *cache*.
- 33 ciclos para la ejecución del segundo segmento, que corresponden a 10 ciclos para acceder a las instrucciones, 10 ciclos para leer la dirección del bloque a cargar, 10 ciclos para transferir de forma efectiva el bloque de instrucciones desde memoria principal a memoria *cache*, más 3 ciclos para ejecutar las instrucciones. Se ha considerado la peor situación posible: no hay *cache* de datos, por lo que las instrucciones de carga acceden siempre a memoria principal, y en cada iteración del bucle las instrucciones han sido desplazadas del buffer temporal por la ejecución del tercer segmento.
- 13 ciclos para la ejecución del tercer segmento, correspondientes a la trasferencia del código desde memoria principal hasta el buffer temporal más la ejecución de tres instrucciones. El simulador utiliza una mejora estructural llamada *branch delay slot* [Hennessy96], lo que hace que la instrucción *nop* se ejecute siempre, independientemente del resultado de la ejecución de la instrucción *beq* que la precede. En este caso también se asume que los segmentos 2 y 3 compiten siempre por el buffer temporal, reemplazándose mútuamente en cada iteración.
- 12 ciclos correspondientes a la ejecución del 4 segmento.

Las características y detalles de los experimentos pueden encontrarse en el apartado 2.5.2 del capítulo anterior, por lo que a continuación se pasa a detallar los resultados obtenidos. Sólo

recordar que en estos experimentos se realiza la evaluación de distintos sistemas de tiempo real que se ejecutan sobre una *locking cache* de instrucciones haciendo un uso dinámico de la misma.

3.5.1 Análisis del determinismo

Para evaluar el error cometido al estimar el tiempo de respuesta de cada tarea se ha definido este error mediante la siguiente ecuación:

$$S = \frac{TRE}{TRS} - 1$$

donde TRE es el tiempo de respuesta estimado por el algoritmo genético y TRS es el tiempo de respuesta obtenido por la simulación de la *locking cache*. Cuanto mayor es el valor de S mayor es el error cometido al estimar el tiempo de respuesta. Valores de S negativos indicarían que el algoritmo genético ha estimado un tiempo de respuesta inferior al real, situación que, en principio, y debido al análisis conservador realizado, no debería suceder. Por esta razón, este error recibe el nombre de sobreestimación.

La Figura 28 muestra el histograma de frecuencias para diferentes intervalos de sobreestimación S. El eje Y indica el numero de tareas cuya sobreestimación se encuentra en el intervalo correspondiente del eje X. Para un número muy importante de tareas la sobreestimación no sobrepasa el 1%, aunque también es importante el número de tareas con una sobreestimación mayor, llegando, para 12 tareas, a superar el 30% de error. Un aspecto importante es que en ningún caso el error es negativo, es decir, para todas las tareas el tiempo de respuesta estimado es igual o superior al simulado.

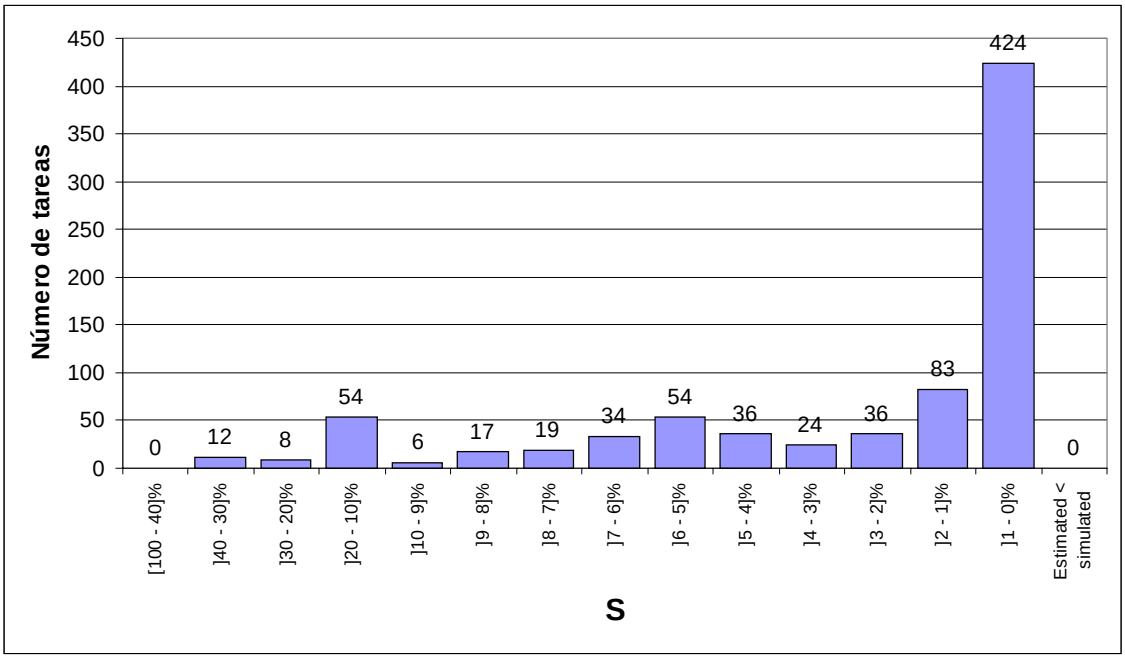


Figura 28. Histograma de frecuencias para la sobreestimación. Cada columna indica el número de tareas con un valor de S comprendido en el intervalo correspondiente del eje X. Uso dinámico.

La Figura 29 presenta la frecuencia acumulada, en porcentaje, para diferentes valores de sobreestimación. El valor del eje X indica el porcentaje de tareas para los que se ha obtenido una sobreestimación mayor o igual al indicado por el valor correspondiente del eje Y. En esta figura se puede observar que para más del 50% de las tareas el error cometido al calcular su tiempo de respuesta es menor de un 1 por cien, y que para aproximadamente el 90% de las

tareas el error cometido no alcanza el 9%, mostrando el alto grado de exactitud de los métodos de análisis presentados y el gran determinismo proporcionado por el uso estático de *locking cache*. Sin embargo, las fuentes de error descritas en los apartados anteriores, especialmente durante el cálculo del *cache-refill penalty* hacen que en un 10% de los casos la sobreestimación cometida sea igual o superior al 10%.

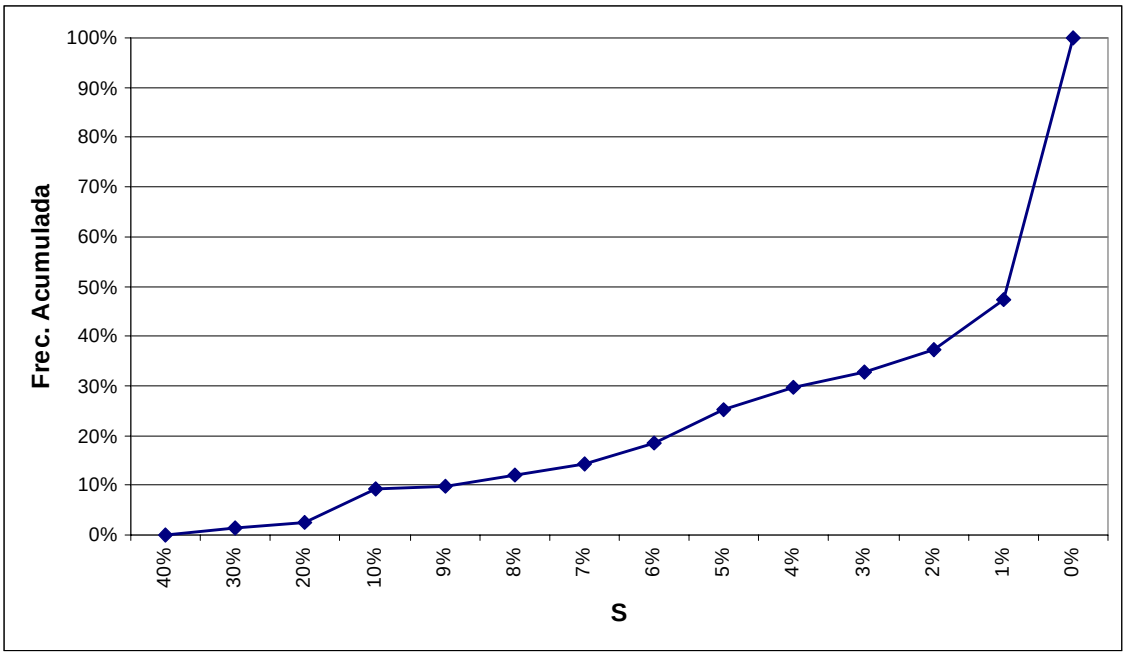


Figura 29. Histograma de frecuencias acumuladas para la sobreestimación. El eje Y indica el porcentaje de tareas con un valor de S igual o superior que el correspondiente del eje X. Uso dinámico.

3.5.2 Análisis de prestaciones

La evaluación del coste del determinismo proporcionado por el uso dinámico de *locking cache*, es decir, la pérdida de prestaciones sufrida, se ha realizado comparando los tiempos de respuesta de cada tarea estimados por el algoritmo genético con los tiempos de respuesta obtenidos de la simulación de una *cache* convencional. Se han utilizado cuatro funciones de correspondencia para *caches* convencionales –directa, asociativa de dos conjuntos, asociativa de cuatro conjuntos y completamente asociativa- escogiendo para cada experimento aquélla que presentaba mejores resultados, es decir, menor tiempo de respuesta. Para la comparación se ha utilizado el ratio de los tiempos de respuesta o *speed-up* calculado como el cociente entre el tiempo de respuesta estimado por el algoritmo genético y el obtenido de la simulación de la *cache* convencional. Valores superiores a 1 indican que el uso dinámico de *locking cache* proporciona mejores prestaciones al presentar un tiempo de respuesta menor. Por el contrario, valores inferiores a 1 indican que el uso dinámico de *locking cache* conlleva una pérdida de prestaciones, al ser su tiempo de respuesta mayor que el obtenido al utilizar una *cache* convencional.

La Figura 30 presenta el histograma de frecuencias para diferentes intervalos del ratio de prestaciones. El eje Y indica el número de tareas que presentan un *speed-up* que se encuentra en el intervalo indicado por el eje X. En la figura se puede apreciar que existe una gran variabilidad, pero que el grueso de los casos se aglutina alrededor del valor 1, indicando una pérdida o ganancia de prestaciones mínima. Aun así, es significativa la existencia de casos extremos, con pérdida de prestaciones cercana al 70% y ganancia de prestaciones superior al 100%, aunque en ambos extremos, para un número de tareas muy reducido.

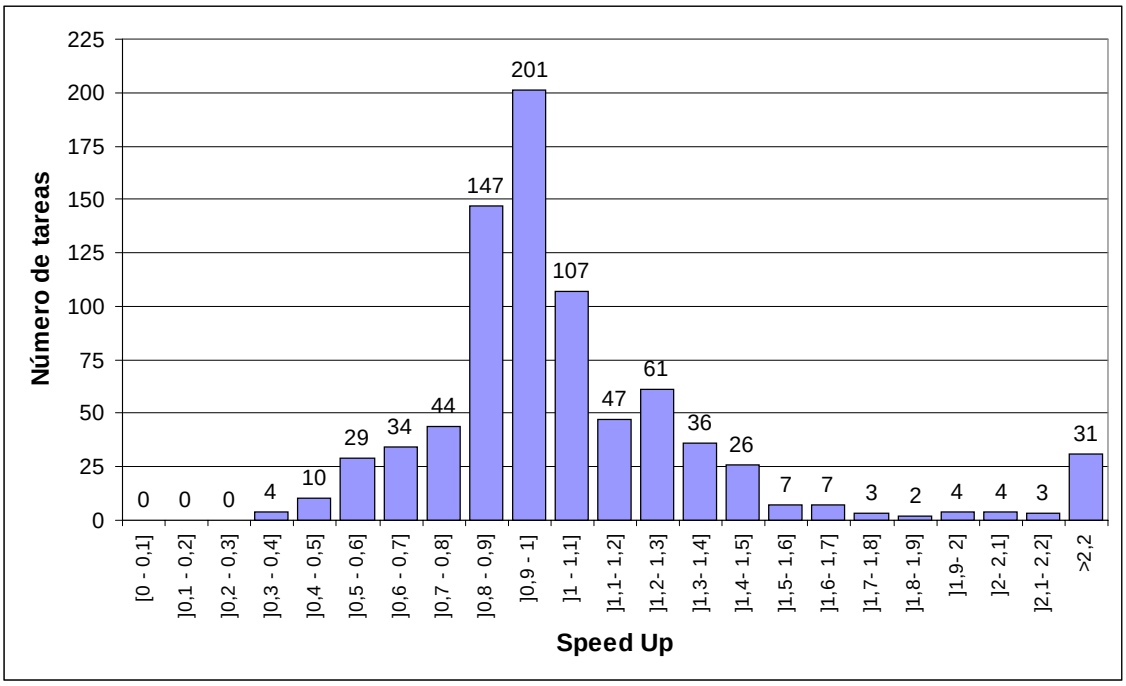


Figura 30. Histograma de frecuencias para la variación de prestaciones. Cada columna indica el número de tareas con un *speed-up* comprendido en el intervalo indicado por el eje X. Uso dinámico.

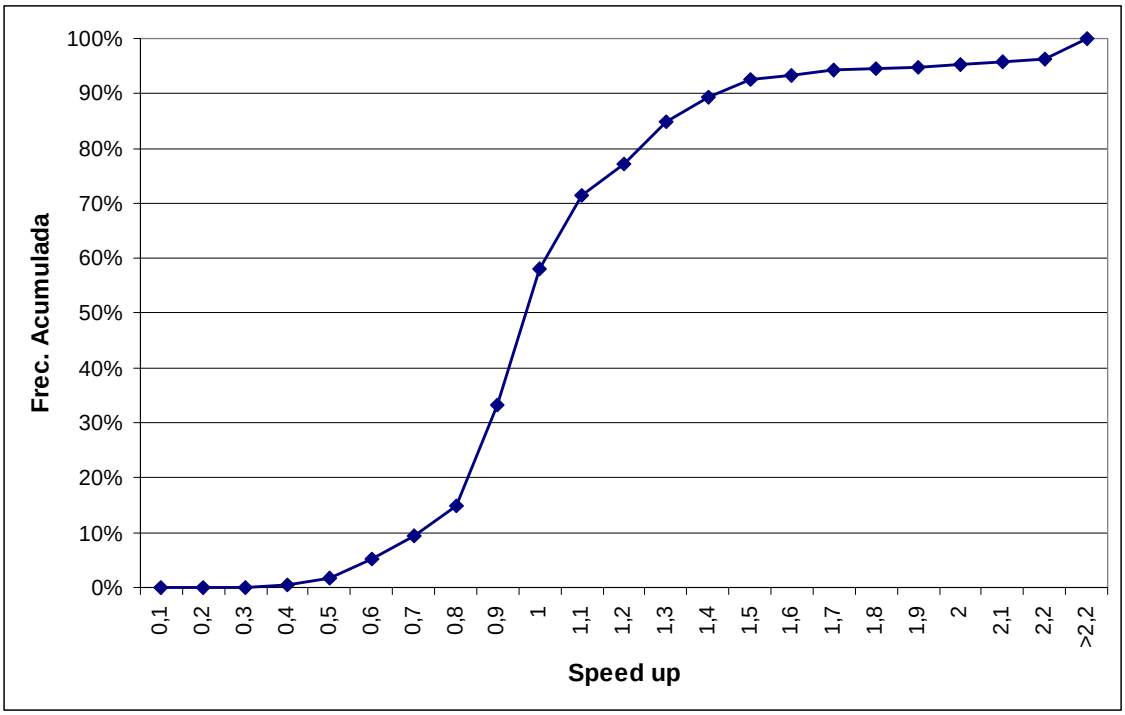


Figura 31. Histograma de frecuencias acumuladas para la variación de prestaciones. El eje Y indica el porcentaje de tareas con un *speed-up* menor o igual que el valor indicado por el eje X. Uso dinámico.

La Figura 31 presenta la frecuencia acumulada, en porcentaje, para diferentes intervalos de *speed-up*. El valor del eje X indica el porcentaje de tareas que presentan un *speed-up* igual o menor al correspondiente valor del eje X. En la figura se puede observar que para algo más del

40% de las tareas no existe pérdida de prestaciones, porcentaje que sube hasta casi el 70% si se consideran valores de *speed-up* superiores a 0,9 como una pérdida de prestaciones no significativa. Sólo para un 10% de tareas se aprecia una pérdida de prestaciones superior al 30%, mientras que en el otro extremo, algo más del 10% de las tareas presentan un aumento de prestaciones superior al 30%.

Aunque el análisis de las prestaciones ofrecidas por el uso dinámico de *locking cache*, considerando las tareas, ofrece resultados interesantes, la posibilidad de que en un mismo sistema convivan tareas con ganancia y pérdida de prestaciones sugiere la realización de un análisis global, evaluando las prestaciones del sistema al completo. Para realizar este análisis se ha comparado, para cada experimento, la utilización planificable obtenida de la simulación de la *cache* convencional frente a la utilización planificable de la *locking cache* obtenida de la ejecución del algoritmo genético. En ambos casos, la utilización planificable se obtiene de los tiempos de respuesta, bien simulados para *caches* convencionales, bien estimados por el algoritmo genético para el uso dinámico de *locking cache*. El proceso y ecuaciones utilizadas para el cálculo de la utilización planificable es el mismo que el descrito en el apartado 2.5.3 del capítulo anterior. La comparación de las prestaciones del uso dinámico de *locking cache* frente a las *caches* convencionales se realiza definiendo el valor Prestaciones (π) como el cociente de la utilización planificable para *cache* convencional entre la utilización planificable para *locking cache* ($\pi = U_c/U_l$). Valores de π mayores que 1 indican que el uso estático de *locking cache* presenta mejores prestaciones que la *cache* convencional al ofrecer una utilización planificable menor.

La Figura 32 muestra el histograma de frecuencias para diferentes intervalos de π. El eje Y indica el número de experimentos que presentan un valor de π que se encuentra en el intervalo indicado por el eje X. En este caso la variabilidad es mucho menor, y los valores extremos se han visto reducidos de manera significativa, ya que la máxima pérdida de prestaciones no alcanza el 50%, al igual que la máxima ganancia de prestaciones, que se queda también en el 50%.

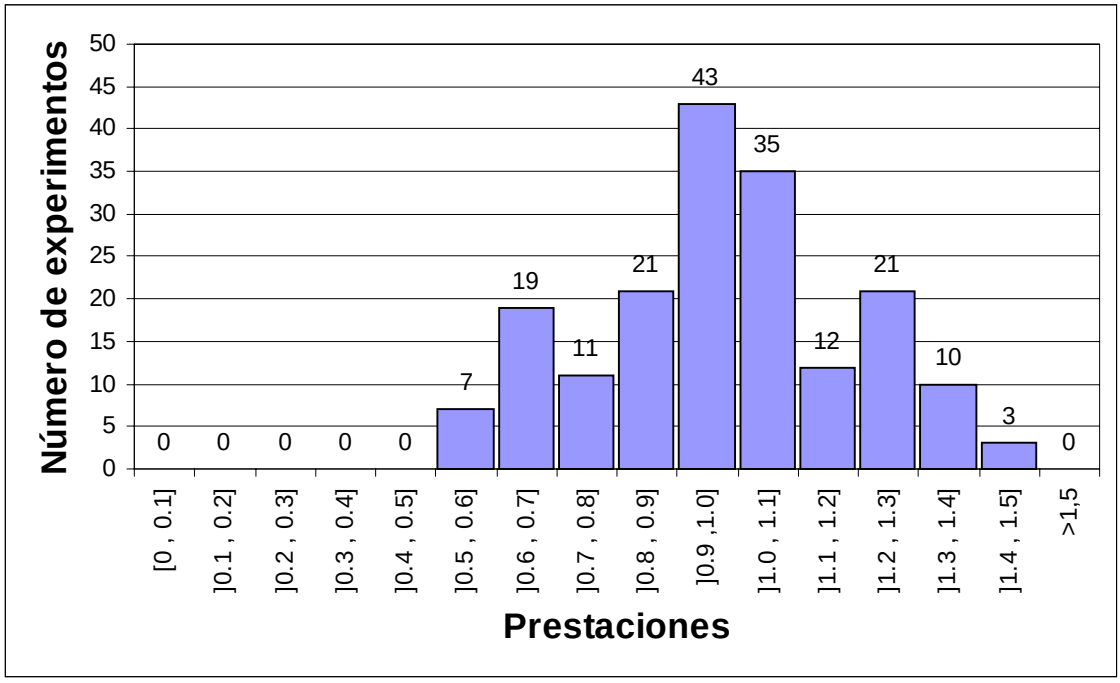


Figura 32. Histograma de frecuencias para U_c/U_l. Cada columna indica el número de sistemas con un valor comprendido en el intervalo indicado por el eje X. Uso dinámico.

La Figura 33 presenta el histograma de frecuencias acumuladas, en porcentaje, para diferentes intervalos de π. El valor del eje X indica el porcentaje de tareas que presentan un valor de π inferior o igual al correspondiente valor del eje X. En la figura se puede observar que para poco

más del 30% de los experimentos existe una pérdida de prestaciones ($\pi \leq 0{,}9$) al utilizar *locking cache*, o lo que es lo mismo, que para casi el 70% de los experimentos, el uso dinámico de *locking cache* proporciona determinismo sin pérdida de prestaciones, y que para cerca del 50% de los experimentos el determinismo se consigue al mismo tiempo que se mejoran las prestaciones ($\pi > 1{,}0$). La curva es similar a la presentada en la Figura 31 donde se comparaban las prestaciones considerando las tareas de forma individual, pero en este caso existe menos variabilidad, la pendiente es más suave, y los valores extremos de π son menores.

Figura 33. Histograma de frecuencias acumuladas para U_c/U_l. El eje Y indica el porcentaje de sistemas con un valor menor o igual que el valor indicado por el eje X. Uso dinámico.

3.5.3 Comparación de prestaciones de uso dinámico vs uso estático

Para verificar que efectivamente el uso dinámico de *locking cache* mejora, en prestaciones, al uso estático, se han comparado la utilización planificable obtenida para cada una de las propuestas. Esta comparación se realiza únicamente para planificadores con prioridades fijas, ya que no se dispone de experimentos para el uso dinámico de *locking cache* bajo planificación EDF.

Una primera aproximación para determinar si es posible que existan diferencias consiste en representar conjuntamente las curvas de π para uso estático π_e y uso dinámico π_d. La Figura 34 muestra ambas curvas y se puede apreciar que existen diferencias entre ellas. En concreto, en el uso dinámico de *locking cache* la pérdida de prestaciones es menor, tanto en valor absoluto como en porcentaje. Para los sistemas donde se obtiene una ganancia de prestaciones, ambos métodos presentan aproximadamente el mismo comportamiento.

Para obtener conclusiones sobre si uno de los dos métodos presenta mejores prestaciones que el otro, y si esta diferencia es significativa, se ha realizado un análisis de datos pareados [Jain91]. Para poder llevar a cabo este análisis se realizan *n* experimentos sobre uso estático, y los mismos *n* experimentos sobre uso dinámico, de forma que el experimento i sobre uso estático es el mismo experimento i sobre uso dinámico. La comparación entre las dos muestras puede realizarse mediante el análisis de una sola muestra, de n experimentos, donde cada valor es la diferencia entre el valor del experimento i sobre uso estático y el valor del experimento i sobre uso dinámico. En concreto, de cada experimento se ha obtenido la utilización planificable, $U_{estático}$ y $U_{dinámico}$, realizándose el análisis sobre la diferencia U_e-U_d.

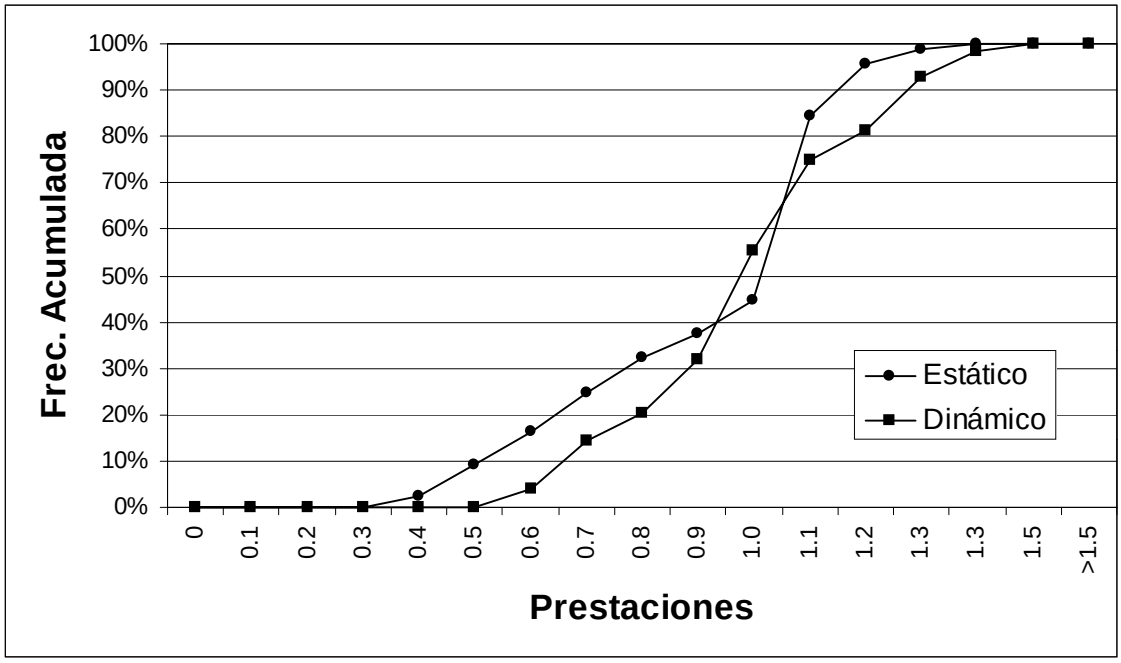


Figura 34. Comparativa de las prestaciones de uso estático frente a uso dinámico. Cada curva representa el histograma de frecuencias acumuladas para U_c/U_l de cada una de las propuestas.

La Tabla 11 muestra los principales estadísticos para la diferencia de las utilizaciones planificables de cada propuesta.

Número de experimentos	**182**
Valor medio	0,0226792
Mediana	0,00920231
Varianza	0,00626576
Desviación típica	0,0791566
Mínimo	-0,227715
Máximo	0,313682
Primer cuartil	-0,0037259
Tercer cuartil	0,0363896

Tabla 11. Resumen de los estadísticos del análisis de datos pareados $U_{estático}$ - $U_{dinámico}$.

La media de la diferencia es positiva, lo que indica que la utilización obtenida por el uso estático es superior a la utilización obtenida por el uso dinámico de *locking cache*, por lo que las prestaciones obtenidas por esta última propuesta son, en principio, mejores, ya que presentan una menor utilización para la misma carga. Sin embargo, para determinar realmente si esta afirmación es cierta, o por el contrario la media es superior a cero debido únicamente a la aleatoriedad de las muestras, es necesario utilizar herramientas estadísticas que nos permitan concluir si realmente existe diferencia en las prestaciones obtenidas por ambos usos de la *locking cache*.

Una herramienta sencilla y a la vez muy eficaz para determinar si la media es significativamente distinta de cero es calcular su intervalo de confianza para un determinado nivel de confianza. El intervalo de confianza representa el rango de valores entre los cuales se encontrará la media verdadera de la población que es representada por las muestras de que se disponen. El nivel de confianza es un indicador de la seguridad con que se obtiene el intervalo de confianza. De este

modo, si el nivel de confianza es del 90%, quiere decir que hay un 10% (100-90) de posibilidades que el valor real de la media se encuentre fuera de dicho intervalo.

Una aplicación muy útil del intervalo de confianza, cuando se calcula para la media de las diferencias de dos muestras pareadas, es la comprobación de si dicho intervalo contiene el valor cero. Si dicho intervalo no contiene el valor cero, se puede asegurar, al nivel confianza utilizado para calcular el intervalo, que la media es significativamente distinta de cero, bien positiva o bien negativa. En el análisis que nos ocupa, el intervalo de confianza para U_e-U_d, a un nivel de confianza del 95%, es [0,0111018;0,0342567]. Puesto que el intervalo de confianza no contiene el cero, puede decirse, con un 95% de confianza, que existen diferencias significativas entre las prestaciones obtenidas por el uso estático y el uso dinámico, ofreciendo mejores prestaciones este último método ya que la media de las diferencias es positiva.

Una vez verificado que existen diferencias entre una propuesta y otra, el histograma de frecuencias y el histograma de frecuencias acumuladas nos permitirán extraer más información sobre esta diferencia.

La Figura 35 presenta el histograma de frecuencias para U_e-U_d. Este histograma muestra que en la mayoría de los casos la diferencia entre las utilizaciones se aglutina alrededor del cero, pero que existe un pequeño número de casos donde la diferencia es negativa, es decir, el uso estático presenta mejores prestaciones que el uso dinámico. Un dato importante es la gran distancia respecto de cero que presentan estos casos, aproximándose al valor de –0,3, claro indicador de que existe una gran variabilidad en los resultados El mismo efecto se puede observar en el otro extremo, aunque el número de casos es ligeramente superior y se alcanza un valor mayor, lo que concuerda con las conclusiones obtenidas del intervalo de confianza.

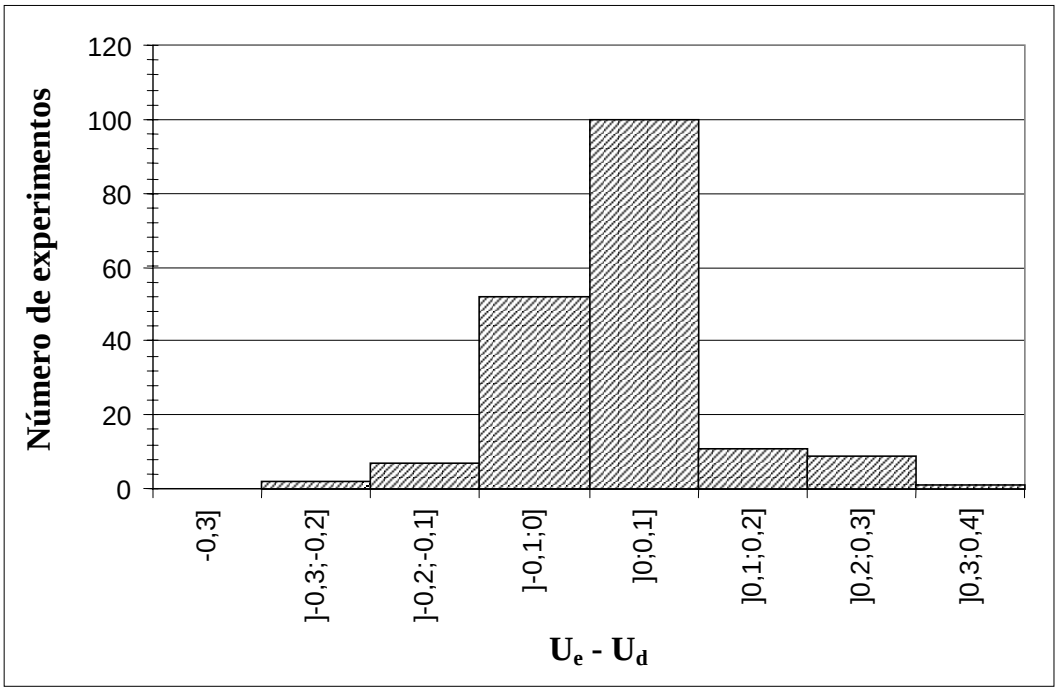


Figura 35. Histograma de frecuencias para U_e-U_d. Cada columna indica el número de sistemas con un valor comprendido en el intervalo correspondiente del eje X.

La Figura 36 presenta el histograma de frecuencias acumuladas. Los valores del eje Y indican el porcentaje de experimentos para los cuales la diferencia U_e-U_d es menor que el valor correspondiente del eje X. En él se puede observar que sólo para el 5% de los casos, la diferencia es inferior a –0,1, y que para menos del 40% de los casos la diferencia es inferior o igual a cero, lo que indica que para más del 60% restante de los casos, la diferencia es superior a cero, mostrando que el uso dinámico presenta una utilización menor para la mayoría de los casos.

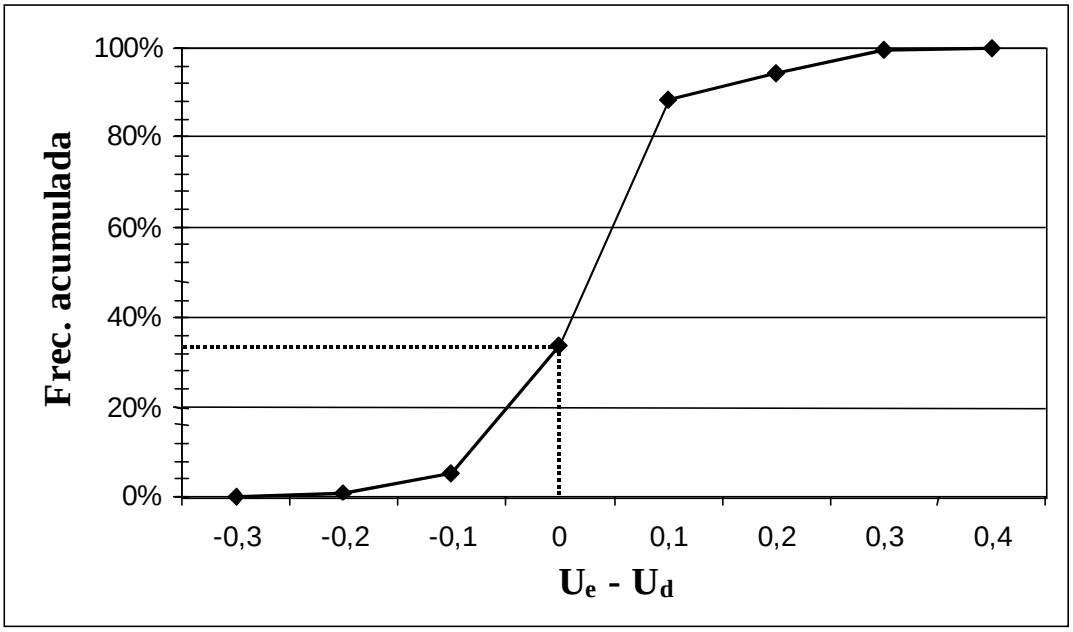


Figura 36. Histograma de frecuencias acumuladas para U_e-U_d. El eje Y indica el porcentaje de sistemas cuyo valor es menor o igual que el indicado por el eje X.

Aunque parezca contradictorio el intervalo de confianza con la aparición de valores negativos al calcular U_e-U_d para cada experimento, esto no es así, ya que el intervalo de confianza presenta el valor que se obtendrá de media, por lo que dicho intervalo de confianza puede interpretarse como que el uso dinámico de *locking cache* presentará mejores prestaciones en más casos, o que en los casos donde el uso dinámico presente mejora de prestaciones ésta será mayor que la pérdida de prestaciones para los casos en que la haya, o ambos escenarios simultáneamente. La Figura 36 de frecuencias acumuladas muestra que serán mayoría los casos donde existirá mejora de prestaciones al realizar un uso dinámico de la *locking cache*, y la Figura 35 de frecuencias muestra que esta mejora será mayor que la pérdida de prestaciones en los casos que presenten esta pérdida de prestaciones.

3.6 Uso dinámico de locking cache para el planificador EDF

De igual modo que el uso dinámico de *locking cache* se ha aplicado en sistemas con planificación basada en prioridades fijas, sería posible utilizarlo en sistemas con prioridades dinámicas, modificando adecuadamente el análisis de planificabilidad presentado en el capítulo anterior para la política de planificación EDF. Sin embargo, existen varias cuestiones, derivadas algunas de ellas de la propuesta de uso dinámico para prioridades fijas, que permiten prever que los resultados obtenidos, en cuanto al ajuste de la estimación con los valores reales, no serán demasiado buenos:

- El error cometido al estimar los tiempos de respuesta cuando se utilizan prioridades fijas es, en muchos casos, muy elevado, debido a la variabilidad en los tiempos de recarga de la *cache* para cada tarea.
- La necesidad de considerar el peor caso hace que las tareas menos prioritarias, en prioridades fijas, tengan una mayor posibilidad de error, ya que el peor caso debe considerar el tiempo de recarga de todas las tareas de mayor prioridad.
- En el análisis propuesto para EDF es imposible determinar la prioridad de las tareas -lógicamente, ya que se trata de prioridades dinámicas- por lo que el peor caso en el cálculo del *cache-refill penalty* para cada tarea incluye el tiempo de recarga de todas las demás tareas del sistema, lo que equivale a considerar el mismo valor para todas las tareas.

- El coste del *cache-refill penalty* se incorpora al tiempo de cómputo de la tarea expulsante, y no de la tarea que sufre la expulsión. Debido a la posibilidad de que exista una gran diferencia entre el peor caso y el caso real de cada tarea, esta aproximación puede ser excesivamente conservadora.
- En el análisis de planificabilidad basado en el Instante Crítico Inicial -ICI- no es posible realizar una contabilidad de las expulsiones que suceden en el sistema, por lo que se considera que todas las tareas, al activarse, generan una expulsión. Esto es claramente una aproximación conservadora.

Todas estas consideraciones se planteaban también para el uso estático de *locking cache*, pero en aquel caso el coste del *cache-refill penalty* era extremadamente bajo, correspondiendo tan sólo al tiempo de un fallo de *cache*, por lo que el cúmulo de sobreestimaciones no representaba un error significativo tal como mostraban los resultados de los experimentos. Sin embargo, en uso dinámico de *locking cache*, el valor del *cache-refill penalty* puede ser elevado e introducir, por tanto, un error significativo al estimar la utilización del sistema. Este efecto se ha manifestado en los experimentos realizados para prioridades fijas, y presumiblemente se verá agravado, por las cuestiones mencionadas anteriormente, con prioridades dinámicas.

Todo esto ha motivado la no aplicación del uso dinámico de *locking cache* a sistemas con planificadores de prioridades dinámicas, a la espera del desarrollo de una nueva arquitectura de *cache*, con menor coste de recarga, o de un nuevo análisis de planificabilidad con menos asunciones conservadoras.

3.7 Conclusiones del capítulo

En este capítulo se ha presentando una nueva forma de utilizar las *caches* con bloqueo o *locking cache*, que recibe el nombre de uso dinámico. Esta propuesta se basa en permitir el reemplazo de los contenidos de *cache*, pero en determinados instantes de la ejecución del sistema y con un intercambio de contenidos conocido a priori, ya que éstos se seleccionan en tiempo de diseño mediante la utilización de un algoritmo genético. Este reemplazo de los contenidos de la *cache* se produce únicamente al inicio de la ejecución de una tarea o en la reanudación de la ejecución tras una expulsión.

El conocimiento tanto de los instantes en que la *cache* es modificada como de los contenidos que son ubicados en cada reemplazo permite utilizar técnicas de estimación del WCET y del tiempo de respuesta de las tareas muy sencillos, al mismo tiempo que proporciona un alto nivel de determinismo, lo que ofrece resultados muy ajustados al estimar los tiempos de respuesta de las tareas. Estas técnicas han sido descritas y evaluadas experimentalmente. Los resultados muestran que para más del 50% de las tareas, la sobreestimación cometida al calcular su tiempo de respuesta es menor del 1%. Sin embargo, los experimentos también muestran que el máximo error cometido se encuentra cercano al 40%, y que para más de un 25% de las tareas, la sobreestimación es igual o inferior al 5%. La principal fuente de error es la existencia de interferencia extrínseca indirecta, la cual obliga, en cada expulsión, a contabilizar el peor caso posible de entre todos los reemplazos de *cache* que pueden realizarse para cada expulsión. La variabilidad en el coste temporal del reemplazo para cada tarea del sistema es la causa principal de la desviación del tiempo real de respuesta que se observa en un número minoritario pero significativo de casos.

El uso dinámico de *locking cache* sacrifica parte del determinismo con el objetivo de mejorar las prestaciones, igualando e incluso superando en más del 55% de los experimentos las prestaciones ofrecidas por las *caches* convencionales.

Frente al uso estático de *locking cache*, la propuesta presentada en este capítulo ofrece niveles de determinismo mucho menores, o lo que es lo mismo, el error cometido al calcular el tiempo de respuesta de las tareas es, en general, mayor, y en algunos casos, mucho mayor. Sin embargo, y en cuanto a las prestaciones ofrecidas por una propuesta y otra, el uso dinámico de *locking*

cache se muestra significativamente mejor que el uso estático, presentando una ligera reducción de la utilización del procesador para más del 60% de los sistemas evaluados.

Esta mejora de prestaciones se obtiene pese al error cometido en el análisis de planificabilidad propuesto para uso dinámico, error debido, fundamentalmente, a la existencia de interferencia extrínseca indirecta, por lo que una reducción de este error aumentaría, teóricamente, las prestaciones obtenidas por el uso dinámico de *locking cache*, convirtiendo este modo de operar en una opción extremadamente interesante frente al uso de *caches* convencionales. La reducción del error puede acometerse desde dos frentes:

- Mejora del análisis: mejorando la contabilidad de las posibles expulsiones que una tarea puede sufrir, y sobre la fuente de dichas expulsiones, es decir, qué tareas pueden generar una expulsión concreta y cuáles no, es posible reducir el número de casos a considerar en el cómputo del coste de la interferencia extrínseca, lo que redundaría en una reducción de las posibilidades de estimar un caso peor al que realmente sucede.
- Mejora de la arquitectura: la reducción del coste temporal de la recarga de la *cache*, mediante las modificaciones pertinentes a la arquitectura del sistema de memoria, implicaría, por una parte, el aumento de las prestaciones del sistema, y por otra parte reduciría el error cometido al estimar el tiempo de respuesta, ya que si el tiempo de recarga de la *cache* se reduce significativamente respecto al tiempo de respuesta total de la tarea, el porcentaje de error se vería también reducido de forma significativa.

La reducción del tiempo de carga de la *cache* proporcionaría no sólo las ventajas anteriormente indicadas, sino que podría hacer posible la utilización del uso dinámico en planificadores dinámicos, al reducir el impacto negativo que supone la consideración del peor caso.

Capítulo 4

Análisis de prestaciones para uso estático de locking cache con prioridades fijas

El objetivo de este capítulo es identificar los factores, tanto *software* como *hardware*, que afectan o deciden las *prestaciones* obtenidas por el uso estático de *cache*s con bloqueo cuando se utiliza un planificador de prioridades fijas. Esta identificación se ha realizado a través de varios análisis estadísticos, obteniendo básicamente un conjunto de modelos del comportamiento de la *locking cache* que permite conocer la pérdida o ganancia de *prestaciones* frente al uso de *cache*s convencionales, sin necesidad de realizar costosos experimentos. Parte del trabajo presentado en este capítulo ha sido presentado en [Martí03b].

La obtención de estos modelos permitirá al diseñador de sistemas de tiempo real evaluar la conveniencia o no de aplicar el uso de *locking cache* a sus proyectos. La utilización de *locking cache* presenta como ventaja indiscutible la sencillez de las herramientas necesarias para la realización del análisis de planificabilidad. Pero esta sencillez lleva aparejada, en muchos casos, una pérdida de *prestaciones* que puede resultar inaceptable en determinados sistemas o proyectos.

Los modelos presentados en este capítulo no ofrecen resultados absolutos, es decir, no calculan los tiempos de respuesta ni el WCET de las tareas, ni contesta a la pregunta ¿es el sistema planificable? El resultado obtenido de los modelos desarrollados es una estimación, sujeta a error, de cuál es la relación entre las *prestaciones* obtenidas para un determinado sistema de tiempo real al utilizar la *locking cache* y las *prestaciones* obtenidas para el mismo sistema si se utilizan memorias *cache* convencionales. Si la respuesta del modelo es que no existe gran diferencia entre los dos esquemas de *cache*, el diseñador puede aprovechar el determinismo y la sencillez de las *cache*s con bloqueo. Si por el contrario, existe una gran pérdida de *prestaciones* al utilizar la *locking cache*, deberá sopesarse la importancia de la pérdida de *prestaciones* frente a la realización de una análisis complejo.

El trabajo desarrollado en este capítulo se basa en dos poderosas herramientas estadísticas, los modelos de regresión y el diseño de experimentos, ambas sustentadas en el análisis de la varianza [Jain91] [Romero93]. Aunque no es objetivo de esta tesis el estudio de dichas herramientas, para facilitar la comprensión del trabajo realizado y las conclusiones obtenidas, se realiza una breve introducción a las herramientas utilizadas antes de proceder con el desarrollo de los modelos. Un estudio más detallado de las múltiples aplicaciones de dichas herramientas y su fundamento estadístico y matemático se puede encontrar en la bibliografía asociada [Jain91] [Romero93] [Chirivella99]

4.1 Introducción a las herramientas estadísticas

De las múltiples áreas en que puede dividirse la estadística, la llamada estadística descriptiva es la más conocida y habitualmente utilizada. Esta estadística descriptiva permite conocer las principales características de una muestra de datos mediante la obtención de valores numéricos y representaciones gráficas, como son la media, la mediana, la desviación típica, los intervalos intercuartílicos o los histogramas de frecuencia presentados en los capítulos anteriores. La utilidad de estos valores y representaciones gráficas es inestimable, ya que permite condensar todo el conocimiento incluido en una gran cantidad de experimentos o datos en un número reducido de valores, fácilmente interpretables, a partir de los cuales extraer conclusiones.

Aunque la información ofrecida por la estadística descriptiva y las conclusiones obtenidas a partir de dicha información es, en numerosos casos, suficiente, se podría decir que se trata de información estática y opaca. Estática porque describe cómo son los datos que representa, pero

no permite prever cómo serán esos datos o cual será su evolución si cambian las condiciones o el entorno donde se generaron. Opaca porque no ofrece ninguna información sobre cuáles son las causas que produjeron los datos.

Esto no significa que la estadística descriptiva no sea útil. Se trata de una primera aproximación, sencilla de realizar y de comprender, a los datos disponibles. Aproximación que en muchos casos es más que suficiente.

Pero en otros casos es necesario, o al menos interesante, conocer no sólo un resumen de los datos, sino en que medida estos datos se ven afectados por diferentes factores, o cómo los datos evolucionarán en función de las modificaciones y variaciones que sufran dichos factores, permitiendo realizar previsiones de cuál será el comportamiento del proceso bajo estudio sin necesidad de que las condiciones esperadas se produzcan realmente, así como la identificación de las condiciones que deben darse para obtener un comportamiento concreto del proceso sin necesidad de realizar una cantidad ingente de experimentos.

La estadística ofrece dos métodos, entre otros, para identificar el efecto que determinados factores, llamados variables explicativas, tienen sobre el comportamiento de un sistema (sea cual sea la naturaleza de éste), y que recibe el nombre de variable dependiente. Estos dos métodos estadísticos son el "diseño de experimentos" y los "modelos de regresión lineal múltiple". Ambos métodos comparten múltiples elementos en común, y realmente, el primer método es una particularización del segundo, mucho más flexible, amplio y complejo. Esta complejidad, especialmente en los cálculos matemáticos necesarios para llevar a cabo el desarrollo de los modelos de regresión, ha relegado habitualmente la realización de modelos de regresión lineal múltiple a casos muy particulares o simples, optándose casi siempre por la utilización del diseño de experimentos. Pero la aparición de computadores con gran potencia de cálculo y herramientas *software* para el análisis estadístico ha dado nueva vida a los modelos de regresión [Romero93].

El diseño de experimentos se basa en el estudio de un conjunto de resultados obtenidos tras la ejecución de una serie de experimentos, experimentos en los que los factores o variables explicativas que se desea estudiar han sido forzados a tomar determinados valores. Por el contrario, cuando es imposible asignar valores concretos a los factores bajo estudio, los modelos de regresión permiten trabajar sobre muestras para los que se conocen los valores de los factores, pero estos valores no se han asignado de forma intencionada. Ambos métodos permiten obtener resultados y conclusiones muy similares, ya que aunque las muestras se obtienen por procedimientos diferentes, se parte del mismo origen: la descripción del proceso bajo estudio y la identificación de los posibles factores o variables explicativas que condicionan el comportamiento del proceso.

Así pues, la utilización de un método u otro no depende del tipo de resultado que se desee obtener, sino de las condiciones del proceso que se desea estudiar.

4.2 Descripción del proceso

El primer paso y fundamental a la hora de utilizar los modelos de regresión y el diseño de experimentos es especificar correctamente el proceso que se desea estudiar, y los factores que condicionarán su comportamiento.

La descripción del proceso está compuesta, por un parte, por una variable dependiente, que representa la salida, característica o resultado del proceso que se desea modelar. Por otra parte, la definición comprenderá una o varias variables independientes, llamadas variables explicativas o factores, que representan las múltiples características y entradas del proceso que definirán el comportamiento de la variable dependiente.

Uno de los puntos más delicados es la identificación de las variables explicativas. Si en la definición del proceso se omiten variables o factores que modifican el comportamiento del sistema, será imposible llegar a un modelo adecuado o correcto del proceso. Si por el contrario se incluyen demasiadas variables o que no afectan el comportamiento del sistema, el análisis

estadístico será más complejo y puede incluso ser erróneo. Por tanto es necesario un buen conocimiento del sistema o proceso que se desea modelar y en especial, de los factores que determinan su comportamiento.

En el caso que nos ocupa, el proceso que se desea modelar es el comportamiento del uso estático de *cache* con bloqueo sobre la que se ejecuta un conjunto de tareas planificadas mediante la política de prioridades fijas, frente al comportamiento de una *cache* convencional ejecutando el mismo conjunto de tareas con la misma política de prioridades. Llamaremos *carga* a un conjunto de tareas con unos determinados valores de periodos y plazos, y recibirá el nombre de sistema la unión de una *carga* y un tamaño concreto de memoria *cache*. Así, la misma *carga* sobre una *cache* de 4Kb y sobre una *cache* de 8Kb serán dos sistemas distintos. Por último, la ejecución de un sistema sobre las diferentes arquitecturas de *cache* evaluadas recibirá el nombre de experimento.

El proceso a modelar será la relación o ratio entre las *prestaciones* obtenidas con una *cache* con bloqueo y las *prestaciones* obtenidas con una *cache* convencional al ejecutar un determinado sistema. De este modo, el modelo que se obtiene indicará qué esquema de memoria *cache* ofrecerá mejores *prestaciones* para un determinado sistema (conjunto de tareas y memoria *cache*). Basándose en los resultados obtenidos, un diseñador de sistemas de tiempo real podrá conocer qué nivel de *prestaciones* sacrificará si opta por utilizar una *cache* con bloqueo en su sistema, sin necesidad de realizar ningún experimento, simulación o prototipo.

4.2.1 Variable dependiente

El primer paso a la hora de definir el proceso es identificar la variable dependiente. En este caso, esta variable es la comparación de las *prestaciones* obtenidas con dos arquitecturas de *cache*. Para medir las *prestaciones* de cada esquema de *cache* y de las diferentes propuestas de utilización de la *locking cache*, se empleará la utilización planificable, calculada esta utilización planificable de la forma descrita en los capítulos anteriores. Por tanto, la variable dependiente o salida del proceso que se desea modelar, y que llamaremos *prestaciones* (π) será la relación entre la utilización planificable obtenida con una *cache* convencional donde las tareas se planifican mediante prioridades fijas Utilización$_{\text{convencional fijas}}$ (U_{cf}) y la utilización planificable obtenida con una *cache* con bloqueo Utilización$_{\textit{locking}\text{ estático fijas}}$ (U_{lef}), realizando un uso estático de la misma y planificando las tareas mediante prioridades fijas. La relación entre las utilizaciones se define como el cociente de ambas, tal como muestra la siguiente ecuación:

$$\textbf{\textit{Prestaciones}}\ \pi = U_{cf}/U_{lef}$$

Si la variable π toma el valor de 1 indicará que la utilización obtenida con ambos esquemas de *cache* es la misma, y que por tanto no existe pérdida o ganancia de *prestaciones* al utilizar una u otra alternativa. Si la variable *prestaciones* toma valores superiores a 1, es decir, $U_{cf} > U_{lrf}$, indicará que existe ganancia de *prestaciones* al utilizar la *cache* con bloqueo, ya que se considera que una menor utilización representa mejores *prestaciones*. Por contra, si la variable *prestaciones* toma valores inferiores a 1, es decir, $U_{cf} < U_{lef}$, indicará que el empleo de una *cache* con bloqueo implica una pérdida de *prestaciones* respecto a la utilización de las arquitecturas de *cache* convencionales. En el texto que sigue, todas las referencias a las *prestaciones* se realizan desde el punto de vista de la *locking cache* mientras no se indique explícitamente lo contrario. Cuando se diga que un sistema mejora (o empeora) sus *prestaciones*, implícitamente se está diciendo que el uso estático de *locking cache* presenta una mejora (o empeoramiento) respecto del uso de *cache*s convencionales.

4.2.2 Variables explicativas

Una vez definida la variable dependiente o salida, la definición del proceso se completa con la especificación y definición de todos aquellos parámetros que pueden influir en los resultados obtenidos. Estos factores pueden dividirse en dos grupos: factores *hardware* y factores *software*. El análisis de cada tipo de factor se realizará de forma diferente, empleándose el modelo de

regresión lineal múltiple para el estudio de los factores *software*, y el diseño de experimentos para el estudio de los factores *hardware*. Esta división se ha realizado atendiendo a la naturaleza de los dos tipos de factores. Así, para los factores *hardware* es sencillo diseñar y realizar experimentos donde el valor de cada parámetro es asignado de forma fija y conocida en el momento de crear el experimento, mientras que los factores *software* toman valores dependientes de las tareas que forman el sistema y que no pueden establecerse a un valor concreto durante la creación de las mismas.

Los diferentes factores o variables explicativas que inicialmente se consideran para la realización de los análisis son:

A. Factores *hardware*:

- Tamaño de la *cache* (CS): expresado en KiloBytes, indica el número total de bytes que pueden cargarse en *cache*. Para la *cache* con bloqueo no incluye el buffer temporal. Aunque no se considerará explícitamente en los diferentes análisis, sí participará en los mismos de forma implícita, ya que la definición de sistema incluía este tamaño como una característica propia de cada experimento.
- Tamaño de línea (LS): tamaño de una línea de *cache*, expresado normalmente en bytes o alternativamente en instrucciones.
- Tiempo de acierto (T_{hit}): tiempo en ejecutar una instrucción desde memoria *cache* o desde el buffer temporal, medido en ciclos de procesador.
- Tiempo de fallo (T_{miss}): tiempo necesario para llevar un bloque desde memoria principal a la *cache* o al buffer temporal, medido en ciclos de procesador. Habitualmente se define el tiempo de fallo T_{hit} con el valor de un ciclo de reloj, por lo que se puede considerar que T_{miss} representa tanto el tiempo de fallo en el acceso a *cache* como la relación entre el tiempo de acceso a *cache* frente al tiempo de acceso a memoria principal.

B. Factores *software*:

- Tamaño_sistema (SS): suma de los tamaños de las diferentes tareas del sistema, expresada en kilobytes.
- Numero_tareas (TN): número de tareas que forman el sistema.
- Tamaño_medio (AS): tamaño medio de las tareas del sistema, usando la media aritmética y expresada en kilobytes. Se obtiene del cociente de SS entre TN.
- Tamaño_ponderado (WS): tamaño medio ponderado de las tareas del sistema, dando mayor peso cuanto más prioritaria es la tarea. Expresado en kilobytes.
- Tamaño_prioritaria (PS): tamaño de la tarea más prioritaria del sistema, expresado en kilobytes.
- Ratio_tamaño_sistema (SSR): este factor identifica la relación entre el tamaño del sistema y el tamaño de la memoria *cache*, obtenido de la división del tamaño de *cache* por el tamaño total del sistema (CS/SS). Aunque el tamaño de la *cache* es claramente un factor *hardware*, la relación entre el tamaño del código y el tamaño de la *cache* se ha considerado un factor *software*. Como se comprobará en los análisis realizados, su importancia es capital a la hora de definir el comportamiento del proceso.
- Ejecución_total (ET): número total de instrucciones ejecutadas por todas las tareas del sistema.
- Ejecutadas_media (EA): media aritmética de las instrucciones ejecutadas por cada tarea del sistema, obtenida del cociente ET/TN

- Ejecutadas_ponderada (EW): número medio de instrucciones ejecutadas por cada tarea del sistema, ponderada por la prioridad de la tarea, dando mayor peso cuanto más prioritaria es la tarea.
- Ejecutadas_tamaño (ES): Media aritmética del cociente entre el número de instrucciones ejecutadas por cada tarea y su tamaño, para todas las tareas de un sistema. Esta variable explicativa o factor ofrece una idea general del número de iteraciones de los bucles que forman las tareas del sistema.
- Ejecutadas_tamaño_ponderado (ESW): número medio ponderado de instrucciones ejecutadas por cada tarea, dividido por el tamaño de la tarea y dando mayor peso cuanto mayor es la prioridad de la tarea.
- Uso_*cache* (LOC): índice general del uso que el conjunto de las tareas del sistema hace de la *cache*, o básicamente, el nivel de localidad del código de las tareas que forman el sistema. Para estimar este índice, pueden definirse cuatro niveles de forma global para un conjunto de tareas. Los sistemas utilizados en los experimentos se clasifican en los dos niveles intermedios:
  - LOC -1: el sistema no tiene localidad temporal en su código, y la localidad espacial es mínima, limitada al tamaño de una línea de *cache*. En estos sistemas, las tareas no tienen ningún bucle, por lo que el aprovechamiento que se obtiene de la *cache* es nulo.
  - LOC 0: el sistema, de forma global, tiene un nivel de localidad bajo, pero suficiente como para hacer un aprovechamiento medio de la existencia de *cache*, mejorando sensiblemente las *prestaciones* frente a su ejecución en un sistema computador sin *cache*.
  - LOC 1: el sistema tiene, de forma global, un alto nivel de localidad, tanto espacial como temporal, lo que redunda en un buen aprovechamiento de la *cache* y en una tasa de fallos baja.
  - LOC 2: representa el máximo nivel de aprovechamiento de la *cache*. Todas las instrucciones se ejecutan un elevado número de veces antes de ser reemplazadas de *cache*, y no vuelven a ejecutarse nunca tras su reemplazo, por lo que sólo existen fallos de *cache* del tipo obligatorio. Al igual que el nivel LOC -1, se trata de casos extremos y teóricos, prácticamente imposibles de producirse en tareas o sistemas reales o representativos de la realidad.
- Interferencia (IRF): este factor, definido también como variable cualitativa pero sólo a dos niveles, indica la relación entre los periodos de las diferentes tareas que forman el sistema, y por tanto, el nivel de interferencia que se generará entre las tareas. Un valor de IRF igual a 0 indica que no hay interferencia entre las tareas, o esta interferencia es mínima. Un valor de IRF igual a 1 indica que sí hay interferencias y que las tareas provocaran y sufrirán un número elevado de expulsiones.

Es posible que existan más factores o variables explicativas, especialmente de tipo *software*, que permitan caracterizar aún con más detalle las tareas y sistemas utilizados en los experimentos. Se ha considerado que las descritas anteriormente son suficientes por varios motivos, siendo el principal de ellos la facilidad en la obtención de los valores concretos para una tarea o sistema determinado. El objetivo es que estos valores puedan calcularse, o al menos estimarse, sin necesidad de ejecutar ninguna tarea, evitando de este modo consideraciones sobre si es necesario ejecutar las tareas en el peor caso o si es necesario considerar explícitamente el *hardware* utilizado. Esta razón es la que ha motivado la definición de los parámetros LOC e IRF como cualitativos y no como cuantitativos, para que sea posible, mediante una simple observación del experimento, estimar en qué categoría clasificarlo.

Otro motivo para no incluir nuevas variables explicativas es el objetivo perseguido con la realización de los análisis, ya que no se trata de predecir el valor exacto de las *prestaciones* (π) sino de identificar el efecto que los diferentes factores ejercen sobre el comportamiento y la

tendencia de dichas *prestaciones*. Otra razón es la posibilidad de que estas nuevas variables sean combinaciones de las ya definidas, por lo que no aportarían nueva información, sino que confundirían el análisis estadístico realizado. A este respecto, es posible que exista alguna relación entre los factores definidos anteriormente, por lo que antes de proceder a realizar cualquier análisis es necesario identificar si existe alguna correlación entre los factores y en caso de ser así, eliminar variables explicativas. La identificación de relaciones lineales estadísticamente significativas puede realizarse mediante la matriz de correlación de las variables explicativas. Posteriormente, la diagonal de la matriz inversa de dicha matriz de correlación indicará la variable explicativa que, por ser una combinación lineal de otras, debe ser eliminada para no confundir los modelos de regresión. Una vez eliminada la variable explicativa indicada por la matriz inversa, se repetirá el proceso generando la matriz de correlación y la matriz inversa para los factores restantes, eliminando un nuevo factor si así lo sugieren los resultados obtenidos, finalizando el proceso cuando la matriz inversa no indica ninguna relación lineal entre los factores restantes.

Este proceso se aplica sólo a las variables cuantitativas, y su resultado depende sólo de los valores de los factores, y no de la variable dependiente. Para los factores *software*, tras la identificación de las relaciones lineales, las siguientes variables explicativas no han mostrado ningún tipo de correlación, por lo que serán las que se utilizarán para la construcción de los modelos de regresión lineal múltiple:

- Tamaño_sistema (SS)
- Numero_tareas (TN)
- Ejecutadas_tamaño (ES)
- Tamaño_prioritaria (PS)
- Ejecutadas_total (ET)
- Ratio_tamaño_sistema (SSR)

En cuanto a los factores *hardware*, no se ha detectado ninguna correlación entre los definidos, por lo que todos ellos podrán emplearse en los correspondientes análisis.

4.3 Análisis de los factores software

Para llevar a cabo los modelos de regresión lineal múltiple, utilizando como variables explicativas los factores *software*, se han utilizado los experimentos desarrollados en el capítulo 2, donde el tamaño de la línea de *cache* se había establecido en 16 bytes (o 4 instrucciones), el tiempo de acierto en *cache* T_{hit} se estableció en 1 ciclo de reloj, y el tiempo de fallo en *cache* T_{miss} se estableció en 10 ciclos de reloj. En cuanto al tamaño de *cache*, se utilizaron siete tamaños diferentes, entre 1 Kbyte y 64 Kbytes. El número total de experimentos realizados y evaluados es de 210, que proporcionan un número de datos suficiente para realizar los modelos de regresión lineal que se presentan a continuación.

Previamente a estos análisis se ha realizado una descomposición del problema, basada en la intuición de que uno de los factores que más peso tendrá sobre las *prestaciones* ofrecidas por el uso estático de *cache* con bloqueo sobre prioridades fijas será la relación entre el tamaño de las tareas que forman el sistema y el tamaño de la *cache* sobre la que se ejecutan.

Para comprobar si la intuición es correcta y esta relación realmente existe, y si aporta alguna información útil a la hora de tomar decisiones, se presenta la Figura 37, donde se muestra el ratio entre la utilización obtenida al usar una *cache* convencional y la utilización estimada por el algoritmo genético al utilizar una *cache* con bloqueo ($\pi = U_{cf}/U_{lef}$), frente al ratio entre el tamaño de la *cache* y el tamaño total de las tareas (SSR= CS/SS)

En la gráfica, valores de π por encima de 1 indican que la utilización del sistema al emplear *cache* con bloqueo es menor que al emplear *cache*s convencionales, tal como se dijo en la

definición de esta variable dependiente. Respecto a SSR, los valores más pequeños representan tamaños de *cache* menores que el tamaño de las tareas. Puesto que el tamaño de la *cache* crece en potencias de dos, la gráfica representa el logaritmo del ratio de los tamaños, para permitir una mejor observación de la distribución de los puntos.

A partir de ahora, y dentro de este apartado, cualquier referencia a la *cache* con bloqueo o *locking cache* se limitará al uso estático de la misma para sistemas con política de planificación de prioridades fijas.

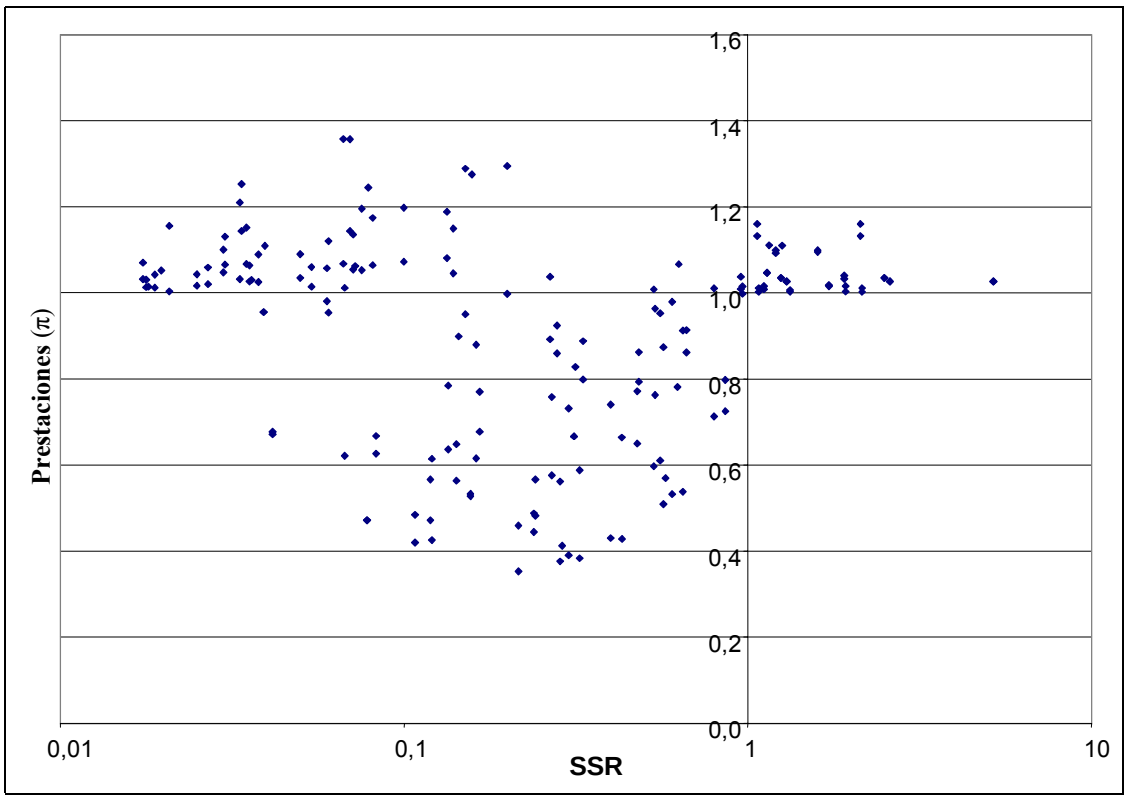


Figura 37. Gráfico de dispersión de $\pi = U_{cf}/U_{lef}$ frente a SSR.

En la Figura 37 pueden identificarse tres zonas nombradas como a, b y c. La primera zona (a) corresponde a experimentos con un valor de SSR menor que 0,1. En este caso, casi la totalidad de los puntos se encuentran por encima de 1, es decir, se obtienen mejores *prestaciones* al emplear *cache* con bloqueo que al utilizar una *cache* convencional. La segunda zona (b) corresponde a experimentos con un conjunto de tareas cuyo tamaño es siempre mayor que el tamaño de la *cache*, con valores de SSR entre 0,1 y menor que 1. En este grupo, la variable π se encuentra tanto por encima y por debajo de 1, aunque aparentemente se observan más puntos por debajo del valor 1, por lo que puede concluirse que presentan una pérdida de prestaciones generalizadas. Además, la variabilidad es muy alta, ya que los valores de π oscilan entre el 0,30 (pérdida de *prestaciones* cercana al 70%) y el 1,40 (ganancia de *prestaciones* cercana al 40%). Por último, la tercera zona (c) corresponde a experimentos cuyo conjunto de tareas tiene un tamaño igual o menor que el tamaño de la *cache*. Para esta situación, siempre existe una ganancia de *prestaciones*, es decir, la utilización del sistema obtenida al emplear *cache* con bloqueo es siempre igual o menor que al utilizar una *cache* convencional. En esta zona también existe variabilidad en la variable dependiente, pero mucho menor que en la zona b.

En una primera aproximación para explicar el comportamiento del proceso, esta gráfica pone de manifiesto que en las situaciones extremas la *cache* con bloqueo proporciona mejores *prestaciones* que la *cache* convencional. En concreto, cuando todo el código de las tareas cabe en *cache* (zona c), la mejora viene dada por la eliminación de los fallos de *cache* obligatorios. Al utilizar *cache* con bloqueo de forma estática, la precarga de las instrucciones en *cache* se realiza en tiempo de arranque del sistema, por lo que el coste temporal de cargar estas

instrucciones no se considera en los tiempos de respuesta de las tareas. Sin embargo, al utilizar *cache*s convencionales, las instrucciones se cargan bajo demanda, es decir, cuando es necesario ejecutarlas, por lo que el primer fallo de *cache* se incorpora al tiempo de respuesta de las tareas, aumentando la utilización que se realiza del sistema.

En cuanto a la zona a, al ser la *cache* extremadamente pequeña en relación al tamaño del sistema, el número de fallos de *cache* que se produce por conflicto o capacidad en una *cache* convencional es tan elevado que, en el caso extremo, ninguna referencia generada por el procesador produce acierto nunca, ya que los bloques de memoria principal son reemplazados de la *cache* por otros bloques antes de volver a ser ejecutados. Por el contrario, al utilizar *cache* con bloqueo, algunos bloques, en principio los más interesantes según el algoritmo genético, se mantienen siempre en *cache*, produciendo siempre acierto, mientras que el resto de bloques, que son mayoría, producen fallo de *cache* cada vez que son ejecutados, ya que deben ser cargados en el buffer temporal en cada instancia. Estos bloques no escogidos por el algoritmo genético presentan el mismo comportamiento en la *locking cache* que en la *cache* convencional, ya que deben ser transferidos desde memoria principal cada vez que deben ser ejecutados. Sin embargo, al utilizar la *locking cache* un pequeño número de bloques toma ventaja de su precarga y bloqueo en *cache*, produciendo la mejora de *prestaciones* apreciada en la gráfica.

Por último, de la zona central (zona b) es imposible extraer ninguna conclusión, aunque parece que las *prestaciones* obtenidas por la *cache* con bloqueo serán peores que las obtenidas por la *cache* convencional. El objetivo del resto del capítulo es encontrar una explicación a la variabilidad en las *prestaciones* obtenidas por las *cache*s con bloqueo, identificando las situaciones donde la probabilidad de obtener determinismo sin pérdida de *prestaciones* sea mayor.

Para aumentar las posibilidades de éxito en el momento de realizar los análisis de regresión, se ha realizado una agrupación de los puntos mostrados en la Figura 37. Se han definido siete grupos, correspondientes cada uno de ellos a los siete tamaños de *cache* utilizados en los experimentos. El tamaño máximo del código de un experimento es de 64KB, por lo que para definir los límites superiores y los experimentos que se integran en cada grupo se han utilizado las siguientes formulas:

- El límite superior del grupo n se define como z/64, donde $z=2^n$ y n = 1,2,3,4,5,6
- El límite superior del grupo n=7 se define con valor infinito, ya que agrupa todos los sistemas cuyo tamaño es igual o mayor que el tamaño de la *cache*.
- Un experimento x, con SSR= r, pertenecerá al grupo n si:
  límite superior del grupo n-1 ≤ r < límite superior del grupo n

Los límites de cada grupo se muestran en la Tabla 12.

Grupo	**Tamaño *cache* Kbytes**	**Límite inferior**	**Límite superior**
1	1	0	0,0313
2	2	0,0313	0,0625
3	4	0,0625	0,125
4	8	0,125	0,25
5	16	0,25	0,5
6	32	0,5	1,0
7	64	1,0	∞

Tabla 12. Valores límite de SSR para la agrupación de $\pi = U_{cf}/U_{lef}$.

La Figura 38 muestra el resultado obtenido de la agrupación, así como la media, mediana, y el primer y tercer cuartil. Además, al realizar está agrupación, se puede obtener un número importante de características estadísticas para cada grupo. Estos resultados se muestran en la Tabla 13.

Grupo	1	2	3	4	5	6	7
Total datos	18	18	28	28	28	26	36
Media	1,051	1,033	0,969	0,773	0,669	0,828	1,049
Mediana	1,043	1,049	1,059	0,663	0,667	0,868	1,03
Des. estándar	0,042	0,151	0,275	0,295	0,188	0,18	0,047
Intervalo de Conf. para la media.	±0,019	±0,07	±0,102	±0,109	±0,07	±0,069	±0,015
Nº de éxitos ($\pi \geq 1$)	18	14	19	7	1	6	36
Probabilidad de éxito	100,00%	77,78%	67,86%	25,00%	3,57%	23,08%	100,00%
Máximo	1,156	1,253	1,358	1,295	1,038	1,067	1,16
Mínimo	1,003	0,672	0,419	0,353	0,376	0,509	1,002
1 ^{er} cuartil	1,018	1,026	0,883	0,532	0,542	0,716	1,016
3er cuartil	1,064	1,104	1,138	1,009	0,806	0,993	1,094
Coeficiente de curtosis	1,351	2,247	-0,383	-1,105	-0,997	-1,152	-0,005
Coeficiente de asimetría	1,325	-1,382	-0,86	0,482	-0,067	-0,47	1,108

Tabla 13. Principales estadísticos de la agrupación de $\pi = U_{cf}/U_{lef}$ en función de SSR.

Tanto en la gráfica como en la tabla de valores estadísticos se puede observar que para valores de SSR próximos a los extremos, el determinismo se obtiene ya no sin pérdida de *prestaciones*, sino incluso con una mejora de *prestaciones* para un número importante de experimentos. Las *prestaciones* obtenidas por la *cache* con bloqueo empeoran en la zona central, donde las *cache*s convencionales realizan un mejor aprovechamiento del espacio disponible debido a su comportamiento dinámico. Los valores de la media y mediana muestran que esta pérdida de *prestaciones* en la zona central no es muy elevada (0,669 y 0,667 respectivamente) pero la probabilidad de éxito para los grupos 4, 5 y 6, con valores inferiores al 25%, llegando al 3%, indican que esta pérdida de *prestaciones* es generalizada para dichos valores de SSR. Es más, los valores mínimos muestran una pérdida de *prestaciones* cercana al 70% en casos extremos, y atendiendo a los intervalos intercuartílicos, el 50% de los experimentos obtiene unas *prestaciones* entre el 50% y el 80% para el grupo 5, una variabilidad elevada que se pone de manifiesto también en el grupo 6, pero especialmente en el grupo 4.

Estos datos muestran la importancia de la relación entre los tamaños del sistema y de la *cache*, ya que en función de la zona en que se sitúe un sistema, las posibilidades de conseguir un sistema determinista sin pérdida significativa de *prestaciones* son mayores o menores. Sin embargo, existe una variabilidad importante en las *prestaciones* obtenidas dentro de cada grupo, por lo que la relación entre el tamaño de la *cache* y los tamaños de las tareas proporciona información insuficiente para tomar una decisión. Así pues, la utilización de métodos estadísticos avanzados, como los modelos de regresión lineal, se hacen indispensables para extraer información útil sobre la ganancia o pérdida de *prestaciones* de un determinado sistema, sin necesidad de simularlo o evaluarlo.

Aunque los valores estadísticos mostrados anteriormente tienen un valor innegable, existe una información todavía más útil para los propósitos de este análisis estadístico que se puede extraer de la agrupación de los resultados presentados en la Figura 38. En dicha figura pueden identificarse cuatro zonas donde los valores de la media, mediana e intervalos intercuartílicos pueden aproximarse a una recta. De esta forma, en lugar de realizar un único modelo de regresión para todos los datos, se realizarán cuatro modelos de regresión, uno para cada uno de los cuatro espacios. Esto permite obtener modelos más ajustados, y sobre todo, más simples, por lo que su interpretación y utilización será más sencilla. Los espacios definidos se muestran de forma aproximada en la Figura 39, y sus límites exactos se describen en la Tabla 14.

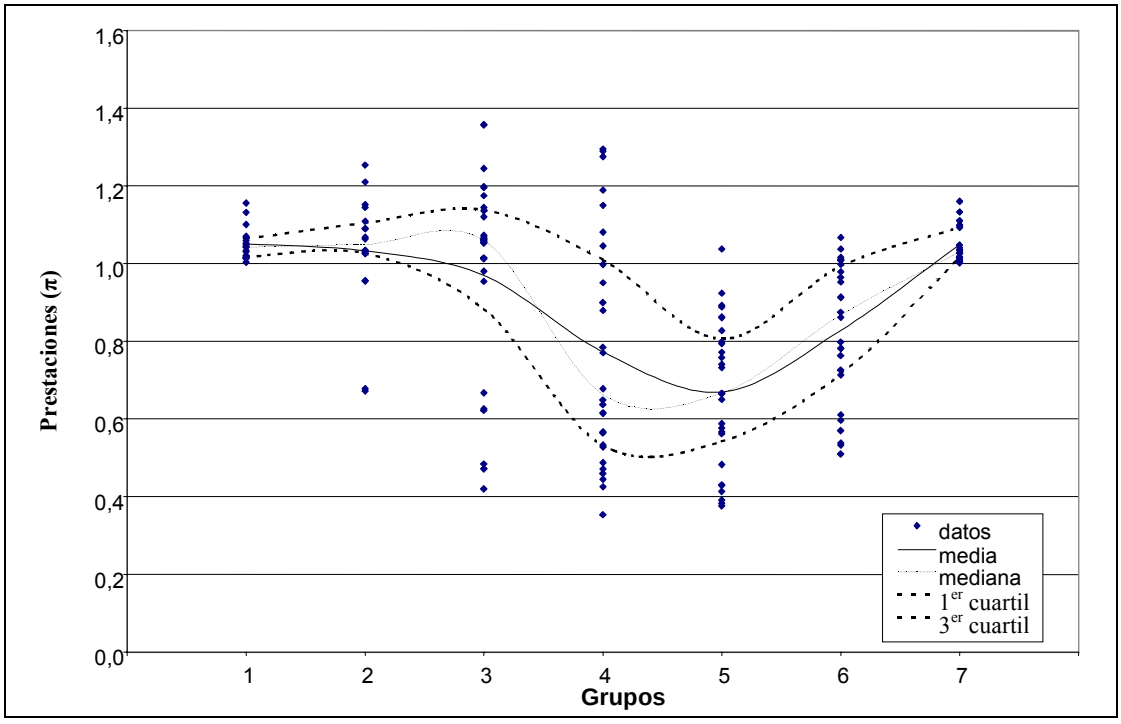


Figura 38. Agrupación de $\pi = U_{cf}/U_{lef}$ en función de SSR.

El límite inferior y superior asignado a cada espacio y mostrados en la Tabla 14 se ha establecido sobre la base de las divisiones mostradas en la Figura 39, pero se han ajustado a sus valores actuales a través de los modelos de regresión, buscando los límites que ofrecían los mejores resultados para cada espacio. La columna "Total datos" contabiliza el número total de experimentos que se enmarcan dentro de cada espacio. Las dos últimas columnas representan los valores de los coeficientes de asimetría y curtosis estándar para cada espacio. Estos valores son importantes para determinar la normalidad o no de los datos. Valores fuera el rango -2 a +2 indican desviaciones importantes respecto a una distribución normal, lo que desaconsejaría e incluso imposibilitaría la utilización de los modelos de regresión lineal.

Espacio	**Límite inferior**	**Límite superior**	**Total datos**	**Asimetría estándar**	**Curtosis estándar**
A	0	≤0,065	42	-4,82	8,42
B	>0,065	<0,3	61	-0,68	-2,12
C	≥0,3	<1	43	-2,12	-0,06
D	≥1	∞	36	2,4	-0,23

Tabla 14. Valores límites de SSR para la creación de los cuatro espacios de $\pi = U_{cf}/U_{lef}$.

A continuación se muestran los modelos de regresión obtenidos para cada uno de los cuatro espacios considerando únicamente los factores *software*. Junto con el modelo se presenta una interpretación de los resultados obtenidos y la validación del modelo a través del análisis de los residuos.

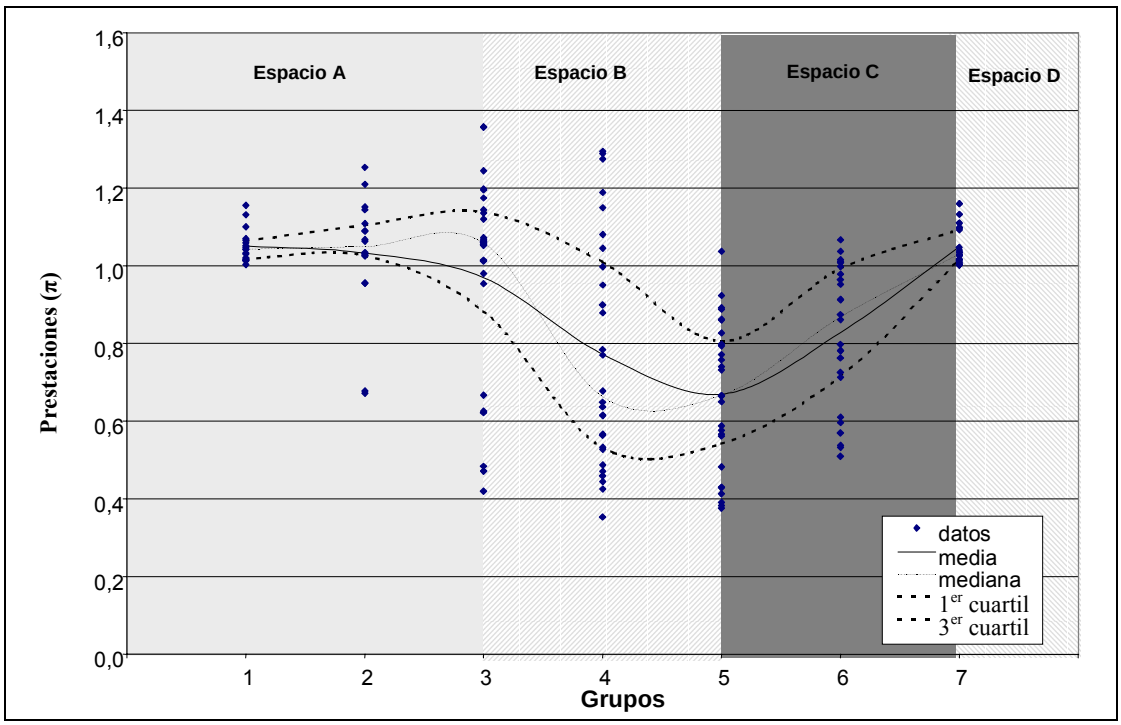


Figura 39. Identificación de los espacios de $\pi = U_{cf}/U_{lef}$ en función de SSR.

4.3.1 Modelo de los factores software para el Espacio A

La teoría de los modelos de regresión lineal indica que, para que los resultados sean válidos, los datos utilizados deben proceder de una distribución normal. Aunque existen muchos métodos para validar la normalidad o no de unos datos, estos tests suelen ser excesivamente estrictos, por lo que es más interesante comprobar que puede asumirse que los datos son normales [Romero93] antes de evaluar exactamente si lo son. Los valores mostrados en la Tabla 14 muestran claramente que el espacio A no se distribuye de forma normal ya que los valores de asimetría y curtosis estándar se encuentran fuera del rango considerado habitual para los datos normales. Sin embargo, la Figura 40 representa en papel probabilístico normal los datos incluidos en este espacio y en dicha figura puede verse que a excepción de dos datos, todos los demás se distribuyen alrededor de una recta, por lo que puede asumirse su normalidad para calcular el modelo de regresión.

Sin embargo, tras las primeras aproximaciones para construir el modelo de regresión lineal del espacio A se observó que los residuos presentaban cierta curvatura. Al igual que la normalidad de los datos para los que se quiere realizar el modelo es una premisa para obtener un modelo válido, la normalidad de los residuos obtenidos del modelo es un indicio de la validez de dicho modelo. La curvatura presentada por los residuos en su representación en papel probabilístico puede ser eliminada utilizando como datos a modelar el logaritmo de los datos originales. Considerando que la variable π presenta valores entre 0 y 2, y que el valor 1 representa el valor cero en la ganancia/pérdida de *prestaciones*, esta transformación era previsible, y los residuos obtenidos en los primeros modelos realizados para el logaritmo de π corroboran esta intuición. La representación de log (π) en papel probabilístico normal se muestra en la Figura 41, y aunque aparentemente no mejora la normalidad de los datos, los diferentes modelos realizados hasta llegar al que se presenta a continuación sí indican una mejora sustancial.

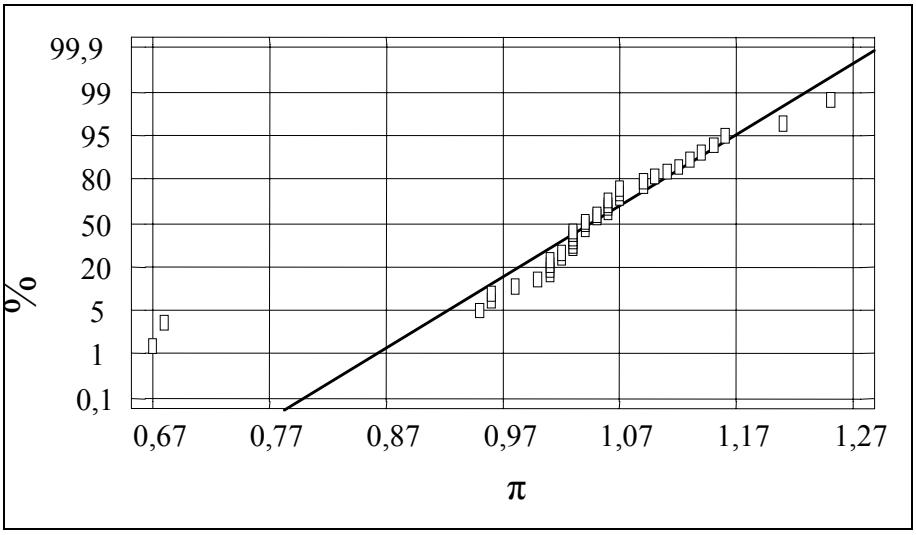


Figura 40. Representación de $\pi = U_{cf}/U_{lef}$ en papel probabilístico normal. Espacio A.

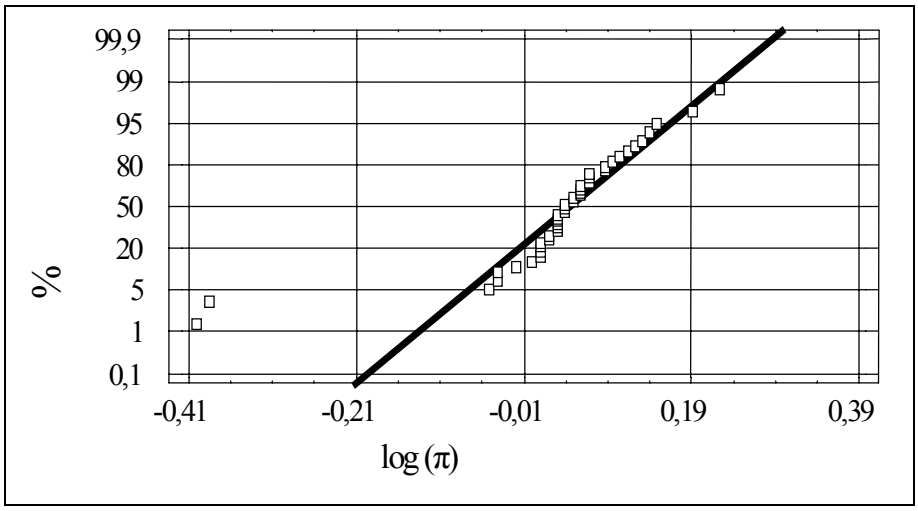


Figura 41. Representación de log ($\pi = U_{cf}/U_{lef}$) en papel probabilístico normal. Espacio A.

Así, aunque la variable bajo estudio es π, el modelo se desarrollará para log (π) sin pérdida de generalidad ni información. El modelo resultante se muestra a continuación:

log(π) = 1,90654 - 19,2279*SSR - 0,408003*TN - 0,000689129*SS - 0,000136394*IRF*ES + 1,99859*IRF*SSR - 0,13865*LOC*TN - 0,000865968*LOC*ES - 3,97403*LOC*SSR - 0,00122654*LOC*PS + 0,000607725*LOC*SS + 0,000107755*TN*ES + 3,3879*TN*SSR - 0,0000505736*TN*PS + 0,000161385*TN*SS + 0,0574987*ES*SSR + 0,00905212* SSR*PS - 0,0000044891*SSR*ET

El modelo contempla 17 variables independientes, con una R-Squared del 93,516%, lo que quiere decir que el modelo explica en algo más del 93% el comportamiento de log (π), es decir, la casi totalidad del comportamiento de log(π). El estadístico Durbin-watson, con un valor de 2,05, superior a 1,4 indica que no existe ningún indicio de autocorrelación en los residuos. La Tabla 15 muestra los coeficientes, el error normalizado y la significación de cada variable independiente, y la Tabla 16 el análisis de la varianza del modelo. En estas tablas se puede observar que todas las variables independientes y sus interacciones tienen un P-value inferior a 0,05, lo que indica que son estadísticamente significativas al 95% de confianza. Así mismo, el

modelo presenta un P-value de 0,0000 que al ser menor que 0,01 indica que existe una relación estadísticamente significativa entre el modelo y la variable dependiente al 99% de confianza.

Factor	Valor Estimado	Standard Error	T-Statistic	P-Value
Constante	1,90654	0,44996	4,23713	0,0003
SSR	-19,2279	8,52248	-2,25614	0,0334
TN	-0,408003	0,0903219	-4,51721	0,0001
SS	-0,000689129	0,000126946	-5,42854	0,0000
IRF*ES	-0,000136394	0,0000478412	-2,85098	0,0088
IRF*SSR	1,99859	0,41462	4,82029	0,0001
LOC*TN	-0,13865	0,0421035	-3,29307	0,0031
LOC*ES	-0,000865968	0,000140472	-6,16469	0,0000
LOC*SSR	-3,97403	1,00153	-3,96797	0,0006
LOC*PS	-0,00122654	0,000350832	-3,49608	0,0019
LOC*SS	0,000607725	0,000151982	3,99867	0,0005
TN*ES	0,000107755	0,000034911	3,08655	0,0050
TN*SSR	3,3879	1,34936	2,51075	0,0192
TN*PS	-0,0000505736	0,0000239378	-2,11271	0,0452
TN*SS	0,000161385	0,000029267	5,51421	0,0000
ES*SSR	0,0574987	0,0113823	5,0516	0,0000
SSR*PS	0,00905212	0,00425811	2,12585	0,0440
SSR*ET	-0,0000044891	6,05615E-7	-7,41246	0,0000

Tabla 15. Detalles del modelo de log ($\pi = U_{cf}/U_{lef}$). Espacio A.

Fuente	Suma de cuadrados	Grados de libertad	Mean Square	F-Ratio	P-Value
Modelo	0,485143	17	0,0285378	20,36	0,0000
Residual	0,0336379	24	0,00140158		
Total(Corr.)	0,518781	41			

Tabla 16. Análisis de la varianza del modelo de log ($\pi = U_{cf}/U_{lef}$). Espacio A.

La obtención del modelo permite realizar predicciones sobre los resultados que se obtendrán para un determinado sistema, simplemente reemplazando los factores del modelo por los valores concretos del sistema que se desea evaluar (el lector no debe olvidar que este modelo es válido sólo para sistemas con un SSR -relación entre el tamaño de la *cache* y el tamaño de las tareas del sistema- menor o igual que 0,065). Pero la misma importancia o más tiene para el diseñador de un sistema de tiempo real conocer la influencia de cada factor sobre el modelo. Esta influencia puede obtenerse directamente de los coeficientes asignados a cada factor, siempre y cuando los parámetros o variables independientes vengan expresadas en la misma magnitud. Dado que este no es el caso, ya que por ejemplo los valores para el número de tareas TN oscila entre 3 y 8, mientras que el tamaño del sistema SS toma valores entre los pocos cientos y más de 3.000, es necesario recurrir a otra herramienta estadística para obtener la importancia de cada factor en el modelo. La Tabla 17 muestra el resultado de la suma de cuadrados condicional, donde valores elevados de la columna F-ratio indican un nivel de significación importante para un determinado factor. Para permitir una interpretación más sencilla del modelo, los factores se

muestran ordenados de forma descendente en función de la F-Ratio, es decir, de mayor a menor importancia, junto con el coeficiente estimado en el modelo para cada factor.

Fuente	F-Ratio	Coeficiente estimado
LOC*SS	91,39	0,00060773
SSR*ET	54,94	-4,4891E-06
SS	43,34	-0,00068913
SSR*ES	26,86	0,0574987
SSR*TN	26,19	3,3879
TN	24,25	-0,408003
LOC*TN	20,94	-0,13865
IRF*SSR	18,05	1,99859
SSR*PS	16,58	0,00905212
SSR	6,77	-19,2279
TN*ES	5,51	0,00010776
LOC*SSR	3,27	-3,97403
TN*SS	3,06	0,00016139
LOC*ES	2,88	-0,00086597
LOC*PS	1,92	-0,00122654
IRF*ES	0,14	-0,00013639
TN*PS	0,06	-5,0574E-05

Tabla 17. Impacto de los factores (orden descendente) sobre el modelo de log ($\pi = U_{cf}/U_{lef}$). Espacio A.

Antes de utilizar el análisis del espacio A para predecir las *prestaciones* obtenidas por la *locking cache* o evaluar cómo afectan los distintos factores a dichas *prestaciones*, es necesario validar el modelo. Esta validación puede realizarse de forma poco rigurosa representando los valores reales de los datos obtenidos de los experimentos realizados, frente a los estimados por el modelo para los sistemas utilizados en la realización de los experimentos. La distribución de los puntos a lo largo de una línea, tal como muestra la Figura 42, indica que el modelo se ajusta a la realidad. Esta representación es, como se ha dicho, poco rigurosa para validar el modelo, siendo mucho más útil el análisis de los residuos. Si el modelo es válido, y no existe ninguna anomalía, los residuos deben distribuirse de forma normal con media 0. La Figura 43 muestra los residuos del modelo en papel probabilístico normal, pudiéndose asumir su normalidad, y la Tabla 18 muestra los valores de media, asimetría y curtosis, que corroboran la normalidad de los residuos.

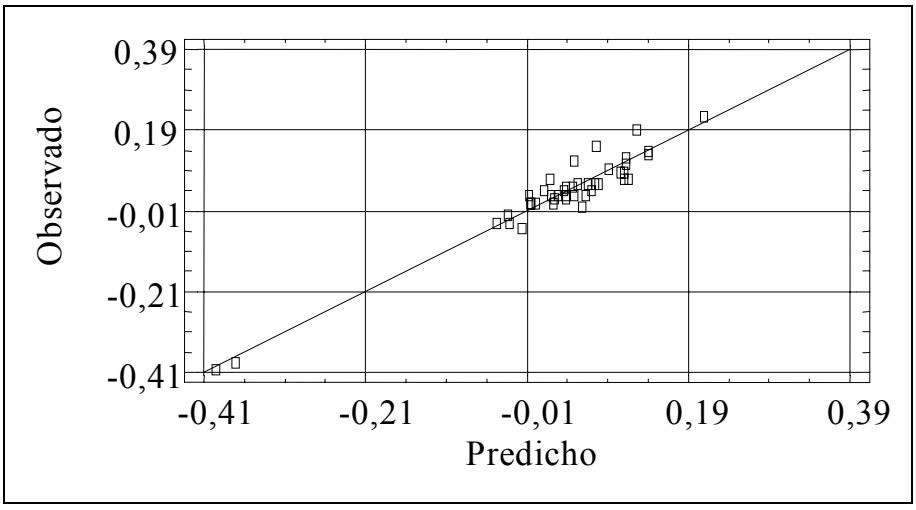


Figura 42. Representación de los valores de log ($\pi = U_{cf}/U_{lef}$) observados frente a los predichos por el modelo para el espacio A.

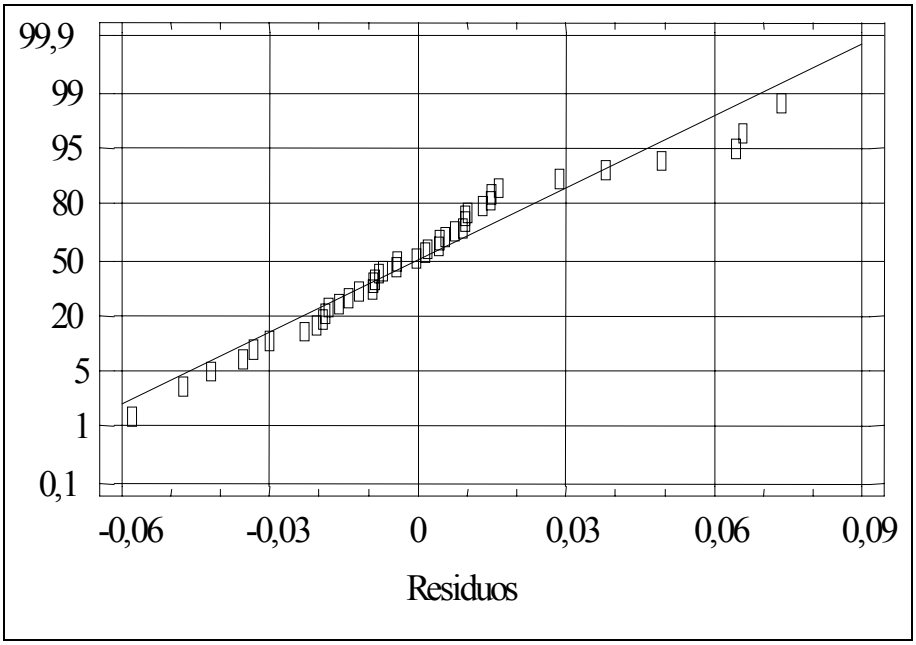


Figura 43. Representación en papel probabilístico normal de los residuos del modelo para el espacio A.

Número de datos	42
Media	1,2381E-9
Varianza	0,000820436
Desviación estándar	0,0286433
Mínimo	-0,0581027
Máximo	0,0734634
Asimetría estándar	1,7932
Curtosis estándar	1,17477

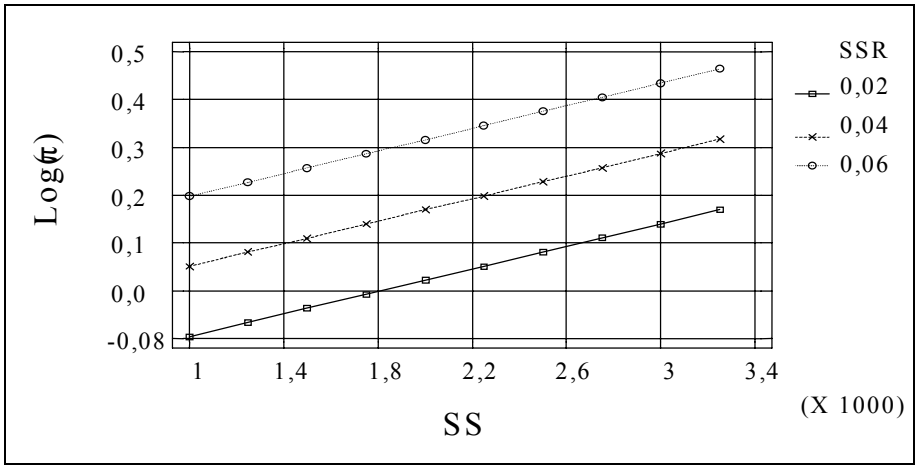
Tabla 18. Principales estadísticos de los residuos del modelo de log ($\pi = U_{cf}/U_{lef}$). Espacio A.

El elevado número de factores significativos y la existencia de interacciones entre variables cuantitativas hacen compleja la interpretación y la extracción de conclusiones. Con el objetivo de discriminar el efecto de las variables explicativas de forma individual, el modelo se ha evaluado modificando una variable y manteniendo constante el resto. Esta forma de proceder es inviable sino se dispone del modelo, ya que el número de experimentos necesarios para extraer conclusiones y detectar las diferentes interacciones entre los factores es ingente, pero al disponer del modelo, este procedimiento puede aplicarse de forma relativamente sencilla. Sin embargo, la complejidad del modelo, especialmente por la existencia de interacciones entre factores cuantitativos, hace casi imposible la justificación del comportamiento de log (π). Las gráficas que se muestran a continuación pretenden servir de ayuda en el conocimiento del comportamiento de las *prestaciones* en el uso estático de *locking cache*, pero sería necesario un análisis mucho más extenso para identificar las causas de dicho comportamiento.

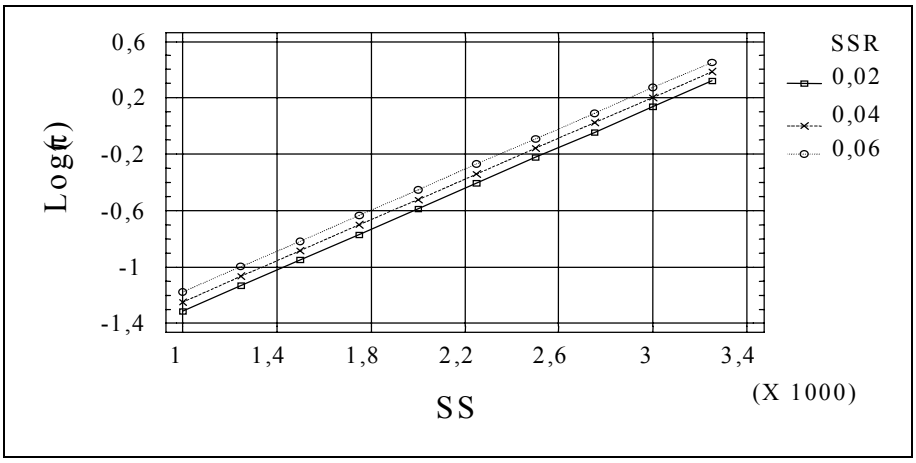
En primer lugar, la variable que por sí sola explica más variabilidad en los datos es el tamaño total de las tareas o sistema (SS). La Figura 44-a) muestra la evolución de las *prestaciones* según crece el tamaño total de las tareas suponiendo que las tareas no hacen un uso intensivo de la *cache* (LOC 0), que tienen un nivel alto de interferencia (IRF 1), y manteniendo constante el resto de características. Se muestran tres sistemas con tres valores de SSR diferentes.

Como se puede apreciar, al aumentar el tamaño total de las tareas las *prestaciones* crecen de forma constante. También puede observarse que la relación entre el tamaño de las tareas y la *cache* –SSR- afecta a las *prestaciones* positivamente, aunque no influye en el efecto del tamaño del sistema, es decir, no existe interacción entre ambos factores, ya que las tres rectas son paralelas. Aunque el modelo muestra que el efecto de SSR es muy bajo, entre los cinco factores más influyentes aparecen tres interacciones que incorporan este factor, lo que indica que realmente es significativo, aunque ponderado por otras características del sistema. La Figura 44–b) muestra los mismos tres sistemas que la Figura 44-a), pero cuando las tareas que lo forman hacen un uso intensivo de la *cache* (LOC 1). Los efectos y tendencias son similares en ambas gráficas, casi idénticos. Aunque visualmente el efecto de SSR parece menor, esto se debe tan sólo a la diferencia de escala en el eje Y. Donde sí existe diferencia es en los valores de log (π) entre ambas figuras, que al compararlos se aprecia que las *prestaciones* obtenidas por la *locking cache* cuando las tareas hacen un aprovechamiento intensivo de la *cache* son menores.

La Tabla 19 muestra los valores asignados a los parámetros que se han mantenido constantes.



a) LOC = 0



b) LOC = 1

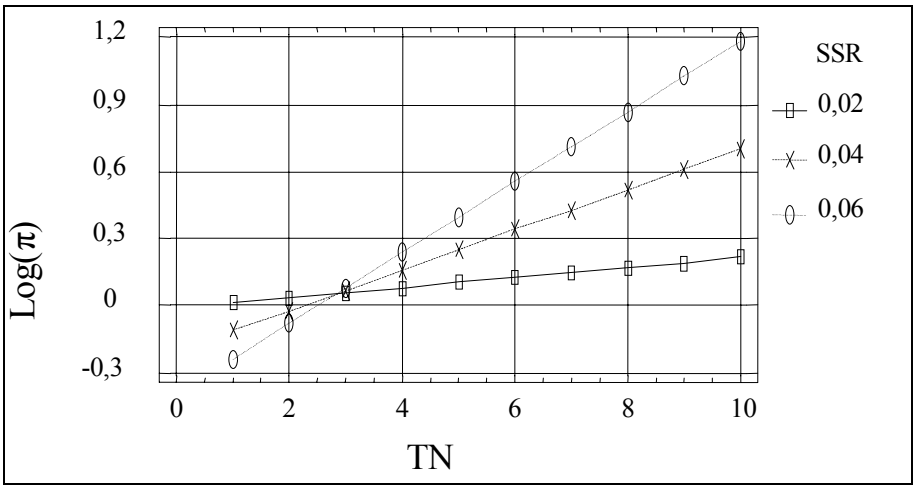
Figura 44. Evolución de log ($\pi = U_{cf}/U_{lef}$) en función de SS y SSR. Espacio A.

Factor	Valor asignado
ET	365281
ES	26,135635
TN	5
IRF	1
PS	857

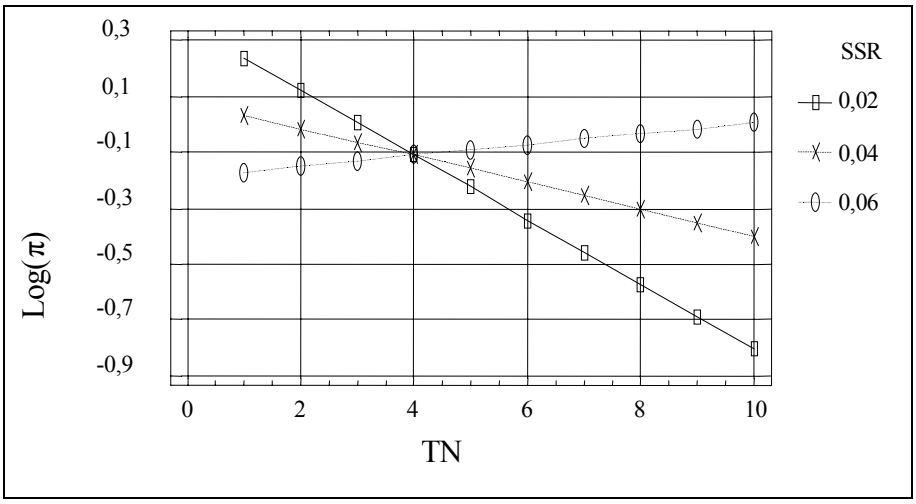
Tabla 19. Valores de los factores para la obtención de la Figura 44.

La Figura 45-a) muestra el efecto del número de tareas (TN) para tres sistemas, con diferentes valores de SSR y que no hacen un uso intensivo de la *cache* (LOC 0). En la figura puede observarse que al aumentar el número de tareas, las *prestaciones* del sistema mejoran, pero su pendiente se encuentra condicionada por el valor de SSR, es decir, existe una clara interacción entre los dos factores. La Figura 45-b) muestra los tres mismos sistemas, pero cuando sus tareas sí hacen un uso intensivo de la *cache* (LOC 1). En este caso, la situación es la contraria. Al aumentar el número de tareas, las *prestaciones* tienden a decrecer, pero condicionado este comportamiento por el valor de SSR, llegando incluso a no perder *prestaciones* cuando el sistema se encuentra en el límite derecho del espacio, es decir, cuanto menor es la diferencia entre el tamaño de las tareas y de la *cache*. La Tabla 20 muestra los valores asignados a los parámetros que se han mantenido constantes para la obtención de la Figura 45-a) y b)

En la Figura 45-a) y b) se puede observar que las líneas se cruzan en un determinado valor de TN. Esto, junto con las interacciones que aparecen en el modelo, y en las que se encuentra presente de forma significativa el número de tareas, indica que esta característica de la carga modificará la influencia que otros factores tendrán sobre las *prestaciones* del sistema, como por ejemplo la inversión en la tendencia del efecto de SSR, o como puede observarse en la Figura 46-a), la inversión del efecto del tamaño del sistema, que para sistemas formados por tres tareas será negativo, contrariamente al efecto observado en la Figura 44-a) para sistemas con 5 tareas. La Figura 46-b) muestra los mismos sistemas que la Figura 46-a) pero cuando éstos hacen un uso intensivo de la *cache* (LOC 1). El efecto del número de tareas es patente y similar en ambas gráficas, pero no llega a invertir la tendencia de las rectas hacia pendientes negativas. La Tabla 21 muestra los valores asignados a los parámetros que se han mantenido constantes.



a) LOC = 0

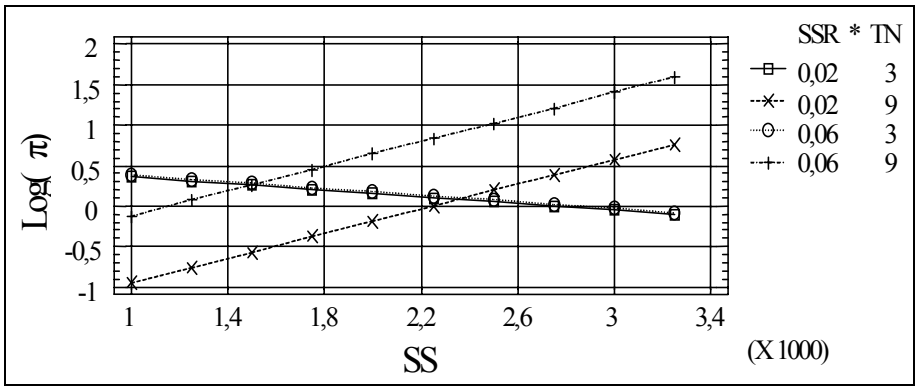


b) LOC = 1

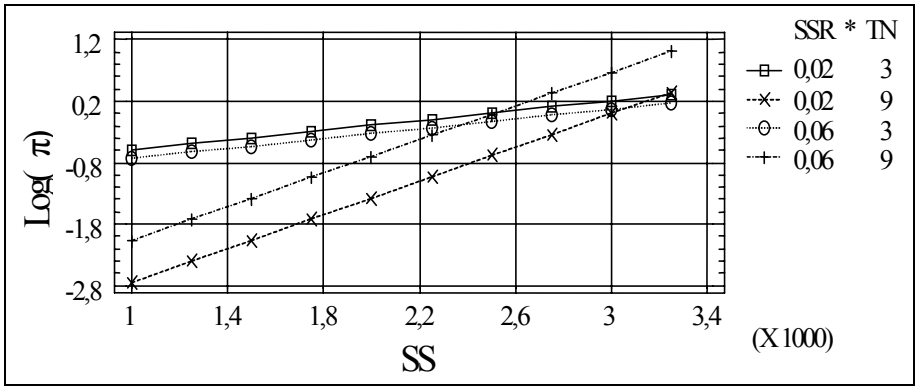
Figura 45. Evolución de log ($\pi = U_{cf}/U_{lef}$) en función de TN y SSR. Espacio A.

Factor	Valor asignado
ET	365281
ES	26,135635
SS	2500
IRF	1
PS	857

Tabla 20. Valores de los factores para la obtención de la figura Figura 45.



a) LOC = 0



b) LOC = 1

Figura 46. Evolución de log ($\pi = U_{cf}/U_{lef}$) en función de SS y SSR * TN. Espacio A.

Factor	Valor asignado
ET	365281
ES	26,135635
IRF	1
PS	857

Tabla 21. Valores de los factores para la obtención de la Figura 46.

En las gráficas anteriores se ha presentado el efecto de aquellas variables explicativas que aparecen de forma independiente en el modelo, es decir, de aquellos factores que directamente influyen en el comportamiento de log(π). Otros factores aparecen en el modelo formando parte de interacciones dobles con un valor elevado de F-Ratio. Este es el caso del número total de instrucciones ejecutadas (ET) y el ratio de instrucciones ejecutadas en función del tamaño de las tareas (ES), ambos en interacción con SSR. En el primer caso, el coeficiente estimado para ET*SSR es negativo, lo que quiere decir que según aumente el número total de instrucciones ejecutadas, las *prestaciones* de la *locking cache* empeorarán. En el segundo caso el coeficiente de ES*SSR es positivo, lo que significa que las *prestaciones* de la *locking cache* aumentarán según aumente el número de instrucciones ejecutadas en relación con el tamaño de las tareas. Esta modificación de las *prestaciones*, para ambos factores, se producirá en relación con el valor de SSR, es decir, modificará ligeramente las pendientes presentadas en las gráficas anteriores, pero no llegará a modificar su signo.

Otro factor que aparece en una interacción doble con SSR es el nivel de interferencia (IRF). Para esta interacción el coeficiente es positivo, lo que indica que para sistemas con un alto nivel

de interferencia, las *prestaciones* de la *locking cache* serán mejores que las de las *cache*s convencionales. Esto es debido a que el *cache-refill penalty* para las *locking cache* es constante y muy bajo (T_{miss}, tiempo de recarga de una línea de *cache*) mientras que para las *cache*s convencionales el valor del *cache-refill penalty* será mayor, pudiendo llegar a ser el tiempo necesario para recargar completamente la *cache*. Por este motivo, cuantas más expulsiones se produzcan (IRF 1) peores serán las *prestaciones* de la *cache* convencional, ya que el coste de cada expulsión es mayor que en el caso de la *locking cache*.

El resto de factores aparecen también en interacciones dobles, y con valores de F-Ratio muy bajos, lo que indica que su efecto en el comportamiento de la variable dependiente será mínimo, modificando ligeramente el efecto de los factores mostrados en las gráficas anteriores.

Es importante remarcar que este análisis del efecto de los factores es útil para identificar las tendencias que producen dichos factores, pero no para evaluar las *prestaciones* obtenidas, es decir, el valor de π ni el del $\log(\pi)$. No debe considerarse en ningún caso las *prestaciones* de forma absoluta, sino de forma relativa, comparándolas entre las diferentes gráficas y rectas que se presentan. Por tanto es posible comparar los valores de $\log(\pi)$ entre la Figura 44-a) y la Figura 44-b), pero no, por ejemplo, entre la Figura 44-a) y la Figura 45-a).

Como conclusión, los sistemas que se encuentren en el espacio A tendrán, en general, mejores *prestaciones* (relativas) al utilizar la *locking cache* cuanto más a la derecha se encuentren, es decir, cuanto menor sea la diferencia entre el tamaño de la *cache* y el tamaño total del sistema -valores elevados de SSR- aunque esta tendencia se ve fuertemente condicionada por el número de tareas en el sistema. Esta mejora en el valor de π se podía apreciar ligeramente en la mediana de la Figura 39. Por otro lado, el efecto del número de tareas del sistema (TN), y del tamaño total del sistema (SS), que se muestran como los dos factores más influyentes, se encuentra condicionado por la utilización que las tareas hagan de la *cache* (LOC). Por tanto, se deberán tratar de forma distinta estos dos tipos de sistema, ya que los efectos de los factores pueden invertirse en función de esta característica, al igual que los sistemas formados por pocas tareas -por debajo de 5 tareas, aproximadamente- para los cuales la pendiente de los efectos puede cambiar de signo, y no sólo en magnitud.

4.3.2 Modelo de los factores software para el Espacio B

Al igual que para el espacio A, antes de presentar el modelo para los datos enmarcados en el espacio B se verificará su normalidad. En este caso, tal como muestra la Tabla 13, el valor de asimetría se encuentra dentro los límites de una distribución normal, mientras que el valor de curtosis supera por poco el límite para considerar que los datos siguen una distribución normal. La Figura 47 muestra la distribución en papel probabilístico normal de los datos pertenecientes a este espacio ($0{,}065 < SSR < 0{,}3$). En esta gráfica se observa que la mayor parte de los datos se distribuyen alrededor de una recta, aunque con cierta curvatura, que se reduce al utilizar el logaritmo de π del mismo modo que se realizó en el espacio A. Pese a esta transformación, que se muestra en la Figura 48, unos pocos datos se desvían de la recta, pero la normalidad de los datos puede aceptarse para la realización del modelo. Estos datos corresponden a aquellos sistemas que por su valor de SSR se encuentran próximos a los límites inferior y superior del espacio, donde la aproximación a una recta es más forzada.

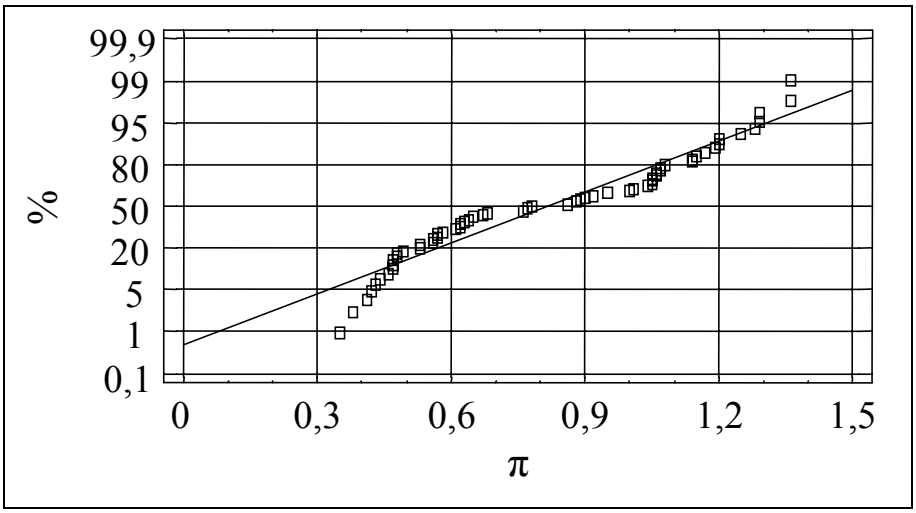


Figura 47. Representación de $\pi = U_{cf}/U_{lef}$ en papel probabilístico normal. Espacio B.

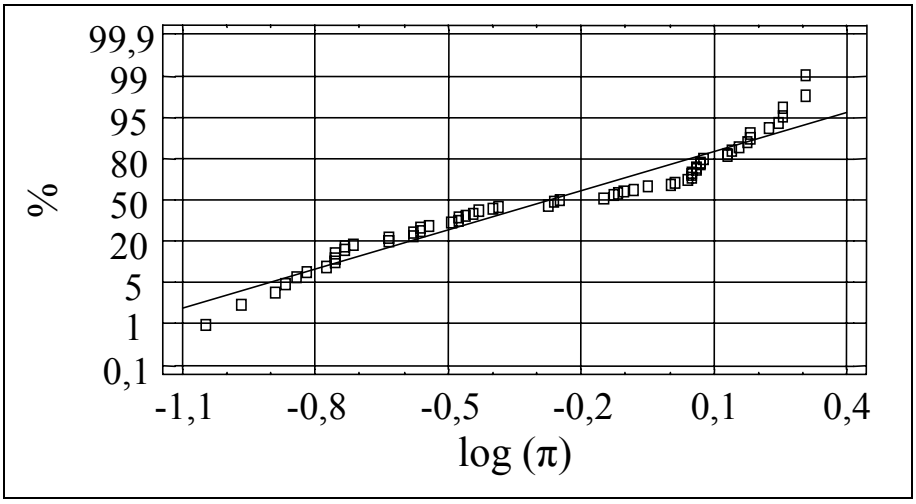


Figura 48. Representación de log ($\pi = U_{cf}/U_{lef}$) en papel probabilístico normal. Espacio B.

El modelo para el log (π) de los sistemas incluidos en el espacio B es el siguiente:

log (π) = 7,61661 - 1,88823*TN + 0,0182397*ES + 0,00279844*PS - 0,00434027*SS -0,000002255*ET - 0,000463993*IRF*ES +1,3525*IRF*SSR - 2,18453*LOC*TN - 0,00435036*LOC*ES - 0,0195533*LOC*PS + 0,00847027*LOC*SS -0,000360763*TN*PS +0,00103008*TN*SS + 1,96222E-7*TN*ET- 0,000973407*SSR*SS +1,57844E-7*SSR*ET

El modelo incorpora 16 variables independientes, con una R-Squared del 90,411%, lo que quiere decir que el modelo explica en algo más del 90% el comportamiento de log(π), o lo que es lo mismo, que el 10% de la variabilidad observada en los datos no puede ser explicada por el modelo obtenido. El estadístico Durbin-watson, con un valor de 3,06, muy superior a 1,4 indica que no existe ningún indicio de autocorrelación en los residuos. La Tabla 22 muestra los coeficientes, el error normalizado y la significación de cada variable independiente, y la Tabla 23 el análisis de la varianza del modelo. En estas tablas se puede observar que todas las variables independientes y las interacciones tienen un P-value inferior a 0,05, lo que indica que son estadísticamente significativas con un nivel de confianza del 95%. Así mismo, el modelo presenta un P-value de 0,0000 que al ser menor que 0,01 indica que existe una relación

estadísticamente significativa entre las variables explicativas y la variable dependiente al 99% de confianza.

Factor	Valor Estimado	Standard Error	T-Statistic	P-Value
Constante	7,61661	3,54117	2,15088	0,0370
TN	-1,88823	0,82177	-2,29775	0,0264
ES	0,0182397	0,00553274	3,29669	0,0019
PS	0,00279844	0,00132266	2,11577	0,0401
SS	-0,00434027	0,0022598	-1,92064	0,0613
ET	-0,000002255	3,80039E-7	-5,93362	0,0000
IRF*ES	-0,000463993	0,000122499	-3,78774	0,0005
IRF*SSR	1,3525	0,278736	4,85226	0,0000
LOC*TN	-2,18453	0,715761	-3,05204	0,0038
LOC*ES	-0,00435036	0,00222883	-1,95186	0,0573
LOC*PS	-0,0195533	0,00604142	-3,23654	0,0023
LOC*SS	0,00847027	0,00274524	3,08544	0,0035
TN*PS	-0,000360763	0,00016455	-2,19242	0,0337
TN*SS	0,00103008	0,000505916	2,03606	0,0478
TN*ET	1,96222E-7	7,82544E-8	2,50749	0,0159
SSR*SS	-0,000973407	0,000116961	-8,32252	0,0000
SSR*ET	1,57844E-7	8,47691E-8	1,86204	0,0693

Tabla 22. Detalles del modelo de log ($\pi = U_{cf}/U_{lef}$). Espacio B.

Fuente	Suma de cuadrados	Grados de libertad	Mean Square	F-Ratio	P-Value
Modelo	8,42265	16	0,526416	25,93	0,0000
Residual	0,893264	44	0,0203014		
Total(Corr.)	9,31591	60			

Tabla 23. Análisis de la varianza del modelo de log ($\pi = U_{cf}/U_{lef}$). Espacio B.

La Tabla 24 muestra la importancia en el modelo de cada uno de los factores que aparecen en el modelo. Para poder interpretar con mayor facilidad el modelo, se presentan ordenados de mayor valor de F-Ratio (más significativo) a menor valor de F-Ratio (menos significativo), junto con el coeficiente estimado para cada factor.

Para validar el modelo obtenido, la Figura 49 representa los valores estimados de log (π) por el modelo frente a los valores obtenidos de los experimentos. Aunque existe una clara dispersión en los puntos representados, todos ellos se distribuyen de forma uniforme alrededor de una recta, por lo que puede asumirse que el modelo tiene un alto nivel de ajuste. La Figura 50 muestra la representación de los residuos en papel probabilístico normal, representación ésta mucho más útil para validar el modelo. La normalidad en la distribución de los residuos indica la validez del modelo. La Tabla 25 muestra los principales estadísticos de los residuos.

Fuente	F-Ratio	Coeficiente estimado
LOC*TN	74,77	-2,18453
TN	73,23	-1,88823
SSR*SS	72,11	-0,00097341
LOC*ES	60,48	-0,00435036
LOC*SS	39,54	0,00847027
TN*SS	36,67	0,00103008
ET	14,32	-2,255E-06
ES	9,96	0,0182397
PS	9,05	0,00279844
LOC*PS	8,52	-0,0195533
TN*PS	5,64	-0,00036076
SS	3,97	-0,00434027
SSR*ET	3,47	1,58E-07
IRF*ES	1,37	-0,00046399
TN*ET	0,9	1,96E-07
IRF*SSR	0,88	1,3525

Tabla 24. Impacto de los factores (orden descendente) sobre el modelo de log ($\pi = U_{cf}/U_{lef}$). Espacio B.

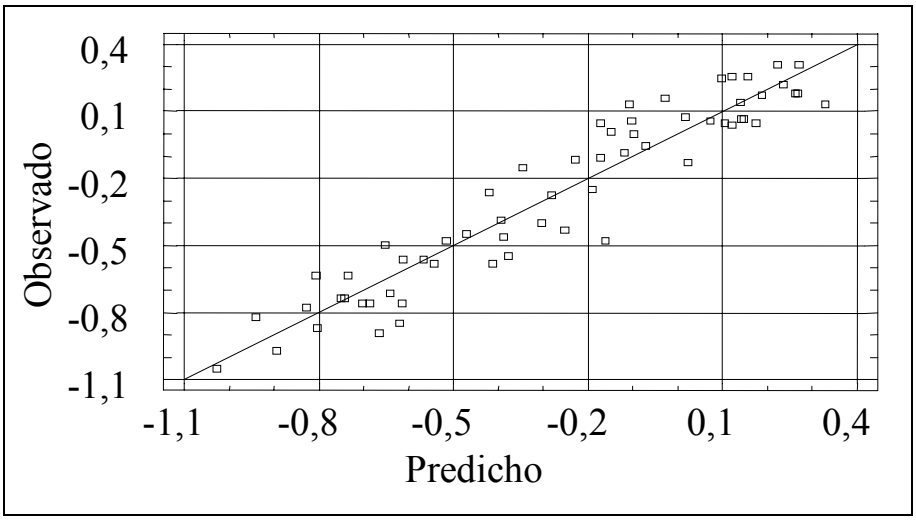


Figura 49. Representación de los valores de log ($\pi = U_{cf}/U_{lef}$) observados frente a los predichos por el modelo para el espacio B.

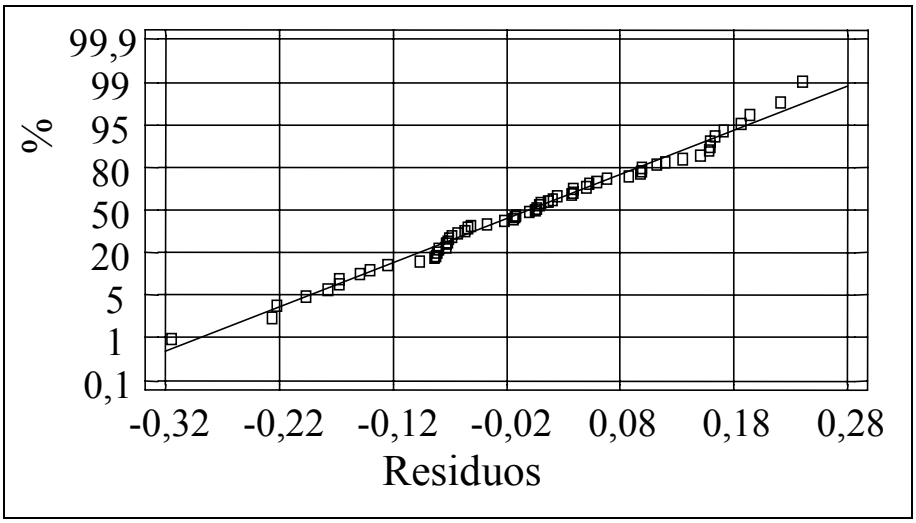


Figura 50. Representación en papel probabilístico normal de los residuos del modelo para el espacio B.

Número de datos	61
Media	2,16652E-8
Varianza	0,0148877
Desviación estándar	0,122015
Mínimo	-0,314822
Máximo	0,240421
Asimetría estándar	-0,504007
Curtosis estándar	-0,541061

Tabla 25. Principales estadísticos de los residuos del modelo de log ($\pi = U_{cf}/U_{lef}$). Espacio B.

La interpretación de este modelo es más sencilla que el correspondiente al espacio A, no porque tenga un menor número de variables explicativas, sino porque el número de interacciones es mucho menor, y se limitan básicamente a interacciones entre variables cualitativas y cuantitativas. El análisis se realizará del mismo modo que en la sección anterior, manteniendo fijos los valores de un conjunto de factores y estudiando el efecto de otro aplicando el modelo a cada combinación de valores de los factores.

La Figura 51 muestra el efecto del número de tareas en las *prestaciones* del sistema, considerando tres valores de SSR y los dos posibles valores de LOC. Este último factor es muy influyente, ya que determina el signo del efecto del número de tareas, siendo positivo para sistemas que no hacen un uso intensivo de la *cache*, y negativo para sistemas que sí hacen un uso intensivo de la *cache*. El efecto del factor SSR es casi inapreciable, como era de esperar al no existir interacción entre este factor y el número de tareas ni el factor LOC. La existencia de un punto de cruce para las dos grupos rectas pone de manifiesto la gran importancia que tiene el número de tareas en las *prestaciones* que se obtendrán, ya que puede invertir la tendencia de otros factores. La Tabla 26 muestra los valores asignados al resto de factores.

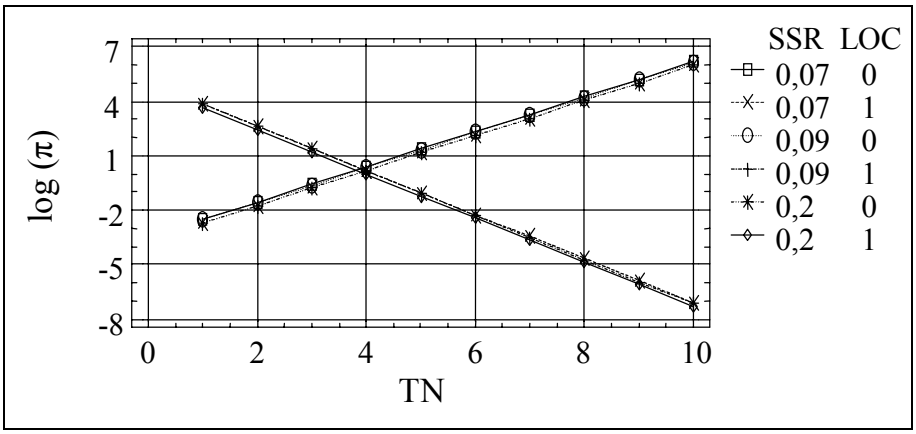


Figura 51. Evolución de log ($\pi = U_{cf}/U_{lef}$) en función de TN, SSR y LOC. Espacio B.

Factor	Valor asignado
ET	365281
ES	26,135635
SS	3000
IRF	1
PS	857

Tabla 26. Valores de los factores para la obtención de la Figura 51.

La Figura 52 muestran el efecto del tamaño de las tareas para tres valores de SSR diferentes y los dos distintos usos de *cache* (LOC 0 y 1). La Tabla 27 muestra los valores asignados a los parámetros que se han mantenido constantes. Los dos tipos de sistemas se comportan de manera muy distinta, no sólo en la pendiente, negativa para el primer tipo y positiva para el segundo, sino en la fuerza del efecto, ya que la pendiente para los sistemas que hacen un uso intensivo de la *cache* (LOC 1) es muy superior. El efecto de SSR es casi inapreciable para los dos tipos de sistemas, aunque la existencia de la interacción SSR*SS indica que el efecto de SSR se verá aumentado según aumente el tamaño del sistema, pero esta modificación es mínima frente al efecto de aumentar el tamaño del sistema.

Sin embargo, la existencia de una interacción entre SS y TN hará que el efecto de este factor se modifique en función del número de tareas, modificando las tendencias de los sistemas que no hacen un uso intensivo de la *cache*, como puede verse en la Figura 53. Los valores de los factores que no se han variado se presentan en la Tabla 28. El comportamiento observado en la Figura 53 podía deducirse de la Figura 51.

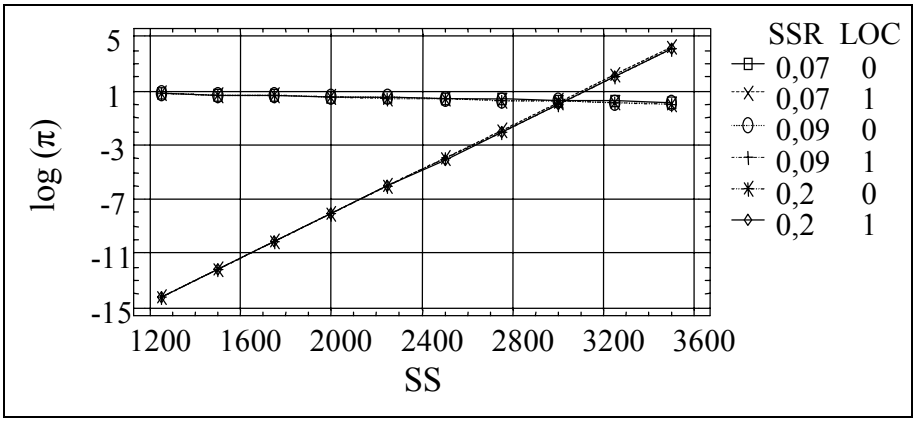


Figura 52. Evolución de log ($\pi = U_{cf}/U_{lef}$) en función de SS, SSR y LOC, con TN = 4. Espacio B.

Factor	Valor asignado
ET	365281
ES	26,135635
TN	**4**
IRF	0
PS	857

Tabla 27. Valores de los factores para la obtención de la Figura 52.

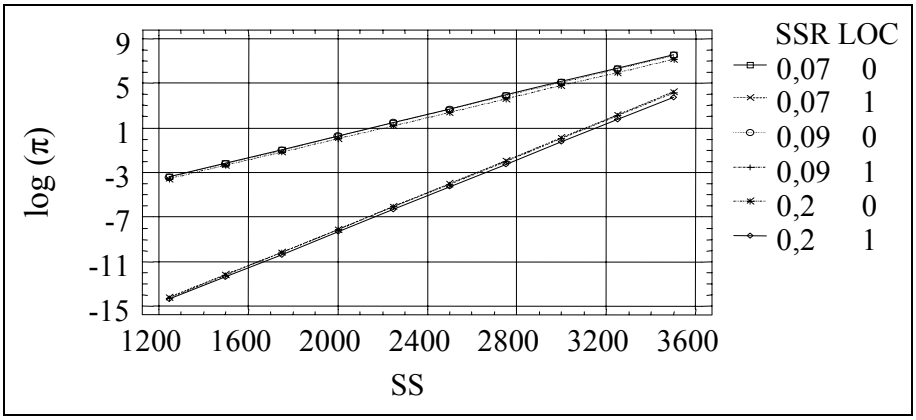


Figura 53. Evolución de log ($\pi = U_{cf}/U_{lef}$) en función de SS, SSR y LOC, con TN = 9. Espacio B.

Factor	Valor asignado
ET	365281
ES	26,135635
TN	**9**
IRF	0
PS	857

Tabla 28. Valores de los factores para la obtención de la Figura 53.

La Figura 54 muestra el efecto del factor ES para dos valores de SSR y para los dos tipos de uso de *cache*. El efecto de SSR es inapreciable, pero no así el de LOC, que hace más importante el efecto de ES para los sistemas que no hacen un aprovechamiento intensivo de la *cache*. La Tabla 29 presenta los valores del resto de los factores.

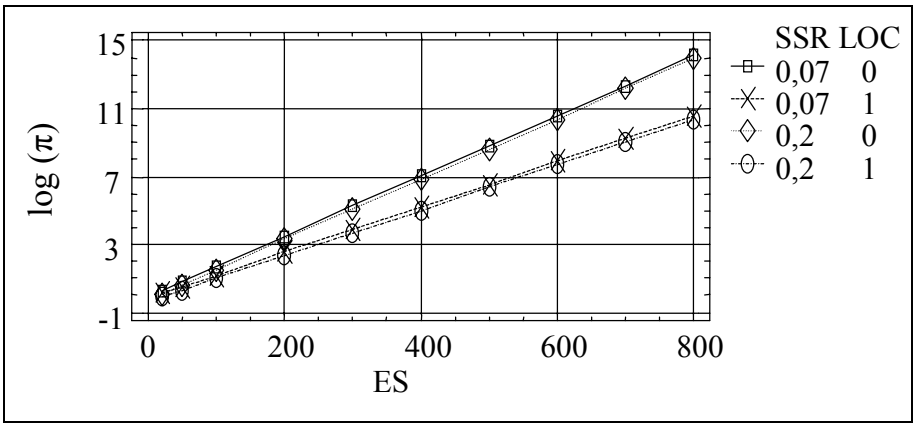


Figura 54. Evolución de log ($\pi = U_{cf}/U_{lef}$) en función de ES, SSR y LOC. Espacio B.

Factor	Valor asignado
ET	365281
SS	3000
TN	4
IRF	1
PS	857

Tabla 29. Valores de los factores para la obtención de la Figura 54.

La Figura 55 muestra el efecto del tamaño de la tarea más prioritaria del sistema. Una vez más, se aprecia el mismo comportamiento que en las gráficas anteriores: el efecto de SSR es inapreciable, y existen severas diferencias entre los sistemas que hacen un uso intensivo de la *cache* –pendiente negativa- de los que no – pendiente positiva casi nula -. Los valores asignados al resto de los factores se muestran en la Tabla 30.

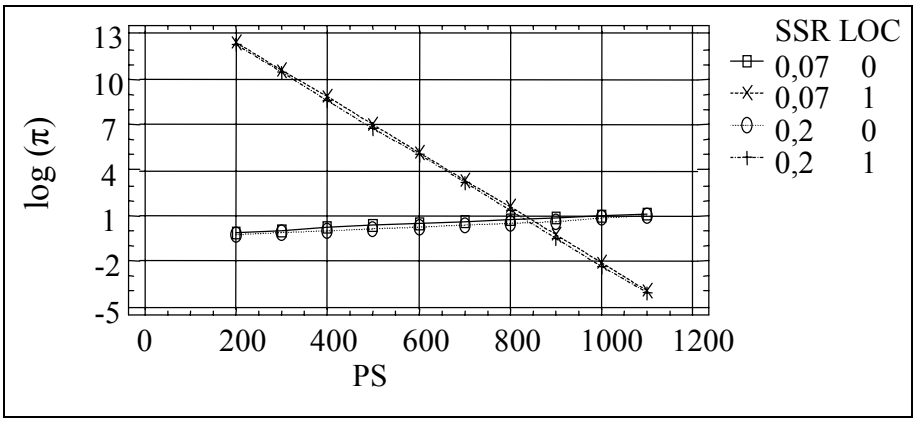


Figura 55. Evolución de log ($\pi = U_{cf}/U_{lef}$) en función de PS, SSR y LOC. Espacio B.

Factor	Valor asignado
ET	365281
SS	3000
TN	4
IRF	1
ES	50

Tabla 30. Valores de los factores para la obtención de la Figura 55.

La Figura 56 muestra el efecto del número total de instrucciones ejecutadas por el sistema. El comportamiento de este factor es diferente al estudiado anteriormente. El efecto es siempre el mismo, a mayor número de instrucciones ejecutadas, menores *prestaciones*. Existen diferencias en función del uso de la *cache* que haga el sistema –uso intensivo de *cache* presenta menores *prestaciones*- y del valor de SSR –a mayor valor, menores *prestaciones*- pero no existe interacción entre estos factores y el número total de instrucciones ejecutadas, ya que las rectas son paralelas. Además, comparando las escalas del eje Y con las figuras anteriores, se observa que la variación es mucho menor en el caso del factor ET. Los valores de los factores que permanecen constantes se detallan en la Tabla 31.

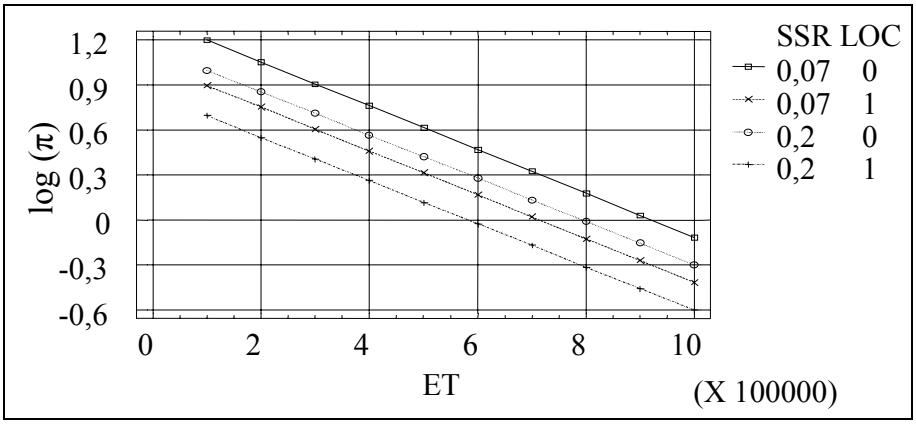


Figura 56. Evolución de log ($\pi = U_{cf}/U_{lef}$) en función de ET, SSR y LOC. Espacio B.

Factor	Valor asignado
PS	857
SS	3000
TN	4
IRF	1
ES	50

Tabla 31. Valores de los factores para la obtención de la Figura 56.

Aunque el factor SSR no aparece de forma individual en el modelo, y en la mayoría de las gráficas presentadas su efecto es nulo o inapreciable, sí existe una interacción en el modelo que incluye este factor con un valor F-ratio elevado, por lo que la Figura 57 muestra el efecto de SSR para cuatro sistemas, con los posibles valores de LOC e IRF. Se puede observar que realmente existe un efecto, siempre negativo, del factor SSR, aunque este es muy ligero, ya que la variación en las *prestaciones* es muy baja. El valor de IRF, es decir, el nivel de interferencia que se provoca entre las tareas del sistema condiciona el efecto de SSR, siendo más pronunciado

para aquellos sistemas que no sufren expulsiones. Y aunque el uso de la *cache* realizado por las tareas del sistema afecta a las *prestaciones* del sistema, no modifica el efecto de SSR. Los valores asignados al resto de factores se presentan en la Tabla 32.

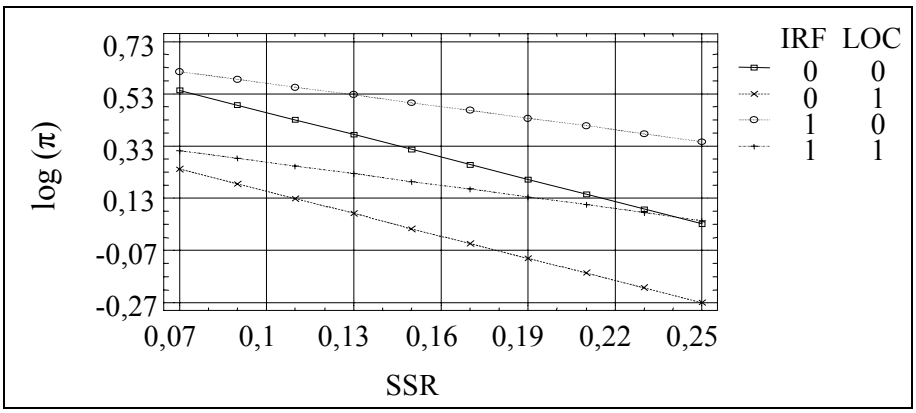


Figura 57. Evolución de log ($\pi = U_{cf}/U_{lef}$) en función de SSR, IRF y LOC. Espacio B.

Factor	Valor asignado
ET	500000
SS	3000
TN	4
PS	1
ES	50

Tabla 32. Valores de los factores para la obtención de la Figura 57.

Los sistemas que se enmarcan en el espacio B muestran una tendencia a empeorar sus *prestaciones* cuanto más a la derecha se encuentran, es decir, cuanto más se parezcan el tamaño de la *cache* y el tamaño del sistema. Esta tendencia se puede observar en la Figura 38, donde la media, mediana y los percentiles muestran una pendiente negativa. Sin embargo, el análisis del modelo muestra que el efecto de esta relación es muy pequeño, al igual que el de IRF, especialmente cuando se compara con otros factores. La importancia de SSR en el modelo no justifica completamente la elevada pendiente que se observa en la Figura 38. Esta pendiente será consecuencia, al igual que la gran variabilidad que presentan los datos de la Figura 39 pertenecientes al espacio B, al gran efecto que tienen otros parámetros y su interacción con SSR. También, al igual que en el espacio A, el comportamiento de los sistemas es muy distinto en función del aprovechamiento que estos hagan de la *cache* –intensivo o no- por lo que el diseñador deberá considerar este factor a la hora de tomar cualquier decisión. El resto de factores e interacciones que aparecen en el modelo presentan un valor de F-Ratio muy bajo, por lo que en principio modificarán el valor absoluto o la posición de las pendientes que se han mostrado en las gráficas anteriores, pero de forma muy ligera, y sin invertir las tendencias.

4.3.3 Modelo de los factores software para el Espacio C

Antes de comenzar el desarrollo del modelo correspondiente al espacio C, la Figura 58 y Figura 59 muestran la representación de π y de $\log(\pi)$ con el objetivo de verificar su normalidad. Como se puede observar, y corroborando los valores de asimetría y curtosis de la Tabla 13, los datos se distribuyen de forma normal. Aunque no se aprecian grandes diferencias entre la utilización o no del logaritmo de π, por coherencia con el resto de análisis se utilizará $\log(\pi)$ como variable dependiente.

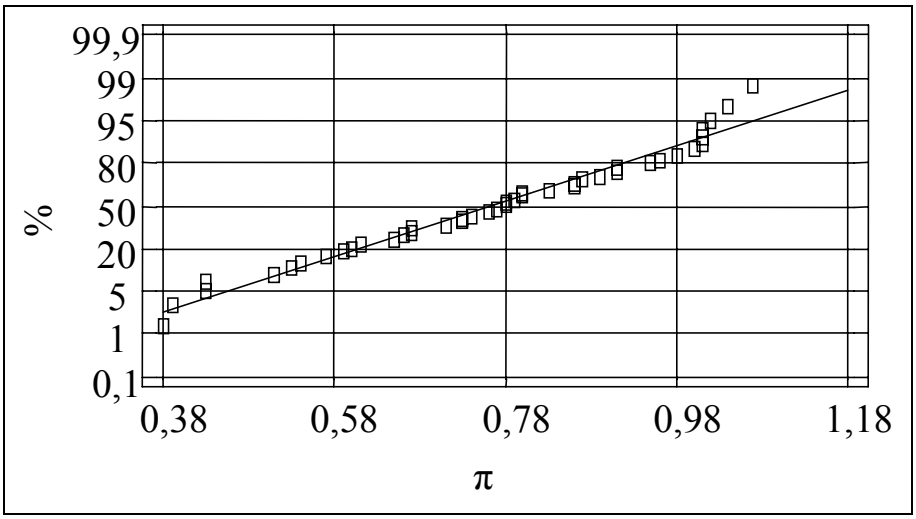


Figura 58. Representación de $\pi = U_{cf}/U_{lef}$ en papel probabilístico normal. Espacio C.

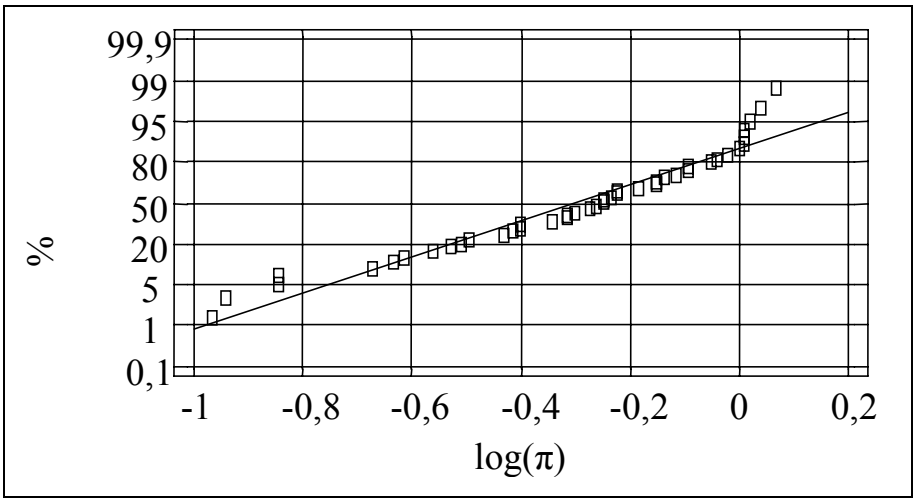


Figura 59. Representación de Log ($\pi = U_{cf}/U_{lef}$) en papel probabilístico normal. Espacio C.

El modelo de las *prestaciones* de los sistemas enmarcados en el espacio C es el siguiente:

Log(π)= -1,3442 + 0,915466*IRF + 0,0116071*ES + 0,937331*SSR - 0,00000179352*ET - 0,111507*IRF*TN -0,000828444*IRF*ES - 0,00196996*TN*ES + 0,0000245098*TN*SS +3,59166E-7*TN*ET - 8,63969E-8*SSR*ET

El modelo incorpora 10 variables independientes, con una R-Squared de 88,03%, lo que quiere decir que el modelo explica en casi el 90% el comportamiento del comportamiento de log(π). El estadístico Durbin-watson, con un valor de 1,47, ligeramente superior a 1,4 indica que no existe ningún indicio de autocorrelación en los residuos. La Tabla 33 muestra los coeficientes, el error normalizado y la significación de cada variable independiente, y la Tabla 34 el análisis de la varianza del modelo. En estas tablas se puede observar que todas las variables independientes tienen un P-value inferior a 0,05, lo que indica que son estadísticamente significativas con un nivel de confianza del 95%. Así mismo, el modelo presenta un P-value de 0,0000 que al ser menor que 0,01 indica que existe una relación estadísticamente significativa entre las variables explicativas y la variable dependiente al 99% de confianza.

Factor	Valor Estimado	Standard Error	T-Statistic	P-Value
Constante	-1,3442	0,0958209	-14,0283	0,0000
IRF	0,915466	0,143666	6,37219	0,0000
ES	0,0116071	0,00434323	2,67245	0,0117
SSR	0,937331	0,11275	8,31336	0,0000
ET	-0,00000179352	5,70307E-7	-3,14483	0,0036
IRF*TN	-0,111507	0,03415	-3,2652	0,0026
IRF*ES	-0,000828444	0,000146555	-5,65278	0,0000
TN*ES	-0,00196996	0,000876036	-2,24872	0,0315
TN*SS	0,0000245098	0,00000600168	4,08383	0,0003
TN*ET	3,59166E-7	1,14008E-7	3,15037	0,0035
SSR*ET	-8,63969E-8	2,7344E-8	-3,15963	0,0034

Tabla 33. Detalles del modelo de log ($\pi = U_{cf}/U_{lef}$). Espacio C.

Fuente	Suma de cuadrados	Grados de libertad	Mean Square	F-Ratio	P-Value
Modelo	2,83264	10	0,283264	23,53	0,0000
Residual	0,38525	32	0,0120391		
Total	3,21789	42			

Tabla 34. Análisis de la varianza del modelo de log ($\pi = U_{cf}/U_{lef}$). Espacio C.

La Tabla 35 muestra la importancia en el modelo de cada uno de los factores que aparecen en el modelo. Para poder interpretar con mayor facilidad el modelo, se presentan ordenados de mayor valor de F-Ratio (más significativo) a menor valor de F-Ratio (menos significativo), junto con el coeficiente estimado para cada factor.

Fuente	F-Ratio	Coeficiente estimado
IRF	62,58	0,915466
SSR	58,92	0,937331
IRF*ES	25,73	-0,00082844
TN*ES	19,36	-0,00196996
ES	17,72	0,0116071
TN*SS	15,43	2,451E-05
ET	10,16	-1,7935E-06
SSR*ET	9,98	-8,64E-08
TN*ET	9,97	3,59E-07
IRF*TN	5,43	-0,111507

Tabla 35. Impacto de los factores (orden descendente) sobre el modelo de log ($\pi = U_{cf}/U_{lef}$). Espacio C.

Lo primero que llama la atención del modelo correspondiente al espacio C es la no aparición, ni de forma individual ni en interacciones, de la variable explicativa LOC. Esto significa que los sistemas enmarcados en este espacio mostrarán el mismo comportamiento, independientemente

del nivel de aprovechamiento que hagan de la *cache*. Esto es debido a que el tamaño de la *cache* se aproxima hasta casi igualar el tamaño del sistema, por lo que la mayoría de las instrucciones del sistema pueden precargarse en *cache*. De este modo, el uso que las tareas hagan de la *cache* es irrelevante, ya que todas las instrucciones significativas para el tiempo de ejecución de todas las tareas se encontrarán bloqueadas en *cache*. Por el contrario, el factor IRF se muestra como el más significativo, además de aparecer en otras dos interacciones, lo que indica que el número de veces que deba recargarse la *cache* convencional influirá en las *prestaciones* obtenidas. Respecto a los espacios A y B, el tamaño de la *cache*, en comparación con el tamaño de las tareas, es mucho mayor, por lo que el coste de una recarga será también mucho mayor.

Al igual que en los análisis anteriores, la validación del modelo se ha realizado mediante la confrontación de los valores observados frente a los predichos, Figura 60, y la verificación de la normalidad de los residuos, Figura 61 y Tabla 36. La distribución de los datos alrededor de una recta y la normalidad de los residuos permiten aceptar la validez del modelo obtenido.

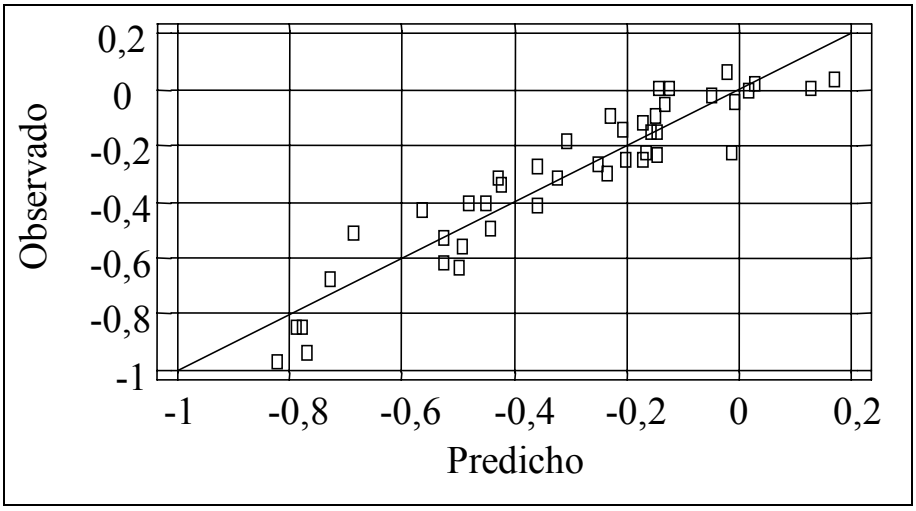


Figura 60. Representación de los valores de log ($\pi = U_{cf}/U_{lef}$) observados frente a los predichos por el modelo para el espacio C.

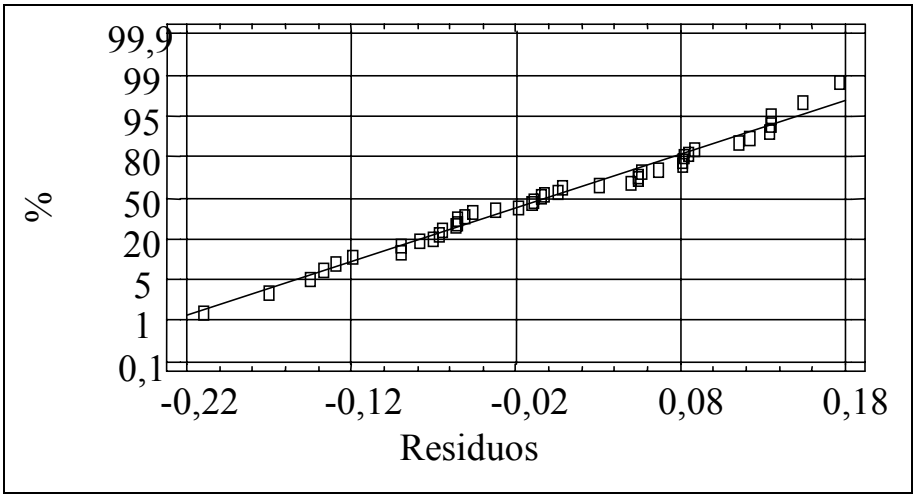


Figura 61. Representación en papel probabilístico normal de los residuos del modelo para el espacio C.

Número de datos	43
Media	-6,97674E-9
Varianza	0,00686544
Desviación estándar	0,082858
Mínimo	-0,166275
Máximo	0,169769
Asimetría estándar	0,0710191
Curtosis estándar	-0,652614

Tabla 36. Principales estadísticos de los residuos del modelo de log ($\pi = U_{cf}/U_{lef}$). Espacio C.

La Figura 62 muestra el efecto del factor SSR en función del nivel de interferencia del sistema IRF y del número de tareas TN. Estos tres factores, además de aparecer de forma individual, intervienen en todas las interacciones presentes en modelo. Tal como muestra la figura, los tres factores afectan el comportamiento del sistema. Como era de prever al observar el modelo, todas las rectas son paralelas, indicando que no existe interacción entre SSR y TN ni entre SSR e IRF. El efecto de SSR es independiente del número de tareas y del grado de expulsiones, mejorando las *prestaciones* de la *locking cache* frente a las *cache*s convencionales según aumenta el valor de SSR. El efecto de TN es similar al de SSR, con mejores *prestaciones* cuanto mayor es el número de tareas, pero en este caso condicionado por el valor de IRF, como puede verse en la figura al no ser igual la distancia entre las diferentes rectas. La Tabla 37 muestra los valores del resto de los factores.

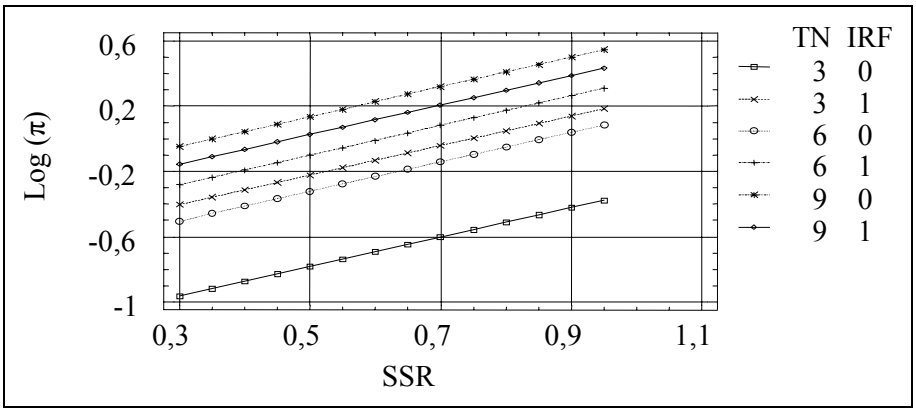


Figura 62. Evolución de log ($\pi = U_{cf}/U_{lef}$) en función de SSR, TN e IRF. Espacio C.

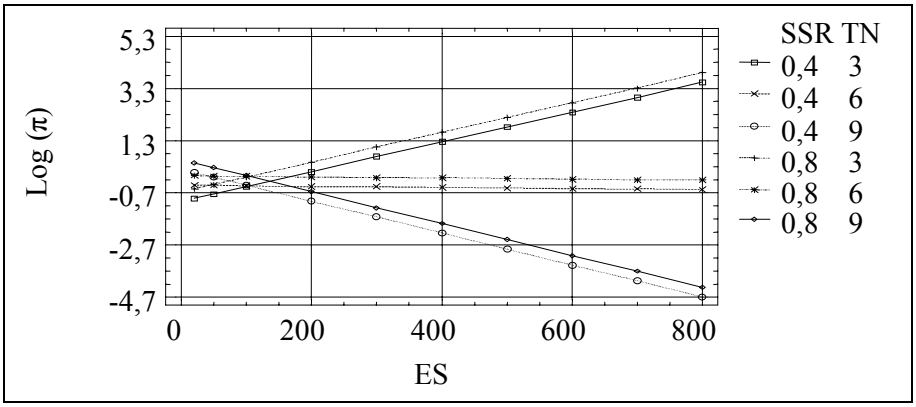
Factor	Valor asignado
PS	857
SS	3000
ET	365281
ES	26,135635

Tabla 37. Valores de los factores para la obtención de la Figura 62.

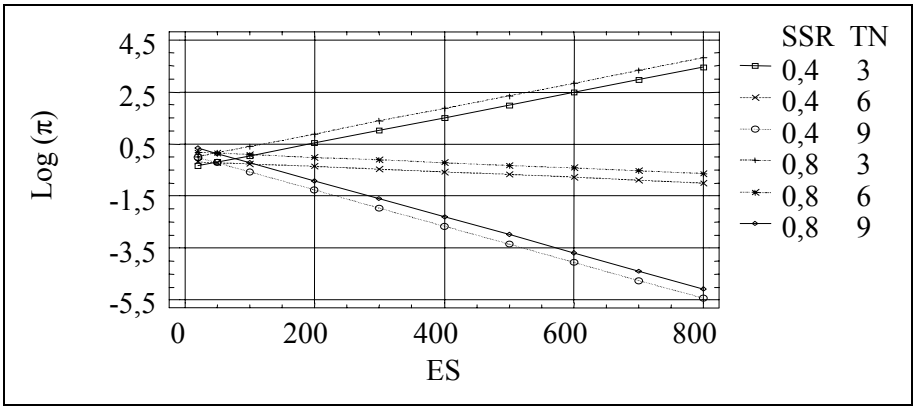
El efecto del factor ES se muestra en la Figura 63-a) para sistemas sin interferencia (IRF 0) y en la Figura 63-b) para sistemas con interferencia (IRF 1). En ambos casos el efecto se muestra

para distintos valores de SSR y para diferentes números de tareas. Los valores asignados a los factores que permanecen constantes se presentan en la Tabla 38.

Comparando las dos gráficas se observa que las tendencias son las mismas, pero que existe una pequeña diferencia en cuanto a los valores absolutos de las *prestaciones*, por lo que el factor IRF afectará a las *prestaciones* de forma ligera, pero no influirá en el efecto que produce ES ni en el efecto que produce el número de tareas del sistema. Sin embargo, el número de tareas del sistema tiene una influencia muy importante, ya que decidirá el signo del efecto de ES. Así, el efecto de ES será positivo, es decir, mejores *prestaciones* cuanto mayor sea su valor, para un número bajo de tareas. Al crecer el número de tareas en el sistema el efecto de ES será menor, siendo casi nulo para un sistema con 6 tareas, y pasará a ser negativo, es decir, pérdida de *prestaciones* según aumenten el número de tareas del sistema por encima de 6.



a) IRF = 0



b) IRF = 1

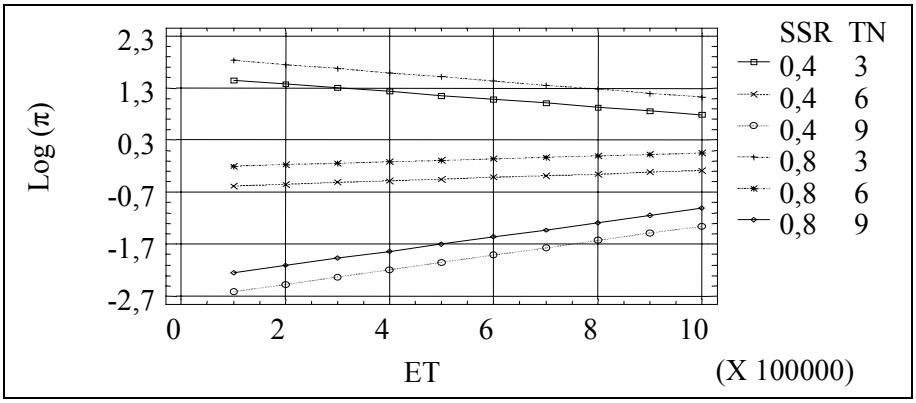
Figura 63. Evolución de log ($\pi = U_{cf}/U_{lef}$) en función de ES, SSR y TN. Espacio C.

Factor	Valor asignado
PS	857
SS	3000
ET	365281

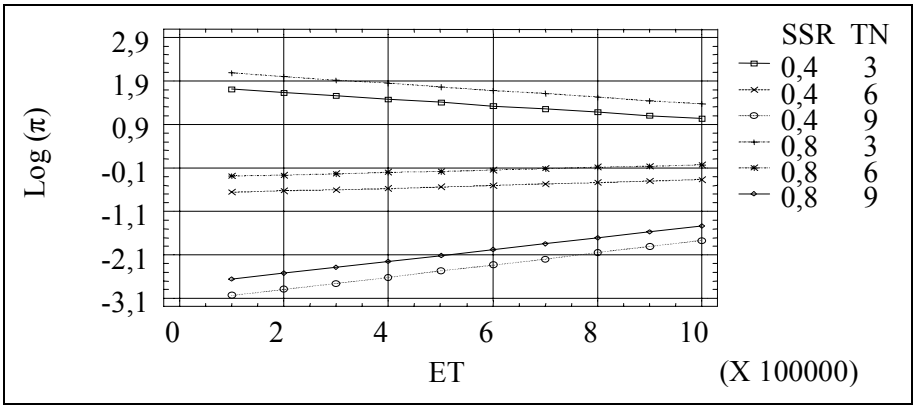
Tabla 38. Valores de los factores para la obtención de la Figura 63.

El efecto del número total de instrucciones ejecutadas se muestra en la Figura 64-a) para sistemas sin interferencia (IRF 0) y en la Figura 64-b) para sistemas con interferencia (IRF 1). Las figuras se muestran muy similares a las anteriores, pero con las tendencias invertidas. El efecto de SSR es mínimo y no modifica el efecto del factor ET, que sí se ve modificado por el

número de tareas del sistema. Para un valor bajo de TN, el efecto es negativo, mientras que para un valor elevado de TN, el efecto del número de instrucciones ejecutadas es positivo, al contrario de lo que sucedía con el factor ES. El efecto del factor IRF es mínimo y no influye en el efecto producido por los otros factores. La Tabla 39 muestra los valores asignados al resto de los factores.



a) IRF = 0



b) IRF = 1

Figura 64. Evolución de log ($\pi = U_{cf}/U_{lef}$) en función de ET, SSR y TN. Espacio C.

Factor	Valor asignado
PS	857
SS	3000
ES	400

Tabla 39. Valores de los factores para la obtención de la Figura 64.

Al igual que en los espacios A y B existe una cierta tendencia en el valor de las *prestaciones* en función de la variable explicativa SSR, siendo en este caso positiva, es decir, mejores *prestaciones* cuanto mayor es el valor de este factor, cuanto más próximos se encuentran los tamaños de la *cache* y del sistema. Esta tendencia también puede observarse en la Figura 38. Sin embargo, y corroborando la variabilidad de las *prestaciones* que se aprecia en la Figura 38, existen otros factores, como el número de instrucciones ejecutadas o el número de tareas, cuyas interacciones con SSR decidirán de forma más significativa las *prestaciones* obtenidas al utilizar una *cache* con bloqueo tal como se ha observado en las gráficas anteriores.

4.3.4 Modelo de los factores software para el Espacio D

Antes de presentar el modelo obtenido para los sistemas incluidos en el espacio D, se estudiará la normalidad de los datos para verificar la aplicabilidad del modelo de regresión. Tal como indican los valores de asimetría y curtosis de la Tabla 13 y la representación en papel probabilístico de la Figura 65 se puede asumir la normalidad de los datos, pero por coherencia con los análisis de los espacios A, B y C se ha preferido utilizar el logaritmo de π, cuya representación en papel probabilístico normal se muestra en la Figura 66.

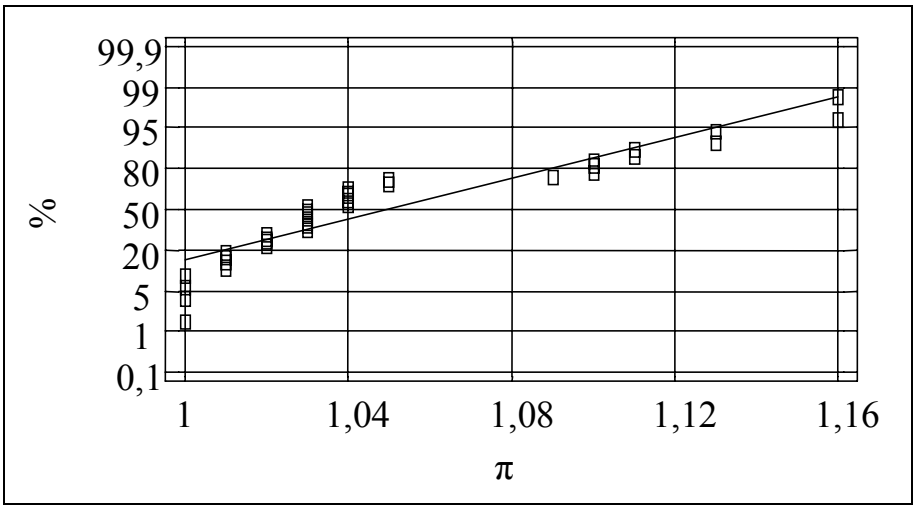


Figura 65. Representación de $\pi = U_{cf}/U_{lef}$ en papel probabilístico normal. Espacio D.

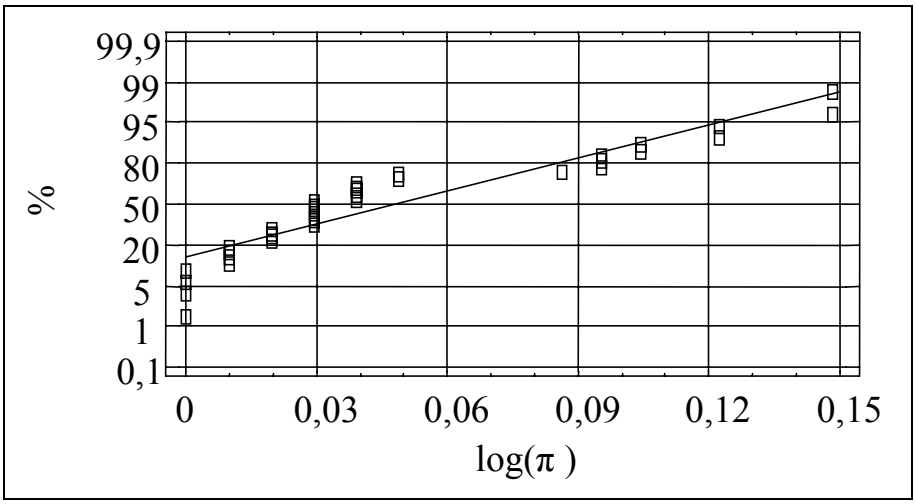


Figura 66. Representación de log ($\pi = U_{cf}/U_{lef}$) en papel probabilístico normal. Espacio D.

El modelo para el logaritmo de π correspondiente a los sistemas enmarcados en el espacio D es el siguiente:

log(π) = 0,661209 + 0,00735613*IRF + 0,0000191324*PS - 0,153268*TN + 0,00679404*ES - 0,000227163*SS - 0,00000114892*ET - 0,102455*LOC*TN - 0,000678915*LOC*PS + 0,00034003*LOC*SS - 0,00112175*TN*ES + 0,0000621048*TN*SS + 2,12598E-7*TN*ET

El modelo incorpora 12 variables independientes, con una R-Squared de 98,85%, lo que quiere decir que el modelo explica la casi totalidad del comportamiento del logaritmo de π. El estadístico Durbin-watson, con un valor de 1,83, ligeramente superior a 1,4 indica que no existe

ningún indicio de autocorrelación en los residuos. La Tabla 40 muestra los coeficientes, el error normalizado y la significación de cada variable independiente, y la Tabla 41 el análisis de la varianza del modelo. En estas tablas se puede observar que todas las variables independientes tienen un P-value inferior a 0,05, lo que indica que son estadísticamente significativas con un nivel de confianza del 95%. Así mismo, el modelo presenta un P-value de 0,0000 que al ser menor que 0,01 indica que existe una relación estadísticamente significativa entre las variables explicativas y la variable dependiente al 99% de confianza.

Factor	Valor Estimado	Standard Error	T-Statistic	P-Value
Constante	0,661209	0,0259337	25,4961	0
IRF	0,00735613	0,00204431	3,59835	0,0015
PS	1,9132E-05	5,1983E-06	3,68051	0,0012
TN	-0,153268	0,00604051	-25,3734	0
ES	0,00679404	0,00055657	12,2071	0
SS	-0,00022716	1,3685E-05	-16,5994	0
ET	-1,1489E-06	8,75E-08	-13,1368	0
LOC*TN	-0,102455	0,00576844	-17,7613	0
LOC*PS	-0,00067892	4,2421E-05	-16,0042	0
LOC*SS	0,00034003	1,9855E-05	17,126	0
TN*ES	-0,00112175	0,00010169	-11,0316	0
TN*SS	6,2105E-05	3,3542E-06	18,5158	0
TN*ET	2,13E-07	1,68E-08	12,6188	0

Tabla 40. Detalles del modelo de log ($\pi = U_{cf}/U_{lef}$). Espacio D.

Fuente	Suma de cuadrados	Grados de libertad	Mean Square	F-Ratio	P-Value
Modelo	0,0658913	12	0,00549094	164,40	0,0000
Residual	0,000768202	23	0,0000334001		
Total	0,0666595	35			

Tabla 41. Análisis de la varianza del modelo de log ($\pi = U_{cf}/U_{lef}$). Espacio D.

La Tabla 42 muestra el resultado de la suma de cuadrados condicional, donde valores elevados de la columna F-ratio indican un nivel de significación importante para un determinado factor. Para permitir una interpretación más sencilla del modelo, los factores se muestran ordenados de forma descendente en función de la F-Ratio, es decir, de mayor a menor importancia, junto con el coeficiente estimado en el modelo para cada factor.

La validación del modelo se ha realizado mediante la comparación de los valores observados y los predichos por el modelo -Figura 67-, y por la comprobación de la normalidad de los residuos -Figura 68 y Tabla 43-. De la observación de dichas figuras y tabla se puede aceptar la validez del modelo obtenido.

Fuente	F-Ratio	Coeficiente estimado
ES	735,33	0,00679404
TN*SS	266,86	6,2105E-05
TN*ES	255,44	-0,00112175
TN	235,27	-0,153268
LOC*PS	178,42	-0,00067892
TN*ET	159,23	2,13E-07
LOC*TN	81,78	-0,102455
IRF	18,36	0,00735613
SS	14,29	-0,00022716
ET	12,57	-1,1489E-06
LOC*SS	7,78	0,00034003
PS	7,47	1,9132E-05

Tabla 42. Impacto de los factores (orden descendente) sobre el modelo de log ($\pi = U_{cf}/U_{lef}$). Espacio D.

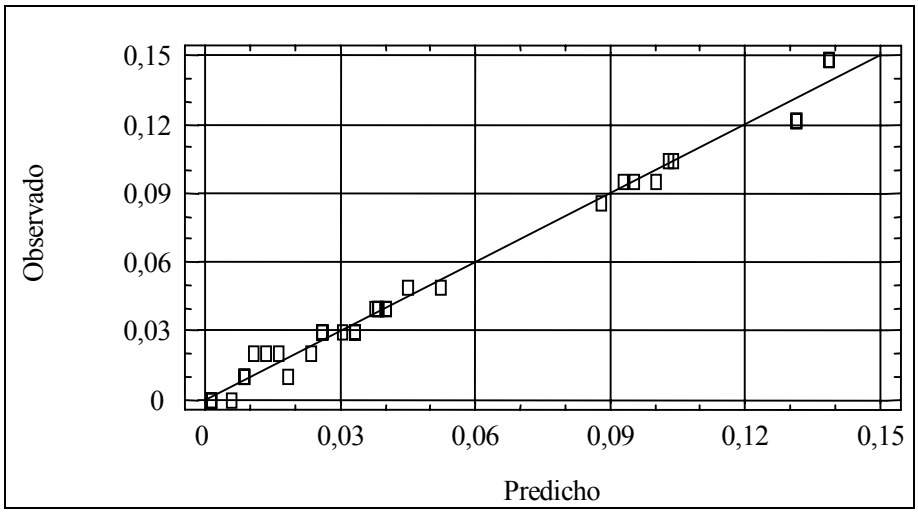


Figura 67. Representación de los valores de log ($\pi = U_{cf}/U_{lef}$) observados frente a los predichos por el modelo para el espacio D.

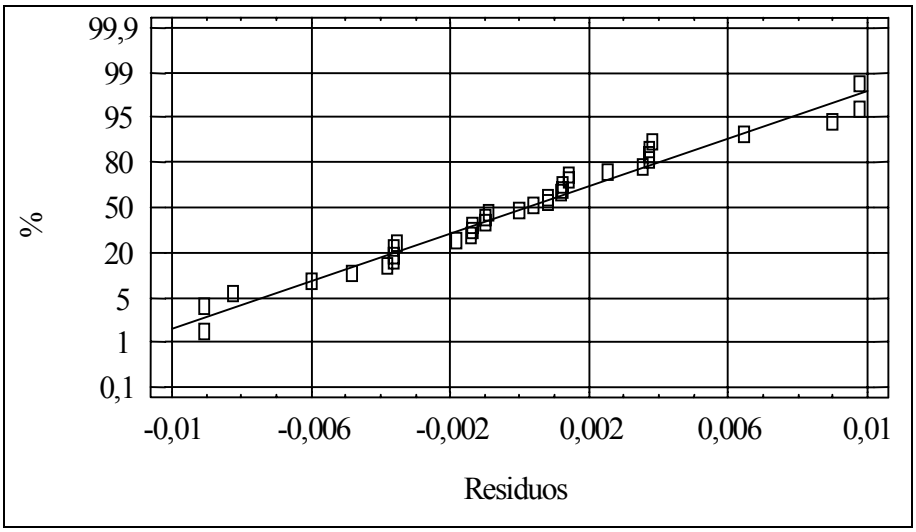


Figura 68. Representación en papel probabilístico normal de los residuos del modelo para el espacio D.

Número de datos	36
Media	4,16667E-11
Varianza	0,0000219486
Desviación estándar	0,00468494
Mínimo	-0,00909406
Máximo	0,00975219
Asimetría estándar	0,343019
Curtosis estándar	0,156833

Tabla 43. Principales estadísticos de los residuos del modelo de log ($\pi = U_{cf}/U_{lef}$). Espacio D.

En el modelo correspondiente al espacio D se puede observar que la variable explicativa SSR no aparece, ni de forma individual ni interactuando con otros factores. Esto es lógico, ya que este espacio corresponde a aquellos sistemas en los que el tamaño de la *cache* es igual o mayor que el tamaño del sistema, por lo que todas las instrucciones de todas las tareas serán precargadas y bloqueadas en *cache*. No tiene ningún efecto sobre el tiempo de ejecución de las tareas ni sobre el tiempo de respuesta, que el tamaño de *cache* sea más o menos mayor que el tamaño del sistema, ya que no pueden cargarse más instrucciones en la *cache*. Simplemente, parte de la *cache* no se utilizará.

Otro detalle importante del modelo es la aparición del factor IRF únicamente de forma individual y con coeficiente positivo. Esto significa que aquellos sistemas que sufran interferencia obtendrán mejores *prestaciones* que aquellos que no. Este efecto se manifestaba también, con mayor o menor intensidad, en los espacios anteriores, pero la causa es diferente. Mientras en los espacios anteriores los sistemas con un nivel alto de interferencia presentaban mejores *prestaciones* debido al menor coste de recarga de la *cache* con bloqueo tras una expulsión, en los sistemas enmarcados en el espacio D no hay recarga de la *cache* tras una expulsión, ya que todas las instrucciones se encuentran en *cache*. Sin embargo, al utilizar la *locking cache* las tareas inician su ejecución con todas las instrucciones precargadas en *cache*, por lo que su WCET será menor que al utilizar una *cache* convencional, que sufrirá los fallos obligatorios necesarios para cargar la *cache*. Este WCET mayor para las *cache* convencionales hará que las expulsiones tengan una mayor repercusión, no por el *cache-refill penalty*, sino por

la ejecución de la tarea o tareas más prioritarias que provocan la expulsión. Así, este efecto será mayor en los sistemas con mayor número de expulsiones. Puesto que el factor IRF aparece sólo de forma individual, no se incluirá en las siguientes gráficas, pero el diseñador deberá tenerlo presente al tomar decisiones sobre su sistema.

La Figura 69 muestra el efecto del tamaño de la tarea más prioritaria PS para los dos posibles usos de *cache*. El efecto de este factor para los sistemas que no hacen un uso intensivo de la *cache* es casi nulo, mientras que para aquellos sistemas que hacen un uso intensivo de la *cache* el efecto es significativo y negativo. La razón de este efecto es similar a la de la aparición del factor IRF. La tarea más prioritaria generará expulsiones a casi todas las tareas del sistema, por lo que su tiempo de cómputo -WCET- repercutirá en los tiempos de respuesta de todas las tareas, condicionando las *prestaciones* obtenidas. En los sistemas que hagan un buen aprovechamiento de la *cache*, el tiempo de los fallos obligatorios se disolverá por el elevado número de aciertos que se producirán, por lo que la eliminación de estos fallos al utilizar *locking cache* no provocará ninguna mejora. Los valores del resto de factores se muestran en la Tabla 44.

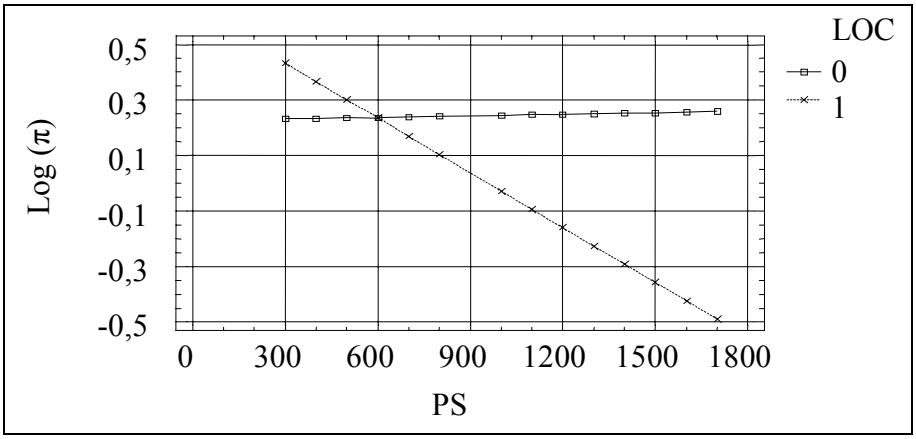


Figura 69. Evolución de log ($\pi = U_{cf}/U_{lef}$) en función de PS y LOC. Espacio D.

Factor	Valor asignado
SS	3000
ET	365281
TN	6
IRF	0
ES	26,13

Tabla 44. Valores de los factores para la obtención de la Figura 69.

La Figura 70 muestra el efecto de ES para sistemas con 3, 6 y 9 tareas, y para los dos posibles usos de *cache*. Existe una clara interacción entre ES y el número de tareas, haciendo que el efecto sea positivo para 3 tareas, nulo para 6 tareas y negativo para 9 tareas. También existe interacción entre el factor LOC y el número de tareas, ya que el efecto de LOC es mayor cuanto mayor es el número de tareas. Por el contrario, LOC influye en las *prestaciones* pero no existe interacción con ES, es decir, no modifica el efecto que ES produce sobre el comportamiento del sistema. La Tabla 45 presenta los valores asignados a los factores que se han mantenido constantes.

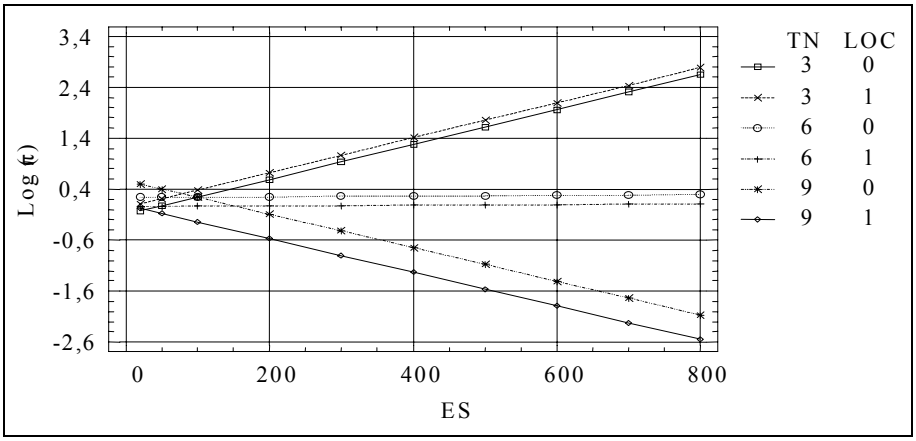


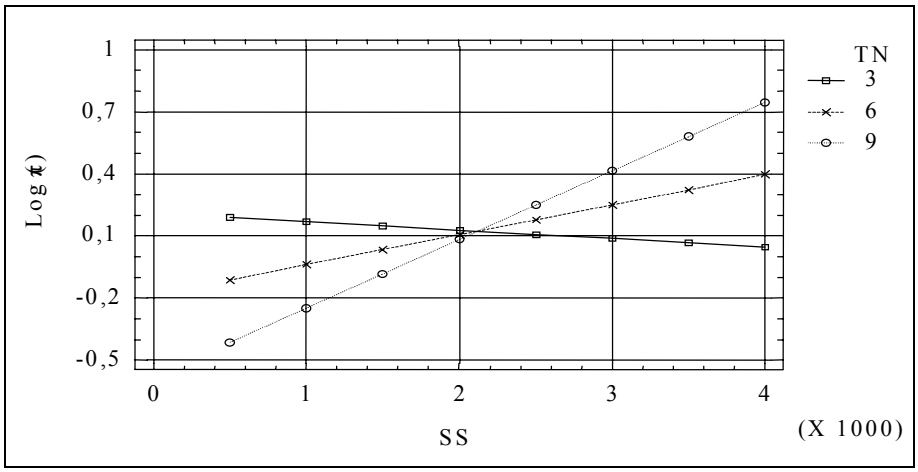
Figura 70. Evolución de log ($\pi = U_{cf}/U_{lef}$) en función de ES, TN y LOC. Espacio D.

Factor	Valor asignado
SS	3000
ET	365281
IRF	0
PS	857

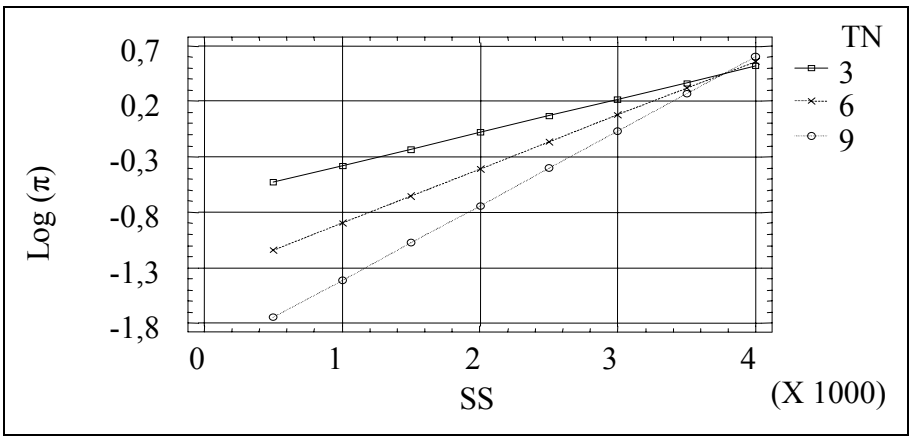
Tabla 45. Valores de los factores para la obtención de la Figura 70.

El efecto del tamaño total de las tareas del sistema se presenta en la Figura 71-a) para sistemas que no hacen un uso intensivo de la *cache*, y en la Figura 71-b) para los sistemas que sí hacen un uso intensivo de la *cache*. Para este último tipo de sistemas, el efecto es siempre positivo, es decir, a mayor tamaño del sistema, mejores *prestaciones*, pero condicionado por el número de tareas, siendo el efecto más importante cuanto más tareas tiene el sistema. Para los sistemas que no hacen un uso intensivo de la *cache*, el comportamiento es similar, pero el efecto del número de tareas en la pendiente es mucho más importante, llegando a anular e incluso invertir ligeramente el efecto del tamaño del sistema. En ambos casos, la intersección de las rectas indica una muy fuerte interacción entre el tamaño del sistema y el número de tareas. Los valores asignados al resto de los factores se muestran en la Tabla 46.

En la Figura 72 se muestra el efecto del número total de instrucciones ejecutadas. Se muestran seis sistemas, con tres números de tareas distintos y los dos usos de *cache* posibles. Al igual que con las variables explicativas anteriores, el número de tareas del sistema condiciona el efecto que produce en las *prestaciones* el número total de instrucciones ejecutadas por el sistema. El efecto de este factor es negativo cuando el sistema está formado por pocas tareas, y positivo cuando está formado por muchas tareas. Sin embargo, las pendientes son muy suaves, lo que indica que el efecto del número total de instrucciones ejecutadas es mínimo, y que el número de tareas se presenta como más significativo en este modelo, ya que la distancia entre las rectas es mucho mayor que la variación entre el inicio (pocas instrucciones ejecutadas) y el final de la recta (muchas instrucciones ejecutadas). Por último, también se observa una interacción entre el número de tareas y el parámetro LOC. La Tabla 47 muestra los valores de los factores que se han mantenido constantes.



a) LOC = 0



b) LOC = 1

Figura 71. Evolución de log ($\pi = U_{cf}/U_{lef}$) en función de SS y TN. Espacio D.

Factor	Valor asignado
PS	857
ET	365281
IRF	0
ES	50

Tabla 46. Valores de los factores para la obtención de la Figura 71.

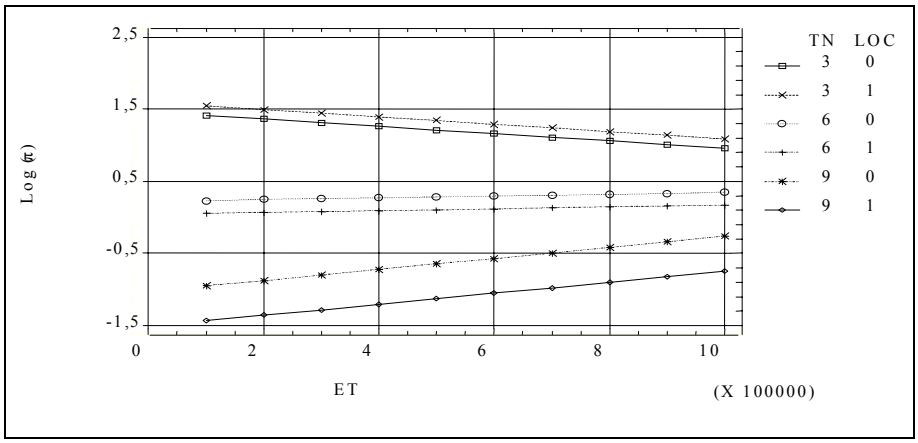


Figura 72. Evolución de log ($\pi = U_{cf}/U_{lef}$) en función de ET, TN y LOC. Espacio D.

Factor	Valor asignado
SS	3000
ES	400
IRF	0
PS	857

Tabla 47. Valores de los factores para la obtención de la Figura 72.

Las características que marcan las *prestaciones* que se obtendrán al utilizar una *cache* con bloqueo, cuando los sistemas se encuentran en el espacio D, son principalmente el tamaño de las tareas y el número de instrucciones ejecutadas, así como el número de tareas en el sistema, que condicionará de forma muy importante el efecto de las demás variables explicativas. Estos factores no influyen de forma individual, sino interactuando entre ellos o en función de la relación que exista entre los diferentes factores. Esto hace que analizar el modelo y encontrar las condiciones óptimas sea una tarea compleja, aunque la mínima variabilidad en los resultados obtenidos relativiza el efecto de los diferentes factores.

4.4 Análisis de los factores hardware

El objetivo del análisis de los factores *hardware* no es obtener un modelo que permita prever las *prestaciones* obtenidas por el uso estático de *locking cache* en función de los valores de dichos factores, sino identificar si estos factores influyen o no en las *prestaciones* obtenidas y cuáles son las tendencias que marcan. Para ello, cada factor se define a dos niveles, realizando un diseño de experimentos en los que se consideran todas las posibles combinaciones de los valores asignados a los factores bajo estudio. Este tipo de diseño se conoce como diseño de experimentos 2^K, donde K es el número de factores bajo estudio. Además, se trata de un diseño con réplicas, ya que existen varios experimentos donde se utiliza la misma combinación de valores de los factores.

El estudio del efecto de los factores *hardware* sobre el comportamiento de las *prestaciones* del uso estático de *locking cache* se ha realizado bajo las siguientes condiciones:

- El proceso bajo estudio o variable dependiente sigue siendo la relación entre la utilización planificable obtenida al emplear el uso estático de *locking cache* frente al empleo de *caches* convencionales, en sistemas de prioridades fijas, definida esta relación como $\pi = U_{cf}/U_{lef}$

- Se ha eliminado el factor "tamaño de *cache*" CS y se ha incorporado el factor SSR, debido a que este último aporta mucha más información sobre el comportamiento de las *prestaciones* π.
- Debido a las diferencias en el comportamiento de π en función de los factores IRF y LOC mostradas en los análisis anteriores, y presumiendo que estos factores interacturarán, en algunos casos, con los factores *hardware*, ambos factores *software* se incluyen en el análisis.
- De los tres factores *hardware* restantes, tamaño de línea de *cache* LS, Tiempo de fallo T_{miss}, y Tiempo de acierto T_{hit}, sólo se estudiarán los dos primeros, ya que, en cuanto a los tiempos de acceso, el factor importante es la relación entre el tiempo de acierto y el tiempo de fallo. Por ello, el valor de T_{hit} se establece en un ciclo de procesador, y se redefine T_{miss} como la relación entre el tiempo de fallo y el tiempo de acierto en *cache*.

Para llevar a cabo el análisis, se han escogido aleatoriamente dos conjuntos de tareas, el primero de ellos –conjunto A- con la característica LOC a nivel 0, es decir, tareas con un nivel bajo de localidad o un aprovechamiento bajo de la *cache*. El segundo conjunto –conjunto B-, con la característica LOC a nivel 1, es decir, tareas con un nivel de localidad alto. Aunque ambos conjuntos de tareas se han escogido, como se ha dicho, aleatoriamente, se ha intentado que su número de tareas, tamaño de las tareas, número de instrucciones ejecutadas y en general, el resto de características a excepción de LOC, fueran lo más parecidas posibles, para poder evaluar adecuadamente y sin interferencia de otros factores el efecto de la característica LOC.

Para cada conjunto de tareas se han definido dos grupos de periodos de modo que, en el primer caso se obtenga el máximo número posible de expulsiones (IRF 1) y en el segundo caso no exista ninguna expulsión durante la ejecución (IRF 0). De este modo se han generado 4 sistemas distintos que, en principio, permiten estudiar los efectos de IRF y LOC. Para poder estudiar el efecto de SSR se ha replicado cada sistema y evaluado su comportamiento con siete tamaños de *cache*s, con valores desde 1Kbyte hasta 64Kbyte, obteniéndose un total de 28 sistemas distintos.

Finalmente, el estudio de los dos factores *hardware* LS y T_{miss} se ha llevado a cabo definiendo cada parámetro a dos valores y evaluando las cuatro posibles combinaciones para los 28 sistemas anteriores. En total, se han realizado 112 nuevos experimentos, para los que se ha obtenido su utilización planificable U_{lef} a partir de las estimaciones del tiempo de respuesta realizadas por el algoritmo genético y su utilización planificable U_{cf} a partir de los tiempos de respuesta obtenidos de la simulación de su ejecución sobre una *cache* convencional.

La Tabla 48 muestra el resumen de los experimentos realizados con los valores asignados a cada factor. Entre paréntesis se muestra el nivel correspondiente a los valores asignados a cada factor o variable explicativa.

Aunque es posible realizar un diseño de experimentos que contemple simultáneamente todos los factores, se ha preferido considerar por separado el efecto de SSR, ya que es posible que la relación entre el tamaño del sistema y el tamaño de la *cache* determine el efecto del resto de los factores. De esta forma se elimina una posible interacción, facilitando por tanto la interpretación de los resultados.

Para estudiar el efecto de SSR se ha realizado el gráfico de dispersión de los valores de $\pi = U_{cf}/U_{lef}$ frente a los valores de SSR, mostrado en la Figura 73. El objetivo de esta gráfica no es mostrar cómo evolucionan los valores de π en función de los valores de SSR, sino comprobar si existe diferencia entre el efecto que los factores LS, T_{miss}, IRF y LOC producen en las *prestaciones* en función de los valores de SSR. Cada "columna" de puntos corresponde al mismo conjunto de tareas con un determinado tamaño de *cache*, y cada punto de cada "columna" corresponde a un experimento con una determinada configuración de los factores bajo estudio. Como era de esperar, los puntos de cada "columna" no se superponen, mostrando claramente que los factores LS, T_{miss}, IRF y LOC afectan a las *prestaciones* obtenidas. Pero también puede observarse que la distancia entre los puntos es diferente para cada columna. Esto indica claramente que existe una interacción entre el factor SSR y el resto de factores, por lo que

el estudio del efecto de los factores restantes se realizará agrupando los experimentos en función de su valor de SSR. Aunque lo ideal sería dividir los experimentos en cuatro espacios, tal como se hizo para el análisis de los factores *software*, el reducido número de experimentos no permite hacer una división tan fina, por lo que para este análisis se considerarán sólo tres espacios, a los que llamaremos ϵ_1, ϵ_2 y ϵ_3. La Tabla 49 muestra los límites inferior y superior de cada espacio.

Conjunto de tareas	**LOC (Nivel)**	**IRF (Nivel)**	**Tamaño de línea LS (Nivel)**	**Relación miss/hit T_{miss} (Nivel)**	**Tamaño de *cache***
A	0 (-)	0 (-)	16 bytes = 4 instr. (-)	5 ciclos (-)	1Kb a 64Kb
A	0 (-)	0 (-)	16 bytes = 4 instr. (-)	10 ciclos (+)	1Kb a 64Kb
A	0 (-)	0 (-)	32 bytes = 8 instr. (+)	5 ciclos (-)	1Kb a 64Kb
A	0 (-)	0 (-)	32 bytes = 8 instr. (+)	10 ciclos (+)	1Kb a 64Kb
A	0 (-)	1 (+)	16 bytes = 4 instr. (-)	5 ciclos (-)	1Kb a 64Kb
A	0 (-)	1 (+)	16 bytes = 4 instr. (-)	10 ciclos (+)	1Kb a 64Kb
A	0 (-)	1 (+)	32 bytes = 8 instr. (+)	5 ciclos (-)	1Kb a 64Kb
A	0 (-)	1 (+)	32 bytes = 8 instr. (+)	10 ciclos (+)	1Kb a 64Kb
B	1 (+)	0 (-)	16 bytes = 4 instr. (-)	5 ciclos (-)	1Kb a 64Kb
B	1 (+)	0 (-)	16 bytes = 4 instr. (-)	10 ciclos (+)	1Kb a 64Kb
B	1 (+)	0 (-)	32 bytes = 8 instr. (+)	5 ciclos (-)	1Kb a 64Kb
B	1 (+)	0 (-)	32 bytes = 8 instr. (+)	10 ciclos (+)	1Kb a 64Kb
B	1 (+)	1 (+)	16 bytes = 4 instr. (-)	5 ciclos (-)	1Kb a 64Kb
B	1 (+)	1 (+)	16 bytes = 4 instr. (-)	10 ciclos (+)	1Kb a 64Kb
B	1 (+)	1 (+)	32 bytes = 8 instr. (+)	5 ciclos (-)	1Kb a 64Kb
B	1 (+)	1 (+)	32 bytes = 8 instr. (+)	10 ciclos (+)	1Kb a 64Kb

Tabla 48. Características de los diferentes experimentos realizados para la evaluación del impacto de los factores hardware sobre $\pi = U_{cf}/U_{lef.}$

Espacio	**Límite inferior**	**Límite superior**
ϵ_1	$0 \leq SSR$	$SSR < 0,1$
ϵ_2	$0,1 \leq SSR$	$SSR < 1$
ϵ_3	$1 \leq SSR$	------

Tabla 49. Valores límites de SSR para la agrupación de $\pi = U_{cf}/U_{lef}$ en los espacios ϵ_1, ϵ_2 y ϵ_3.

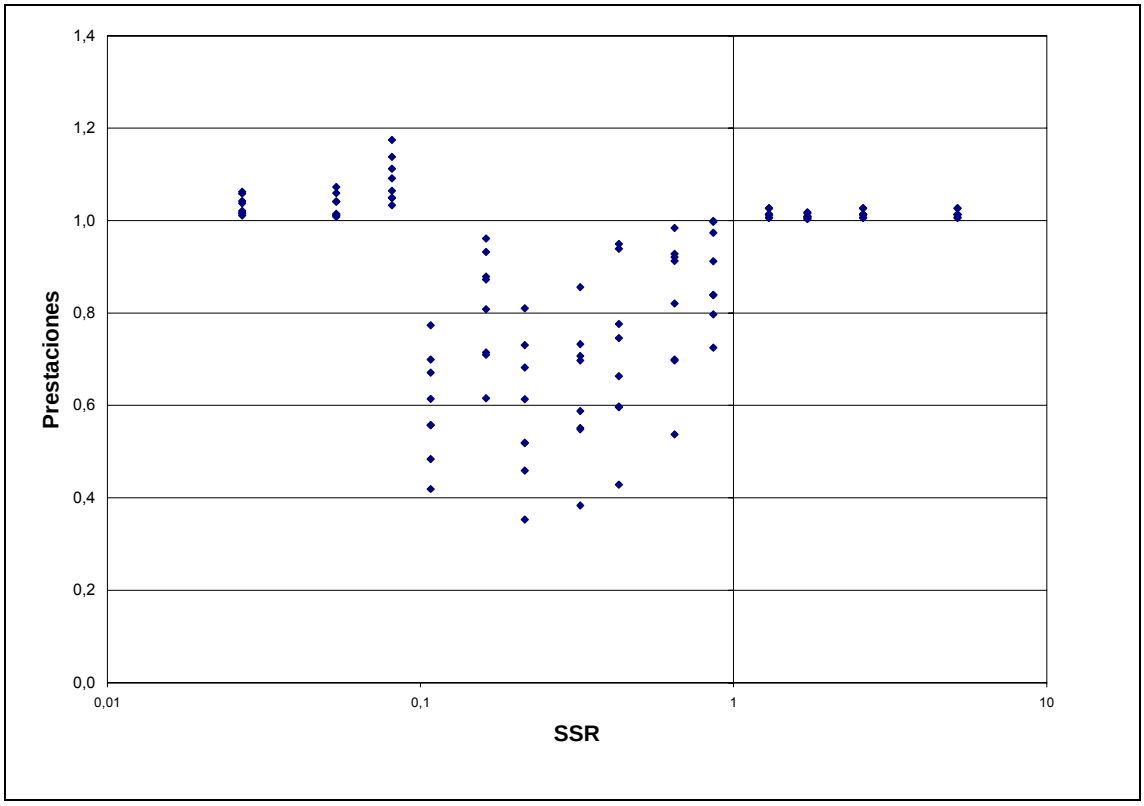


Figura 73. Gráfico de dispersión de $\pi = U_{cf}/ U_{lef}$ frente a SSR.

En los análisis que se presentarán a continuación no se ha verificado, previamente a la realización del análisis, la normalidad de los datos. Esto es debido a dos razones. La primera, a la importante robustez del diseño de experimentos frente a la no normalidad de los datos. Desviaciones importantes respecto de una distribución normal de los datos utilizados para el análisis no influyen de manera significativa en los resultados obtenidos del análisis de la varianza de un diseño de experimentos [Romero93]. La segunda se deriva de la verificación de la normalidad de los datos realizada durante los análisis de los factores *software*. Durante estos análisis se pudo verificar que la asunción de que los datos provenían de una distribución normal era apropiada. Si bien es cierto que en los análisis siguientes se utilizan nuevos experimentos, y los espacios en que se han agrupado los datos son ligeramente diferentes, al tratarse de muestras proveniente de la misma población es posible asumir que, si existe desviación respecto de una distribución normal, esta desviación no será excesiva como para invalidar los resultados obtenidos.

4.4.1 Análisis de los factores hardware para el espacio ϵ_1

La Tabla 50 muestra el resultado del análisis de los experimentos enmarcados en el espacio ϵ_1. En esta tabla se muestra el efecto de cada factor y de las interacciones dobles sobre el valor de π, así como el error normalizado al estimar dicho efecto. Sólo se incluyen interacciones dobles, ya que normalmente interacciones de orden superior no son significativas ni aportan información importante [Romero93]. En esta tabla, el signo positivo del coeficiente estimado para un factor indica que el valor de π será mayor cuando el factor se encuentre a nivel "+", mientras que el signo negativo del coeficiente estimado para un factor indica que el valor de π será mayor cuando dicho factor se encuentre a nivel "–". Los niveles asignados a cada valor de los factores bajo estudio pueden determinarse en la Tabla 48.

Factor	Efecto	Standard error
Media	1,06108	+/- 0,00605171
A:T_{miss}	0,0241197	+/- 0,0121034
B:LS	-0,0113372	+/- 0,0121034
C:IRF	0,0223918	+/- 0,0121034
D:LOC	-0,0562104	+/- 0,0121034
AB	0,00068058	+/- 0,0114112
AC	0,00745675	+/- 0,0114112
AD	-0,0107218	+/- 0,0121034
BC	-0,0244474	+/- 0,0114112
BD	0,0109043	+/- 0,0121034
CD	0,0157898	+/- 0,0121034

Tabla 50. Efecto de los factores hardware sobre $\pi = U_{cf}/U_{lef}$. Espacio ϵ_1.

Antes de interpretar los resultados obtenidos es necesario determinar la importancia de cada factor y si su efecto es estadísticamente significativo. Para ello, el gráfico de Pareto muestra el efecto de cada factor ordenado de mayor a menor importancia –Figura 74- y el análisis de la varianza permite discriminar los factores o interacciones que son significativas. La Tabla 51 muestra el resumen del *anova* (*analyis of variance*) realizado, donde valores menores de 0,05 en la columna P-Value indican que el efecto del factor es significativo a un nivel de confianza del 95%. La columna F-ratio tiene el mismo significado que la mostrada en las tablas de los modelos de regresión lineal múltiple.

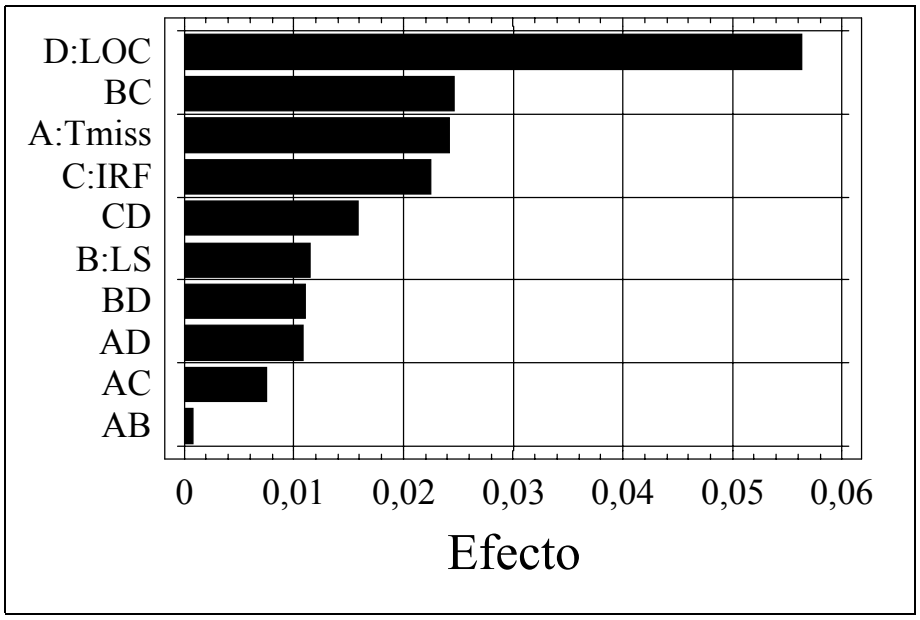


Figura 74. Gráfico de Pareto para $\pi = U_{cf}/ U_{lef}$. Espacio ϵ_1.

Factor	F-Ratio	P-Value
A:T_{miss}	3,97	0,0677
B:LS	0,88	0,3660
C:IRF	3,42	0,0872
D:LOC	21,57	0,0005
AB	0,00	0,9533
AC	0,43	0,5249
AD	0,78	0,3918
BC	4,59	0,0517
BD	0,81	0,3840
CD	1,70	0,2147

Tabla 51. *Anova* del diseño de experimentos para $\pi = U_{cf}/U_{lef}$. Espacio ϵ_1.

En el *anova* se ha obtenido un valor de R-Squared del 75,22%, lo que significa que el efecto de los factores utilizados en el análisis no explica el 25% de la variabilidad presentada por los datos. Esta variabilidad es debida a las diferencias entre las tareas que forman los dos tipos de sistemas estudiados (LOC=0 y LOC=1), como por ejemplo el número de instrucciones ejecutadas en total por el sistema. Aunque ambos tipos de sistemas se han escogido intentando que las diferencias fueran mínimas para poder estudiar con precisión el efecto de los factores *hardware*, es inevitable que existan variaciones, aunque el valor de R-Squared muestra que estas diferencias son mínimas, tal como se perseguía.

En cuanto a los factores considerados en el análisis, sólo la variable explicativa LOC se muestra claramente significativa (P-Value <0,05), aunque los efectos de la interacción BC (LS*IRF), y de las variables independientes T_{miss} e IRF se muestran muy importantes y casi significativos. El efecto de LOC e IRF ya se comentó al realizar el modelo de regresión en el apartado 4.3.1, y coincide con el mostrado por este nuevo análisis: en aquellos sistemas con valor de LOC igual a 1, el uso estático de *locking cache* ofrece peores *prestaciones*, al igual que para aquellos sistemas con valor de IRF igual a 0.

El factor T_{miss} aparece con coeficiente positivo, lo que indica que las *prestaciones* mejorarán cuando esta variable se encuentre a nivel +1, es decir, cuando la relación entre el tiempo de acierto y el tiempo de fallo sea de 10, frente a aquellos sistemas en los que el valor de T_{miss} sea de 5. Esto es debido a que el uso estático de *locking cache* sufre menos fallos en los accesos a *cache* que el esquema convencional, ya que un reducido número de bloques permanece de forma constante en la *locking cache*, mientras que en las *cache*s convencionales, y debido al pequeñísimo tamaño de la *cache* en el espacio ϵ_1, la mayoría de instrucciones son reemplazadas de la *cache* antes de ser referenciadas una segunda vez. Por tanto, al aumentar la diferencia de velocidad entre la memoria principal y la memoria *cache*, el bloqueo de *cache* sufre una menor penalización.

La interacción LS*IRF también se muestra casi significativa. Para interpretar mejor el significado de esta interacción, la Figura 75 muestra la gráfica de interacciones. En ella se puede observar que el efecto del factor LS es muy bajo para los sistemas con IRF a nivel –1, mientras que para los sistemas con factor IRF nivel +1 es mucho mayor. Esto significa que los sistemas en los que se produzca un número elevado de expulsiones serán más sensibles al tamaño de línea de *cache*, y al ser la pendiente negativa, cuanto mayor sea el tamaño de línea de *cache* peores *prestaciones* ofrecerá la *locking cache* frente a las *cache*s convencionales. Este comportamiento está relacionado directamente con el presentado por T_{miss}. Al aumentar el tamaño de la línea de *cache* se reduce el número de bloques de memoria principal que forman el sistema, por lo que el número de conflictos se reduce. En principio, ambos modelos de *cache* –

locking y convencional- se benefician de la reducción del número de conflictos. Pero una de las principales ventajas del uso estático de *locking cache* frente a las *cache*s convencionales es la no existencia de reemplazos durante las expulsiones. Por ello, cuando se trata de sistemas con muchas expulsiones, el uso estático de *locking cache* no obtiene beneficio del aumento del tamaño de línea debido a que no afecta a la recarga de la *cache* tras la expulsión, mientras que las *cache*s convencionales obtienen un beneficio importante, ya que al aumentar el tamaño de línea se reduce el número de bloques que deben transferirse para recargar la *cache* y por tanto se reduce el *cache-refill penalty*. Así pues, no se trata de que el uso estático de *locking cache* se vea perjudicado al aumentar el tamaño de línea, sino que la *cache* convencional obtiene un mayor beneficio.

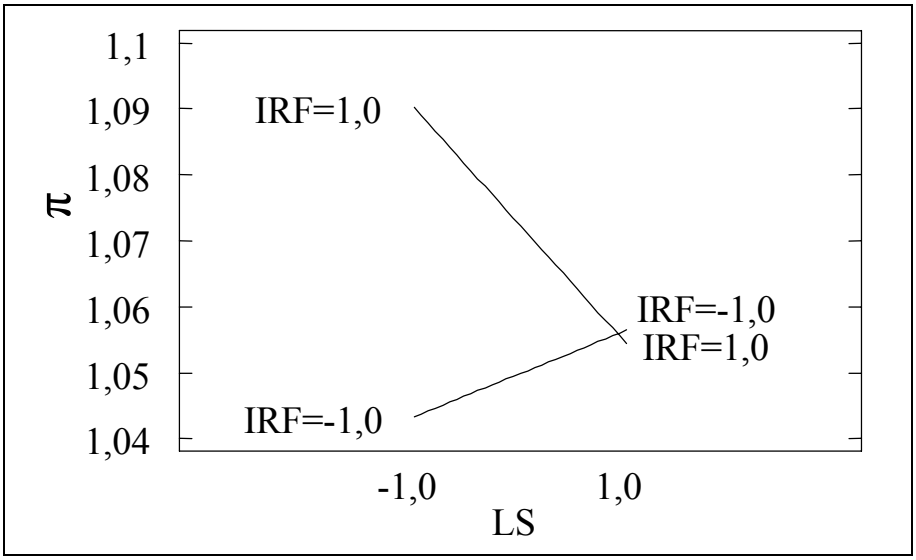


Figura 75. Efecto de los factores IRF y LS sobre $\pi = U_{cf}/ U_{lef}$. Espacio ϵ_1.

4.4.2 Análisis de los factores hardware para el espacio ϵ_2

La Tabla 52 muestra el resultado del análisis de los experimentos enmarcados en el espacio ϵ_2. En esta tabla se muestra el efecto de cada factor y de las interacciones dobles sobre el valor de π, así como el error normalizado al estimar dicho efecto.

Factor	Efecto	Standard error
Media	0,723829	+/- 0,0191845
A:T_{miss}	-0,105542	+/- 0,0383689
B:LS	0,135801	+/- 0,0383689
C:IRF	0,0827192	+/- 0,0383689
D:LOC	-0,0572484	+/- 0,0383689
AB	0,0146892	+/- 0,0379754
AC	0,0183384	+/- 0,0379754
AD	-0,0111376	+/- 0,0383689
BC	-0,0747758	+/- 0,0379754
BD	0,0266851	+/- 0,0383689
CD	0,0586483	+/- 0,0383689

Tabla 52. Efecto de los factores hardware sobre $\pi = U_{cf}/U_{lef}$. Espacio ϵ_2.

Antes de interpretar los resultados obtenidos es necesario determinar la importancia de cada factor y si su efecto es estadísticamente significativo. Para ello, el gráfico de Pareto muestra el efecto de cada factor ordenado de mayor a menor importancia –Figura 76- y el análisis de la varianza permite discriminar los factores o interacciones que son significativas. La Tabla 53 muestra el resumen del *anova* realizado, donde valores menores de 0,05 en la columna P-Value indican que el efecto del factor es significativo a un nivel de confianza del 95%.

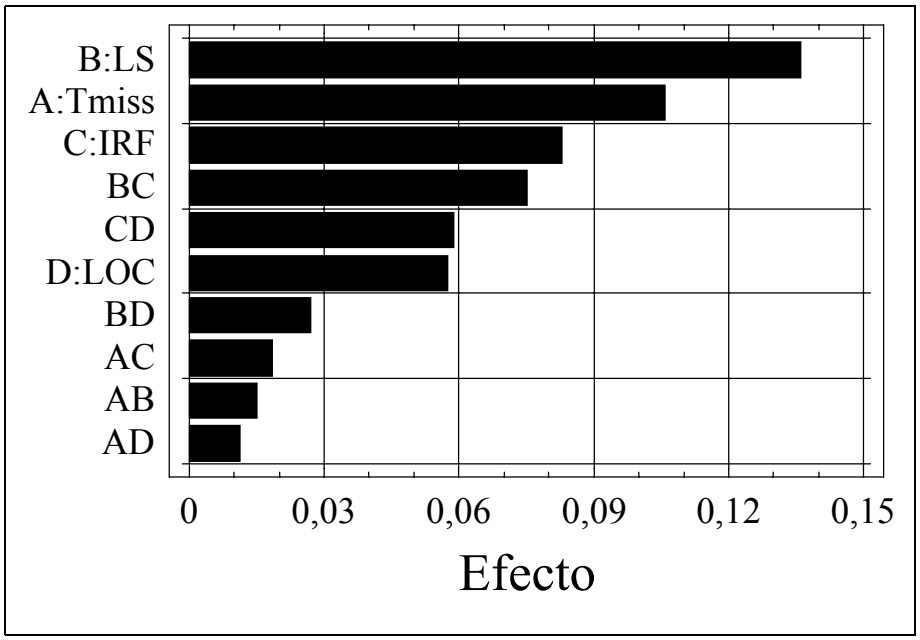


Figura 76. Gráfico de Pareto para $\pi = U_{cf}/ U_{lef}$. Espacio ϵ_2.

Factor	F-Ratio	P-Value
$A:T_{miss}$	7,57	0,0085
B:LS	12,53	0,0009
C:IRF	4,65	0,0365
D:LOC	2,23	0,1427
AB	0,15	0,7007
AC	0,23	0,6315
AD	0,08	0,7729
BC	3,88	0,0551
BD	0,48	0,4903
CD	2,34	0,1334

Tabla 53. *Anova* del diseño de experimentos para $\pi = U_{cf}/U_{lef}$. Espacio ϵ_2.

En el *anova* se ha obtenido un valor de R-Squared del 44,86%, lo que significa que la mayor parte de la variabilidad observada en los valores de π se debe a factores y características que no han sido consideradas en este análisis, como el número de instrucciones ejecutadas en total por el sistema o el tamaño concreto de cada tarea, características que varían entre los dos conjuntos de tareas seleccionados para este análisis. Aunque las diferencias en estas características se han intentado minimizar, su efecto sobre el comportamiento del sistema es muy importante.

La importancia del estadístico R-Squared es relativa, ya que el objetivo de este análisis no es encontrar un modelo que explique el comportamiento de la variable dependiente, sino determinar cuáles son los factores que influyen de forma significativa en dicho comportamiento. En concreto, de los factores e interacciones consideradas en este análisis, tres son estadísticamente significativos (P-Value <0,05), y uno de ellos está muy próximo a serlo. Estos factores son LS, T_{miss}, IRF y la interacción LS*IRF.

La variable explicativa IRF, con signo positivo en su coeficiente estimado, indica que aquellos sistemas que tengan esta característica a nivel 1, es decir, que sufran un elevado número de expulsiones, obtendrán mejores *prestaciones* al utilizar la *locking cache* que aquellos sistemas que no sufran expulsiones. Este comportamiento se podía observar en los modelos de regresión correspondiente al espacio B y especialmente al espacio C.

El efecto de la variable LS es el más importante de todos, y se muestra con coeficiente positivo. Esto indica que cuanto mayor es el tamaño de línea de *cache*, mejores *prestaciones* se obtienen con el uso estático de la *locking cache* frente a las *cache*s convencionales. Este comportamiento es debido a que en este espacio, el tamaño de la *cache* es relativamente grande comparado con el tamaño del sistema, por lo que las *cache*s convencionales hacen un buen uso del espacio disponible resultando en un número menor de fallos al acceder a *cache* que el uso estático de *locking cache*. Al aumentar el tamaño de línea se reduce el número de fallos en *cache*, ya que se reduce el número de bloques de memoria principal que compiten por el espacio de *cache*. Aunque esta reducción en el número de fallos favorece a ambos esquemas de *cache*, la *cache* convencional no obtiene un gran beneficio de esta reducción, ya que su comportamiento adaptativo le permite, con tamaños de línea pequeños, optimizar el tráfico de bloques entre memoria principal y memoria *cache*.

La interacción BC (LS*IRF) es casi significativa, lo que indica que el efecto de LS descrito en el párrafo anterior se verá condicionado por el número de expulsiones que sufren las tareas. La Figura 77 muestra que en ambos casos el efecto es del mismo signo, pero mucho más importante para aquellos sistemas que no sufren expulsiones. Esto es debido a que el uso estático de *locking cache* compensa su mal comportamiento en cuanto a interferencia intrínseca con su excelente comportamiento en cuanto a interferencia extrínseca, ya que tras una expulsión no es necesario recargar ningún bloque de memoria principal (excepto el buffer temporal). Al aumentar el tamaño de línea, la *cache* convencional debe mover un número menor de bloques desde memoria principal a *cache*, por lo que el valor del *cache-refill penalty* se reduce de forma importante, por lo que la ventaja (sobre el WCET) que obtiene el uso estático de *locking cache* al aumentar el tamaño de línea se ve compensada por la ventaja (sobre el *cache-refill penalty*) que obtienen los sistemas con IRF = 1cuando se ejecutan en la *cache* convencional.

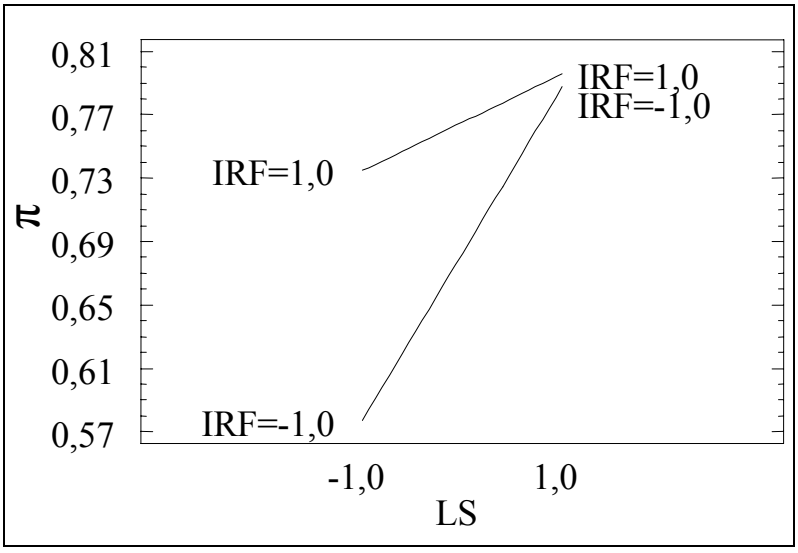


Figura 77. Efecto de los factores IRF y LS para $\pi = U_{cf} / U_{lef}$. Espacio ϵ_2.

Finalmente, el efecto de T_{miss} aparece con coeficiente negativo, lo que quiere decir que al aumentar el tiempo de fallo en los accesos a *cache*, las *prestaciones* del uso estático de *locking cache* se reducen frente al uso de *cache*s convencionales. Como se ha comentado anteriormente, el número de fallos que sufre el uso de *locking cache* es mayor que el que se produce al utilizar *cache* convencional, por lo que aumentar este tiempo de fallo producirá un impacto negativo mayor sobre el primer esquema de *cache*.

4.4.3 Análisis de los factores hardware para el espacio ϵ_3

La Tabla 54 muestra el resultado del análisis de los experimentos enmarcados en el espacio ϵ_3. En esta tabla se muestra el efecto de cada factor y de las interacciones dobles sobre el valor de π, así como el error normalizado al estimar dicho efecto.

Factor	Efecto	Standard error
Media	1,01217	+/- 0,0000930835
A:T_{miss}	0,00850037	+/- 0,000186167
B:LS	-0,00825638	+/- 0,000186167
C:IRF	0,00077438	+/- 0,000186167
D:LOC	-0,00541438	+/- 0,000186167
AB	-0,00292556	+/- 0,000161225
AC	0,00012131	+/- 0,000161225
AD	-0,00179462	+/- 0,000186167
BC	9,38E-07	+/- 0,000161225
BD	0,00206512	+/- 0,000186167
CD	0,00045288	+/- 0,000186167

Tabla 54. Efecto de los factores hardware sobre $\pi = U_{cf}/U_{lef}$. Espacio ϵ_3.

Antes de interpretar los resultados obtenidos es necesario determinar la importancia de cada factor y si su efecto es estadísticamente significativo. Para ello, el gráfico de Pareto muestra el efecto de cada factor ordenado de mayor a menor importancia –Figura 78- y el análisis de la varianza permite discriminar los factores o interacciones que son significativas. La Tabla 55 muestra el resumen del *anova* realizado, donde valores menores de 0,05 en la columna P-Value indican que el efecto del factor es significativo a un nivel de confianza del 95%.

En el *anova* se ha obtenido un valor de R-Squared del 99,74%, lo que significa que los cuatro factores incluidos en el análisis, junto con sus interacciones, explican la casi totalidad de la variación de los datos.

Este resultado puede parecer contradictorio frente al obtenido en el análisis de los factores *software* para los experimentos enmarcados en el espacio D, donde también se obtenía un valor de R-Squared cercano al 100%, por lo que se decía que el modelo obtenido explicaba la casi totalidad de la variabilidad de los datos. Esta contradicción es sólo aparente, ya que se está hablando de fuentes de variabilidad distintas. En el presente análisis se ha modificado el valor de los factores *hardware*, mientras que la gran mayoría de los parámetros *software* se han mantenido constantes. Sólo alguna característica *software* sufre ligeras variaciones, que para los experimentos enmarcados en este espacio no ejercen influencia alguna. Por contra, en el análisis de los factores *software*, eran las características *hardware* las que se mantenían constantes, por lo que es imposible que se manifestarán sus efectos.

De los factores considerados en el presente análisis, 8 efectos, incluyendo los cuatro simples y cuatro interacciones, aparecen con un P-Value menor de 0,05, indicando que son

estadísticamente significativas. Sin embargo, observando con detalle el gráfico de Pareto o los valores de F-Ratio del resumen del *anova*, dos factores simples se muestran con diferencia como los más importantes respecto al resto de factores e interacciones. Estos factores son T_{miss} y LS.

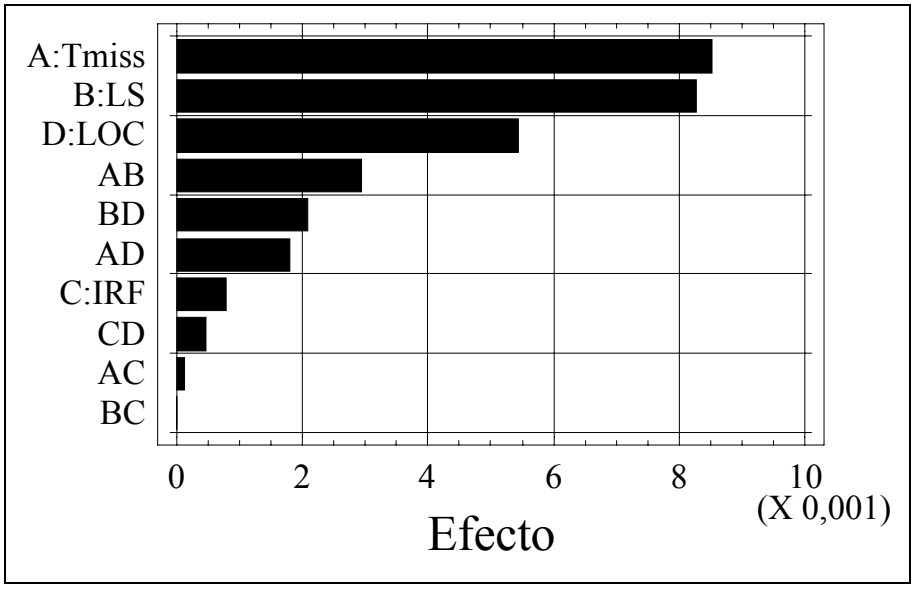


Figura 78. Gráfico de Pareto para $\pi = U_{cf}/ U_{lef}$. Espacio ϵ_3.

Factor	F-Ratio	P-Value
A:T_{miss}	2084,83	0
B:LS	1966,86	0
C:IRF	17,3	0,0004
D:LOC	845,85	0
AB	329,27	0
AC	0,57	0,4601
AD	92,93	0
BC	0	0,9954
BD	123,05	0
CD	5,92	0,024

Tabla 55. *Anova* del diseño de experimentos para $\pi = U_{cf}/U_{lef}$. Espacio ϵ_3.

El coeficiente estimado para el factor T_{miss} es positivo, por lo que al aumentar el tiempo de fallo las *prestaciones* del uso estático de *locking cache* mejorarán respecto al uso de las *caches* convencionales. Esto es debido a que en el uso estático no se produce ningún fallo de *cache*, ya que todo el sistema cabe completamente en *cache*, y ésta es precargada durante el tiempo de arranque y antes de la ejecución del sistema. Por tanto, este factor no afecta a las *prestaciones* ofrecidas por la *locking cache*. Sin embargo, al utilizar una *cache* convencional, ésta se encuentra inicialmente vacía, por lo que cada bloque de memoria principal producirá un fallo hasta que todos ellos hayan sido transferidos a la memoria *cache*. Estos fallos, llamados obligatorios, no suceden en el uso estático de *locking cache*, por lo que el incremento en el valor de T_{miss} influirá sobre las *prestaciones* ofrecidas por una *cache* convencional, pero no sobre las *prestaciones* obtenidas en el uso estático de *locking cache*.

En cuanto al efecto del factor LS, éste aparece con signo negativo, es decir, al aumentar el tamaño de línea las *prestaciones* del uso estático de *locking cache* empeoran respecto al uso de *cache*s convencionales. La causa de este comportamiento es la misma comentada para el factor T_{miss}. Las *cache*s convencionales sufren *mandatory misses*, mientras que el uso estático de *locking cache* no sufre ningún fallo en *cache*. De este modo, la *locking cache* es insensible a la variación en las características de la *cache*, mientras que la *cache* convencional se beneficia del aumento del tamaño de la línea de *cache*, ya que se producen menos fallos obligatorios.

4.5 Conclusiones del capítulo

En este capítulo se ha presentado el análisis del comportamiento, en cuanto a *prestaciones*, del uso estático de *locking cache* frente a la utilización de memorias *cache* convencionales, utilizando en ambos casos un planificador de prioridades fijas.

Este análisis se ha realizado para la relación entre la utilización ofrecida por la *locking cache* frente a la utilización de *cache*s convencionales. Esta relación se ha definido como *prestaciones* = $\pi = U_{cf}/U_{lef}$ donde U_{cf} es la utilización del sistema al emplear *cache* convencional y U_{lef} es la utilización del sistema al emplear *locking cache*. Ambas utilizaciones corresponden con la máxima utilización planificable, lo que ha permitido realizar los análisis considerando el peor escenario posible.

Al realizar los análisis para la relación o ratio entre las utilizaciones se limita el ámbito de las conclusiones obtenidas. Así, a partir de estos análisis puede decirse que para unas determinadas condiciones *software* y *hardware* del sistema, la *locking cache* ofrecerá mejores -o peores- *prestaciones* que las *cache*s convencionales, pero no pueden compararse entre sí las *prestaciones* ofrecidas por la *locking cache* para diferentes situaciones. Esta limitación no es tal, ya que el objetivo era identificar el coste, en cuanto a *prestaciones*, que supone la obtención de determinismo al emplear *locking cache*, y en ningún momento realizar un modelo o una predicción de la utilización que presentará un determinado conjunto de tareas al ejecutarse sobre una memoria *cache* con bloqueo.

El análisis del comportamiento de la variable bajo estudio se ha divido en tres análisis distintos, empleando tres herramientas estadísticas diferentes: un simple gráfico de dispersión, los modelos de regresión lineal múltiple, y el diseño de experimentos. Cada herramienta se ha utilizado en función de los objetivos perseguidos y de los datos disponibles.

El primer análisis presentado, una representación gráfica de π en función del ratio entre el tamaño del sistema y el tamaño de la *cache* (SSR) ofrece una gran cantidad de información, delimitando claramente situaciones en las que la ganancia o pérdida de *prestaciones* es generalizada. Además, este primer análisis ha permitido dividir el problema en cuatro zonas o espacios para el análisis de los factores *software*, y en tres zonas o espacios para el análisis de los factores *hardware*, facilitando la realización de dichos análisis, y lo que es más importante, permitiendo revelar de forma más intensa los efectos ejercidos por cada factor o variable independiente. La primera y principal ventaja de esta división en espacios ha sido la posibilidad de obtener modelos lineales para un conjunto de datos cuyo comportamiento no es, en conjunto, lineal. La segunda y no menos importante ventaja es el descubrimiento de un comportamiento básico de la *locking cache*, ya que esta gráfica ha permitido identificar la importancia de la interferencia intrínseca y extrínseca en los diferentes sistemas.

A través de los modelos de regresión lineal múltiple, segundo análisis presentado, se ha podido constatar que existen tres factores o características del sistema cuya influencia en las *prestaciones* ofrecidas por la *locking cache* es importante. Estos factores son el grado de interferencia o expulsiones que se producen en el sistema (IRF), el nivel de localidad o aprovechamiento que de la *cache* hace el sistema (LOC), y el número de tareas que componen el sistema (TN). Aunque las demás características *software* aparecen, con mayor o menor importancia en casi todos los modelos, los tres factores mencionados definen no sólo subconjuntos de sistemas con comportamientos particulares, sino que modifican el efecto que el resto de características ejercen sobre las *prestaciones* de la *locking cache*. Así, el número de

tareas es capaz de invertir el efecto producido por otra característica, mientras que el comportamiento de un sistema puede variar radicalmente según el valor de LOC o IRF. La realización de modelos de regresión múltiple independientes para diferentes subcojuntos en función de los valores de estas variables explicativas, tal como se ha hecho con el factor SSR, permitiría obtener resultados más ajustados, sencillos y fáciles de interpretar, aunque exigiría aumentar el número de experimentos para poder trabajar con los niveles de confianza necesarios.

De forma generalizada es posible afirmar que los sistemas con un alto grado de localidad presentarán peores prestaciones que los sistemas con un bajo grado de localidad al utilizarse una cache con bloqueo. Del mismo modo, los sistemas en que se producen pocas expulsiones ofrecerán peores prestaciones al ejecutarse en una cache con bloqueo que aquellos sistemas en los que se produzca un elevado número de expulsiones.

En cuanto al factor SSR, aunque se encuentra implícito en cada análisis al ser la característica utilizada para dividir los datos en espacios, los modelos han mostrado que para los espacios A, B y C define también el comportamiento de las *prestaciones*. Este resultado corrobora lo observado en la gráfica de π frente a SSR, donde la pendiente de los datos en dichos espacios era siempre distinta de cero.

En el tercer análisis presentado, correspondiente al efecto de los factores *hardware* y realizado mediante el diseño de experimentos, se ha constatado que el efecto del tamaño de la línea de *cache* y de la relación entre el tiempo de acierto y el tiempo de fallo en el acceso a *cache* es altamente dependiente del valor de SSR que tenga el sistema. Y esta dependencia no se limita a la intensidad del efecto, sino también al signo. Por ejemplo, en algunos casos, tener un tiempo de fallo mayor hará que el uso estático de *locking cache* presente una menor pérdida de *prestaciones* frente a las *cache*s convencionales, mientras que en otros casos el efecto será el contrario, es decir, la pérdida de prestaciones será mayor cuanto mayor sea el tiempo de fallo. Esto es debido a que los factores *hardware* actúan principalmente en función del número de fallos que se produzca al acceder a *cache*, y esta tasa depende de la relación entre el tamaño de las tareas y el tamaño de la *cache*. Aunque de forma general se puede afirmar que el tamaño de línea de *cache* y el tiempo de fallo influirán en la ganancia o pérdida de *prestaciones* presentada por el uso estático de *locking cache*, dicha ganancia o pérdida dependerá del valor de la característica SSR que presente un determinado sistema.

Los análisis realizados y presentados en este capítulo permiten estimar, en función de las características *software* y *hardware* de un sistema, cuál será la ganancia o pérdida de prestaciones que se obtendrá al emplear de forma estática una cache con bloqueo frente a la utilización de una cache convencional. De este modo, el diseñador de sistemas de tiempo real puede conocer, de forma aproximada, el coste que deberá pagar por disponer de un sistema determinista que le permita realizar el análisis de planificabilidad de una forma sencilla.

Capítulo 5

Análisis de prestaciones para uso estático de locking cache con prioridades dinámicas EDF

El principal objetivo del trabajo presentado en este capítulo es el estudio del comportamiento de las prestaciones del uso estático de *locking cache* cuando la política de planificación empleada corresponde con prioridades dinámicas EDF. Este estudio persigue la identificación de los factores del sistema que participan y determinan el comportamiento de las prestaciones. Estos factores incluyen tanto características *software* como *hardware*.

Al igual que en el capítulo anterior, la identificación de estos efectos se realizará mediante el uso de dos herramientas estadísticas: los modelos de regresión lineal múltiple y el diseño de experimentos 2^K. El análisis de los resultados se realizará mediante la comparación con los datos y conclusiones obtenidas en el capítulo anterior, permitiendo de este modo discriminar si la utilización de un planificador diferente (prioridades fijas frente a prioridades dinámicas EDF) introduce alguna variación en el comportamiento, desde el punto de vista de las prestaciones, en el uso estático de *locking cache*.

Tanto los factores o características estudiadas como la medida de prestaciones empleada en este capítulo coinciden con las definidas en el capítulo anterior, por lo que antes de presentar los análisis realizados se realiza un breve recordatorio del proceso bajo estudio.

5.1 Descripción del proceso

El proceso que se desea modelar es el comportamiento de una *cache* con bloqueo sobre la que se ejecuta un conjunto de tareas, frente al comportamiento de una *cache* convencional ejecutando el mismo conjunto de tareas, empleando en ambos casos una política de planificación de prioridades dinámicas EDF. Llamaremos sistema al conjunto de tareas ejecutándose sobre una *cache* con tamaño concreto. Así, el mismo conjunto de tareas ejecutándose en una *cache* de 4Kb y en una *cache* de 8Kb serán dos sistemas distintos. Este proceso estará formado por un variable dependiente o salida del proceso y por un conjunto de variables independientes o explicativas que determinarán el comportamiento del proceso.

5.1.1 Variable dependiente

La variable dependiente o salida del proceso es la comparación de las prestaciones obtenidas con dos arquitecturas de *cache*: uso estático de *locking cache* y memorias *cache* convencionales. Al igual que en el capítulo anterior, la medida de prestaciones empleada será la utilización. Sin embargo, en este caso no se empleará la utilización planificable, sino de la utilización real, obtenida dicha utilización de la simulación del hiperperiodo para la *cache* convencional, y de la ejecución del algoritmo genético para la *locking cache*. Estas utilizaciones no representan la máxima utilización en el peor de los casos, pero el método utilizado para la realización del análisis de planificabilidad, y que fue descrito en el capítulo 3, no permite la obtención de la utilización en el peor de los casos.

Por tanto, la variable dependiente o salida del proceso que se desea modelar, y que llamaremos *prestaciones* (π) será la relación entre la utilización obtenida con una *cache* convencional y prioridades dinámicas EDF Utilización$_{\text{convencional dinámicas}}$ (U_{cd}) y la utilización obtenida con una *cache* con bloqueo haciendo un uso estático de la misma y con prioridades dinámicas EDF Utilización$_{\textit{locking}\ \text{estático dinámicas}}$ (U_{led}), La relación entre ambas utilizaciones se define como el cociente, tal como muestra la siguiente ecuación:

$$\textbf{\textit{Prestaciones}}\ \pi = U_{cd}/U_{led}$$

Si la variable *prestaciones* π toma el valor de 1 indicará que la utilización obtenida con ambos esquemas de *cache* es la misma, y que por tanto no existe pérdida o ganancia de *prestaciones* al utilizar una u otra alternativa. Si la variable *prestaciones* toma valores superiores a 1, es decir, $U_{cd} > U_{led}$, indicará que existe ganancia de *prestaciones* al utilizar la *cache* con bloqueo, ya que se considera que una menor utilización representa mejores *prestaciones*. Por contra, si la variable *prestaciones* toma valores inferiores a 1, es decir, $U_{cd} < U_{led}$, indicará que el empleo de una *cache* con bloqueo implica una pérdida de *prestaciones* respecto a la utilización de las arquitecturas de *cache* convencionales. En el texto que sigue, todas las referencias a las *prestaciones* se realizan desde el punto de vista de la *locking cache*, mientras no se indique explícitamente lo contrario. Cuando se diga que un sistema mejora o empeora sus *prestaciones*, implícitamente se está diciendo que el uso estático de *locking cache* presenta una mejora o empeoramiento respecto del uso de *cache*s convencionales.

5.1.2 Variables explicativas

Las variables explicativas o independientes se dividen en dos grupos: características o factores *hardware* y características o factores *software*. Esta división permitirá realizar un análisis de la influencia que ejercen sobre la variable dependiente de forma más detallada y sencilla.

A. Factores *hardware*:

- Tamaño de la *cache* (CS): expresado en KiloBytes, indica el número total de bytes que pueden cargarse en *cache*. Para la *cache* con bloqueo no incluye el buffer temporal. Aunque no se considerará explícitamente en los diferentes análisis, sí participará en los mismos de forma implícita, ya que la definición de sistema incluía este tamaño como una característica propia de cada experimento.
- Tamaño de línea (LS): tamaño de una línea de *cache*, expresado normalmente en bytes o alternativamente en instrucciones.
- Tiempo de acierto (T_{hit}): tiempo en ejecutar una instrucción desde memoria *cache* o desde el buffer temporal, medido en ciclos de procesador.
- Tiempo de fallo (T_{miss}): tiempo necesario para llevar un bloque desde memoria principal a la *cache* o al buffer temporal, medido en ciclos de procesador.

B. Factores *software*:

- Tamaño_sistema (SS)
- Número_tareas (TN)
- Ejecutadas_tamaño (ES)
- Tamaño_prioritaria (PS)
- Ejecutadas_total (ET)
- Ratio_tamaño_sistema (SSR)

Las variables independientes enumeradas anteriormente son las resultantes de la eliminación de aquellas que, a través de la matriz de correlación, presentaron indicios claros de correlación lineal. Estas correlaciones dependen únicamente del valor que toman las variables explicativas, y no de los valores de la variable dependiente, por lo que el proceso realizado en el capítulo anterior es completamente válido para los análisis presentados en este capítulo, ya que se ha utilizado el mismo conjunto de tareas y experimentos.

5.2 Análisis de los factores software

Al igual que para el uso estático con prioridades fijas, antes de obtener los modelos de regresión de las *prestaciones* $\pi = U_{cd}/U_{led}$ se ha realizado un estudio del efecto de la relación entre el tamaño de la *cache* y el tamaño del sistema (SSR= CS/SS). El gráfico de dispersión se presenta en la Figura 79, donde se aprecia una clara distribución de los datos muy similar a la observada para prioridades fijas.

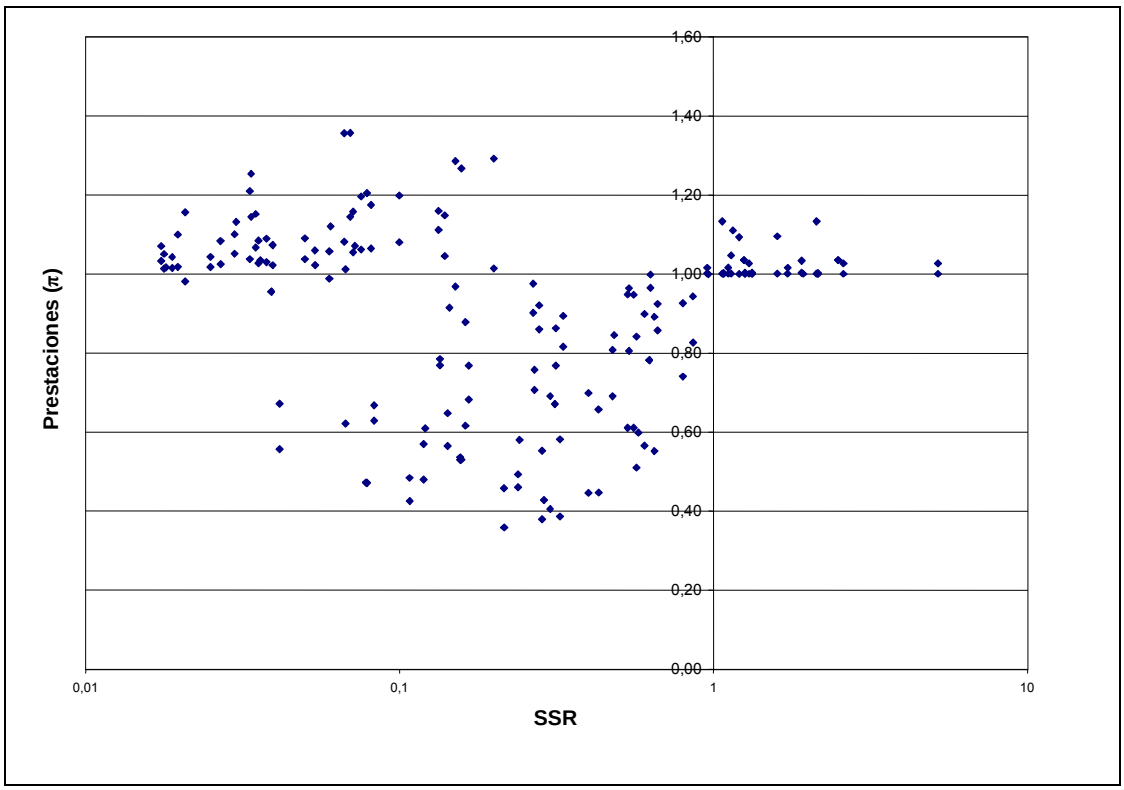


Figura 79. Gráfico de dispersión de $\pi = U_{cd}/U_{led}$ frente a SSR.

Para estudiar mejor el comportamiento de π en función de SSR se han definido los 7 mismos grupos que se definieron para prioridades fijas, y cuyos límites se muestran en la Tabla 56.

Grupo	Tamaño *cache*	Límite inferior	Límite superior
1	1	0	0,0313
2	2	0,0313	0,0525
3	4	0,0525	0,1150
4	8	0,1150	0,2400
5	16	0,2400	0,4900
6	32	0,4900	1,0000
7	64	1,0000	∞

Tabla 56. Valores límite de SSR para la agrupación de $\pi = U_{cd}/U_{led}$.

En la Tabla 57 se muestra un conjunto de estadísticos para cada grupo, y en la Figura 80 la representación gráfica de la agrupación, junto con la media, mediana y el primer y tercer cuartil.

Grupo	1	2	3	4	5	6	7
Total datos	18,000	19,000	28,000	28,000	27,000	26,000	36,000
Media	1,052	1,026	0,954	0,784	0,681	0,827	1,025
Mediana	1,043	1,038	1,061	0,725	0,691	0,874	1,003
Des. Std	0,046	0,164	0,290	0,289	0,180	0,160	0,039
Intervalo de. Conf	0,021	0,074	0,107	0,107	0,068	0,061	0,013
Nº de éxitos ($\pi \geq 1$)	17,000	15,000	19,000	8,000	0,000	2,000	32,000
Probabilidad de éxito	94,4%	78,9%	67,9%	28,6%	0,0%	7,7%	88,9%
Máximo	1,156	1,253	1,357	1,292	0,975	1,016	1,133
Mínimo	0,981	0,557	0,425	0,358	0,379	0,510	1,000
1er cuartil	1,017	1,024	0,658	0,534	0,566	0,751	1,000
3er cuartil	1,080	1,090	1,148	1,022	0,830	0,948	1,035
Coeficiente de curtosis	0,157	3,702	-0,805	-1,146	-1,035	-0,802	2,231
Coeficiente de asimetría	0,813	-1,757	-0,760	0,445	-0,284	-0,731	1,782

Tabla 57. Principales estadísticos de la agrupación de $\pi = U_{cd}/U_{led}$ en función de SSR.

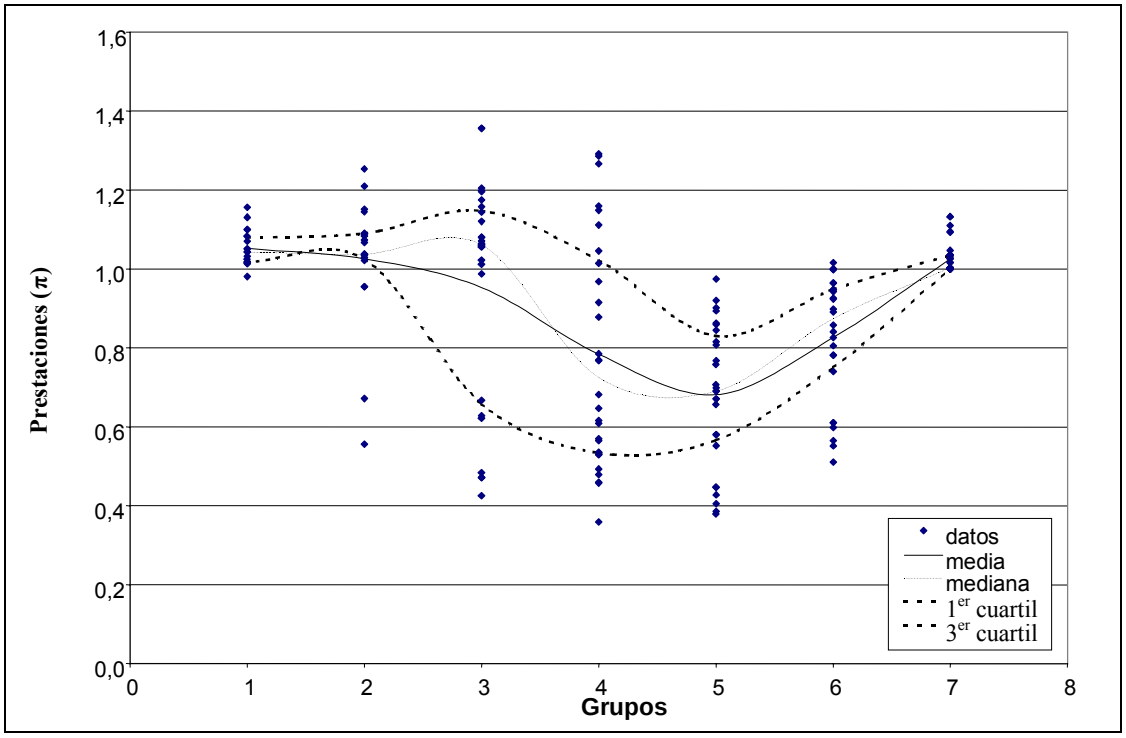


Figura 80. Agrupación de U_{cd}/U_{led} en función de SSR.

Comparando la Figura 38 -prioridades fijas- con la Figura 80 -prioridades dinámicas- se puede observar una gran similitud en la distribución de los puntos así como en las curvas de la media, mediana y los cuartiles. En ambas figuras hay una ganancia generalizada de *prestaciones* en los

extremos derecho e izquierdo, mientras que en la zona central se observa un valle donde las *prestaciones* ofrecidas por la *locking cache* son peores -mayor utilización- que las ofrecidas por las *cache*s convencionales. Además, la variabilidad en los datos es también similar, siendo muy elevada en la zona central y reduciéndose en los extremos. A simple vista no es descabellado decir que ambas gráficas son la misma.

Sin embargo, una observación más detallada, y sobre todo el resumen estadístico mostrado en la Tabla 57 pone de manifiesto ligeras diferencias. Concretamente se detecta una ligera disminución en el número de experimentos que presentan un valor de π superior a 1. Mientras que para prioridades fijas los grupos 1 y 7 presentaban un 100% de éxito ($\pi \geq 1$), esto no es así para prioridades dinámicas. Además, el grupo 5 presenta un 0% de probabilidad de éxito para prioridades dinámicas, mientras que para prioridades fijas este grupo alcanzaba algo más del 3%. También existen diferencias entre los valores de otros estadísticos, pero a simple vista no es posible identificar una tendencia concreta.

La similitud en el comportamiento de las *prestaciones*, independientemente del tipo de planificador utilizado, plantea la posibilidad de utilizar los modelos desarrollados para prioridades fijas en el caso de prioridades dinámicas. Por tanto, se definirán los mismos cuatro espacios, y el análisis de cada uno de los cuatro se realizará de la siguiente manera:

- En primer lugar se realizará un análisis pareado entre los valores de π para prioridades fijas y prioridades dinámicas, para verificar si existe o no diferencia en los datos. Puesto que se han utilizado los mismos experimentos, este análisis pareado es completamente factible.
- Si no existen diferencias, se aplicará el mismo modelo que prioridades fijas, verificando su corrección y comparando el efecto y los coeficientes de los factores. Por el "mismo modelo" se entiende el mismo conjunto de variables explicativas e interacciones, aunque el valor de los coeficientes puede variar.
- Si el análisis pareado muestra que existen diferencias entre los resultados de prioridades fijas y dinámicas, se desarrollará un nuevo modelo para el espacio correspondiente.
- Se utilizará directamente el logaritmo de π para poder comparar los modelos y las conclusiones obtenidas para prioridades fijas con los nuevos modelos obtenidos para prioridades dinámicas.

5.2.1 Modelo de los factores software para el Espacio A

La Tabla 58 muestra la comparación de los principales estadísticos del espacio A para prioridades fijas y prioridades dinámicas. Las únicas diferencias apreciables se encuentran en los valores de asimetría y curtosis, por lo que todo parece indicar que ambas muestras tienen un comportamiento similar. Para confirmar esta impresión se ha realizado el análisis pareado de ambas muestras. El intervalo de confianza para la media, a un 95% de confianza, es [-0,00788639;0,0074102], que al contener el valor 0 permite asumir que ambas muestras tienen la misma media. Así pues, parece factible aplicar el modelo de regresión lineal múltiple obtenido para prioridades fijas a los valores de π obtenidos para prioridades dinámicas. Este modelo se ha aplicado al logaritmo de π, tal como se hizo para prioridades fijas. La Figura 81 muestra la representación en papel probabilístico normal de $\log(\pi)$, que a excepción de dos datos, se distribuye alrededor de una recta, por lo que puede asumirse su normalidad.

Estadístico	Prioridades fijas U_{cd}/U_{lef}	Prioridades dinámicas U_{cd}/U_{led}	Diferencia U_{cd}/U_{led} - U_{cd}/U_{lef}
Nº de experimentos	42	42	0,00000
Media	1,04	1,03976	-0,00024
Mediana	1,04	1,04	0,00000
Varianza	0,010722	0,0130219	0,00230
Desviación típica	0,103547	0,114114	0,01057
Mínimo	0,67	0,56	-0,11000
Máximo	1,25	1,25	0,00000
Primer cuartil	1,01	1,02	0,01000
Tercer cuartil	1,09	1,09	0,00000
Asimetría estándar	-4,82589	-6,47267	-1,64678
Curtosis estándar	8,42719	12,2846	3,85741

Tabla 58. Principales estadísticos de la comparación entre U_{cd}/U_{led} y U_{cd}/U_{lef}. Espacio A.

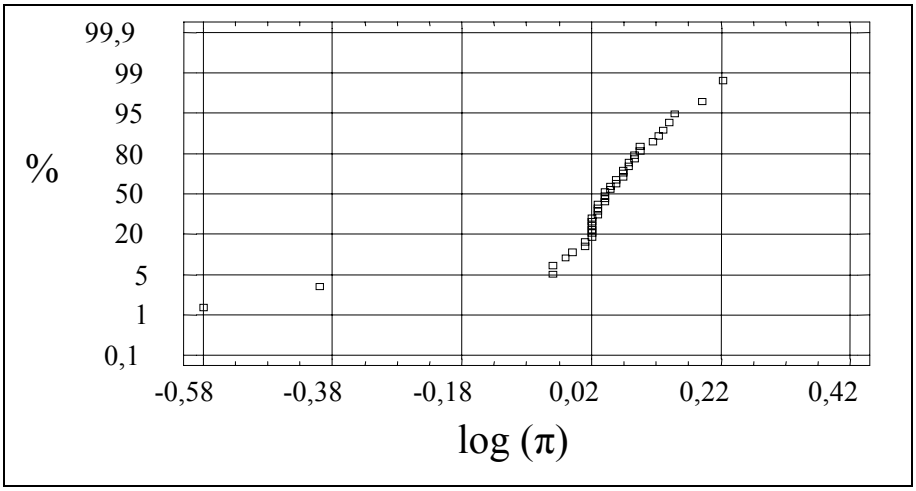


Figura 81. Representación de log ($\pi = U_{cd}/U_{led}$) en papel probabilístico normal. Espacio A.

El modelo obtenido para $\log(\pi) = \log (U_{cd}/U_{led})$ es el siguiente:

Log (π) = 2,52772 - 27,1598*SSR - 0,540008*TN - 0,000888979*SS - 0,000172445*IRF*ES + 1,59306*IRF*SSR - 0,176199*LOC*TN - 0,0011468*LOC*ES - 4,57596*LOC*SSR - 0,00152989*LOC*PS + 0,000763566*LOC*SS + 0,00012422*TN*ES + 4,84317*TN*SSR - 0,0000671937*TN*PS + 0,000206987*TN*SS + 0,073279*ES*SSR + 0,0124053*SSR*PS - 0,00000553075*SSR*ET

El modelo incorpora 17 variables independientes, con una R-Squared del 92,33% -muy similar al valor correspondiente a prioridades fijas-, lo que quiere decir que el modelo explica en más del 90% el comportamiento del logaritmo de π. El estadístico Durbin-watson, con un valor de 1,85, ligeramente superior a 1,4 indica que no existe ningún indicio de autocorrelación en los residuos. La Tabla 59 muestra los coeficientes, el error normalizado y la significación de cada variable independiente, y la Tabla 60 el análisis de la varianza del modelo. En estas tablas se puede observar que todas las variables independientes tienen un P-value inferior a 0,05, lo que indica que son estadísticamente significativas con un nivel de confianza del 95%. Así mismo, el

modelo presenta un P-value de 0,0000 que al ser menor que 0,01 indica que existe una relación estadísticamente significativa entre las variables independientes y la variable dependiente al 99% de confianza.

Factor	Valor Estimado	Standard Error	T-Statistic	P-Value
Constante	2,52772	0,573976	4,40389	0,0002
SSR	-27,1598	10,8714	-2,49828	0,0197
TN	-0,540008	0,115216	-4,68692	0,0001
SS	-0,00088898	0,00016193	-5,48978	0
IRF*ES	-0,00017245	6,1027E-05	-2,82572	0,0094
IRF*SSR	1,59306	0,528895	3,01205	0,006
LOC*TN	-0,176199	0,0537079	-3,2807	0,0032
LOC*ES	-0,0011468	0,00017919	-6,39999	0
LOC*SSR	-4,57596	1,27756	-3,5818	0,0015
LOC*PS	-0,00152989	0,00044753	-3,41855	0,0023
LOC*SS	0,00076357	0,00019387	3,93854	0,0006
TN*ES	0,00012422	4,4533E-05	2,78939	0,0102
TN*SSR	4,84317	1,72126	2,81373	0,0096
TN*PS	-6,7194E-05	3,0535E-05	-2,20052	0,0376
TN*SS	0,00020699	3,7334E-05	5,54427	0
ES*SSR	0,073279	0,0145194	5,04698	0
SSR*PS	0,0124053	0,00543171	2,28386	0,0315
SSR*ET	-5,5308E-06	7,73E-07	-7,15926	0

Tabla 59. Detalles del modelo de log ($\pi = U_{cf}/U_{led}$). Espacio A.

Fuente	Suma de cuadrados	Grados de libertad	Mean Square	F-Ratio	P-Value
Modelo	0,658922	17	0,0387601	17	0,0000
Residual	0,0547353	24	0,00228064		
Total (Corr.)	0,713657	41			

Tabla 60. Análisis de la varianza del modelo de log ($\pi = U_{cf}/U_{led}$). Espacio A.

La Tabla 61 muestra la comparativa de los coeficientes estimados para los factores correspondientes al modelo de regresión para prioridades dinámicas frente a los coeficientes estimados para los factores del modelo de regresión para prioridades fijas. Para cada factor se muestra el coeficiente estimado y el valor de F-Ratio. Además se muestra la variación entre los coeficientes de cada uno de los modelos como el cociente entre el valor estimado para prioridades fijas y el valor estimado para prioridades dinámicas. Cuanto más cercano a 1 sea este cociente, menor diferencia existe. Además, el signo positivo de este cociente indica que ambos coeficientes tienen el mismo signo, mientras que un valor negativo para el cociente indica signos opuestos en los coeficientes.

	Prioridades fijas		Prioridades dinámicas		
Fuente	**F-Ratio**	**Coeficiente estimado**	**F-Ratio**	**Coeficiente estimado**	**Variación coeficientes**
IRF*ES	0,14	-0,00013639	1,26	-0,00017245	0,790895912
IRF*SSR	18,05	1,99859	5,96	1,59306	1,254560406
LOC*ES	2,88	-0,00086597	0,66	-0,0011468	0,755118591
LOC*PS	1,92	-0,00122654	4,24	-0,00152989	0,801717771
LOC*SS	91,39	0,00060773	80,18	0,00076357	0,795906073
LOC*SSR	3,27	-3,97403	2,31	-4,57596	0,868458203
LOC*TN	20,94	-0,13865	20,1	-0,176199	0,786894364
SS	43,34	-0,00068913	38,16	-0,00088898	0,775191793
SSR	6,77	-19,2279	3,47	-27,1598	0,707954403
SSR*ES	26,86	0,0574987	20,76	0,073279	0,784654539
SSR*ET	54,94	-4,49E-06	51,26	-5,53E-06	0,811654733
SSR*PS	16,58	0,00905212	13,08	0,0124053	0,72969779
SSR*TN	26,19	3,3879	18,25	4,84317	0,699521181
TN	24,25	-0,408003	19,21	-0,540008	0,755549918
TN*ES	5,51	0,00010776	8,44	0,00012422	0,867493157
TN*PS	0,06	-5,06E-05	0,06	-6,72E-05	0,752656487
TN*SS	3,06	0,00016139	1,54	0,00020699	0,779699502

Tabla 61. Comparativa de los coeficientes estimados para el modelo de prioridades fijas y para el modelo de prioridades dinámicas. Espacio A.

La Tabla 61 muestra ligeras diferencias en los valores de F-Ratio de los coeficientes de ambos modelos, lo que significa que la importancia de cada factor varía para las dos políticas de planificación. Esto se ve corroborado por las diferencias en los valores estimados de los coeficientes, que aunque son relativamente pequeñas, existen. Sin embargo, todos los signos de los coeficientes -por parejas- coinciden, por lo que las tendencias que marcará cada variable explicativa serán las mismas, más o menos pronunciadas, pero del mismo signo, independientemente de la política de planificación utilizada. De este modo, las conclusiones extraídas para el espacio A con planificador de prioridades fijas serán válidas también para los datos incluidos en el espacio A con prioridades dinámicas EDF.

La validación del modelo se ha realizado mediante la confrontación de los valores observados frente a los valores predichos por el modelo -Figura 82- y la verificación de la normalidad de los residuos del modelo -Figura 83 y Tabla 62-.

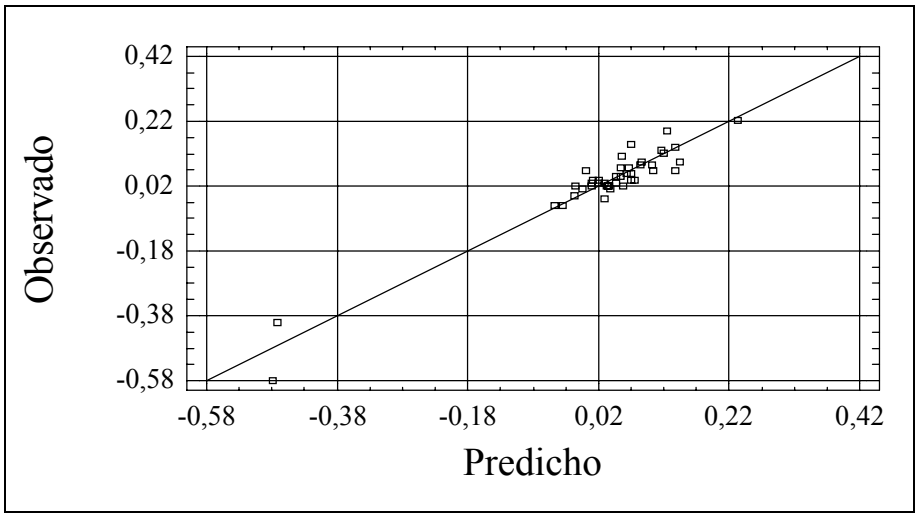


Figura 82. Representación de los valores de log ($\pi = U_{cd}/U_{led}$) observados frente a los predichos por el modelo para el espacio A.

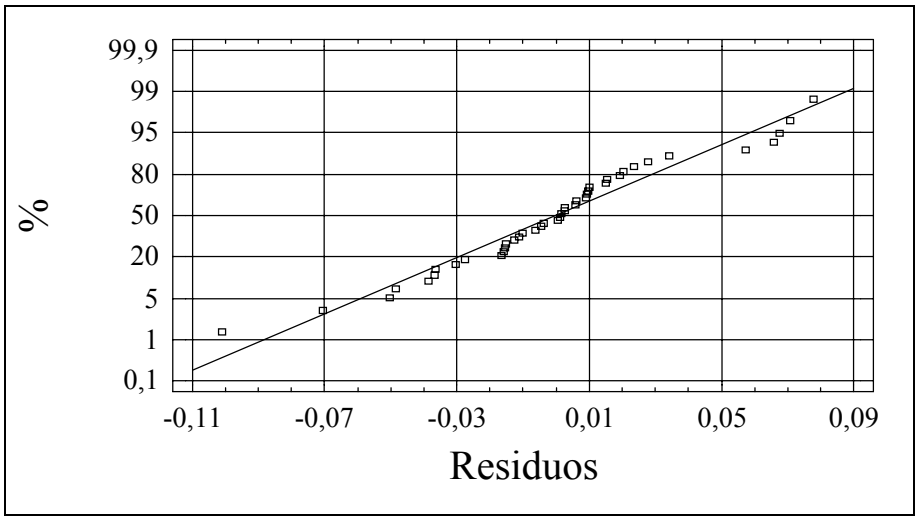


Figura 83. Representación en papel probabilístico normal de los residuos del modelo para el espacio A.

Número de datos	42
Media	-3,09524E-10
Varianza	0,00133501
Desviación estándar	0,0365378
Mínimo	-0,101019
Máximo	0,077691
Asimetría estándar	-0,205498
Curtosis estándar	1,22618

Tabla 62. Principales estadísticos de los residuos del modelo de log ($\pi = U_{cf}/U_{led}$). Espacio A.

5.2.2 Modelo de los factores software para el Espacio B

La Tabla 63 muestra la comparación de los principales estadísticos del espacio B para prioridades fijas y prioridades dinámicas.

Estadístico	**Prioridades fijas U_{cd}/U_{lef}**	**Prioridades dinámicas U_{cd}/U_{led}**	**Diferencia U_{cd}/U_{led} - U_{cd}/U_{lef}**
Nº de experimentos	61	61	0,00000
Media	0,820328	0,827213	0,00689
Mediana	0,78	0,78	0,00000
Varianza	0,0923966	0,0885638	-0,00383
Desviación típica	0,303968	0,297597	-0,00637
Mínimo	0,35	0,36	0,01000
Máximo	1,36	1,36	0,00000
Primer cuartil	0,56	0,55	-0,01000
Tercer cuartil	1,07	1,08	0,01000
Asimetría estándar	0,45976	0,386066	-0,07369
Curtosis estándar	-2,22691	-2,19932	0,02759

Tabla 63. Principales estadísticos de la comparación entre U_{cd}/U_{led} y U_{cd}/U_{lef}. Espacio B.

Las diferencias para los valores de todos los estadísticos son mínimas, por lo que todo parece indicar que ambas muestras se comportan de igual manera. Para confirmar esta impresión se ha realizado el análisis pareado de ambas muestras. El intervalo de confianza para la media, a un 95% de confianza, es [-0,000306315;0,0140768], que al contener el valor 0 permite asumir que ambas muestras tienen la misma media. Así pues, parece factible aplicar el modelo de regresión lineal múltiple obtenido para prioridades fijas a los valores de π obtenidos para prioridades dinámicas. Este modelo se ha aplicado al logaritmo de π, tal como se hizo para prioridades fijas. La Figura 84 muestra la representación en papel probabilístico normal de log(π). Aunque aparece cierta curvatura en los datos, se puede aceptar la normalidad de los mismos, y el modelo explicará esta curvatura al hacer distinción entre los sistemas que hacen un buen aprovechamiento de la *cache* (factor LOC = 1) de los que no (factor LOC = 0).

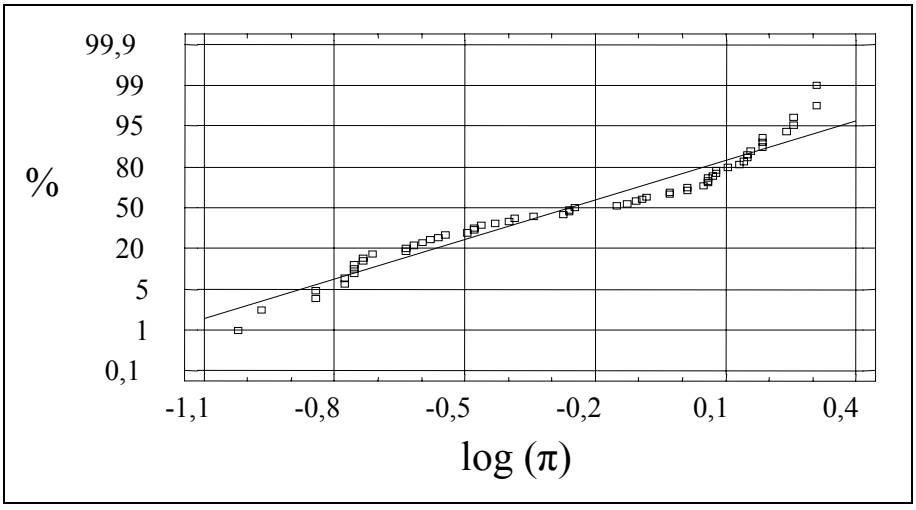


Figura 84. Representación de log ($\pi = U_{cd}/U_{led}$) en papel probabilístico normal. Espacio B.

El modelo obtenido para $\log(\pi) = \log (U_{cd}/U_{led})$ es el siguiente:

$$\log(\pi) = 7,42841 - 1,84399*TN + 0,0179658*ES + 0,00277669*PS - 0,00422605*SS - 0,00000228502*ET - 0,000342227*IRF*ES + 1,15487*IRF*SSR - 2,12456*LOC*TN - 0,00424903*LOC*ES - 0,0190689*LOC*PS + 0,00825092*LOC*SS - 0,000359933*TN*PS + 0,00100434*TN*SS + 2,05264E-7*TN*ET - 0,0009643*SSR*SS + 2,07207E-7*SSR*ET$$

El modelo incorpora 16 variables independientes, con una R-Squared de 90,2% -casi idéntico al correspondiente a prioridades fijas-, lo que quiere decir que el modelo explica aproximadamente el 90% de la variabilidad del logaritmo de π. El estadístico Durbin-watson, con un valor de 3,08, muy superior a 1,4 indica que no existe ningún indicio de autocorrelación en los residuos. La Tabla 64 muestra los coeficientes, el error normalizado y la significación de cada variable independiente, y la Tabla 65 el análisis de la varianza del modelo. En estas tablas se puede observar que todas las variables independientes y las interacciones incluidas en el modelo tienen un P-value inferior a 0,05, lo que indica que son estadísticamente significativas con un nivel de confianza del 95%. Así mismo, el modelo presenta un P-value de 0,0000 que al ser menor que 0,01 indica que existe una relación estadísticamente significativa entre las variables independientes y la variable dependiente al 99% de confianza.

Factor	Valor Estimado	Standard Error	T-Statistic	P-Value
Constante	7,42841	3,47651	2,13674	0,0382
TN	-1,84399	0,806766	-2,28566	0,0271
ES	0,0179658	0,00543172	3,30757	0,0019
PS	0,00277669	0,00129851	2,13837	0,0381
SS	-0,00422605	0,00221854	-1,90488	0,0633
ET	-2,285E-06	3,73E-07	-6,12442	0
IRF*ES	-0,00034223	0,00012026	-2,84568	0,0067
IRF*SSR	1,15487	0,273647	4,22031	0,0001
LOC*TN	-2,12456	0,702692	-3,02347	0,0042
LOC*ES	-0,00424903	0,00218814	-1,94185	0,0586
LOC*PS	-0,0190689	0,00593111	-3,21507	0,0024
LOC*SS	0,00825092	0,00269512	3,06143	0,0037
TN*PS	-0,00035993	0,00016155	-2,22806	0,031
TN*SS	0,00100434	0,00049668	2,02211	0,0493
TN*ET	2,05E-07	7,68E-08	2,67181	0,0105
SSR*SS	-0,0009643	0,00011483	-8,39799	0
SSR*ET	2,07E-07	8,32E-08	2,48983	0,0166

Tabla 64. Detalles del modelo de log ($\pi = U_{cf}/U_{led}$). Espacio B.

Fuente	Suma de cuadrados	Grados de libertad	Mean Square	F-Ratio	P-Value
Modelo	7,94417	16	0,496511	25,38	0,0000
Residual	0,860942	44	0,0195669		
Total (Corr.)	8,80512	60			

Tabla 65. Análisis de la varianza del modelo de log ($\pi = U_{cf}/U_{led}$). Espacio B.

La Tabla 66 muestra la comparativa de los coeficientes estimados para los factores correspondientes al modelo de regresión para prioridades dinámicas frente a los coeficientes estimados para los factores del modelo de regresión para prioridades fijas. Para cada factor se muestra el coeficiente estimado y el valor de F-Ratio. Además se muestra la variación entre los coeficientes de cada modelo como el cociente entre el valor estimado para prioridades fijas y el valor estimado para prioridades dinámicas. Canto más cercano a 1 sea este cociente, menor diferencia existe. Además, el signo positivo de este cociente indica que ambos coeficientes tienen el mismo signo, mientras que una valor negativo para el cociente indica signos opuestos en los coeficientes.

	Prioridades fijas		Prioridades dinámicas		
Fuente	F-Ratio	Coeficiente estimado	F-Ratio	Coeficiente estimado	Variación coeficientes
ES	9,96	0,0182397	4,93	0,0179658	1,015245633
ET	14,32	-2,26E-06	15,85	-2,285E-06	0,986862259
IRF*ES	1,37	-0,00046399	0,21	-0,00034223	1,355796007
IRF*SSR	0,88	1,3525	0,21	1,15487	1,171127486
LOC*ES	60,48	-0,00435036	59,36	-0,00424903	1,023847796
LOC*PS	8,52	-0,0195533	8,74	-0,0190689	1,025402619
LOC*SS	39,54	0,00847027	33,61	0,00825092	1,026584914
LOC*TN	74,77	-2,18453	76,96	-2,12456	1,028227021
PS	9,05	0,00279844	9,39	0,00277669	1,007833067
SS	3,97	-0,00434027	3,51	-0,00422605	1,027027603
SSR*ET	3,47	1,58E-07	6,2	2,07E-07	0,762522502
SSR*SS	72,11	-0,00097341	67,69	-0,0009643	1,009447267
TN	73,23	-1,88823	73,11	-1,84399	1,023991453
TN*ET	0,9	1,96E-07	1,32	2,05E-07	0,954867877
TN*PS	5,64	-0,00036076	6,71	-0,00035993	1,00229765
TN*SS	36,67	0,00103008	38,21	0,00100434	1,025628771

Tabla 66. Comparativa de los coeficientes estimados para el modelo de prioridades fijas y para el modelo de prioridades dinámicas. Espacio B.

La Tabla 66 muestra que las diferencias en los valores de F-Ratio como en los valores estimados de los coeficientes de ambos modelos son mínimas para la gran mayoría de las variables explicativas y sus interacciones, lo que significa que la importancia de cada factor variará sólo ligeramente en función de la política de planificación empleada. Además, todos los signos de los coeficientes -por parejas- coinciden, por lo que las tendencias que marcará cada variable explicativa y las correspondientes interacciones serán las mismas, más o menos

pronunciadas, pero del mismo signo. De este modo, las conclusiones extraídas para el espacio B con planificador de prioridades fijas serán válidas también para sistemas planificados con prioridades dinámicas EDF.

La validación del modelo se ha realizado mediante la confrontación de los valores observados frente a los valores predichos por el modelo -Figura 85- y la verificación de la normalidad de los residuos del modelo -Figura 86 y Tabla 67-.

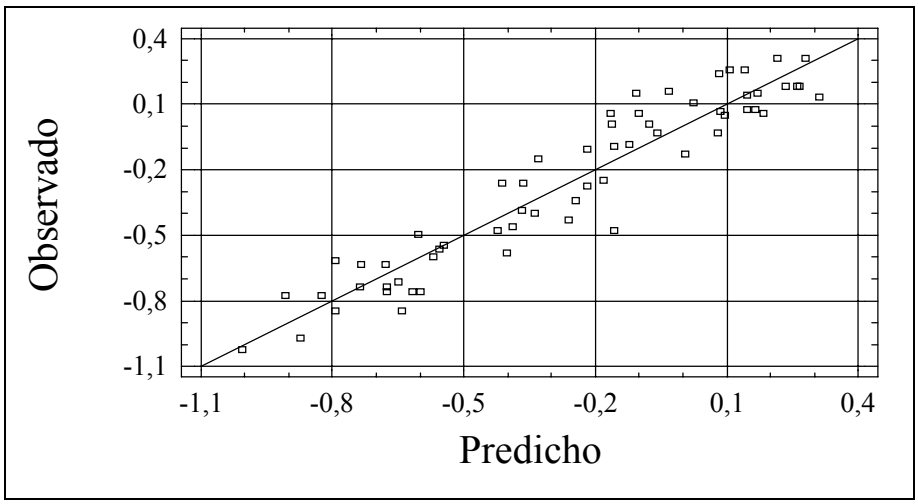


Figura 85. Representación de los valores de log ($\pi = U_{cd}/U_{led}$) observados frente a los predichos por el modelo para el espacio B.

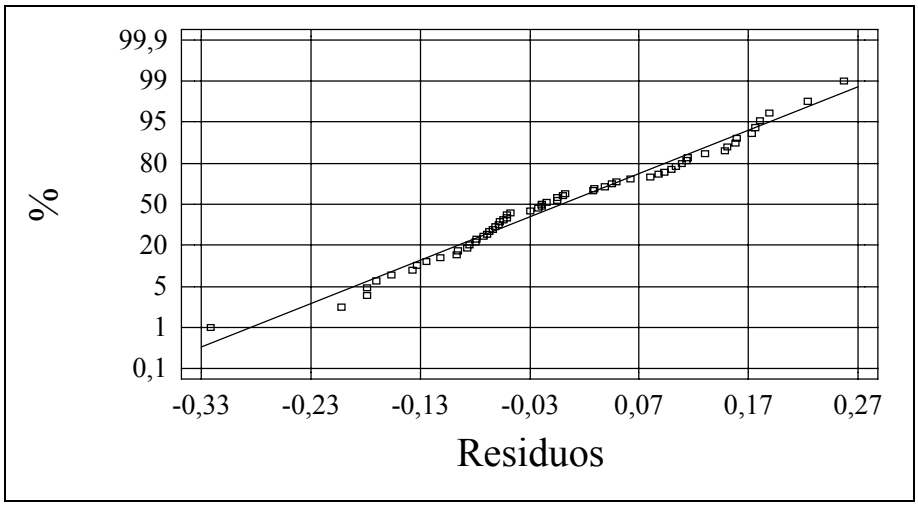


Figura 86. Representación en papel probabilístico normal de los residuos del modelo para el espacio B.

Número de datos	61
Media	9,41475E-9
Varianza	0,014349
Desviación estándar	0,119787
Mínimo	-0,321165
Máximo	0,256286
Asimetría estándar	0,021328
Curtosis estándar	-0,501639

Tabla 67. Principales estadísticos de los residuos del modelo de log ($\pi = U_{cf}/U_{led}$). Espacio B.

5.2.3 Modelo de los factores software para el Espacio C

La Tabla 68 muestra la comparación de los principales estadísticos del espacio C para prioridades fijas y prioridades dinámicas.

Estadístico	**Prioridades fijas U_{cd}/U_{lef}**	**Prioridades dinámicas U_{cd}/U_{led}**	**Diferencia U_{cd}/U_{led} - U_{cd}/U_{lef}**
Nº de experimentos	43	43	0,00000
Media	0,762791	0,764419	0,00163
Mediana	0,78	0,81	0,03000
Varianza	0,0365301	0,0313872	-0,00514
Desviación típica	0,191129	0,177164	-0,01397
Mínimo	0,38	0,39	0,01000
Máximo	1,07	1,02	-0,05000
Primer cuartil	0,61	0,61	0,00000
Tercer cuartil	0,91	0,92	0,01000
Asimetría estándar	-0,800296	-1,29314	-0,49284
Curtosis estándar	-1,04563	-1,02337	0,02226

Tabla 68. Principales estadísticos de la comparación entre U_{cd}/U_{led} y U_{cd}/U_{lef}. Espacio C.

Las diferencias para los valores de todos los estadísticos son mínimas, por lo que todo parece indicar que ambas muestras se comportan de igual manera. Para confirmar esta impresión se ha realizado el análisis pareado de ambas muestras. El intervalo de confianza para la media, a un 95% de confianza, es [-0,012052;0,0153078], que al contener el valor 0 permite asumir que ambas muestras tienen la misma media. Así pues, parece factible aplicar el modelo de regresión lineal múltiple obtenido para prioridades fijas a los valores de π obtenidos para prioridades dinámicas. Este modelo se ha aplicado al logaritmo de π, tal como se hizo para prioridades fijas. La Figura 87 muestra la representación en papel probabilístico normal de log (π). Al igual que para los valores de log (π) enmarcados en el espacio B aparece cierta curvatura, aunque en este caso es menos pronunciada.

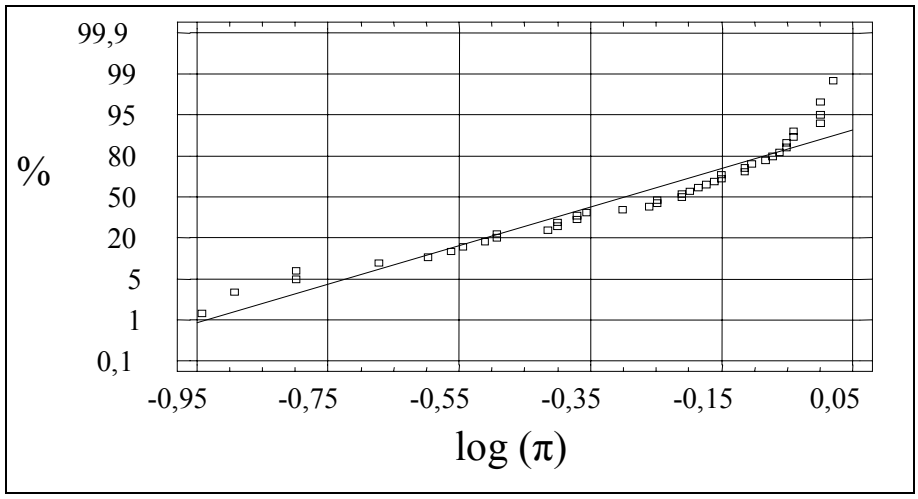


Figura 87. Representación de log ($\pi = U_{cd}/U_{led}$) en papel probabilístico normal. Espacio C.

El modelo obtenido para $\log(\pi) = \log (U_{cd}/U_{led})$ es el siguiente:

$$\log(\pi) = -1{,}29007 + 0{,}772682*IRF + 0{,}0113383*ES + 0{,}967581*SSR - 0{,}00000169967*ET - 0{,}0968254*IRF*TN - 0{,}000636002*IRF*ES - 0{,}00196577*TN*ES + 0{,}0000212244*TN*SS + 3{,}42783E\text{-}7*TN*ET - 9{,}0983E\text{-}8*SSR*ET$$

El modelo incorpora 10 variables independientes, con una R-Squared del 87,95% -casi idéntico al correspondiente a prioridades fijas-, lo que quiere decir que el modelo explica aproximadamente el 88% de la variabilidad del logaritmo de π, mientras que el resto de la variabilidad es debido a factores no considerados en este análisis o a variaciones aleatorias en los datos. El estadístico Durbin-watson, con un valor de 1,84, por encima de 1,4, indica que no existe ningún indicio de autocorrelación en los residuos. La Tabla 69 muestra los coeficientes, el error normalizado y la significación de cada variable independiente, y la Tabla 70 el análisis de la varianza del modelo. En estas tablas se puede observar que todas las variables independientes tienen un P-value inferior a 0,05, lo que indica que son estadísticamente significativas con un nivel de confianza del 95%. Así mismo, el modelo presenta un P-value de 0,0000 que al ser menor que 0,01 indica que existe una relación estadísticamente significativa entre la variable dependiente y las variables independientes al 99% de confianza.

Factor	Valor Estimado	Standard Error	T-Statistic	P-Value
Constante	-1,29007	0,0896114	-14,3963	0
IRF	0,772682	0,134356	5,75101	0
ES	0,0113383	0,00406178	2,79145	0,0088
SSR	0,967581	0,105443	9,1763	0
ET	-1,6997E-06	5,33E-07	-3,18679	0,0032
IRF*TN	-0,0968254	3,19E-02	-3,03176	0,0048
IRF*ES	-0,000636	0,000137058	-4,64039	0,0001
TN*ES	-0,00196577	0,000819266	-2,39943	0,0224
TN*SS	2,12244E-05	5,61275E-06	3,78145	0,0006
TN*ET	3,43E-07	1,07E-07	3,21501	0,003
SSR*ET	-9,10E-08	2,56E-08	-3,55791	0,0012

Tabla 69. Detalles del modelo de log ($\pi = U_{cf}/U_{led}$). Espacio C.

Fuente	Suma de cuadrados	Grados de libertad	Mean Square	F-Ratio	P-Value
Modelo	2,46122	10	0,246122	23,37	0,0000
Residual	0,336937	32	0,0105293		
Total (Corr.)	2,79816	42			

Tabla 70. Análisis de la varianza del modelo de log ($\pi = U_{cf}/U_{led}$). Espacio C.

La Tabla 71 muestra la comparativa de los coeficientes estimados para los factores correspondientes al modelo para prioridades dinámicas frente a los coeficientes estimados para los factores del modelo de prioridades fijas. Para cada factor se muestra el coeficiente estimado y el valor de F-Ratio. Además se muestra la variación entre los coeficientes de los modelos como el cociente entre el valor estimado para prioridades fijas y el valor estimado para prioridades dinámicas. Cuanto más cercano a 1 sea este cociente, menor diferencia existe. Además, el signo positivo de este cociente indica que para ambos modelos, ambos coeficientes tienen el mismo signo, mientras que un valor negativo para el cociente indica signos opuestos en los coeficientes.

	Prioridades fijas		Prioridades dinámicas		
Fuente	**F-Ratio**	**Coeficiente estimado**	**F-Ratio**	**Coeficiente estimado**	**Variación coeficientes**
ES	17,72	0,0116071	27,46	0,0116071	1
ET	10,16	-1,79E-06	10,18	-1,7935E-06	0,999988849
IRF	62,58	0,915466	51,95	0,915466	1
IRF*ES	25,73	-0,00082844	16,61	-0,00082844	0,999995172
IRF*TN	5,43	-0,111507	4,68	-0,111507	1
SSR	58,92	0,937331	72,46	0,937331	1
SSR*ET	9,98	-8,64E-08	12,66	-8,64E-08	1,000035881
TN*ES	19,36	-0,00196996	14,23	-0,00196996	1
TN*ET	9,97	3,59E-07	10,39	3,59E-07	0,999537818
TN*SS	15,43	2,45E-05	13,13	2,45098E-05	1,00000816

Tabla 71. Comparativa de los coeficientes estimados para el modelo de prioridades fijas y para el modelo de prioridades dinámicas. Espacio C.

La Tabla 71 muestra que aunque existen diferencias en los valores de F-Ratio, no sucede así con los valores estimados para los coeficientes. Además, todos los signos de los coeficientes -por parejas- coinciden, por lo que las tendencias que marcará cada variable explicativa y las correspondientes interacciones serán las mismas, más o menos pronunciadas, pero del mismo signo. De este modo, las conclusiones extraídas para el espacio C con planificador de prioridades fijas serán válidas también para prioridades dinámicas EDF.

La validación del modelo se ha realizado mediante la confrontación de los valores observados frente a los valores predichos por el modelo -Figura 88- y la verificación de la normalidad de los residuos del modelo -Figura 89 y Tabla 72-.

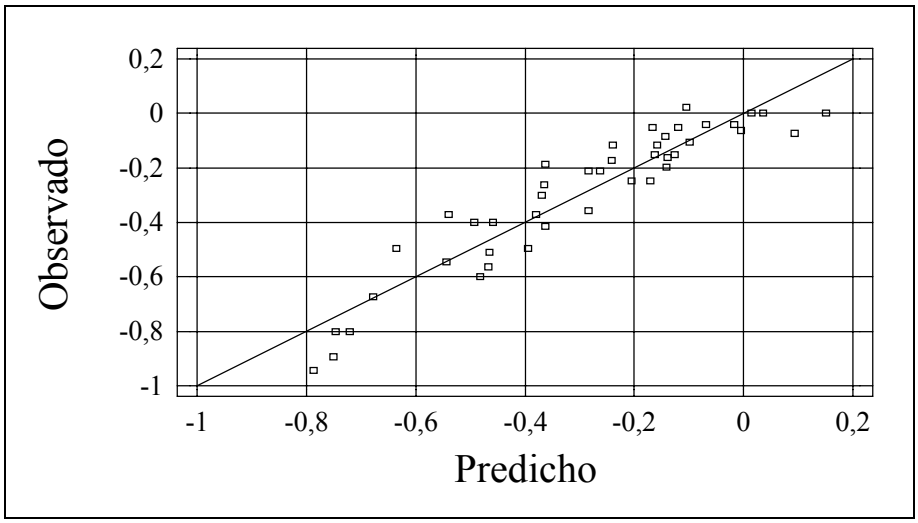


Figura 88. Representación de los valores de log ($\pi = U_{cd}/U_{led}$) observados frente a los predichos por el modelo para el espacio C.

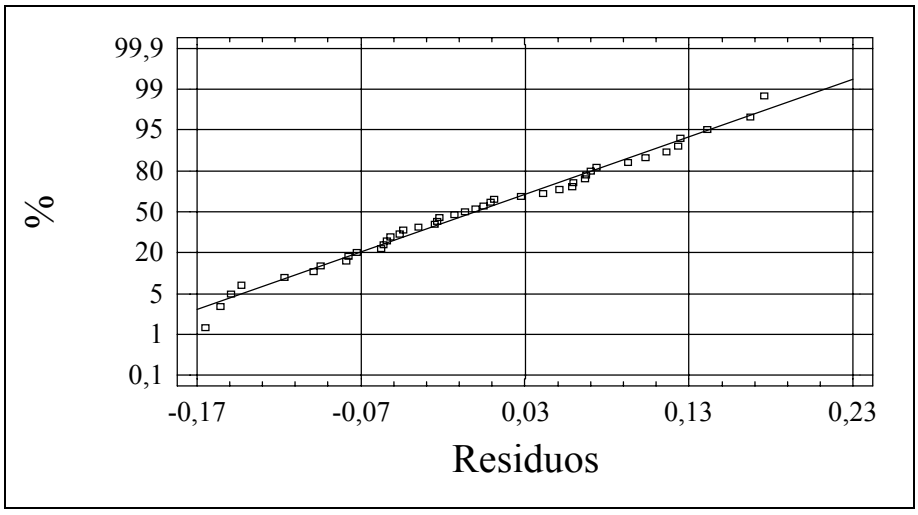


Figura 89. Representación en papel probabilístico normal de los residuos del modelo para el espacio C.

Número de datos	43
Media	4,14419E-8
Varianza	0,00802231
Desviación estándar	0,0895673
Mínimo	-0,164757
Máximo	0,175998
Asimetría estándar	0,195555
Curtosis estándar	-0,975145

Tabla 72. Principales estadísticos de los residuos del modelo de log ($\pi = U_{cf}/U_{led}$). Espacio C.

5.2.4 Modelo de los factores software para el Espacio D

La Tabla 73 muestra la comparación de los principales estadísticos del espacio D para prioridades fijas y prioridades dinámicas.

Estadístico	**Prioridades fijas U_{cd}/U_{lef}**	**Prioridades dinámicas U_{cd}/U_{led}**	**Diferencia U_{cd}/U_{led} - U_{cd}/U_{lef}**
Nº de experimentos	36	36	0,00000
Media	1,05056	1,02583	-0,02473
Mediana	1,03	1,0	-0,03000
Varianza	0,00218825	0,00152214	-0,00067
Desviación típica	0,0467788	0,0390146	-0,00776
Mínimo	1,0	1,0	0,00000
Máximo	1,16	1,13	-0,03000
Primer cuartil	1,02	1,0	-0,02000
Tercer cuartil	1,095	1,04	-0,05500
Asimetría estándar	2,53683	4,00489	1,46806
Curtosis estándar	-0,0755459	2,12556	2,20111

Tabla 73. Principales estadísticos de la comparación entre U_{cd}/U_{led} y U_{cd}/U_{lef}. Espacio D.

Las diferencias para los valores de todos los estadísticos son mínimas, por lo que todo parece indicar que ambas muestras se comportan de igual manera. Para confirmar esta impresión se ha realizado el análisis pareado de ambas muestras. El intervalo de confianza para la media, a un 95% de confianza, es [-0,0397776;-0,00966681], que al no contener el valor 0 indica que existen diferencias significativas en la media de ambas muestras, y posiblemente sea un indicio de diferencias importantes en el comportamiento de los datos. La aplicación del modelo desarrollado para el espacio D de prioridades fijas sobre los datos de prioridades dinámicas presenta un valor de R-Squared del 33,9%, muy inferior al 98,8% obtenido por el mismo modelo cuando se aplicó a los datos obtenidos con la política de planificación de prioridades fijas. Comprobando la normalidad de los datos de $\pi = U_{cd}/U_{led}$ -Figura 90- mediante el gráfico de probabilidad normal, se puede apreciar que una parte importante de los datos no se distribuyen alrededor de una recta, tomando todos estos datos el mismo valor de 1. Esta falta de normalidad no puede corregirse mediante simples transformaciones como se ha hecho en los espacios anteriores aplicando el logaritmo a los valores de π, por lo que se ha intentado encontrar una relación entre alguna de las variables explicativas y los valores de π. La Figura 91 muestra la representación de $\pi = U_{cd}/U_{led}$ frente a los valores de SSR, identificando, además, los experimentos por su valor de IRF. En este gráfico no se puede apreciar relación aparente entre SSR y π, pero sí es trivial la identificación de una relación entre π y los valores de IRF: para todos aquellos sistemas con valor de IRF igual a 1, las *prestaciones* obtenidas al utilizar *locking cache* con prioridades dinámicas EDF son las mismas que al utilizar *cache*s convencionales, es decir, $\pi = 1$.

Esta relación tan extremadamente directa entre π y la variable explicativa IRF es debida a que la evaluación de los experimentos se ha realizado empleando la utilización real obtenida de las simulaciones de las *cache*s convencionales y la utilización estimada por el análisis de planificabilidad de la *cache* con bloqueo, y no la máxima utilización planificable. Esta utilización real o estimada depende del tiempo de funcionamiento del sistema que se simule, ya que el tiempo de cómputo de una tarea varía o puede variar en cada ejecución. Para las *cache*s convencionales, la primera ejecución de cada tarea incorporará los fallos obligatorios necesarios para cargar la *cache*, mientras que las siguientes ejecuciones consumirán un tiempo menor de procesador, ya que la *cache* se encuentra cargada, y al ser el tamaño de esta mayor que el código

de las tareas, no se produce ningún conflicto. Para la *cache* con bloqueo el tiempo de cómputo de las tareas es el mismo para todas las ejecuciones, ya que dicha *cache* es precargada antes de iniciarse la ejecución del sistema, por lo que no existe ningún fallo obligatorio. Resumiendo, la primera ejecución de cada tarea tendrá un coste temporal superior que las restantes para las *cache*s convencionales, mientras que para la *locking cache* el coste temporal será siempre constante.

Tal como se indicó al especificar los experimentos, los sistemas sin interferencia tienen asignado el mismo periodo a todas las tareas, por lo que el hiperperiodo corresponde a una sola activación de cada tarea. Si sólo se considera una activación de cada tarea, la utilización presentada por las *cache*s convencionales será mayor que la presentada por la *locking cache*, ya que esta primera activación incluye los fallos obligatorios. Por el contrario, en los sistemas con interferencia, el hiperperiodo contendrá múltiples ejecuciones de cada tarea -decenas e incluso cientos- por lo que la utilización de las *cache*s convencionales tenderá a ser la ofrecida por la segunda y posteriores ejecuciones, con un coste menor que la primera ejecución. Estas ejecuciones tienen además el mismo coste que la primera ejecución -y todas las restantes- de la *locking cache*, ya que se producen en las mismas condiciones: la *cache* cargada y sin coste de recarga tras una expulsión. Por este motivo, la utilización de los sistemas para los que se simula un largo periodo de tiempo y en los que se producen múltiples ejecuciones de cada tarea tiende a ser la misma, independientemente del esquema de *cache –locking* o convencional- utilizado.

Este efecto no se producía en los experimentos con planificador estático, ya que para estos sistemas se empleaba la utilización planificable, que considera el peor caso posible, correspondiente a la primera activación de todas las tareas en las *cache*s convencionales.

Pese a que la influencia del factor IRF se debe al modo en que se realizan los experimentos, el análisis de los datos enmarcados en el espacio D se realizará de forma independiente para cada tipo de sistema: IRF = 0 e IRF = 1. Este análisis independiente para cada tipo de sistema permitirá, con los datos disponibles, obtener resultados y conclusiones más ajustadas y válidas. Sin embargo, conociendo la verdadera causa del efecto de IRF y las condiciones en que se ejecutan las tareas, se puede asumir que, en el peor caso y si se considerara la planificación utilizable, ambos tipos de sistemas se comportan de igual manera, independientemente del valor de IRF, y que este comportamiento corresponde al mostrado por los sistemas cuyo factor IRF toma el valor 0.

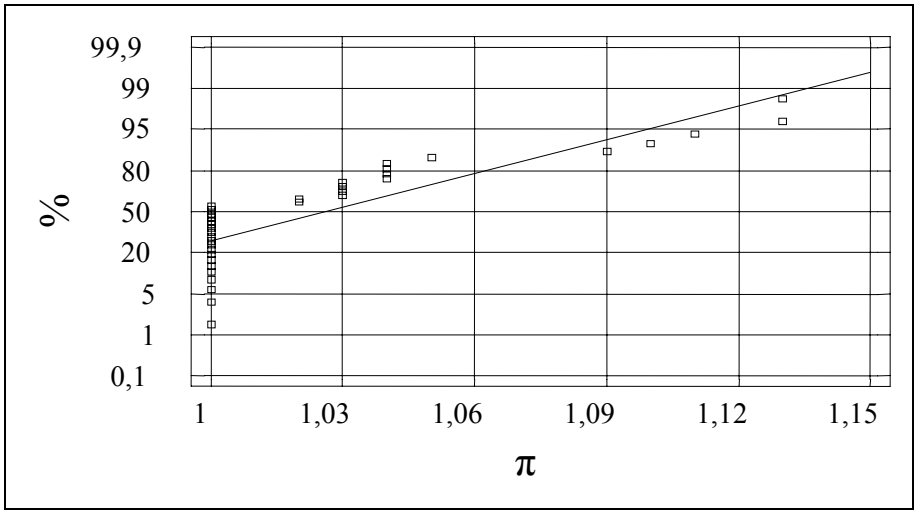


Figura 90. Representación de $\pi = U_{cd}/U_{led}$ en papel probabilístico normal. Espacio D.

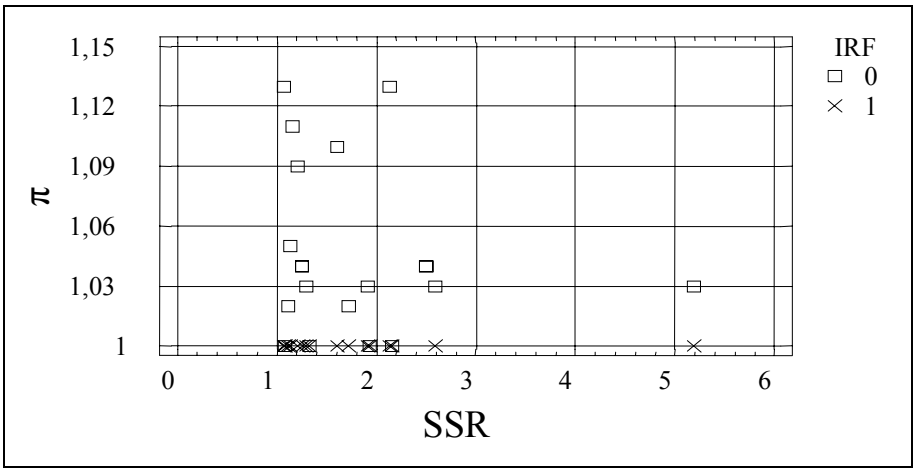


Figura 91. Gráfico de dispersión de $\pi = U_{cd}/U_{led}$ frente a SSR y especificado el valor de IRF. Espacio D.

A. Análisis para sistemas con IRF igual a cero

El primer paso es comparar los valores de π para los sistemas con prioridades fijas frente a los correspondientes valores para los sistemas con prioridades dinámicas. La Figura 92 muestra esta comparación, y en ella puede observarse que los valores son exactamente los mismos para planificador con prioridades fijas que para planificador con prioridades dinámicas. Esto es lógico, ya que se trata de sistemas sin interferencia, es decir, sin que se produzcan expulsiones, por lo que el tipo de planificador no ejerce ninguna influencia sobre el comportamiento de los datos.

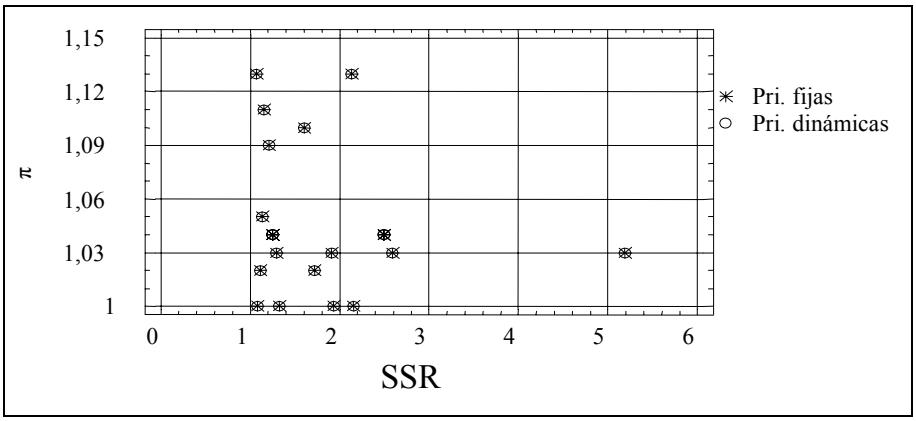


Figura 92. Gráfico de dispersión de $\pi = U_{cd}/U_{led}$ y $\pi = U_{cf}/U_{lef}$ frente a SSR. Espacio D.

Por tanto, el modelo descrito para los datos del espacio D cuando se utiliza un planificador de prioridades fijas será completamente válido, tanto en los factores y sus interacciones como en los coeficientes estimados, para los datos del mismo espacio correspondientes a un planificador de prioridades dinámicas, con la única salvedad que el valor del factor IRF será siempre cero para poder aplicar dicho modelo.

B. Análisis para sistemas con IRF igual a uno

No tiene sentido estudiar un modelo de regresión para estos datos, ya que como se mostraba en la Figura 91 todos los experimentos presentan un valor de π igual o extremadamente próximo a 1 independientemente de sus características. Tampoco tiene sentido realizar ningún tipo de comparación entre las *prestaciones* obtenidas al utilizar ambas políticas de planificación, ya que la métrica empleada para calcular las *prestaciones* es distinta y, en este caso concreto, ofrece resultados sesgados para los experimentos planificados mediante prioridades dinámicas EDF.

5.3 Análisis de los factores hardware

El estudio del efecto de los factores *hardware* sobre el comportamiento de las *prestaciones* del uso estático de *locking cache* cuando se utiliza un planificador de prioridades dinámicas EDF se ha realizado bajo las mismas condiciones que para el uso estático de *locking cache* con planificador de prioridades fijas presentado en el capítulo anterior. Como recordatorio, se ha realizado un diseño de experimentos 2^K, y a continuación se describen brevemente las características más importantes de este análisis:

- El proceso bajo estudio o variable dependiente sigue siendo la relación entre la utilización obtenida al emplear el uso estático de *locking cache* frente al empleo de *cache*s convencionales, en sistemas de prioridades dinámicas, definida esta relación como $\pi = U_{cd}/U_{led}$
- La imposibilidad de obtener la utilización planificable a partir de los resultados del algoritmo genético hace que los valores de U_{cd} y U_{led} correspondan con la utilización real del sistema, bien simulada para las *cache*s convencionales, bien estimada por el algoritmo genético para las *cache*s con bloqueo.
- Se ha eliminado el factor "tamaño de *cache*" CS y se ha incorporado el factor SSR, debido a que este último aporta mucha más información sobre el comportamiento de las *prestaciones* π.
- Debido a las diferencias en el comportamiento de π en función de los factores IRF y LOC mostradas en los análisis anteriores, y presumiendo que estos factores interactuarán, en algunos casos, con los factores *hardware*, ambos factores *software* se incluyen en el análisis.
- De los tres factores *hardware* restantes, tamaño de línea de *cache* LS, Tiempo de fallo T_{miss}, y Tiempo de acierto T_{hit}, sólo se estudiarán los dos primeros, ya que, en cuanto a los tiempos de acceso, el factor importante es la relación entre el tiempo de acierto y el tiempo de fallo. Por ello, el valor de T_{hit} se establece en un ciclo de procesador, y se redefine T_{miss} como la relación entre el tiempo de fallo y el tiempo de acierto en *cache*.
- Se han llevado a cabo 112 nuevos experimentos asignando los factores bajo estudio a dos niveles. Estos experimentos, y los valores y niveles asignados a cada variable independiente se describen en la Tabla 74.

Aunque es posible realizar un diseño de experimentos que contemple simultáneamente todos los factores, se ha preferido considerar por separado el efecto de SSR, ya que es posible que la relación entre el tamaño del sistema y el tamaño de la *cache* determine el efecto del resto de los factores. De esta forma se elimina una posible interacción, facilitando por tanto la interpretación de los resultados.

Para estudiar el efecto de SSR se ha realizado el gráfico de dispersión de los valores de $\pi = U_{cd}/U_{led}$ frente a los valores de SSR, mostrado en la Figura 93. El objetivo de esta gráfica no es mostrar cómo evolucionan los valores de π en función de los valores de SSR, sino comprobar si existe diferencia entre el efecto que los factores LS, T_{miss}, IRF y LOC producen en las *prestaciones* π en función de los valores de SSR. Cada "columna" de puntos corresponde al mismo conjunto de tareas con un determinado tamaño de *cache*, y cada punto de cada "columna" corresponde a un experimento con una determinada configuración de los factores bajo estudio. Como era de esperar, los puntos de cada "columna" no se superponen, mostrando claramente que los factores LS, T_{miss}, IRF y LOC afectan a las *prestaciones* obtenidas. Pero también puede observarse que la distancia entre los puntos difiere para cada columna, en función de la posición de la misma, es decir, del valor de SSR de cada experimento. Esto indica claramente que existe una interacción entre el factor SSR y el resto de factores. Por este motivo el estudio del efecto de los factores restantes se realizará agrupando los experimentos en función de su valor de SSR. Aunque lo ideal sería dividir los experimentos en cuatro espacios, tal como se hizo para el análisis de los factores *software*, el reducido número de experimentos no permite

hacer una división tan fina, por lo que para este análisis se considerarán sólo tres espacios, a los que llamaremos ϵ_1, ϵ_2 y ϵ_3. La Tabla 75 muestra los límites inferior y superior de cada espacio.

Conjunto de tareas	LOC (Nivel)	IRF (Nivel)	Tamaño de línea LS (Nivel)	Relación miss/hit T_{miss} (Nivel)	Tamaño de *cache*
A	0 (-)	0 (-)	16 bytes = 4 instr. (-)	5 ciclos (-)	1Kb a 64Kb
A	0 (-)	0 (-)	16 bytes = 4 instr. (-)	10 ciclos (+)	1Kb a 64Kb
A	0 (-)	0 (-)	32 bytes = 8 instr. (+)	5 ciclos (-)	1Kb a 64Kb
A	0 (-)	0 (-)	32 bytes = 8 instr. (+)	10 ciclos (+)	1Kb a 64Kb
A	0 (-)	1 (+)	16 bytes = 4 instr. (-)	5 ciclos (-)	1Kb a 64Kb
A	0 (-)	1 (+)	16 bytes = 4 instr. (-)	10 ciclos (+)	1Kb a 64Kb
A	0 (-)	1 (+)	32 bytes = 8 instr. (+)	5 ciclos (-)	1Kb a 64Kb
A	0 (-)	1 (+)	32 bytes = 8 instr. (+)	10 ciclos (+)	1Kb a 64Kb
B	1 (+)	0 (-)	16 bytes = 4 instr. (-)	5 ciclos (-)	1Kb a 64Kb
B	1 (+)	0 (-)	16 bytes = 4 instr. (-)	10 ciclos (+)	1Kb a 64Kb
B	1 (+)	0 (-)	32 bytes = 8 instr. (+)	5 ciclos (-)	1Kb a 64Kb
B	1 (+)	0 (-)	32 bytes = 8 instr. (+)	10 ciclos (+)	1Kb a 64Kb
B	1 (+)	1 (+)	16 bytes = 4 instr. (-)	5 ciclos (-)	1Kb a 64Kb
B	1 (+)	1 (+)	16 bytes = 4 instr. (-)	10 ciclos (+)	1Kb a 64Kb
B	1 (+)	1 (+)	32 bytes = 8 instr. (+)	5 ciclos (-)	1Kb a 64Kb
B	1 (+)	1 (+)	32 bytes = 8 instr. (+)	10 ciclos (+)	1Kb a 64Kb

Tabla 74. Características de los diferentes experimentos realizados para la evaluación del impacto de los factores *hardware* sobre $\pi = U_{cf}/U_{led}$.

Espacio	Límite inferior	Límite superior
ϵ_1	$0 \leq SSR$	$SSR < 0,1$
ϵ_2	$0,1 \leq SSR$	$SSR < 1$
ϵ_3	$1 \leq SSR$	------

Tabla 75. Valores límites de SSR para la agrupación de $\pi = U_{cf}/U_{led}$ en los espacios ϵ_1, ϵ_2 y ϵ_3.

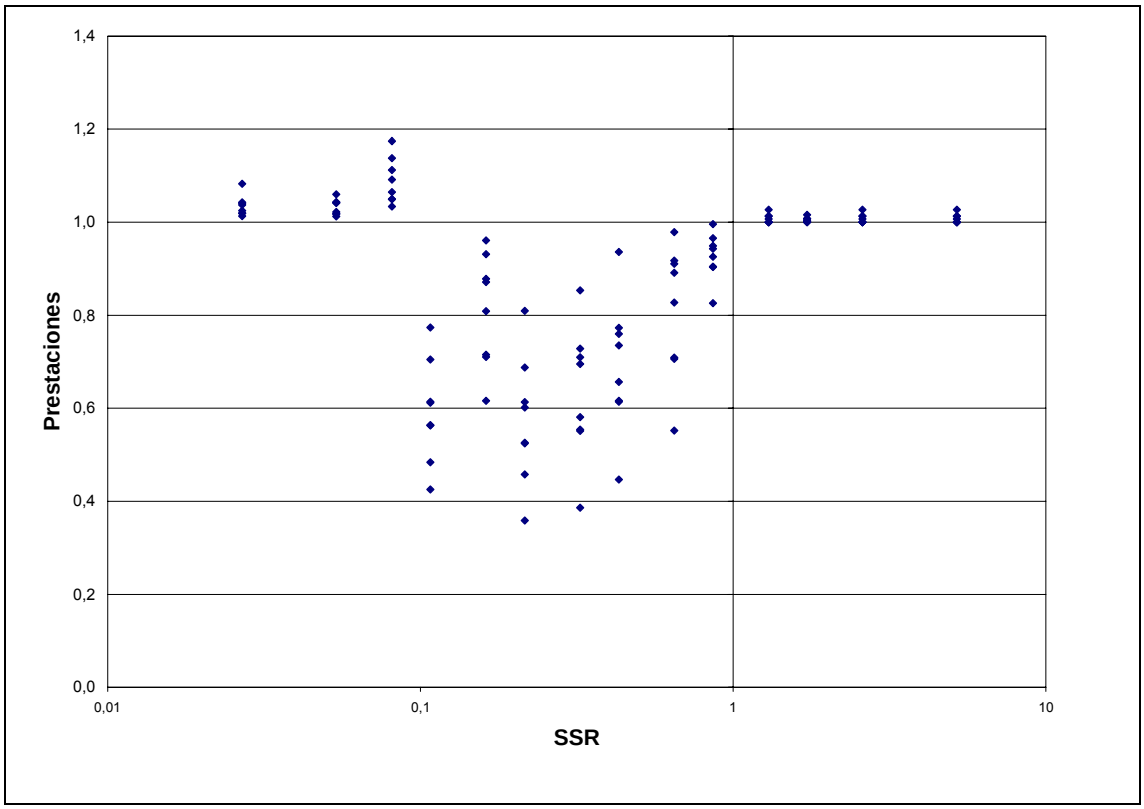


Figura 93. Gráfico de dispersión de $\pi = U_{cd}/U_{led}$ frente a SSR.

5.3.1 Análisis de los factores hardware para el espacio ϵ_1

La Tabla 76 muestra el resultado del análisis de los experimentos enmarcados en el espacio ϵ_1. En esta tabla se muestra el efecto de cada factor y de las interacciones dobles sobre el valor de π, así como el error normalizado al estimar dicho efecto. Sólo se incluyen interacciones a dos niveles, ya que normalmente interacciones de orden superior no son significativas ni aportan información importante [Romero93]. En esta tabla, el signo positivo del coeficiente estimado para un factor indica que el valor de π será mayor cuando el factor se encuentre a nivel +, mientras que el signo negativo del coeficiente estimado para un factor indica que el valor de π será mayor cuando dicho factor se encuentre a nivel –. Los valores concretos asignados a cada nivel pueden encontrarse en la Tabla 74.

Antes de interpretar los resultados obtenidos es necesario determinar la importancia de cada factor y si su efecto es estadísticamente significativo. Para ello, el gráfico de Pareto muestra el efecto de cada factor ordenado de mayor a menor importancia –Figura 94- y el análisis de la varianza permite discriminar los factores o interacciones que son significativas. La Tabla 77 muestra el resumen del *anova* (*analyis of variance*) realizado, donde valores menores de 0,05 en la columna P-Value indican que el efecto del factor es significativo a un nivel de confianza del 95%.

En el *anova* se ha obtenido un valor de R-Squared del 78,09%, por lo que la mayor parte de la variabilidad presentada por los datos es debida a los factores bajo estudio. De estos factores, sólo dos se muestran claramente significativos (P-Value <0,05): LOC y la interacción BC correspondiente a la interacción entre los factores LS e IRF. En cuanto al factor T_{miss}, aunque no aparece como significativo se encuentra muy próximo a serlo.

Factor	Efecto	Standard error
Media	1,06132	+/- 0,00555311
A:T_{miss}	0,0227578	+/- 0,0111062
B:LS	-0,0164042	+/- 0,0111062
C:IRF	0,0185069	+/- 0,0111062
D:LOC	-0,0557073	+/- 0,0111062
AB	-0,00557558	+/- 0,0104711
AC	0,00476608	+/- 0,0104711
AD	-0,012078	+/- 0,0111062
BC	-0,0302566	+/- 0,0104711
BD	0,005893	+/- 0,0111062
CD	0,0119156	+/- 0,0111062

Tabla 76. Efecto de los factores *hardware* sobre $\pi = U_{cf}/U_{led}$. Espacio ϵ_1.

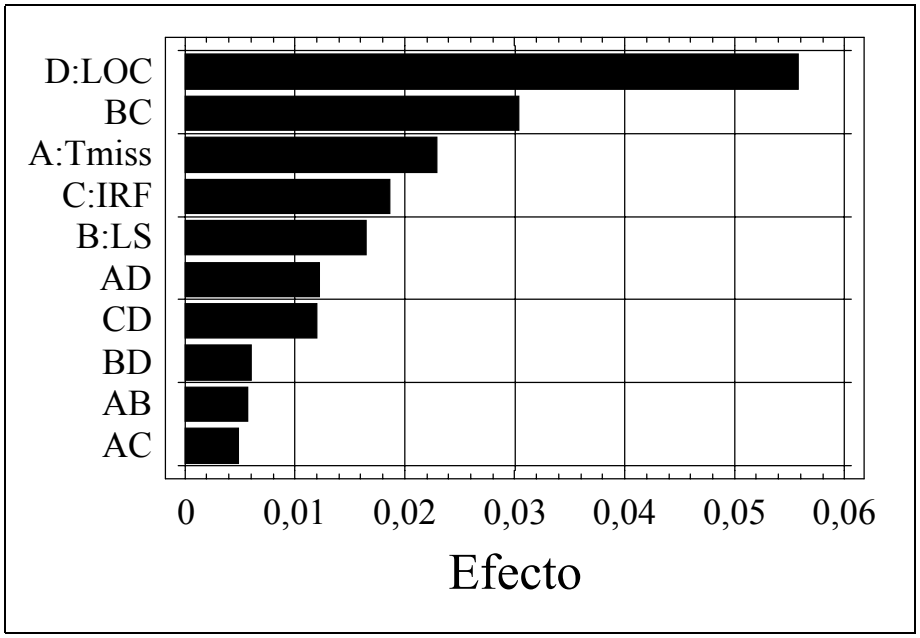


Figura 94. Gráfico de Pareto para $\pi = U_{cd}/U_{led}$. Espacio ϵ_1.

Factor	F-Ratio	P-Value
A:T_{miss}	4,2	0,0612
B:LS	2,18	0,1635
C:IRF	2,78	0,1195
D:LOC	25,16	0,0002
AB	0,28	0,6034
AC	0,21	0,6565
AD	1,18	0,2966
BC	8,35	0,0127
BD	0,28	0,6046
CD	1,15	0,3028

Tabla 77. *Anova* del diseño de experimentos para $\pi = U_{cf}/U_{led}$. Espacio ϵ_1.

Estos tres factores (LOC, T_{miss} y la interacción BC) aparecían también en el análisis del espacio ϵ_1 para prioridades fijas, en el mismo orden y con los mismos signos estimados para su efecto, por lo que se puede concluir que el efecto de los factores *hardware*, para los sistemas enmarcados en el espacio ϵ_1, es el mismo independientemente del tipo de planificador utilizado, y sólo existirán mínimas diferencias en cuanto a la intensidad de los efectos, pero no en lo relativo a la mejora o pérdida de *prestaciones* ni a las causas de la variación en los valores de π. Por ello, se remite al lector al apartado 4.3.1 para conocer las causas de dichos efectos.

Esta similitud en el comportamiento de las *prestaciones* del uso estático de *locking cache* ya se mostró en el análisis de los factores *software*, donde no se apreciaron diferencias estadísticamente significativas en las *prestaciones* ofrecidas por el uso estático de *locking cache* al emplear un planificador de prioridades fijas o el planificador EDF de prioridades dinámicas. Es importante recordar que en los análisis realizados, las *prestaciones* ofrecidas por la *locking cache* están referidas a las *prestaciones* ofrecidas por las *cache*s convencionales. Por tanto, al afirmar que no existe diferencia en las *prestaciones* obtenidas por un planificador u otro, realmente se dice que se obtiene la misma ganancia o pérdida de *prestaciones* al utilizar de forma estática la *locking cache* independientemente del planificador empleado. Pero no puede extraerse ninguna conclusión sobre las *prestaciones*, en términos absolutos, que se obtendrán al emplear un determinado planificador.

5.3.2 Análisis de los factores hardware para el espacio ϵ_2

La Tabla 78 muestra el resultado del análisis de los experimentos enmarcados en el espacio ϵ_2. En esta tabla se muestra el efecto de cada factor y de las interacciones dobles sobre el valor de π, así como el error normalizado al estimar dicho efecto.

Antes de interpretar los resultados obtenidos es necesario determinar la importancia de cada factor y si su efecto es estadísticamente significativo. Para ello, el gráfico de Pareto muestra el efecto de cada factor ordenado de mayor a menor importancia –Figura 95- y el análisis de la varianza permite discriminar los factores o interacciones que son significativas. La Tabla 79 muestra el resumen del *anova* realizado, donde valores menores de 0,05 en la columna P-Value indican que el efecto del factor es significativo a un nivel de confianza del 95%.

Factor	Efecto	Standard error
Media	0,723795	+/- 0,0202438
A:T_{miss}	-0,113827	+/- 0,0404877
B:LS	0,115189	+/- 0,0404877
C:IRF	0,0585121	+/- 0,0404877
D:LOC	-0,0558704	+/- 0,0404877
AB	-0,00714989	+/- 0,0400724
AC	0,00417461	+/- 0,0400724
AD	-0,0182926	+/- 0,0404877
BC	-0,089702	+/- 0,0400724
BD	0,00516119	+/- 0,0404877
CD	0,0367295	+/- 0,0404877

Tabla 78. Efecto de los factores *hardware* sobre $\pi = U_{cf}/U_{led}$. Espacio ϵ_2.

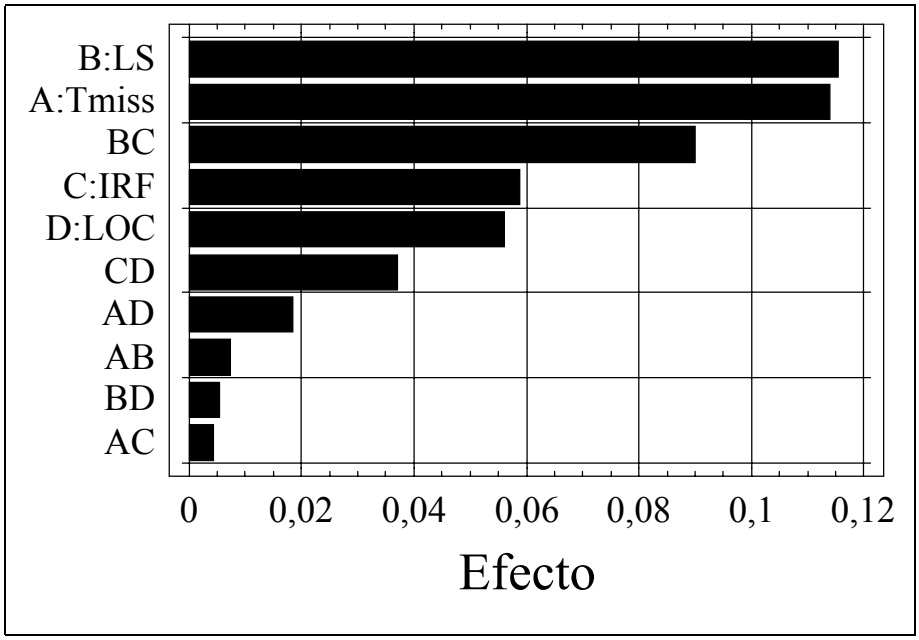


Figura 95. Gráfico de Pareto para $\pi = U_{cd}/U_{led}$. Espacio ϵ_2.

Factor	F-Ratio	P-Value
A:T_{miss}	7,9	0,0073
B:LS	8,09	0,0067
C:IRF	2,09	0,1553
D:LOC	1,9	0,1744
AB	0,03	0,8592
AC	0,01	0,9175
AD	0,2	0,6536
BC	5,01	0,0302
BD	0,02	0,8991
CD	0,82	0,3691

Tabla 79. *Anova* del diseño de experimentos para $\pi = U_{cf}/U_{led}$. Espacio ϵ_2.

En el *anova* se ha obtenido un valor de R-Squared del 37,8%, lo que significa que la mayor parte de la variabilidad observada en los valores de π se debe a factores y características que no han sido consideradas en este análisis, tal como sucedió en el análisis de los experimentos enmarcados en el espacio ϵ_2 para planificador de prioridades fijas. Aun así, tres factores se muestran claramente significativos (P-value <0,005) por lo que su influencia en el comportamiento de π no será determinante pero sí importante. Estos tres factores son LS, T_{miss} y la interacción LS*IRF.

En el análisis realizado para este mismo espacio pero utilizando un planificador de prioridades fijas aparecían también estos tres factores, con el mismo signo para sus coeficientes estimados, por lo que el efecto sobre el comportamiento de la *locking cache* frente a las *cache*s convencionales será el mismo que el descrito en el correspondiente apartado 4.3.2. Sin embargo, existe una ligera diferencia. El efecto del factor IRF se muestra como no significativo al utilizar un planificador con prioridades dinámicas mientras que sí aparecía como significativo cuando se utilizaba un planificador de prioridades fijas. La diferencia entre ambos casos no es muy significativa, ya que el factor IRF ha descendido una posición en la lista de relevancia, y lo que es más importante, el signo del efecto estimado en ambos casos es el mismo, por lo que el efecto producido por IRF será en la misma dirección, aunque con diferente intensidad. Esta ligera diferencia es debida, posiblemente, al empleo de la utilización real en lugar de la utilización planificable para evaluar las *prestaciones* ofrecidas por la *locking cache*.

5.3.3 Análisis de los factores hardware para el espacio ϵ_3

La Tabla 80 muestra el resultado del análisis de los experimentos enmarcados en el espacio ϵ_3. En esta tabla se muestra el efecto de cada factor y de las interacciones dobles sobre el valor de π, así como el error normalizado al estimar dicho efecto.

Antes de interpretar los resultados obtenidos es necesario determinar la importancia de cada factor y si su efecto es estadísticamente significativo. Para ello, el gráfico de Pareto muestra el efecto de cada factor ordenado de mayor a menor importancia –Figura 96- y el análisis de la varianza permite discriminar los factores o interacciones que son significativas. La Tabla 81 muestra el resumen del *anova* realizado, donde valores menores de 0,05 en la columna P-Value indican que el efecto del factor es significativo a un nivel de confianza del 95%.

Factor	Efecto	Standard error
Media	1,0061	+/- 0,000830598
A:T_{miss}	0,00420962	+/- 0,0016612
B:LS	-0,00387562	+/- 0,0016612
C:IRF	-0,00658288	+/- 0,0016612
D:LOC	-0,00282363	+/- 0,0016612
AB	-0,00141994	+/- 0,00143864
AC	-0,00188506	+/- 0,00143864
AD	-0,00081013	+/- 0,0016612
BC	0,0119001	+/- 0,00143864
BD	0,00126363	+/- 0,0016612
CD	-0,00172063	+/- 0,0016612

Tabla 80. Efecto de los factores *hardware* sobre $\pi = U_{cf}/U_{led}$. Espacio ϵ_3.

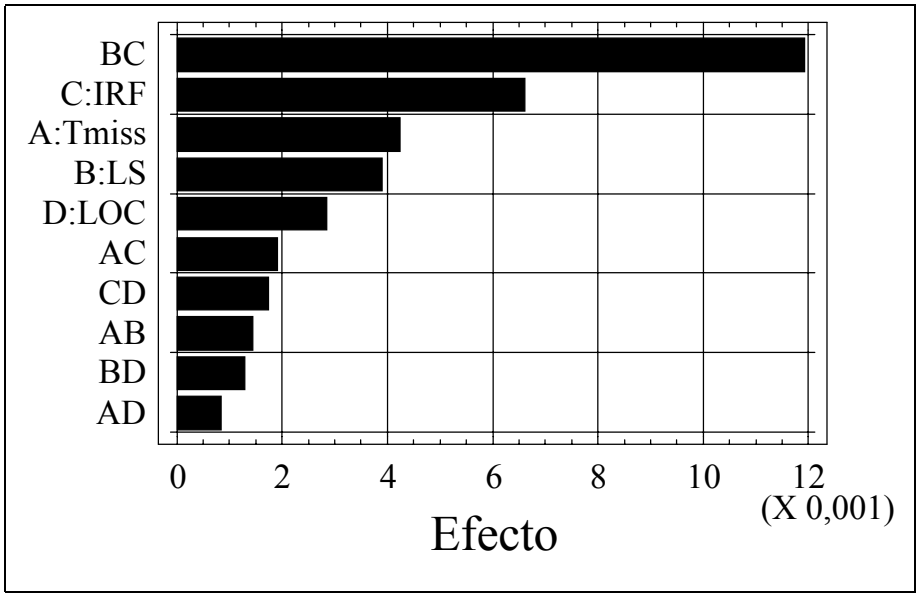


Figura 96. Gráfico de Pareto para $\pi = U_{cd}/U_{led}$. Espacio ϵ_3.

Factor	F-Ratio	P-Value
A:T_{miss}	6,42	0,0193
B:LS	5,44	0,0297
C:IRF	15,7	0,0007
D:LOC	2,89	0,1039
AB	0,97	0,3349
AC	1,72	0,2042
AD	0,24	0,6308
BC	68,42	0
BD	0,58	0,4553
CD	1,07	0,3121

Tabla 81. *Anova* del diseño de experimentos para $\pi = U_{cf}/U_{led}$. Espacio ϵ_3.

En el *anova* se ha obtenido un valor de R-Squared del 84,18%, lo que significa que los cuatro factores incluidos en el análisis, junto con sus interacciones, explican la gran parte de la variabilidad de los datos. En concreto, 4 efectos aparecen con un P-Value menor de 0,05, indicando que son estadísticamente significativos, especialmente y con diferencia respecto a los demás, la interacción correspondiente a los factores LS*IRF.

Comparando el resultado de este análisis con los resultados correspondientes al mismo espacio para prioridades fijas, las diferencias son evidentes ya que no sólo no coincide el número de efectos significativos, sino que tampoco coincide la importancia de los factores ni en muchos casos el signo de cada efecto.

Estas diferencias son debidas, tal como se comentó en el análisis de los factores *software* correspondiente al espacio D de prioridades dinámicas, al empleo de la utilización real como métrica, en lugar de la utilización planificable. La utilización planificable es independiente del número de ejecuciones que se produzcan en el sistema, ya que considera únicamente aquella ejecución, para cada tarea, que representa el peor tiempo de respuesta. Sin embargo, la utilización exacta o real puede considerarse que representa la media de los distintos tiempos de respuesta que presenta cada ejecución de las tareas.

La aparición del factor IRF como muy significativo viene motivada porque el hiperperiodo de los sistemas que no sufren interferencia extrínseca contempla una sola ejecución de cada tarea, por lo que su utilización real o exacta coincide con la máxima utilización planificable. Sin embargo, en los sistemas con IRF = 1 (nivel +) el hiperperiodo contemplará múltiples ejecuciones de las tareas del sistema, alejando, a la baja, la utilización real del sistema de la máxima utilización planificable.

Una vez identificada la razón de la aparición del factor IRF, éste y la interacción con LS pueden ser eliminados, quedando como factores más importantes T_{miss} y LS, el primero de ellos con coeficiente positivo y el segundo con coeficiente negativo. Tanto el orden como el signo de estos dos efectos coincide con el obtenido en el análisis del espacio ϵ_3 para prioridades fijas, por lo que es aceptable concluir que el comportamiento de π frente a los factores *hardware* será el mismo independientemente del tipo de planificador empleado.

5.4 Conclusiones del capítulo

En este capítulo se han presentado diferentes análisis estadísticos con el objetivo de determinar las causas del comportamiento de las *prestaciones* del uso estático de *locking cache* con planificador de prioridades dinámicas EDF. Las *prestaciones* se han definido de forma relativa,

como la ganancia o pérdida de *prestaciones* sufrida al utilizar una *locking cache* frente al empleo de *cache*s convencionales con un comportamiento dinámico y adaptativo, pero no determinista. Esta pérdida o ganancia de *prestaciones* se ha evaluado mediante la relación $\pi = U_{cd}/U_{led}$ donde U_{cd} es la utilización del sistema al emplear *cache* convencional y U_{led} es la utilización del sistema al emplear *locking cache*. En este caso concreto, las utilizaciones empleadas corresponden con la utilización real o exacta, obtenida de la simulación de la ejecución del sistema o estimada por el algoritmo genético.

Todos los análisis realizados han mostrado que el comportamiento de π, para aquellos sistemas con tamaño de *cache* menor que el tamaño de las tareas, es el mismo cuando se utiliza un planificador EDF como cuando se utiliza un planificador de prioridades fijas. Esta igualdad en el comportamiento de π se ha verificado para el efecto del factor SSR, para el valor medio de π, y para el efecto de los factores *software* y *hardware*. Sólo se han apreciado ligeras diferencias, nunca significativas, en algunos estadísticos de las muestras estudiadas, y en la intensidad, que no en la tendencia, del efecto producido por algunos factores, tanto *software* como *hardware*.

En el caso de los sistemas en los que el tamaño total de las tareas es menor que el tamaño de la *cache* se han apreciado diferencias importantes en el comportamiento de π cuando se utiliza un planificador EDF frente al comportamiento de π cuando se utiliza un planificador de prioridades fijas. Sin embargo, estas diferencias sólo se manifestaban en aquellos sistemas en los que el nivel de interferencia extrínseca –factor IRF- era elevado, mientras que para el otro grupo de sistemas los valores de π eran exactamente los mismos independientemente de la política de planificación utilizada. Además de existir diferencias significativas entre los resultados para los dos tipos de planificadores, también aparecen diferencias muy importantes para los dos tipos de sistemas –IRF = 1 e IRF = 0- cuando se utilizan prioridades dinámicas como política de planificación. Aunque esta diferenciación entre los dos tipos de sistemas también aparecía cuando se empleaban prioridades fijas, su importancia era mucho menor que la mostrada en prioridades dinámicas.

Estas diferencias o comportamiento particular de un determinado tipo de sistema era consecuencia de la métrica utilizada para evaluar los sistemas. La imposibilidad de obtener, a partir de los resultados del algoritmo genético, la máxima utilización planificable o la utilización en el peor de los casos, motivaba el empleo de la utilización real o exacta como parámetro de evaluación. Aunque para los sistemas sin interferencia extrínseca el empleo de una u otra métrica era indiferente ya que ambas coincidían, para los sistemas con un alto grado de interferencia extrínseca la utilización real no representa la utilización del sistema en el peor caso, sino la media de la utilización de todas y cada una de las ejecuciones realizadas durante la simulación o evaluación del sistema. Esta situación no representa ningún problema si fuera cierta en los dos esquemas de *cache* que se pretende comparar. Sin embargo, la utilización real sigue coincidiendo con la utilización planificable para los sistemas ejecutados sobre una *locking cache*, independientemente del nivel de interferencia extrínseca existente, ya que los contenidos de la *cache* son estables desde antes de iniciar la ejecución, y por tanto no hay variación en el tiempo de respuesta de las tareas entre las diferentes ejecuciones o instanciaciones. Por contra, al ejecutar el sistema sobre una *cache* convencional, la existencia de fallos obligatorios durante la primera ejecución de cada tarea, pero no en las posteriores, hace que la utilización real y la utilización en el peor caso difieran en mayor medida cuanto mayor es el número de ejecuciones o instanciaciones consideradas.

Así pues, puede concluirse que las diferencias apreciadas en la ganancia o pérdida de *prestaciones* sufrida por la *locking cache* al emplear un determinado planificador no son reales, sino motivadas por la utilización de métricas diferentes, pudiéndose afirmar que el comportamiento del uso estático de *locking cache*s es independiente del tipo de planificador utilizado, bien prioridades fijas, bien prioridades dinámicas EDF.

Por último, indicar que de los análisis realizados no puede extraerse ninguna conclusión sobre cuál de los dos tipos de planificadores realiza un mejor aprovechamiento de los recursos del sistema o cuáles son las ventajas que ofrece uno sobre otro, ya que para ello habría sido

necesario comparar las utilizaciones o *prestaciones* del mismo esquema de *cache* al utilizar las dos políticas. Esta comparación o análisis no se ha realizado ya que la determinación de qué política de planificación es mejor no es un objetivo de este trabajo.

La conclusión que sí puede extraerse de los análisis realizados es que el comportamiento, en cuanto a prestaciones, de la *locking cache* es independiente del tipo de planificador utilizado, por lo que la decisión de utilizar uno u otro deberá tomarla el diseñador del sistema en función de otros parámetros.

Capítulo 6

Análisis de prestaciones para uso dinámico de locking cache

El trabajo presentado a continuación muestra el análisis del comportamiento de las prestaciones de la *locking cache*, cuando ésta se emplea de forma dinámica y con una política de planificación basada en prioridades fijas. Las prestaciones del uso dinámico de *locking cache* se definen como la ganancia o pérdida de prestaciones frente a la utilización de *cache*s convencionales. La métrica empleada para el cálculo de prestaciones ha sido la máxima utilización planificable o utilización en el peor caso, obtenida a partir de los tiempos de respuesta estimado por el algoritmo genético para el caso de la *locking cache*, o de los tiempos de respuesta obtenidos por las simulaciones de las *cache*s convencionales.

Con los análisis realizados se ha podido identificar el efecto que producen diferentes características *software* y *hardware* de los sistemas sobre las prestaciones ofrecidas por el uso dinámico de la *locking cache*. Puesto que los experimentos empleados en estos análisis, así como los propios análisis, son los mismos que se utilizaron al estudiar las prestaciones del uso estático de *locking cache*, se ha podido comparar con sumo detalle las diferencias entre ambos esquemas de *cache*. Esto ha permitido identificar las situaciones concretas en que la utilización de un esquema de *locking cache* será más interesante, siempre desde el punto de vista de las prestaciones.

6.1 Descripción del proceso

Al igual que en los dos capítulos anteriores, el proceso que se desea analizar se divide en dos partes: una variable dependiente o salida del proceso bajo estudio, y un conjunto de variables independientes, también llamadas factores o variables explicativas, que corresponden con diferentes características del sistema y que determinarán el comportamiento y los valores que tomará la variable dependiente. A continuación se describen, de forma breve, estas variables.

6.1.1 Variable dependiente

En los análisis que se realizarán en los apartados siguientes, la salida del proceso o variable dependiente corresponde con la ganancia o pérdida de prestaciones presentada por el uso dinámico de *locking cache* frente a las *cache*s convencionales, siempre con una política de planificación basada en prioridades fijas.

Esta ganancia o pérdida de prestaciones se define como el cociente entre la utilización planificable obtenida por la *cache* convencional y la utilización planificable obtenida por el uso dinámico de *locking cache*:

$$\textbf{\textit{Prestaciones}}\ \pi = U_{cf}/U_{ldf}$$

Si la variable *prestaciones* π toma el valor de 1 indicará que la utilización obtenida con ambos esquemas de *cache* es la misma, y que por tanto no existe pérdida o ganancia de *prestaciones* al utilizar una u otra alternativa. Si la variable *prestaciones* toma valores superiores a 1, es decir, $U_{cf} > U_{ldf}$, indicará que existe ganancia de *prestaciones* al utilizar la *cache* con bloqueo, ya que se considera que una menor utilización representa mejores *prestaciones*. Por contra, si la variable *prestaciones* toma valores inferiores a 1, es decir, $U_{cf} < U_{ldf}$, indicará que el empleo de una *cache* con bloqueo implica una pérdida de *prestaciones* respecto a la utilización de las arquitecturas de *cache* convencionales. En el texto que sigue, todas las referencias a las *prestaciones* se realizan desde el punto de vista de la *locking cache* mientras no se indique explícitamente lo contrario. Por tanto, cuando se diga que un sistema mejora o empeora sus

prestaciones, implícitamente se está diciendo que el uso dinámico de *locking cache* presenta una mejora o empeoramiento respecto del uso de *cache*s convencionales.

6.1.2 Variables explicativas

Las variables independientes o factores utilizados en los análisis realizados en este capítulo se dividen en dos grupos:

A. Factores *hardware*:

- Tamaño de la *cache* (CS): expresado en KiloBytes, indica el número total de bytes que pueden cargarse en *cache*. Para la *cache* con bloqueo no incluye el buffer temporal. Aunque no se considerará explícitamente en los diferentes análisis, sí participará en los mismos de forma implícita, ya que la definición de sistema incluía este tamaño como una característica propia de cada experimento.
- Tamaño de línea (LS): tamaño de una línea de *cache*, expresado normalmente en bytes o alternativamente en instrucciones.
- Tiempo de acierto (T_{hit}): tiempo en ejecutar una instrucción desde memoria *cache* o desde el buffer temporal, medido en ciclos de procesador.
- Tiempo de fallo (T_{miss}): tiempo necesario para llevar un bloque desde memoria principal a la *cache* o al buffer temporal, medido en ciclos de procesador.

B. Factores *software*:

- Tamaño_sistema (SS)
- Número_tareas (TN)
- Ejecutadas_tamaño (ES)
- Tamaño_prioritaria (PS)
- Ejecutadas_total (ET)
- Ratio_tamaño_sistema (SSR)

La lista de factores empleado es el resultado de aplicar la matriz de correlación a un conjunto mayor de variables explicativas, y de la eliminación de aquéllas que presentaban indicios de relación lineal, tal como se describió en el capítulo 4.

6.2 Análisis de los factores software

El primer paso en el análisis del comportamiento de las *prestaciones* es un estudio del efecto de la relación entre el tamaño de la *cache* y el tamaño del sistema (SSR = CS/SS). El gráfico de dispersión de π frente a SSR se muestra en la Figura 97, donde se puede observar una clara tendencia en la distribución de los datos. Para identificar mejor esta tendencia, se han reunido los valores de π en siete grupos, tal como se ha hecho en el análisis del uso estático de *locking cache*. La Tabla 83 muestra los principales estadísticos de cada grupo, y la Figura 98 su representación gráfica en función de SSR. La Tabla 82 muestra los límites inferior y superior de cada grupo.

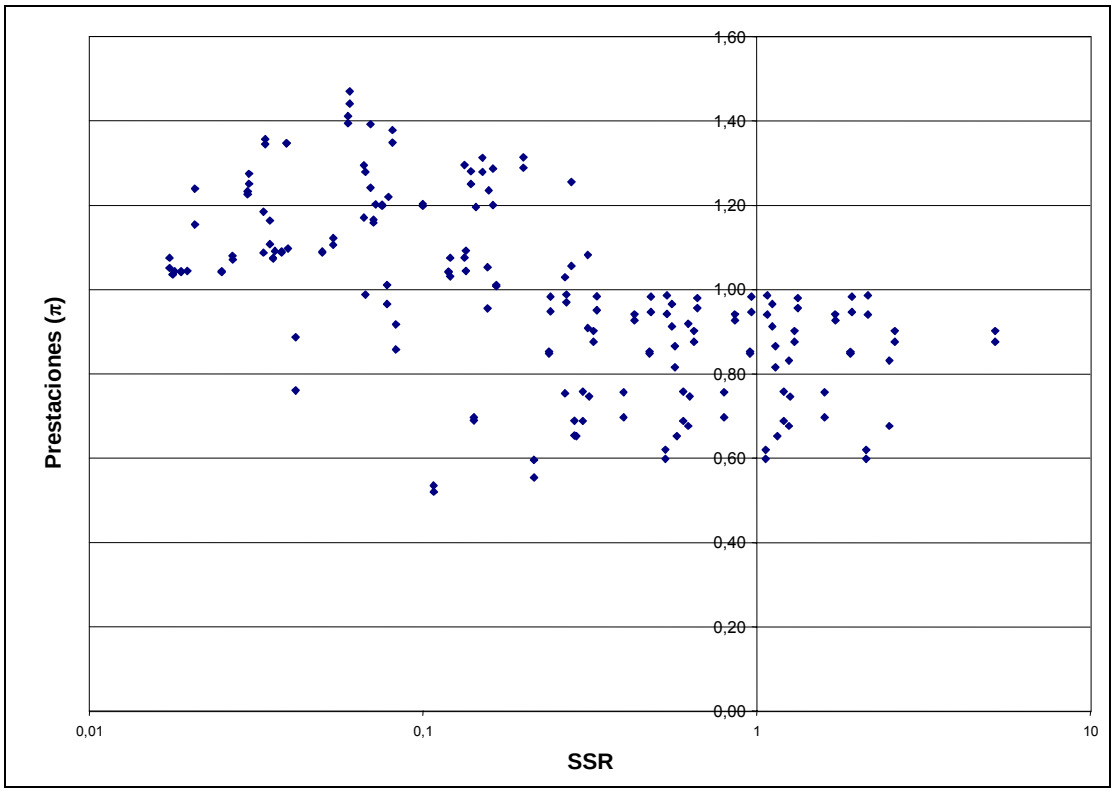


Figura 97. Gráfico de dispersión de $\pi = U_{cd}/U_{ldf}$ frente a SSR.

Grupo	Tamaño *cache*	Límite inferior	Límite superior
1	1	0	0,0313
2	2	0,0313	0,0525
3	4	0,0525	0,1150
4	8	0,1150	0,2400
5	16	0,2400	0,4900
6	32	0,4900	1,0000
7	64	1,0000	∞

Tabla 82. Valores límite de SSR para la agrupación de $\pi = U_{cf}/U_{ldf}$.

La distribución de los datos en la Figura 98 puede dividirse en tres zonas. La primera zona, a la izquierda, donde la mayoría de los datos se encuentran por encima de 1, con una ligera tendencia ascendente según aumenta el valor de SSR. La segunda zona, en el centro de la gráfica, con una clara pendiente negativa, alta variabilidad y con valores tanto por encima como por debajo del 1. La tercera zona, en el extremo derecho de la gráfica, casi sin pendiente, con una variabilidad menor y todos los datos por debajo del valor 1. Pese a que son sólo tres las zonas o espacios discriminados visualmente, los análisis posteriores se realizarán para los mismos cuatro espacios que se han utilizado en los capítulos anteriores. Tres razones motivan esta decisión:

A. La utilización de los mismos espacios permitirá comparar las *prestaciones* obtenidas por el uso dinámico frente a las *prestaciones* obtenidas por el uso estático de forma más precisa.

B. El valor de SSR igual a 1 es un límite muy significativo, por lo que conviene estudiar de forma separada los valores que se encuentran a uno y otro lado de este límite.

C. Las primeras aproximaciones para la obtención de los modelos de regresión lineal múltiple mostraron que el comportamiento de los espacios C y D, correspondientes a la zona derecha de la Figura 98 es demasiado diferente como para emplear un único modelo.

A continuación se describe el análisis realizado para cada espacio -los límites de cada espacio se muestran en la Tabla 84-. Para cada espacio se ha realizado un análisis de datos pareados entre los valores de π para uso estático y para uso dinámico -ambos con planificador de prioridades fijas-, para verificar si existe o no diferencia en las *prestaciones* medias obtenidas. Puesto que se han utilizado los mismos experimentos en ambos usos de la *locking cache*, este análisis pareado es completamente factible. A continuación se presentará el modelo de regresión lineal múltiple obtenido para cada espacio, y se realizará una breve comparación con el modelo del espacio correspondiente para uso estático.

Grupo	1	2	3	4	5	6	7
Total datos	18	18	28	28	28	26	36
Media	1,110	1,127	1,157	1,057	0,887	0,839	0,832
Mediana	1,061	1,091	1,200	1,064	0,918	0,871	0,871
Des. Std	0,091	0,155	0,238	0,223	0,147	0,124	0,125
Intervalo de. Conf	0,042	0,072	0,088	0,082	0,054	0,048	0,041
Nº de éxitos ($\pi \geq 1$)	18	16	22	21	4	0	0
Probabilidad de éxito	100%	88,9%	78,6%	75,0%	14,3%	0,0%	0,0%
Máximo	1,275	1,357	1,471	1,314	1,255	0,986	0,986
Mínimo	1,036	0,761	0,520	0,554	0,653	0,598	0,598
1 ^{er} cuartil	1,043	1,088	1,082	0,995	0,756	0,750	0,734
3er cuartil	1,208	1,179	1,308	1,258	0,983	0,942	0,941
Coeficiente de curtosis	-1,102	0,890	1,832	-0,080	-0,061	-1,034	-1,003
Coeficiente de asimetría	0,880	-0,340	-1,287	-0,830	0,165	-0,583	-0,579

Tabla 83. Principales estadísticos de la agrupación de $\pi = U_{cf}/U_{idf}$ en función de SSR.

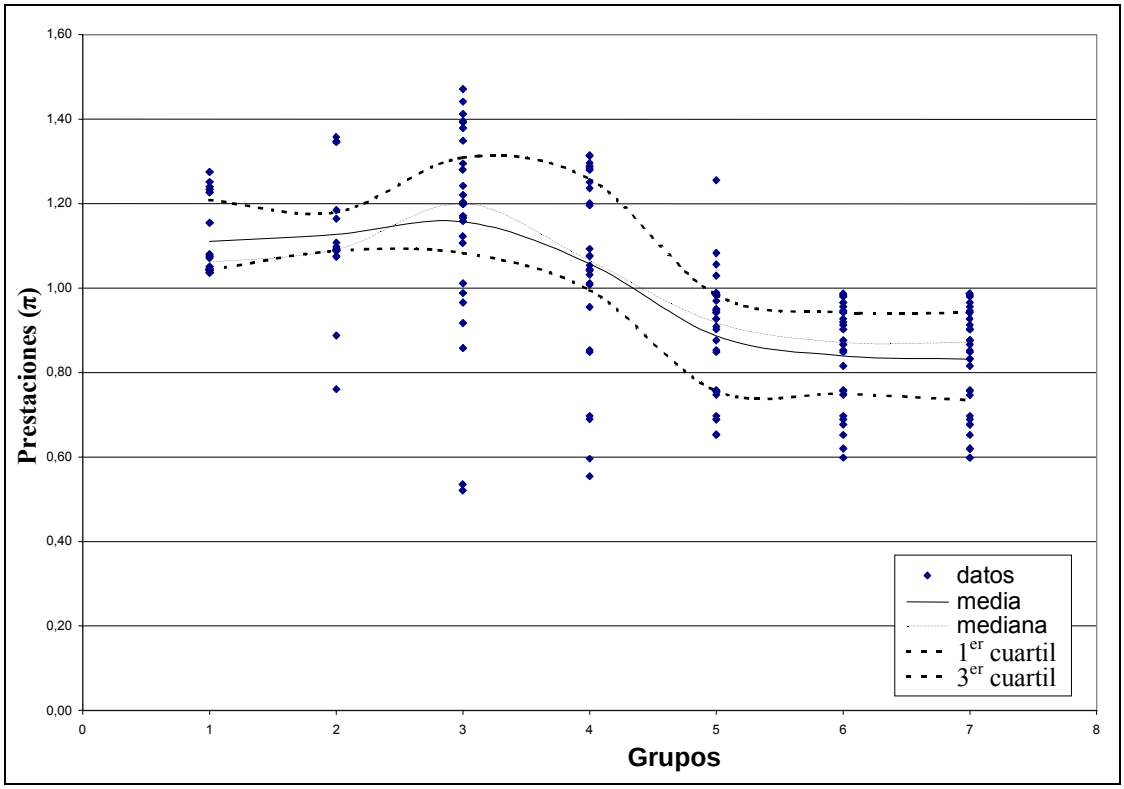


Figura 98. Agrupación de U_{cd}/U_{ldf} en función de SSR.

Espacio	Límite inferior	Límite superior	Total datos	Asimetría Standard	Curtosis Standard
A	0	≤0,065	42	-4,82	8,42
B	>0,065	<0,3	61	-0,68	-2,12
C	≥0,3	<1	43	-2,12	-0,06
D	≥1	∞	36	2,4	-0,23

Tabla 84. Valores límites de SSR para la creación de los cuatro espacios de $\pi = U_{cf}/U_{ldf}$.

6.2.1 Modelo de los factores software para el Espacio A

La Tabla 85 muestra los principales estadísticos para el análisis de datos pareados (π estático - π dinámico), incluyendo el intervalo de confianza al 95%. Dicho intervalo no contiene el valor cero, por lo que existen diferencias significativas en las medias de los datos de ambos usos de la *locking cache*. Puesto que la media es negativa, indica que los valores de π para el uso dinámico son mayores, es decir, se obtendrán, de media, mejores *prestaciones* al utilizar de forma dinámica la *locking cache*. Es importante recordar que el valor de π indica la ganancia o pérdida de *prestaciones* al utilizar la *locking cache* frente al uso de *cache*s convencionales.

Estadístico	Valor
Nº de experimentos	42
Media	-0,108095
Mediana	-0,05
Varianza	0,0187085
Desviación típica	0,136779
Mínimo	-0,52
Máximo	0,03
Primer cuartil	-0,15
Tercer cuartil	-0,02
Asimetría estándar	-4,0732
Curtosis estándar	2,06321
Intervalo de confianza	[-0,150719;-0,0654718]

Tabla 85. Principales estadísticos del análisis de datos pareados de $\pi = U_{cf}/U_{lef}$ y $\pi = U_{cf}/U_{ldf}$. Espacio A

Antes de desarrollar el modelo de regresión correspondiente a los datos enmarcados en el espacio A se ha verificado la normalidad de estos. Por coherencia con los análisis realizados anteriormente, y para facilitar las comparaciones, se ha empleado el logaritmo de π. La Figura 99 muestra la representación en papel probabilístico normal de log (π), y la Tabla 86 los principales estadísticos del espacio. La representación en papel probabilístico muestra ciertas curvaturas en los datos así como un cúmulo de valores que indican que la normalidad de la muestra es cuestionable. Sin embargo, los valores de asimetría y curtosis se encuentran dentro de los límites (-2;+2) correspondientes a una muestra con una distribución normal. Pese a la incertidumbre sobre la normalidad de los datos, se ha desarrollado el correspondiente modelo de regresión lineal múltiple, prestando especial atención a su validación mediante el análisis de los residuos, con objeto de determinar si la asunción de la normalidad de log (π) es aceptable o por el contrario debe desecharse el modelo obtenido.

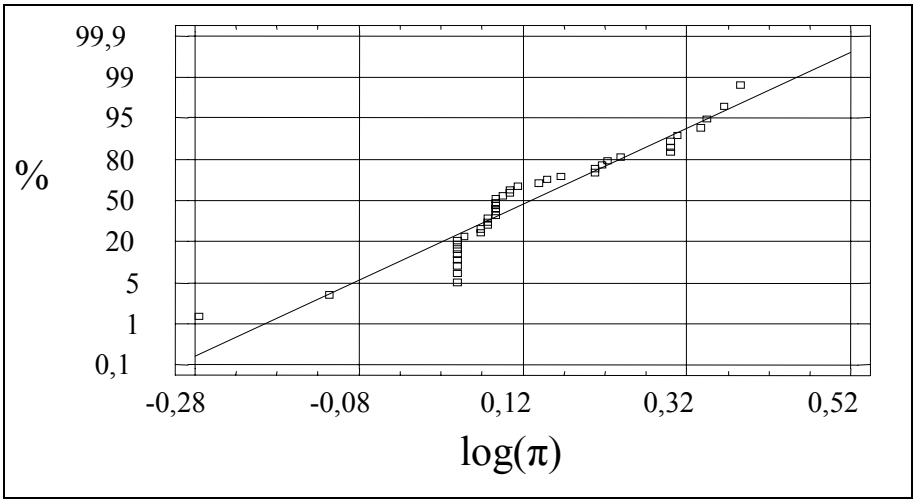


Figura 99. Representación de log ($\pi = U_{cd}/U_{ldf}$) en papel probabilístico normal. Espacio A.

Estadístico	**valor**
Nº de experimentos	42
Media	0,12988
Mediana	0,0861777
Varianza	0,0169643
Desviación típica	0,130247
Mínimo	-0,274437
Máximo	0,385262
Primer cuartil	0,0487902
Tercer cuartil	0,215111
Asimetría estándar	-0,514712
Curtosis estándar	1,71054

Tabla 86. Principales estadísticos de log ($\pi = U_{cf}/U_{ldf}$). Espacio A

El modelo obtenido para $\log(\pi) = \log (U_{cf}/U_{lff})$ es el siguiente:

$$\log(\pi) = 0{,}305994 - 4{,}02203*LOC + 6{,}77513E\text{-}7*ET + 1{,}53864*LOC*TN - 3{,}38502*LOC*SSR + 0{,}010485*LOC*PS - 0{,}00362697*LOC*SS - 0{,}00208394*TN*ES + 7{,}1197*TN*SSR + 0{,}0720573*ES*SSR - 0{,}00391257*SSR*SS - 0{,}00000495733*SSR*ET - 0{,}0010964*PS + 0{,}0000548492*TN*PS - 28{,}5712*SSR + 0{,}0203683*SSR*PS + 0{,}00141202*LOC*ES + 0{,}000521117*SS - 0{,}000103911*TN*SS$$

El modelo incorpora 18 variables independientes, con una R-Squared del 96,73%, lo que quiere decir que el modelo explica la casi totalidad del comportamiento del logaritmo de π. El estadístico Durbin-watson, con un valor de 2,47, muy por encima de 1,4 indica que no existe ningún indicio de autocorrelación en los residuos. La Tabla 87 muestra los coeficientes, el error normalizado y la significación de cada variable independiente, y la Tabla 88 el análisis de la varianza del modelo. En estas tablas se puede observar que todas las variables independientes tienen un P-value inferior a 0,05, lo que indica que son estadísticamente significativas con un nivel de confianza del 95%. Así mismo, el modelo presenta un P-value de 0,000 que al ser menor que 0,01 indica que existe una relación estadísticamente significativa al 99% de confianza entre las variables explicativas y la variable dependiente.

La Tabla 89 muestra la importancia en el modelo de cada uno de los factores. Para poder interpretar con mayor facilidad el modelo, se presentan ordenados de mayor valor de F-Ratio (más significativo) a menor valor de F-Ratio (menos significativo), junto con el coeficiente estimado para cada factor.

Factor	Valor Estimado	Standard Error	T-Statistic	P-Value
Constante	0,305994	0,149796	2,04274	0,0527
LOC	-4,02203	0,634182	-6,34207	0
ET	6,78E-07	1,04E-07	6,5367	0
LOC*TN	1,53864	0,239056	6,43632	0
LOC*SSR	-3,38502	0,952459	-3,55398	0,0017
LOC*PS	0,010485	0,00165519	6,33458	0
LOC*SS	-0,00362697	0,00058443	-6,20598	0
TN*ES	-0,00208394	0,00034046	-6,12095	0
TN*SSR	7,1197	1,28141	5,55614	0
ES*SSR	0,0720573	0,0100119	7,1972	0
SSR*SS	-0,00391257	0,00104674	-3,73788	0,0011
SSR*ET	-4,9573E-06	5,21E-07	-9,50961	0
PS	-0,0010964	0,00023599	-4,64605	0,0001
TN*PS	5,4849E-05	2,1194E-05	2,58793	0,0164
SSR	-28,5712	8,55735	-3,33879	0,0029
SSR*PS	0,0203683	0,00524581	3,88278	0,0008
LOC*ES	0,00141202	0,0003026	4,66631	0,0001
SS	0,00052112	0,00010445	4,98936	0
TN*SS	-0,00010391	2,2946E-05	-4,52846	0,0002

Tabla 87. Detalles del modelo de log ($\pi = U_{cf}/U_{ldf}$). Espacio A.

Fuente	Suma de cuadrados	Grados de libertad	Mean Square	F-Ratio	P-Value
Modelo	0,672815	18	0,0373786	37,83	0,0000
Residual	0,0227231	23	0,00098796		
Total (Corr.)	0,695538	41			

Tabla 88. Análisis de la varianza del modelo de log ($\pi = U_{cf}/U_{ldf}$). Espacio A.

Fuente	F-Ratio	Coeficiente estimado
LOC*SS	266,18	-0,00362697
TN*SSR	141,8	7,1197
SSR*ET	135,86	-4,9573E-06
LOC*PS	33,02	0,010485
ES*SSR	29,11	0,0720573
TN*SS	20,51	-0,00010391
TN*ES	19,18	-0,00208394
LOC*SSR	12,07	-3,38502
LOC*TN	10,38	1,53864
SS	4,5	0,000521117
ET	3,97	6,78E-07
PS	2,57	-0,0010964
LOC	0,74	-4,02203
SSR	0,36	-28,5712
LOC*ES	0,33	0,00141202
TN*PS	0,27	5,48492E-05
SSR*SS	0,1	-0,00391257
SSR*PS	0,05	0,0203683

Tabla 89. Impacto de los factores (orden descendente) sobre el modelo de log ($\pi = U_{cf}/U_{ldf}$). Espacio A.

La validación del modelo se ha realizado comparando los valores observados frente a los calculados por el modelo -Figura 100- y verificando la normalidad de los residuos -Figura 101, Figura 102 y Tabla 90-. En esta última figura se aprecia que un pequeño número de residuos se alejan de la recta, situación que aparece reflejada en la tabla de estadísticos correspondientes, en concreto en el valor de la curtosis estándar, que se aleja de forma importante del valor de +2. La Figura 102 muestra el histograma de frecuencias de los residuos, donde se aprecia claramente que los datos son leptocúrtidos. Pese a esto, la validez del modelo puede ser aceptada para los objetivos perseguidos en este análisis por varios motivos: son muy pocos los residuos que se desvían de la recta que marca la normalidad, y muy posiblemente están relacionados con los valores de log (π) que también se desviaban de la recta normal; la magnitud del error es –relativamente- baja frente a los valores predichos por el modelo; no se ha detectado ninguna tendencia en la representación de los residuos del modelo frente a las variables independientes; por último, el objetivo perseguido con este análisis no es encontrar un modelo que ofrezca predicciones exactas, sino una estimación de cuáles son los factores que determinan el comportamiento de las *prestaciones* del uso dinámico de *locking cache*.

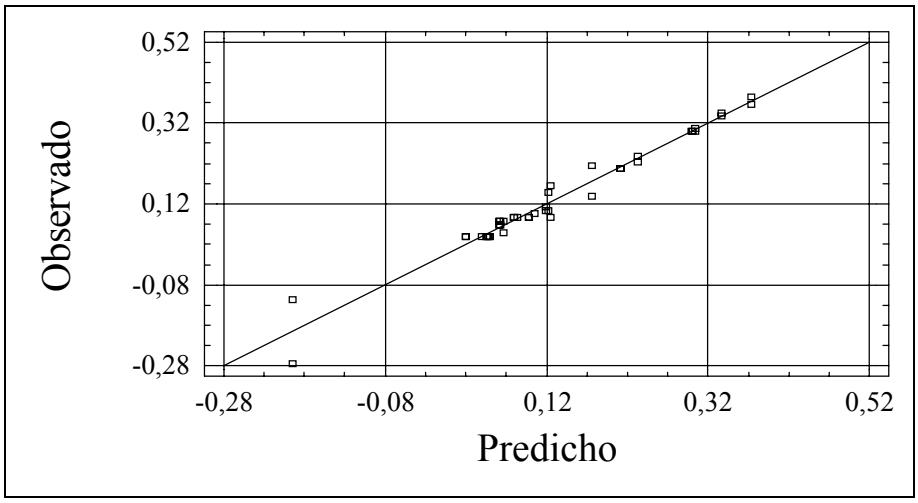


Figura 100. Representación de los valores de log ($\pi = U_{cd}/U_{ldf}$) observados frente a los predichos por el modelo para el espacio A.

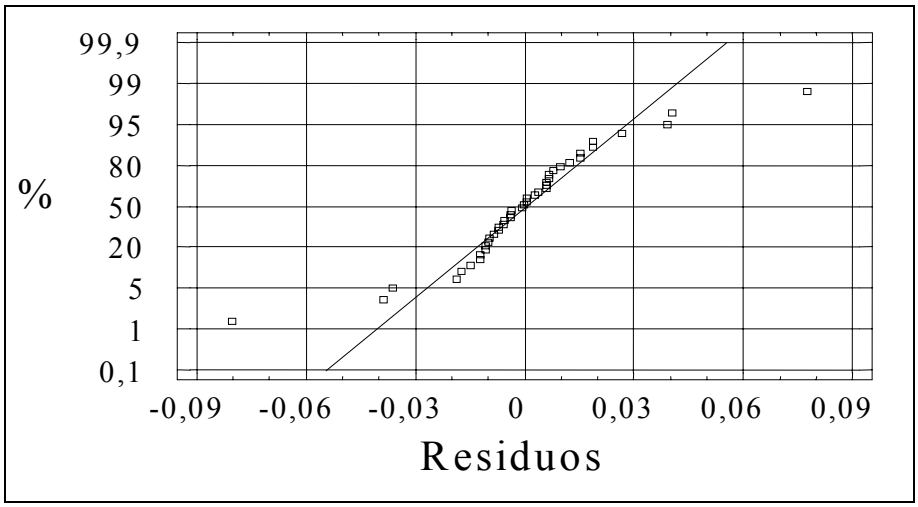


Figura 101. Representación en papel probabilístico normal de los residuos del modelo para el espacio A.

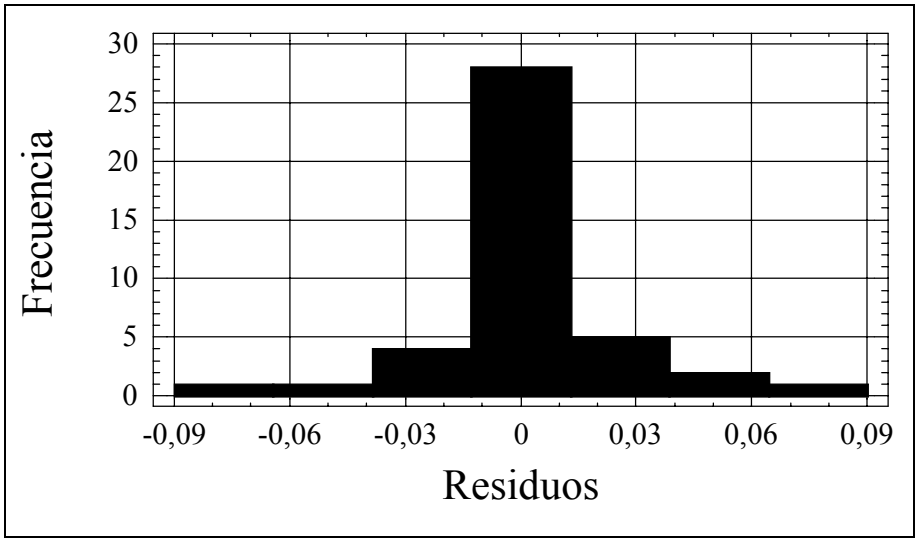


Figura 102. Histograma de frecuencias de los residuos correspondientes al modelo del espacio A.

Número de datos	42
Media	1,40476E-9
Varianza	0,000554222
Desviación estándar	0,0235419
Mínimo	-0,0803675
Máximo	0,0775355
Asimetría estándar	-0,0794929
Curtosis estándar	6,44294

Tabla 90. Principales estadísticos de los residuos del modelo de log ($\pi = U_{cf}/U_{ldf}$). Espacio A.

Las diferencias con el modelo desarrollado para el mismo espacio del uso estático de *locking cache* son múltiples, tanto en los factores que aparecen como en el signo de los coeficientes, llamando especialmente la atención los siguientes casos:

- No aparece, ni de forma individual ni en ninguna interacción, el factor IRF, indicando que el número de expulsiones que se produzcan en el sistema no afecta a las *prestaciones* obtenidas. Esto era de esperar, ya que tanto al utilizar *cache*s convencionales como *locking cache*, hay que recargar parte o incluso la totalidad de los contenidos de *cache* tras una expulsión, por lo que ninguno de los dos esquemas de *cache* presenta ventaja respecto al otro por motivo del número de expulsiones que se produzcan durante la ejecución del sistema.
- Además de aparecer de forma individual el factor LOC, las interacciones donde este factor aparece muestran, para el coeficiente estimado, signos contrarios, por lo que el efecto del nivel de localidad de las tareas sobre las *prestaciones* del sistema será el contrario que el observado para el uso estático de *locking cache*. Este cambio de signo no se produce para la interacción LOC*SSR.
- Desaparece de forma individual el factor TN, pero se mantienen las interacciones con otros factores, aunque también con el signo cambiado, excepto para la interacción TN*SSR.
- Aparecen de forma individual las variables explicativas ET y PS.

- Existen diferencias importantes en el efecto que producen la mayoría de las variables explicativas y sus interacciones.

La variable explicativa SSR aparece en ambos modelos con el mismo signo en todos los casos, por lo que el efecto será de forma general el mismo, es decir, mejores *prestaciones* cuanto más próximos se encuentren los tamaños de la *cache* y del sistema (siempre dentro de los límites que definen el espacio A).

La Tabla 91 muestra la comparación de los diferentes factores que aparecen como significativos en ambos modelos. Además se muestra la variación entre los coeficientes de los modelos como el cociente entre el valor estimado para uso estático y el valor estimado para uso dinámico. Cuanto más cercano a 1 sea este cociente, menor diferencia existe. Además, el signo positivo de este cociente indica que para ambos modelos, el coeficiente tiene el mismo signo, mientras que un valor negativo para el cociente indica signos opuestos en los coeficientes estimados en cada modelo.

Modelo para uso estático			**Modelo para uso dinámico**			**Coef. Estático / Coef. Dinámico**
Fuente	**F-Ratio**	**Coeficiente estimado**	**Fuente**	**F-Ratio**	**Coeficiente estimado**	**Variación coeficientes**
IRF*ES	0,14	-0,00013639	ET	3,97	6,78E-07	No aplicable
IRF*SSR	18,05	1,99859	LOC	0,74	-4,02203	No aplicable
TN	24,25	-0,408003	PS	2,57	-0,0010964	No aplicable
			SSR*SS	0,1	-0,00391257	No aplicable
LOC*ES	2,88	-0,00086597	LOC*ES	0,33	0,00141202	-0,613
LOC*PS	1,92	-0,00122654	LOC*PS	33,02	0,010485	-0,117
LOC*SS	91,39	0,00060773	LOC*SS	266,18	-0,00362697	-0,168
LOC*SSR	3,27	-3,97403	LOC*SSR	12,07	-3,38502	1,174
LOC*TN	20,94	-0,13865	LOC*TN	10,38	1,53864	-0,090
SS	43,34	-0,00068913	SS	4,5	0,000521117	-1,322
SSR	6,77	-19,2279	SSR	0,36	-28,5712	0,673
SSR*ES	26,86	0,0574987	SSR*ES	29,11	0,0720573	0,798
SSR*ET	54,94	-4,49E-06	SSR*ET	135,86	-4,96E-06	0,906
SSR*PS	16,58	0,00905212	SSR*PS	0,05	0,0203683	0,444
TN*ES	5,51	0,00010776	TN*ES	19,18	-0,00208394	-0,052
TN*PS	0,06	-5,06E-05	TN*PS	0,27	5,48E-05	-0,922
TN*SS	3,06	0,00016139	TN*SS	20,51	-0,00010391	-1,553
TN*SSR	26,19	3,3879	TN*SSR	141,8	7,1197	0,476

Tabla 91. Comparación de los modelos obtenidos para log ($\pi = U_{cf}/U_{lef}$) y log ($\pi = U_{cf}/U_{ldf}$). Espacio A.

El comportamiento de los sistemas enmarcados en el espacio A, para uso dinámico de *locking cache* es similar al presentando por el uso estático, pero exclusivamente en relación al factor

SSR. Para el resto de características *software* del sistema, el efecto que producen en el comportamiento de log (π) es de forma general el contrario. Esta inversión en el efecto de las diferentes variables explicativas es debido a que el uso dinámico de *locking cache* se comporta, dentro del espacio A, de un modo asimilable al de las *cache*s convencionales, por lo que al comparar el uso dinámico con el uso estático se puede considerar que se está comparando el uso estático con *cache*s convencionales, pero desde una perspectiva invertida. Es decir, los factores que mejoraban las *prestaciones* del uso estático de *locking cache* empeoraban las *prestaciones* desde el punto de vista de las *cache*s convencionales y al ser similar el comportamiento del uso dinámico y las *cache*s convencionales, podemos decir que el efecto será el contrario para el uso dinámico de *locking cache*.

6.2.2 Modelo de los factores software para el Espacio B

La Tabla 92 muestra los principales estadísticos para el análisis de datos pareados (π estático - π dinámico), incluyendo el intervalo de confianza al 95%. Dicho intervalo no contiene el valor cero, por lo que existen diferencias significativas en las medias de los datos de ambos usos de la *locking cache*. Puesto que la media es negativa, indica que los valores de π para el uso dinámico son mayores, es decir, se obtendrán, de media, mejores *prestaciones* al utilizar de forma dinámica la *locking cache*. Es importante recordar que el valor de π indica la ganancia o pérdida de *prestaciones* al utilizar la *locking cache* frente al uso de *cache*s convencionales, por lo que el análisis de datos pareados nos indica que el uso dinámico de *locking cache* presentará una menor pérdida de *prestaciones* o una mayor ganancia de *prestaciones* que el uso estático.

Estadístico	**Valor**
Nº de experimentos	61
Media	-0,229672
Mediana	-0,23
Varianza	0,0373399
Desviación típica	0,193235
Mínimo	-0,6
Máximo	0,29
Primer cuartil	-0,38
Tercer cuartil	-0,1
Asimetría estándar	0,190981
Curtosis estándar	-0,585106
Intervalo de confianza	[-0,279162;-0,180182]

Tabla 92. Principales estadísticos del análisis de datos pareados de $\pi = U_{cf}/U_{lef}$ y $\pi = U_{cf}/U_{ldf}$. Espacio B.

Antes de desarrollar el modelo de regresión correspondiente a los datos enmarcados en el espacio B, se ha verificado la normalidad de estos. Por coherencia con los análisis realizados anteriormente, y para facilitar las comparaciones, se ha empleado el logaritmo de π. La Figura 103 muestra la representación en papel probabilístico normal de Log(π), y la Tabla 93 los principales estadísticos del espacio. La representación en papel probabilístico normal muestra que aunque los datos se aproximan bastante a una recta, existen ciertas curvaturas que podrían invalidar el modelo desarrollado para estos datos. La existencia de este problema y la posible causa se discuten a continuación, una vez desarrollado el modelo.

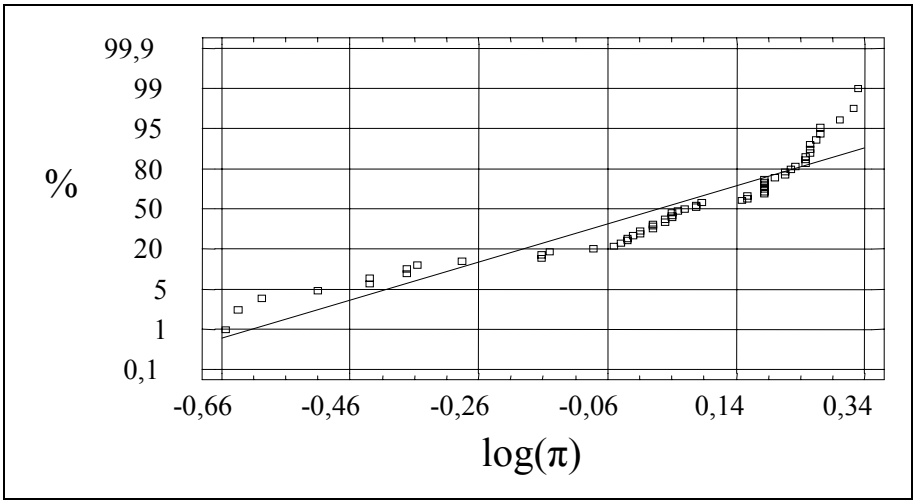


Figura 103. Representación de log ($\pi = U_{cd}/U_{ldf}$) en papel probabilístico normal. Espacio B.

Estadístico	**Valor**
Nº de experimentos	61
Media	0,0202642
Mediana	0,0582689
Varianza	0,0639176
Desviación típica	0,252819
Mínimo	-0,653926
Máximo	0,329304
Primer cuartil	-0,040822
Tercer cuartil	0,215111
Asimetría estándar	-3,75791
Curtosis estándar	1,03229

Tabla 93. Principales estadísticos de log ($\pi = U_{cf}/U_{ldf}$). Espacio B.

El modelo obtenido para $\log(\pi) = \log (U_{cf}/U_{lff})$ es el siguiente:

$$\log(\pi) = 2,13669 - 2,84056*LOC - 0,411135*TN + 0,00443513*ES - 1,60835*SSR + 0,00133737*PS - 0,00125054*SS - 3,29567E\text{-}7*ET - 0,00117215*LOC*ES - 0,00357349*LOC*PS + 0,00192384*LOC*SS - 0,000186192*TN*PS + 0,000265782*TN*SS + 1,7756E\text{-}7*SSR*ET$$

El modelo incorpora 13 variables independientes, con una R-Squared de 83,46, inferior a todos los valores obtenidos en los modelos anteriores -e inferior a los dos modelos que se desarrollarán posteriormente-. Este –relativamente- bajo valor puede deberse a varias razones, como la exclusión de variables explicativas importantes, o la cuestionable normalidad de los datos bajo estudio. Antes de describir con detalle el modelo, se verificará su corrección mediante la comparación de los valores observados frente a los calculados por el modelo -Figura 104- y verificando la normalidad de los residuos -Figura 105, y Tabla 94-.

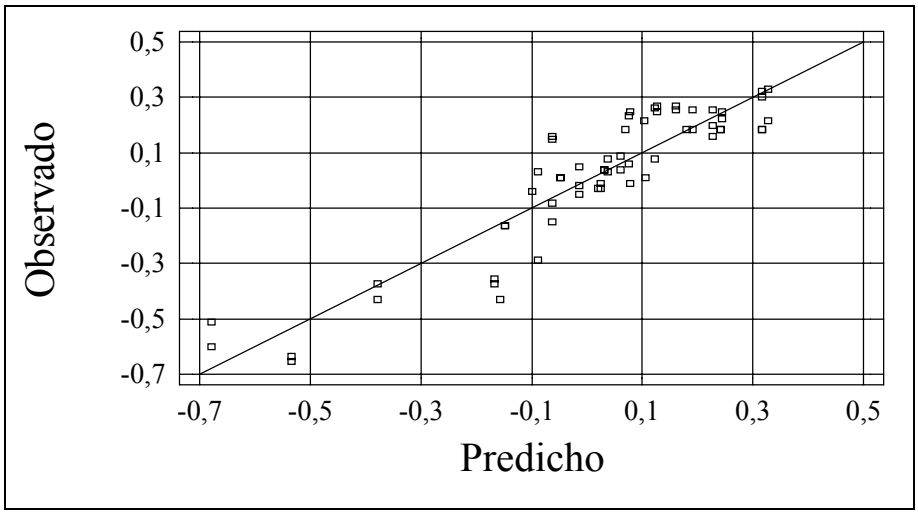


Figura 104. Representación de los valores de log ($\pi = U_{cd}/U_{ldf}$) observados frente a los predichos por el modelo para el espacio B.

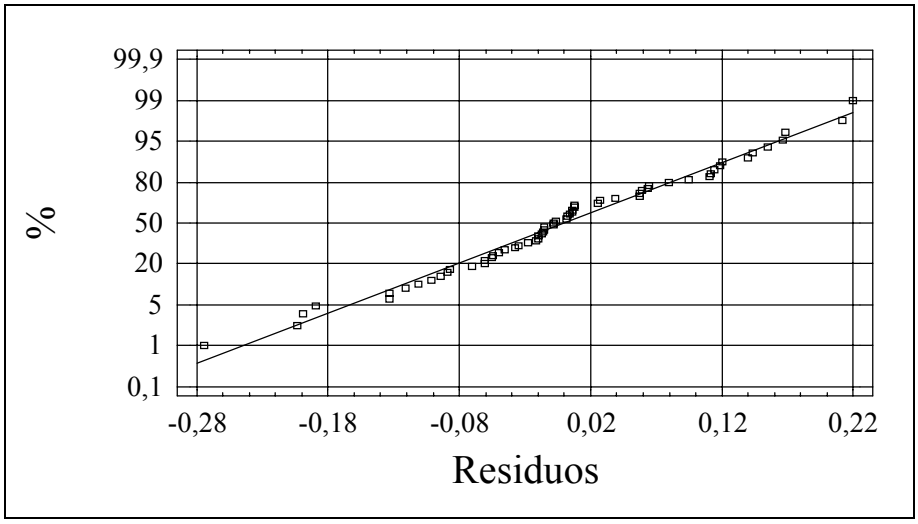


Figura 105. Representación en papel probabilístico normal de los residuos del modelo para el espacio B.

Número de datos	61
Media	2,95082E-9
Varianza	0,0105734
Desviación estándar	0,102827
Mínimo	-0,274577
Máximo	0,219839
Asimetría estándar	-0,372983
Curtosis estándar	0,223571

Tabla 94. Principales estadísticos de los residuos del modelo de log ($\pi = U_{cf}/U_{ldf}$). Espacio B.

Aparentemente el modelo parece ser correcto, pero tanto la Figura 104 como los valores del mínimo y máximo error confirman que se ajusta -relativamente- poco a la realidad.

Esta falta de ajuste puede deberse al error cometido en las estimaciones del tiempo de respuesta de las tareas realizadas por el algoritmo genético, error que se incorpora a la utilización planificable del uso dinámico por el modo en que se ha calculado dicha utilización. Este error se mostró en el capítulo 3, y se vio que no era constante, por lo que puede darse el caso de que sistemas con las mismas características ofrezcan ratios U_{cf}/U_{ldf} diferentes, por lo que el modelo intentará ajustarse a todos ellos, pero sin acercarse realmente a ninguno, ni por tanto a la realidad. La Figura 106 muestra el gráfico de dispersión en función de SSR del error cometido al estimar U_{ldf} (error estimado) para cada uno de los cuatro espacios.

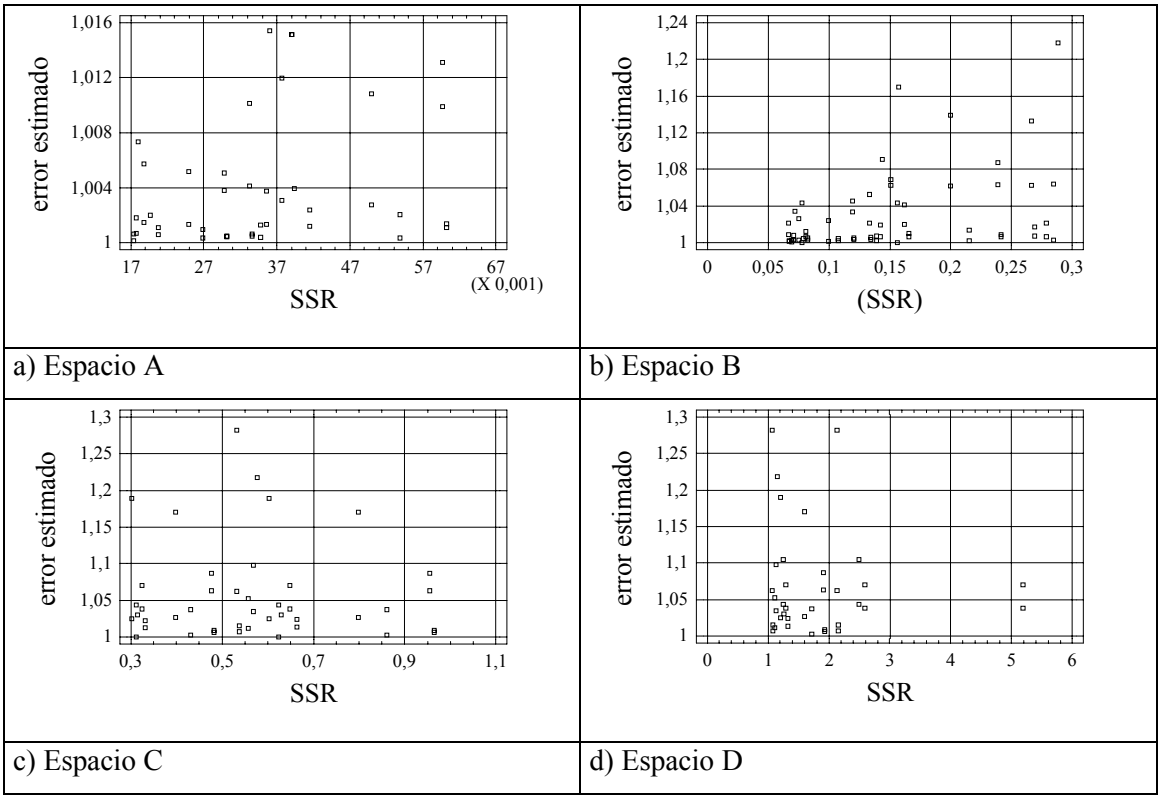


Figura 106. Gráfica de dispersión del error cometido al estimar la utilización de la *locking cache*, frente a SSR.

En la Figura 106-b), correspondiente al error cometido al estimar la utilización de los experimentos enmarcados en el espacio B, se puede apreciar claramente una tendencia ascendente en la variabilidad del error estimado al aumentar el valor de SSR. Sin embargo, el incremento del error cometido no es constante, sino sólo el rango del error cometido, por lo que la misma *carga*, con diferentes tamaños de *cache*, puede sufrir el mismo error o diferente error al estimar su utilización. Puesto que el error cometido no es una de las variables explicativas, ni presenta una distribución uniforme como en el resto de los espacios, el modelo es incapaz de considerarlo o incluirlo en las estimaciones realizadas, por lo que el modelo no es válido, ni tampoco cualquier conclusión que se extraiga del mismo.

Tres alternativas se plantean para la obtención de un modelo para los resultados del espacio B:

A. Utilizar los resultados de la simulación de la *locking cache* en lugar de los estimados, ya que éstos no incorporan ningún tipo de error.

B. Incluir el error cometido al estimar la utilización como una variable explicativa más. El error puede cuantificarse mediante la comparación de los valores estimados por el algoritmo genético frente a los simulados de la *locking cache*.

C. Subdividir el espacio B en espacios más pequeños con el objetivo de mantener uniforme el error cometido al estimar la utilización.

La utilización de los valores simulados, tal como se plantea en las dos primeras alternativas, es desaconsejable debido a la poca fiabilidad de los resultados de la simulación -hablando en términos generales- ya que no existe garantía total de que se haya simulado el peor caso. Por otro lado, es posible que las conclusiones obtenidas a partir de los resultados de la simulación no sean extrapolables a los resultados estimados, que son los únicos que el diseñador de sistemas de tiempo real puede utilizar para sus decisiones.

La última alternativa reduciría el número de datos en cada modelo, probablemente por debajo del umbral necesario para llevar a cabo el análisis con garantías.

Lo expuesto anteriormente recomienda declarar la imposibilidad de obtener, con los experimentos desarrollados en este trabajo, un modelo del comportamiento de los sistemas enmarcados en el espacio B.

6.2.3 Modelo de los factores software para el Espacio C

La Tabla 95 muestra los principales estadísticos para el análisis de datos pareados (π estático - π dinámico), incluyendo el intervalo de confianza al 95%. Dicho intervalo no contiene el valor cero, por lo que existen diferencias significativas en las medias de los datos de ambos usos de la *locking cache*. Puesto que la media es negativa, indica que los valores π para el uso dinámico son mayores, es decir, se obtendrán, de media, mejores *prestaciones* al utilizar de forma dinámica la *locking cache*.

Estadístico	**Valor**
Nº de experimentos	43
Media	-0,0909302
Mediana	-0,09
Varianza	0,041342
Desviación típica	0,203327
Mínimo	-0,5
Máximo	0,39
Primer cuartil	-0,24
Tercer cuartil	0,01
Asimetría estándar	0,525869
Curtosis estándar	-0,0093564
Intervalo de confianza	[-0,153505;-0,0283552]

Tabla 95. Principales estadísticos del análisis de datos pareados de $\pi = U_{cf}/U_{lef}$ y $\pi = U_{cf}/U_{ldf}$. Espacio C.

Antes de desarrollar el modelo de regresión correspondiente a los datos enmarcados en el espacio C, se ha verificado la normalidad de estos. Por coherencia con los análisis realizados anteriormente, y para facilitar las comparaciones, se ha empleado el logaritmo de π. La Figura 107 muestra la representación en papel probabilístico normal de Log(π), y la Tabla 96 los principales estadísticos del espacio.

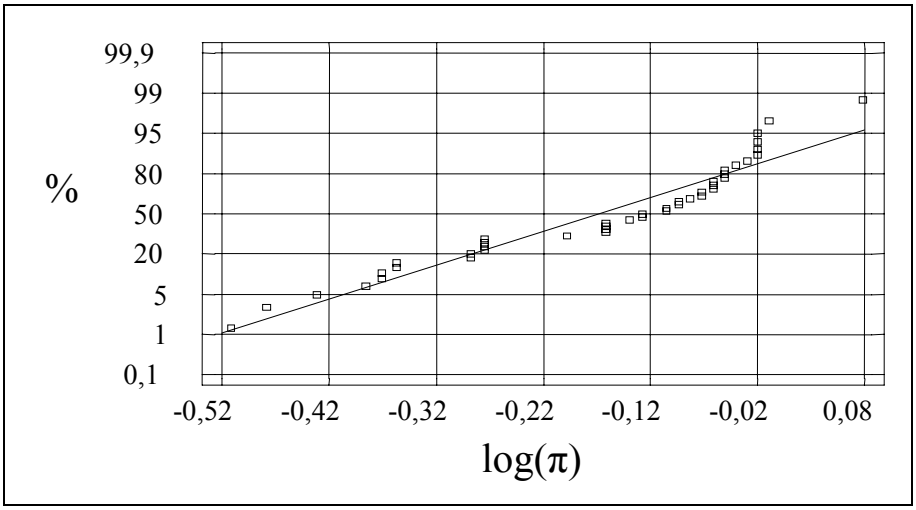


Figura 107. Representación de log ($\pi = U_{cd}/U_{ldf}$) en papel probabilístico normal. Espacio C.

Estadístico	**Valor**
Nº de experimentos	43
Media	-0,168137
Mediana	-0,127833
Varianza	0,0211658
Desviación típica	0,145485
Mínimo	-0,510826
Máximo	0,076961
Primer cuartil	-0,274437
Tercer cuartil	-0,0512933
Asimetría estándar	-1,91667
Curtosis estándar	-0,636053

Tabla 96. Principales estadísticos de log ($\pi = U_{cf}/U_{ldf}$). Espacio C.

El modelo obtenido para $\log(\pi) = \log (U_{cf}/U_{lff})$ es el siguiente:

$$\log(\pi) = -0{,}165506 + 0{,}00638223*ES + 0{,}357221* SSR + 0{,}00154156* PS - 0{,}000886989* SS + 0{,}0000240339* IRF*SS - 1{,}14591E\text{-}8* IRF*ET - 0{,}169577* LOC*TN - 0{,}00110728* LOC*ES - 0{,}00133012* LOC*PS + 0{,}000649172* LOC*SS - 0{,}00121879* TN*ES - 0{,}0736506* TN*SSR - 0{,}000198904* TN*PS + 0{,}000158232* TN*SS - 0{,}000514123* SSR*PS + 0{,}000104233* SSR*SS$$

El modelo incorpora 16 variables independientes, con una R-Squared de 99,3, lo que quiere decir que el modelo explica totalmente el comportamiento del logaritmo de π. El estadístico Durbin-watson, con un valor de 1,7, ligeramente superior a 1,4 indica que no existe ningún indicio de autocorrelación en los residuos. La Tabla 97 muestra los coeficientes, el error normalizado y la significación de cada variable independiente, y la Tabla 98 el análisis de la varianza del modelo. En estas tablas se puede observar que todas las variables independientes tienen un P-value inferior a 0,05, lo que indica que son estadísticamente significativas con un

nivel de confianza del 95%. Así mismo, el modelo presenta un P-value de 0,000 que al ser menor que 0,01 indica que existe una relación estadísticamente significativa al 99% de confianza entre las variables explicativas y la variable dependiente.

Factor	Valor Estimado	Standard Error	T-Statistic	P-Value
Constante	-0,165506	0,0287356	-5,7596	0
ES	0,00638223	0,000190413	33,5179	0
SSR	0,357221	0,0820441	4,35401	0,0002
PS	0,00154156	6,71452E-05	22,9585	0
SS	-0,00088699	2,72E-05	-32,5696	0
IRF*SS	2,40339E-05	2,40006E-06	10,0139	0
IRF*ET	-1,15E-08	1,47E-09	-7,79003	0
LOC*TN	-0,169577	0,00715838	-23,6893	0
LOC*ES	-0,00110728	3,65298E-05	-30,3116	0
LOC*PS	-0,00133012	0,000063608	-20,9112	0
LOC*SS	0,000649172	2,45699E-05	26,4214	0
TN*ES	-0,00121879	4,09081E-05	-29,7934	0
TN*SSR	-0,0736506	0,0105637	-6,97205	0
TN*PS	-1,99E-04	8,35E-06	-23,8168	0
TN*SS	0,000158232	5,42833E-06	29,1494	0
SSR*PS	-5,14E-04	5,08E-05	-10,1187	0
SSR*SS	0,000104233	2,44146E-05	4,2693	0,0002

Tabla 97. Detalles del modelo de log ($\pi = U_{cf}/U_{ldf}$). Espacio C.

Fuente	Suma de cuadrados	Grados de libertad	Mean Square	F-Ratio	P-Value
Modelo	0,882324	16	0,0551452	215,94	0,0000
Residual	0,00663959	26	0,000255369		
Total (Corr.)	0,888964	42			

Tabla 98. Análisis de la varianza del modelo de log ($\pi = U_{cf}/U_{ldf}$). Espacio C.

La Tabla 99 muestra la importancia en el modelo de cada uno de los factores que aparecen en el modelo. Para poder interpretar con mayor facilidad el modelo, se presentan ordenados de mayor valor de F-Ratio (más significativo) a menor valor de F-Ratio (menos significativo), junto con el coeficiente estimado para cada factor.

Fuente	F-Ratio	Coeficiente estimado
ES	1147,73	0,00638223
TN*SS	916,73	0,000158232
LOC*ES	611,46	-0,00110728
LOC*TN	169,92	-0,169577
LOC*SS	125,92	0,000649172
TN*SSR	111,29	-0,0736506
SS	107,12	-0,00088699
SSR*PS	84,51	-5,14E-04
LOC*PS	75,52	-0,00133012
TN*ES	37,56	-0,00121879
PS	28,28	0,00154156
IRF*SS	19,88	2,40339E-05
SSR*SS	18,23	0,000104233
IRF*ET	0,46	-1,15E-08
SSR	0,41	0,357221
TN*PS	0,07	-1,99E-04

Tabla 99. Impacto de los factores (orden descendente) sobre el modelo de log ($\pi = U_{cf}/U_{ldf}$). Espacio C.

La validación del modelo se ha realizado comparando los valores observados frente a los calculados por el modelo -Figura 108- y verificando la normalidad de los residuos -Figura 109 y Tabla 100-.

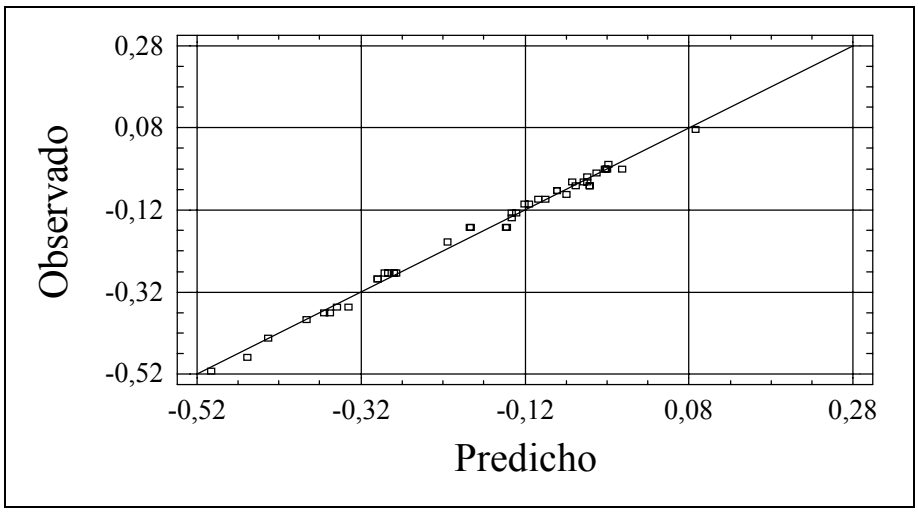


Figura 108. Representación de los valores de log ($\pi = U_{cd}/U_{ldf}$) observados frente a los predichos por el modelo para el espacio C.

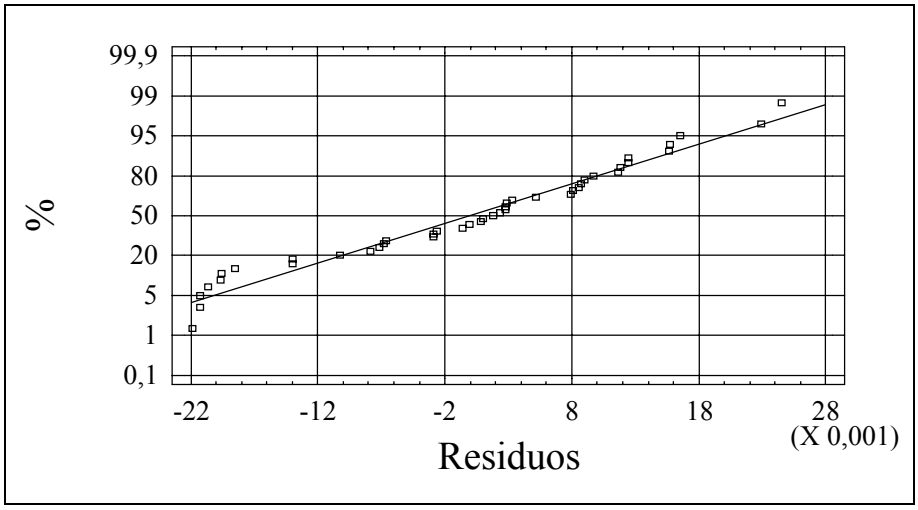


Figura 109. Representación en papel probabilístico normal de los residuos del modelo para el espacio C.

Número de datos	43
Media	3,75116E-9
Varianza	0,000158086
Desviación estándar	0,0125732
Mínimo	-0,0219031
Máximo	0,0245219
Asimetría estándar	-0,593764
Curtosis estándar	-0,962026

Tabla 100. Principales estadísticos de los residuos del modelo de log ($\pi = U_{cf}/U_{ldf}$). Espacio C.

Las diferencias con el modelo desarrollado para el mismo espacio C con uso estático son significativa y notorias en todos los aspectos: número de factores, factores e interacciones que aparecen como significativas, coeficientes, etc. Estas diferencias hacen impracticable una comparación directa entre ambos modelos, por lo que a continuación se realizará una descripción de los aspectos más significativos del modelo correspondiente al espacio C de uso dinámico de *locking cache*:

- El factor más significativo es ES, con coeficiente positivo, por lo que cuanto mayor sea el número de instrucciones en relación con el tamaño de cada tarea, mejores *prestaciones* presentará el uso dinámico de *locking cache*. El mejor aprovechamiento del espacio disponible de *cache* que realiza la *locking cache*, al eliminar la interferencia intrínseca, es la causa de este efecto. Cuantas más veces se ejecuten las instrucciones bloqueadas en *cache*, mayor será la ventaja obtenida. Sin embargo, la existencia de la interacción LOC*ES con coeficiente negativo y muy significativa, compensará para los sistemas que realicen un buen aprovechamiento de la *cache* el efecto manifestado por ES. Al tratarse LOC de una variable cualitativa, es posible comparar los coeficientes estimados para ES y para LOC*ES. Ambos coeficientes son del mismo orden de magnitud, pero el coeficiente de ES es sensiblemente superior, por lo que el efecto de ES se verá reducido, pero no anulado, para los sistemas con LOC igual a 1.
- En el modelo aparece el factor SSR de forma individual, así como la interacción SSR*SS, ambas con coeficiente positivo, pero un valor muy bajo de F-Ratio. Por el contrario, las

interacciones SSR*TN y SSR*PS aparecen mucho más significativas y con coeficientes negativos, lo que compensará e incluso hará tender a la baja el valor de log (π) según aumente el valor de SSR. Esta tendencia a la baja se puede apreciar en la Figura 98, y viene motivada por el mayor coste que sufre la *locking cache* al recargar sus contenidos, ya que recargar un bloque requiere del orden de 4 veces más tiempo que recargar un bloque en una *cache* convencional, cuyo coste es simplemente el valor de T_{miss}. Cuanto mayor es la *cache* en relación con el tamaño del sistema -mayores valores de SSR- más bloques se recargarán, por lo que el efecto de está diferencia será mayor.

- El factor IRF aparece sólo en dos interacciones, y ambas con valores de F-Ratio muy bajos, lo que indica que esta característica del sistema no afectará las *prestaciones* obtenidas, por la misma razón que se comentó para los datos del espacio A.
- El factor LOC aparece, pero sólo en interacciones con otros parámetros. Esto significa que existirán diferencias entre los dos tipos de sistemas definidos por la variable LOC, aunque condicionadas a las características definidas por los factores que forman parte de las interacciones. Estos factores verán reducido el efecto que producen sobre π, pero no llegará a invertirse.
- El número de tareas aparece en varias interacciones, algunas de ellas muy significativas, por lo que al igual que en el caso del uso estático de *locking cache*, el factor TN determinará el efecto que sobre las *prestaciones* tendrán otras variables explicativas.

6.2.4 Modelo de los factores software para el Espacio D

La Tabla 101 muestra los principales estadísticos para el análisis de datos pareados (π estático - π dinámico), incluyendo el intervalo de confianza al 95%. Dicho intervalo no contiene el valor cero, por lo que existen diferencias significativas en las medias de los datos de ambos usos de la *locking cache*. Puesto que la media es positiva, indica que los valores de π para el uso dinámico son menores, es decir, se obtendrán, de media, peores *prestaciones* al utilizar de forma dinámica la *locking cache*. Aparentemente existe una contradicción con el análisis de datos pareados realizado en el capítulo 3 al presentar la propuesta de uso dinámico de *locking cache*, análisis en el que se concluía que el uso dinámico presentaba mejores *prestaciones* que el uso estático. Tal contradicción no existe, ya que el ámbito de aplicación de los dos análisis es distinto, y por tanto la lectura que debe hacerse de cada uno de ellos. El análisis presentado en el capítulo 3 incluía todos los experimentos desarrollados, sin realizar ninguna distinción en función del valor de SSR de cada sistema. Por tanto, la conclusión que se extraía de aquel análisis era la siguiente: de forma general y como media, el uso dinámico de *locking cache* presentará mejores *prestaciones* que el uso estático para todos los sistemas, independientemente de sus características. En cambio, el análisis aquí presentado se limita a aquellos experimentos para los que el tamaño de la *cache* es igual o mayor que el tamaño total del código, por lo que la conclusión que se extrae es la siguiente: de forma general y como media, el uso dinámico de *locking cache* presentará peores *prestaciones* que el uso estático para aquellos sistemas con valores de SSR igual o mayor que 1.

Es importante indicar que el análisis de datos pareados identifica diferencias en la media, por lo que en ningún momento se pueden realizar afirmaciones absolutas del tipo "las *prestaciones serán siempre* mejores...".

Esta pérdida de *prestaciones* es debido al elevado coste de carga de la *cache* al iniciar la ejecución de cada tarea, acrecentado por cada expulsión que sufra cada una de las tareas. Cuanto mayor sea el tamaño de la *cache*, más bloques de memoria principal son susceptibles de ser seleccionados por el algoritmo genético, por lo que mayor será este coste de recarga. Sin embargo, en uso estático, la carga inicial de la *cache* se realiza durante la inicialización del sistema, por lo que no consume tiempo de ejecución de las tareas, y tras cada expulsión sólo es necesario recargar el buffer temporal, cuyo coste de recarga es insignificante frente a los tiempos de ejecución y respuesta de las tareas. Al igual que se vio en el análisis realizado para el espacio C, el número de bloques que se cargan en la *locking cache* cuando se emplea de forma

dinámica representa una muy seria penalización de las prestaciones frente al uso estático de *locking cache* o al empleo de *caches* convencionales. Y, lógicamente, cuanto mayor sea el tamaño de la *cache* más bloques serán susceptibles de ser cargados, por lo que se obtendrán peores prestaciones. Pero este problema es especial cuando todo el código de un sistema cabe en la *locking cache*, ya que al no existir interferencia intrínseca ni extrínseca, ningún bloque es reemplazado de cache, por lo que su recarga en cada ejecución es completamente innecesaria. Por tanto, el empleo dinámico de *locking cache* en esta situación no tiene ningún sentido.

Estadístico	**Valor**
Nº de experimentos	36
Media	0,217778
Mediana	0,165
Varianza	0,0284521
Desviación típica	0,168677
Mínimo	0,01
Máximo	0,54
Primer cuartil	0,075
Tercer cuartil	0,36
Asimetría estándar	1,60422
Curtosis estándar	-0,988713
Intervalo de confianza	[0,136811;0,220029]

Tabla 101. Principales estadísticos del análisis de datos pareados de $\pi = U_{cf}/U_{lef}$ y $\pi = U_{cf}/U_{ldf}$. Espacio D.

Antes de desarrollar el modelo de regresión correspondiente a los datos enmarcados en el espacio D, se ha verificado la normalidad de estos. Por coherencia con los análisis realizados anteriormente, y para facilitar las comparaciones, se ha empleado el logaritmo de π. La Figura 110 muestra la representación en papel probabilístico normal de Log(π), y la Tabla 102 los principales estadísticos del espacio.

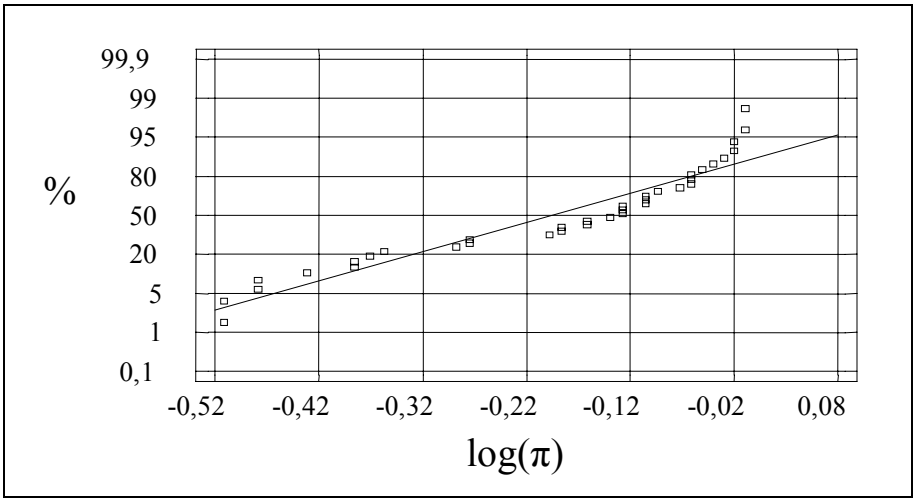


Figura 110. Representación de log ($\pi = U_{cd}/U_{ldf}$) en papel probabilístico normal. Espacio D.

Estadístico	Valor
Nº de experimentos	36
Media	-0,194853
Mediana	-0,133548
Varianza	0,025391
Desviación típica	0,159345
Mínimo	-0,510826
Máximo	-0,0100503
Primer cuartil	-0,322179
Tercer cuartil	-0,0618754
Asimetría estándar	-1,82288
Curtosis estándar	-0,932051

Tabla 102. Principales estadísticos de log ($\pi = U_{cf}/U_{ldf}$). Espacio D.

El modelo obtenido para $\log(\pi) = \log (U_{cf}/U_{lff})$ es el siguiente:

$$\log(\pi) = 2{,}84025 + 4{,}19881*LOC - 0{,}814334* TN + 0{,}013254* ES + 0{,}00213865* PS - 0{,}00234831* SS - 0{,}00000191542* ET + 0{,}0000229934* IRF*SS - 1{,}12984E\text{-}8* IRF*ET - 1{,}99636* LOC*TN - 0{,}00402479* LOC*ES - 0{,}0144211* LOC*PS + 0{,}00547967* LOC*SS - 0{,}000315423* TN*PS + 0{,}000538322* TN*SS + 2{,}23077E\text{-}7* TN*ET$$

El modelo incorpora 15 variables independientes, con una R-Squared de 99,7, lo que quiere decir que el modelo explica totalmente el comportamiento del logaritmo de π. El estadístico Durbin-watson, con un valor de 2,5, superior a 1,4 indica que no existe ningún indicio de autocorrelación en los residuos. La Tabla 103 muestra los coeficientes, el error normalizado y la significación de cada variable independiente, y la Tabla 104 el análisis de la varianza del modelo. En estas tablas se puede observar que todas las variables independientes tienen un P-value inferior a 0,05, lo que indica que son estadísticamente significativas con un nivel de confianza del 95%. Así mismo, el modelo presenta un P-value de 0,000 que al ser menor que 0,01 indica que existe una relación estadísticamente significativa entre las variables explicativas y la variable dependiente al 99% de confianza.

La Tabla 105 muestra la importancia de cada uno de los factores que aparecen en el modelo. Para poder interpretar con mayor facilidad el modelo, se presentan ordenados de mayor valor de F-Ratio (más significativo) a menor valor de F-Ratio (menos significativo), junto con el coeficiente estimado para cada factor.

Factor	Valor Estimado	Standard Error	T-Statistic	P-Value
Constante	2,84025	0,427296	6,64702	0
LOC	4,19881	0,957281	4,38618	0,0003
TN	-0,814334	0,116784	-6,97298	0
ES	0,013254	0,00239249	5,53981	0
PS	0,00213865	1,44E-04	14,8353	0
SS	-0,00234831	0,000248058	-9,46678	0
ET	-1,92E-06	6,27E-07	-3,05374	0,0063
IRF*SS	2,29934E-05	2,15298E-06	10,6798	0
IRF*ET	-1,13E-08	1,25E-09	-9,04257	0
LOC*TN	-1,99636	0,342862	-5,82264	0
LOC*ES	-0,00402479	0,000436998	-9,2101	0
LOC*PS	-0,0144211	0,00231681	-6,22456	0
LOC*SS	0,00547967	0,000821353	6,67152	0
TN*PS	-3,15E-04	1,88E-05	-16,7956	0
TN*SS	0,000538322	6,00262E-05	8,96812	0
TN*ET	2,23E-07	9,76E-08	2,28549	0,0333

Tabla 103. Detalles del modelo de log ($\pi = U_{cf}/U_{ldf}$). Espacio D.

Fuente	Suma de cuadrados	Grados de libertad	Mean Square	F-Ratio	P-Value
Modelo	0,885986	15	0,0590657	437,75	0,0000
Residual	0,00269863	20	0,000134931		
Total (Corr.)	0,888685	35			

Tabla 104. Análisis de la varianza del modelo de log ($\pi = U_{cf}/U_{ldf}$). Espacio D.

La validación del modelo se ha realizado comparando los valores observados frente a los calculados por el modelo -Figura 111- y verificando la normalidad de los residuos -Figura 112 y Tabla 106-.

Fuente	F-Ratio	Coeficiente estimado
LOC	1653,92	4,19881
TN*PS	947,22	-3,15E-04
ES	933,13	0,013254
LOC*TN	856,39	-1,99636
LOC*ES	620,47	-0,00402479
PS	514,03	0,00213865
LOC*SS	389,75	0,00547967
TN	210,23	-0,814334
SS	139,03	-0,00234831
IRF*ET	117,69	-1,13E-08
IRF*SS	100,62	2,30E-05
TN*SS	78,43	0,000538322
TN*ET	5,22	2,23E-07
LOC*PS	0,06	-0,0144211
ET	0	-1,92E-06

Tabla 105. Impacto de los factores (orden descendente) sobre el modelo de log ($\pi = U_{cf}/U_{ldf}$). Espacio D.

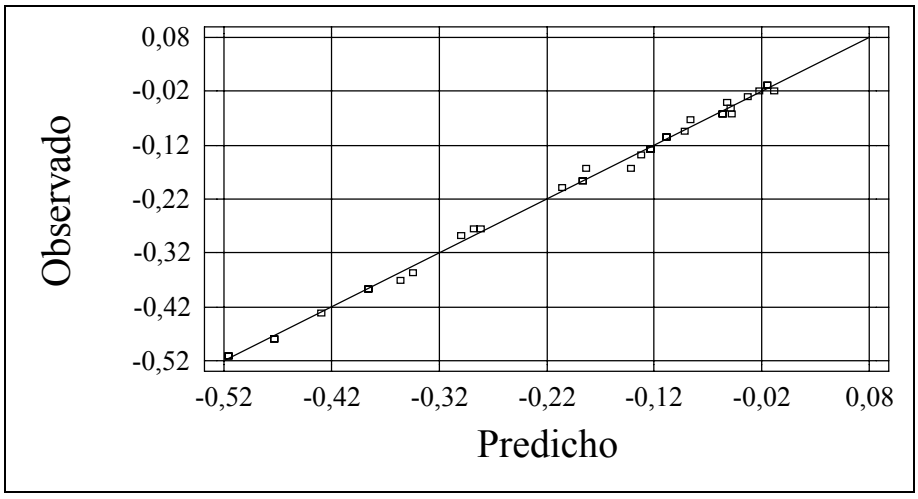


Figura 111. Representación de los valores de log ($\pi = U_{cd}/U_{ldf}$) observados frente a los predichos por el modelo para el espacio C.

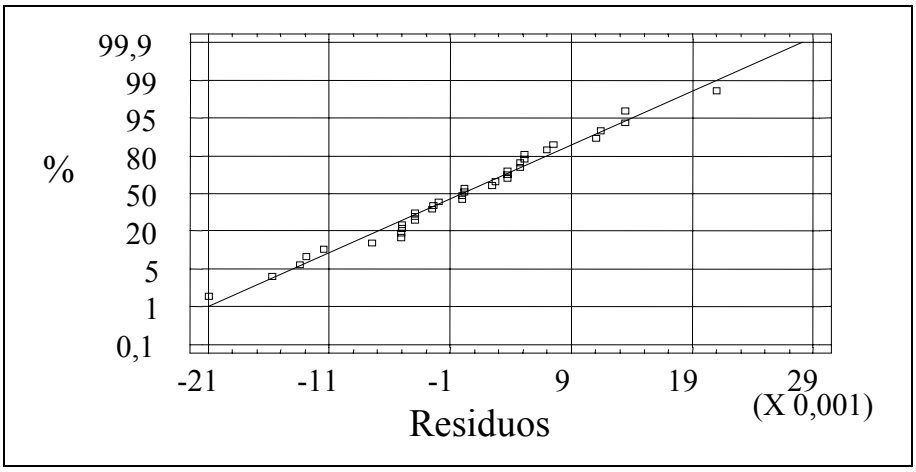


Figura 112. Representación en papel probabilístico normal de los residuos del modelo para el espacio D.

Número de datos	36
Media	5,11E-10
Varianza	7,71037E-05
Desviación estándar	0,00878087
Mínimo	-0,0209961
Máximo	0,0209961
Asimetría estándar	-0,218876
Curtosis estándar	0,455414

Tabla 106. Principales estadísticos de los residuos del modelo de log ($\pi = U_{cf}/U_{ldf}$). Espacio D.

El modelo correspondiente a los datos pertenecientes al espacio D presenta múltiples coincidencias con el desarrollado en el apartado anterior para los datos pertenecientes al espacio C, ambos para uso dinámico de *locking cache*, por lo que las conclusiones extraídas para dicho espacio son aplicables al espacio D. Aun así, existen dos diferencias importantes que se describen a continuación

- La variable explicativa SSR desaparece completamente, ya que una vez la *cache* es mayor que el código del sistema, no importa cuanto mayor es.
- La variable LOC aparece de forma individual, presentándose como el factor más importante. Su coeficiente positivo indica que los sistemas con un alto grado de localidad, es decir, que realicen un buen aprovechamiento de la *cache*, mejorarán sensiblemente sus *prestaciones*. Sin embargo, la existencia de interacciones de signo negativo y que incluyen a LOC amortiguará esta mejora, incluso eliminándola completamente. Al no existir interferencia intrínseca en ninguno de los dos esquema de *cache*, el tiempo de carga de la *locking cache* en cada activación de las tareas se hace significativo. Cuanto más tareas existan, por ejemplo, más importante será la influencia de este tiempo de carga sobre las *prestaciones* del uso dinámico de *locking cache*.

6.3 Análisis de los factores hardware

Las condiciones del análisis de los factores *hardware* para el uso dinámico de *locking cache*, sobre planificador con prioridades fijas, son las mismas que las definidas para el análisis de los

mismos factores en el uso estático de *locking cache*, describiéndose brevemente a continuación las principales características del diseño de experimentos 2^K realizado:

- El proceso bajo estudio o variable dependiente es la relación entre la máxima utilización planificable obtenida al emplear el uso dinámico de *locking cache* y la máxima utilización planificable al emplear *cache*s convencionales, en sistemas de prioridades fijas, definida esta relación como $\pi = U_{fd}/U_{ldf}$
- Los factores bajo estudio son la relación entre el tamaño total del sistema y el tamaño de *cache* –SSR-, la relación entre el tiempo de acierto y el tiempo de fallo al acceder a *cache* – T_{miss}-, el tamaño de la línea de *cache* –LS-, el nivel de interferencia del sistema –IRF- y el grado de localidad del sistema –LOC-. Aunque las dos últimas variables explicativas son características *software* del sistema, su inclusión permitirá detectar variaciones en los efectos de los factores *hardware* debido a dichas características, que de forma general se han mostrado significativas en los análisis realizados previamente.
- Se han llevado a cabo 112 nuevos experimentos asignando los factores bajo estudio a dos niveles, descritos en la Tabla 107.

Conjunto de tareas	**LOC (Nivel)**	**IRF (Nivel)**	**Tamaño de línea LS (Nivel)**	**Relación miss/hit T_{miss} (Nivel)**	**Tamaño de *cache***
A	0 (-)	0 (-)	16 bytes = 4 instr. (-)	5 ciclos (-)	1Kb a 64Kb
A	0 (-)	0 (-)	16 bytes = 4 instr. (-)	10 ciclos (+)	1Kb a 64Kb
A	0 (-)	0 (-)	32 bytes = 8 instr. (+)	5 ciclos (-)	1Kb a 64Kb
A	0 (-)	0 (-)	32 bytes = 8 instr. (+)	10 ciclos (+)	1Kb a 64Kb
A	0 (-)	1 (+)	16 bytes = 4 instr. (-)	5 ciclos (-)	1Kb a 64Kb
A	0 (-)	1 (+)	16 bytes = 4 instr. (-)	10 ciclos (+)	1Kb a 64Kb
A	0 (-)	1 (+)	32 bytes = 8 instr. (+)	5 ciclos (-)	1Kb a 64Kb
A	0 (-)	1 (+)	32 bytes = 8 instr. (+)	10 ciclos (+)	1Kb a 64Kb
B	1 (+)	0 (-)	16 bytes = 4 instr. (-)	5 ciclos (-)	1Kb a 64Kb
B	1 (+)	0 (-)	16 bytes = 4 instr. (-)	10 ciclos (+)	1Kb a 64Kb
B	1 (+)	0 (-)	32 bytes = 8 instr. (+)	5 ciclos (-)	1Kb a 64Kb
B	1 (+)	0 (-)	32 bytes = 8 instr. (+)	10 ciclos (+)	1Kb a 64Kb
B	1 (+)	1 (+)	16 bytes = 4 instr. (-)	5 ciclos (-)	1Kb a 64Kb
B	1 (+)	1 (+)	16 bytes = 4 instr. (-)	10 ciclos (+)	1Kb a 64Kb
B	1 (+)	1 (+)	32 bytes = 8 instr. (+)	5 ciclos (-)	1Kb a 64Kb
B	1 (+)	1 (+)	32 bytes = 8 instr. (+)	10 ciclos (+)	1Kb a 64Kb

Tabla 107. Características de los diferentes experimentos realizados para la evaluación del impacto de los factores *hardware* sobre $\pi = U_{cf}/U_{ldf.}$

El efecto del factor SSR se estudia de forma separada a la del resto de los factores, presentando en la Figura 113 el gráfico de dispersión de los valores de $\pi = U_{cf}/ U_{ldf}$ frente a los valores de SSR. El objetivo de esta gráfica es discriminar si los efectos producidos por los factores

hardware dependen de los valores de SSR. Esta gráfica muestra claramente que existe una ligera variación en las *prestaciones* ofrecidas por el uso dinámico de *locking cache* en función del valor que tomen los factores bajo estudio, ya que los puntos mostrados en cada "columna" no llegan a superponerse de forma completa. Lo que no es tan evidente es que el factor SSR modifique el efecto producido por el resto de factores, ya que la variabilidad mostrada por los datos es similar en todas las columnas, exceptuando, si cabe, las columnas centrales. Sin embargo, esta gráfica no es suficiente para descartar la interacción de SSR con los factores *hardware*, por lo que al igual que en los análisis realizados para uso estático se considerarán tres espacios diferentes, definidos por el valor de SSR de cada experimento. Estos espacios reciben el nombre de espacio ϵ_1, espacio ϵ_2 y espacio ϵ_3. La Tabla 108 muestra los límites inferior y superior de cada espacio.

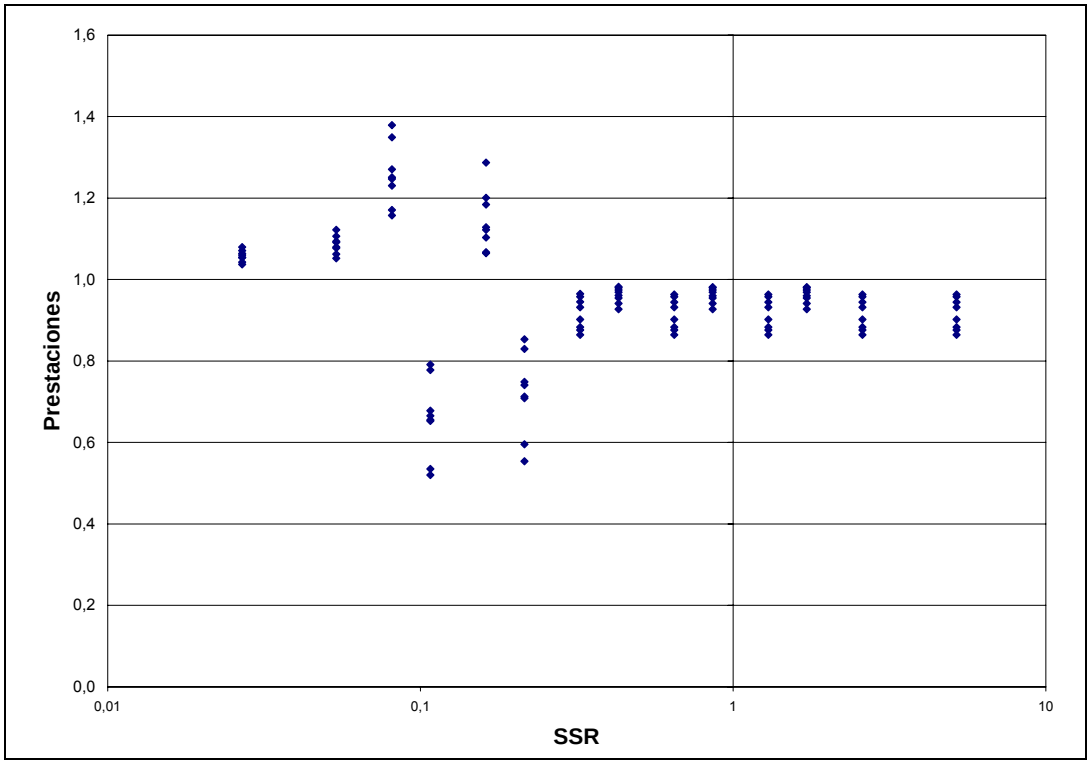


Figura 113. Gráfico de dispersión de $\pi = U_{cd}/U_{ldf}$ frente a SSR.

Espacio	Límite inferior	Límite superior
ϵ_1	$0 \leq SSR$	$SSR < 0,1$
ϵ_2	$0,1 \leq SSR$	$SSR < 1$
ϵ_3	$1 \leq SSR$	------

Tabla 108. Valores límites de SSR para la agrupación de $\pi = U_{cf}/U_{ldf}$ en los espacios ϵ_1, ϵ_2 y ϵ_3.

6.3.1 Análisis de los factores hardware para el espacio ϵ_1

La Tabla 109 muestra el resultado del análisis de los experimentos enmarcados en el espacio ϵ_1. En esta tabla se muestra el efecto de cada factor y de las interacciones dobles sobre el valor de π, así como el error normalizado al estimar dicho efecto. Sólo se incluyen interacciones a dos niveles, ya que normalmente interacciones de orden superior no son significativas ni aportan información importante [Romero93]. En esta tabla, el signo positivo del coeficiente estimado para un factor indica que el valor de π será mayor cuando el factor se encuentre a nivel +,

mientras que el signo negativo del coeficiente estimado para un factor indica que el valor de π será mayor cuando dicho factor se encuentre a nivel –. Los valores asignados a cada nivel pueden encontrarse en la Tabla 107.

Factor	Efecto	Standard error
Media	1,16426	+/- 0,00406994
A:T_{miss}	0,0672112	+/- 0,00813988
B:LS	-0,0557033	+/- 0,00813988
C:IRF	-0,00319644	+/- 0,00813988
D:LOC	-0,185212	+/- 0,00813988
AB	-0,0045265	+/- 0,00767435
AC	-0,00090317	+/- 0,00767435
AD	-0,0434811	+/- 0,00813988
BC	0,00673667	+/- 0,00767435
BD	0,0332279	+/- 0,00813988
CD	0,00014831	+/- 0,00813988

Tabla 109. Efecto de los factores *hardware* sobre $\pi = U_{cf}/U_{ldf}$. Espacio ϵ_1.

Antes de interpretar los resultados obtenidos es necesario determinar la importancia de cada factor y si su efecto es estadísticamente significativo. Para ello, el gráfico de Pareto muestra el efecto de cada factor ordenado de mayor a menor importancia -Figura 114- y el análisis de la varianza permite discriminar los factores o interacciones que son significativas. La Tabla 110 muestra el resumen del *anova* (*analyis of variance*) realizado, donde valores menores de 0,05 en la columna P-Value indican que el efecto del factor es significativo a un nivel de confianza del 95%.

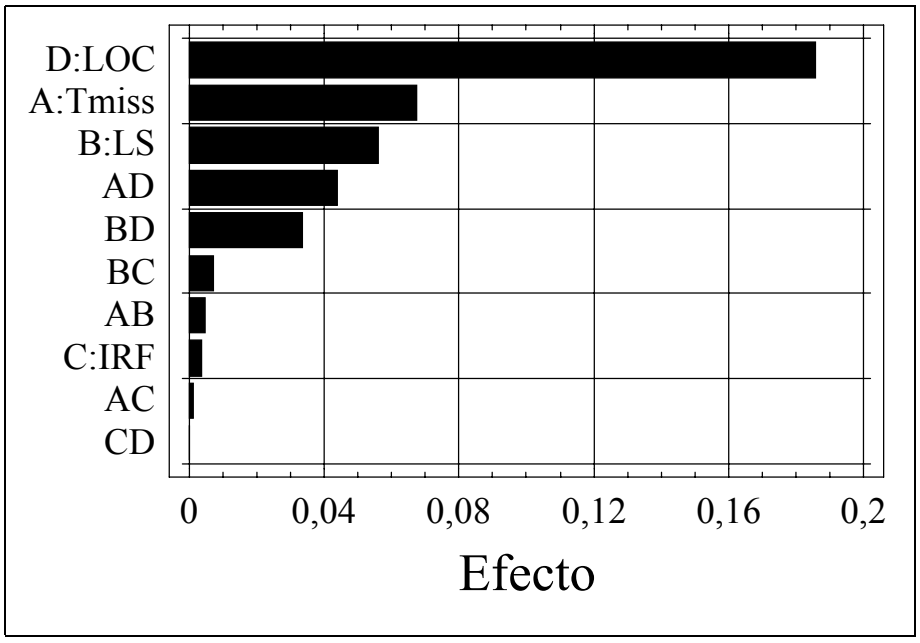


Figura 114. Gráfico de Pareto para $\pi = U_{cd}/U_{ldf}$. Espacio ϵ_1.

Factor	F-Ratio	P-Value
A:T_{miss}	68,18	0
B:LS	46,83	0
C:IRF	0,15	0,7009
D:LOC	517,73	0
AB	0,35	0,5654
AC	0,01	0,9081
AD	28,53	0,0001
BC	0,77	0,396
BD	16,66	0,0013
CD	0	0,9857

Tabla 110. *Anova* del diseño de experimentos para $\pi = U_{cf}/U_{ldf}$. Espacio ϵ_1.

En el *anova* se ha obtenido un valor de R-Squared del 98,02%, por lo que se puede asumir que la totalidad de la variabilidad presentada por los datos es debida a los factores bajo estudio y sus interacciones. Particularmente, y aunque cinco factores aparecen como significativos, el efecto de LOC es determinante en el comportamiento de π, ya que se muestra como el factor más importante y participa en las dos interacciones que se muestran significativas. Este resultado coincide con el obtenido en el análisis de los factores *software* para el espacio A, donde el factor LOC y sus interacciones aparecían con niveles de significación importante. Sin embargo, esta comparación no es muy adecuada, ya el espacio ϵ_1 comprende experimentos enmarcados tanto en el espacio A como en el espacio B.

El factor T_{miss} se presenta con coeficiente positivo, lo que indica que al aumentar el tiempo de fallo la *locking cache* presentará mejores *prestaciones*, mientras que el factor LS aparece con coeficiente negativo, lo que significa que al aumentar el tamaño de la línea de *cache*, el uso dinámico de *locking cache* perderá *prestaciones* frente a las *cache*s convencionales. En ningún momento esto es debido al número de expulsiones que sufren las tareas, ya que en el uso dinámico ambos esquemas de *cache* necesitan recargar los contenidos tras cada expulsión, situación corroborada por el bajo impacto que presenta el factor IRF en el análisis realizado. El efecto de T_{miss} y LS es debido al número de fallos en *cache* que se producen durante la ejecución al utilizar un modelo u otro de *cache*. Puesto que el espacio ϵ_1 comprende sistemas donde la *cache* es mucho más pequeña que el conjunto de las tareas, las *cache*s convencionales presentan una elevadísima tasa de fallos, ya que las instrucciones se reemplazan unas a otras de forma constante, mientras que al utilizar una *locking cache* un pequeño pero importante número de instrucciones se mantienen en *cache*, mientras que el resto de instrucciones compiten por el buffer temporal, presentando el mismo comportamiento que si estuvieran en una *cache* convencional, por lo que el número de fallos al acceder a *cache* es menor al emplear *locking cache*. De este modo, aumentar el tiempo de fallo perjudica a la *cache* convencional (y por tanto beneficia a la *locking cache*), ya que sufre un número mayor de fallos con un mayor coste temporal por cada fallo, mientras que aumentar el tamaño de línea beneficia a la *cache* convencional (perjudicando relativamente a la *locking cache*) ya que se reduce el tráfico entre memoria principal y *cache* y por tanto se reduce la tasa de fallos.

El coeficiente negativo de LOC, indicando que aquellos sistemas que hacen un uso intensivo de la *cache* presentan peores *prestaciones* al utilizar la *locking cache*, confirma la causa de los efectos *hardware* descrita en el párrafo anterior. Además, las interacciones T_{miss}*LOC y LS*LOC, que se muestran en la Figura 115 y la Figura 116 muestran que el efecto de los factores *hardware* será mayor (pendiente más pronunciada) en aquellos sistemas con un nivel de localidad bajo, debido a que son éstos los que más fallos de *cache* sufren.

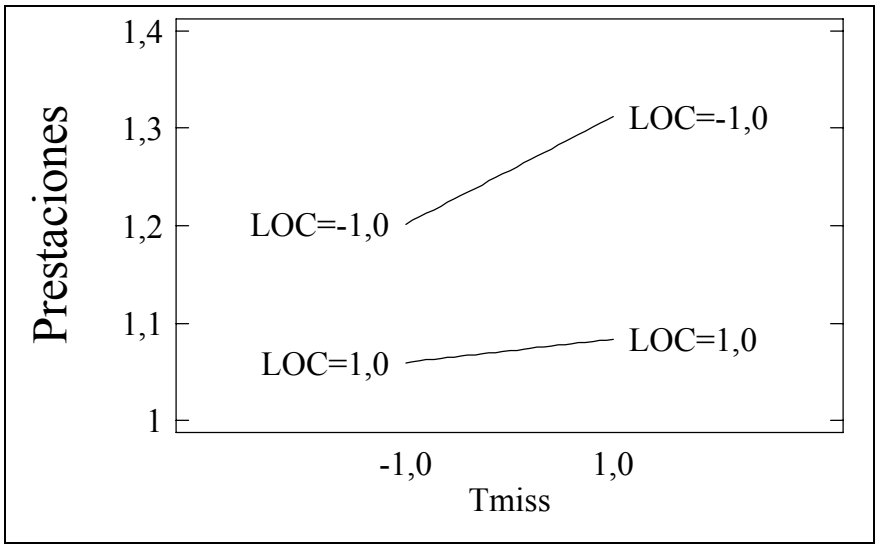


Figura 115. Efecto de los factores LOC y T_{miss} para $\pi = U_{cf}/ U_{ldf}$. Espacio ϵ_1.

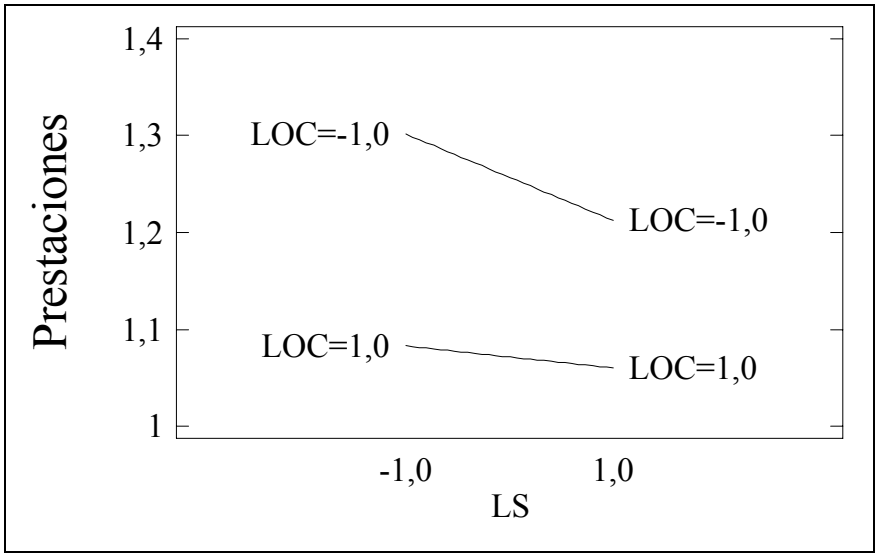


Figura 116. Efecto de los factores LOC y LS para $\pi = U_{cf}/ U_{ldf}$. Espacio ϵ_1.

6.3.2 Análisis de los factores hardware para el espacio ϵ_2

La Tabla 111 muestra el resultado del análisis de los experimentos enmarcados en el espacio ϵ_2. En esta tabla se muestra el efecto de cada factor y de las interacciones dobles sobre el valor de π, así como el error normalizado al estimar dicho efecto.

Antes de interpretar los resultados obtenidos es necesario determinar la importancia de cada factor y si su efecto es estadísticamente significativo. Para ello, el gráfico de Pareto muestra el efecto de cada factor ordenado de mayor a menor importancia –Figura 117- y el análisis de la varianza permite discriminar los factores o interacciones que son significativas. La Tabla 112 muestra el resumen del *anova* realizado, donde valores menores de 0,05 en la columna P-Value indican que el efecto del factor es significativo a un nivel de confianza del 95%.

Factor	Efecto	Standard error
Media	0,908417	+/- 0,0199266
A:T_{miss}	-0,018754	+/- 0,0398532
B:LS	0,0560086	+/- 0,0398532
C:IRF	0,0120755	+/- 0,0398532
D:LOC	-0,167143	+/- 0,0398532
AB	-0,00528175	+/- 0,0394445
AC	0,00245461	+/- 0,0394445
AD	-0,0532499	+/- 0,0398532
BC	-0,0140922	+/- 0,0394445
BD	0,0263963	+/- 0,0398532
CD	0,00353754	+/- 0,0398532

Tabla 111. Efecto de los factores *hardware* sobre $\pi = U_{cf}/U_{ldf}$. Espacio ϵ_2.

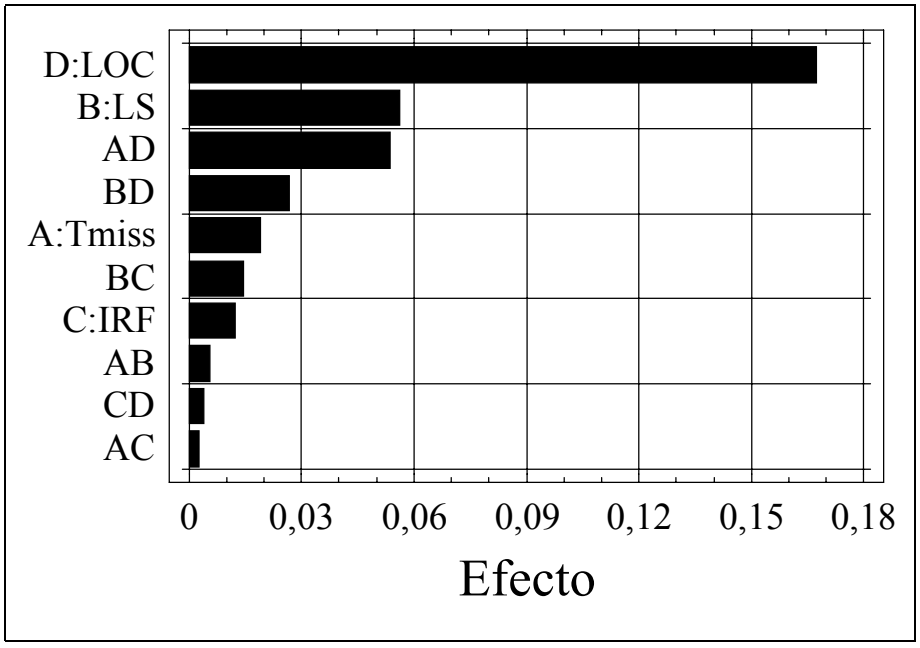


Figura 117. Gráfico de Pareto para $\pi = U_{cd}/U_{ldf}$. Espacio ϵ_2.

En el *anova* se ha obtenido un valor de R-Squared del 33,64%, lo que indica que la gran parte de la variabilidad observada en los datos es debida a otros factores y características no contempladas en este análisis, donde sólo aparece como significativo el factor LOC.

Factor	F-Ratio	P-Value
A:T_{miss}	0,22	0,6402
B:LS	1,98	0,1668
C:IRF	0,09	0,7633
D:LOC	17,59	0,0001
AB	0,02	0,8941
AC	0	0,9507
AD	1,79	0,1882
BC	0,13	0,7226
BD	0,44	0,5111
CD	0,01	0,9297

Tabla 112. *Anova* del diseño de experimentos para $\pi = U_{cf}/U_{ldf}$. Espacio ϵ_2.

Pese a la falta de significación estadística de los factores LS y T_{miss}, es posible estudiar el efecto que producirán sobre las *prestaciones* del uso dinámico de *locking cache* siempre que se mantenga la conciencia sobre su casi inapreciable influencia. Comparando los signos de dichos coeficientes frente a los obtenidos en el espacio ϵ_1 se aprecia que tienen signo contrario, es decir, en este caso el aumento del tiempo de fallo perjudicará a la *locking cache*, mientras que el aumento del tamaño de línea perjudicará a las *cache*s convencionales. Esta influencia es debida a que en este espacio el tamaño de *cache* es bastante grande como para albergar partes importantes del código de la tarea, pero no lo suficiente como para albergar todas las zonas importantes de código de forma simultánea. La *cache* convencional, con su comportamiento dinámico, realiza un mejor aprovechamiento del espacio disponible, por lo que sufre un menor número de fallos al acceder a *cache*, contrariamente a como sucedía en el espacio ϵ_1. El incremento en el tiempo de fallo perjudica en mayor medida al esquema de *cache* que sufre un número mayor de fallos (*locking cache*) mientras que el aumento en el tamaño de línea reduce el número de fallos que se producen, beneficiando en este caso a la *locking cache* al ser el esquema que sufre, en principio, un número de fallos mayor.

Pese a que el análisis realizado en el párrafo anterior es coherente con los resultados obtenidos, es importante recordar que en el espacio ϵ_2 están incluidos experimentos pertenecientes al espacio B del análisis de los factores *software*, y que para dichos experimentos fue imposible encontrar un modelo válido debido a la irregularidad del error cometido al estimar los valores de U_{ldf}. Esta es posiblemente la causa del bajísimo valor del R-Squared obtenido en el presente análisis, y un indicador claro de que las conclusiones obtenidas deben considerarse con cierto escepticismo y mucha cautela.

6.3.3 Análisis de los factores hardware para el espacio ϵ_3

La Tabla 113 muestra el resultado del análisis de los experimentos enmarcados en el espacio ϵ_3. En esta tabla se muestra el efecto de cada factor y de las interacciones dobles sobre el valor de π, así como el error normalizado al estimar dicho efecto.

Antes de interpretar los resultados obtenidos es necesario determinar la importancia de cada factor y si su efecto es estadísticamente significativo. Para ello, el gráfico de Pareto muestra el efecto de cada factor ordenado de mayor a menor importancia –Figura 118- y el análisis de la varianza permite discriminar los factores o interacciones que son significativas. La Tabla 114 muestra el resumen del *anova* realizado, donde valores menores de 0,05 en la columna P-Value indican que el efecto del factor es significativo a un nivel de confianza del 95%.

Factor	Efecto	Standard error
Media	0,938145	+/- 0,0016159
A:T_{miss}	-0,00983975	+/- 0,0032318
B:LS	0,0492967	+/- 0,0032318
C:IRF	0,006712	+/- 0,0032318
D:LOC	0,045501	+/- 0,0032318
AB	-0,012415	+/- 0,00279882
AC	0,00085913	+/- 0,00279882
AD	-0,0064335	+/- 0,0032318
BC	-0,0124249	+/- 0,00279882
BD	-0,0187885	+/- 0,0032318
CD	-0,00007975	+/- 0,0032318

Tabla 113. Efecto de los factores *hardware* sobre $\pi = U_{cf}/U_{ldf}$. Espacio ϵ_3.

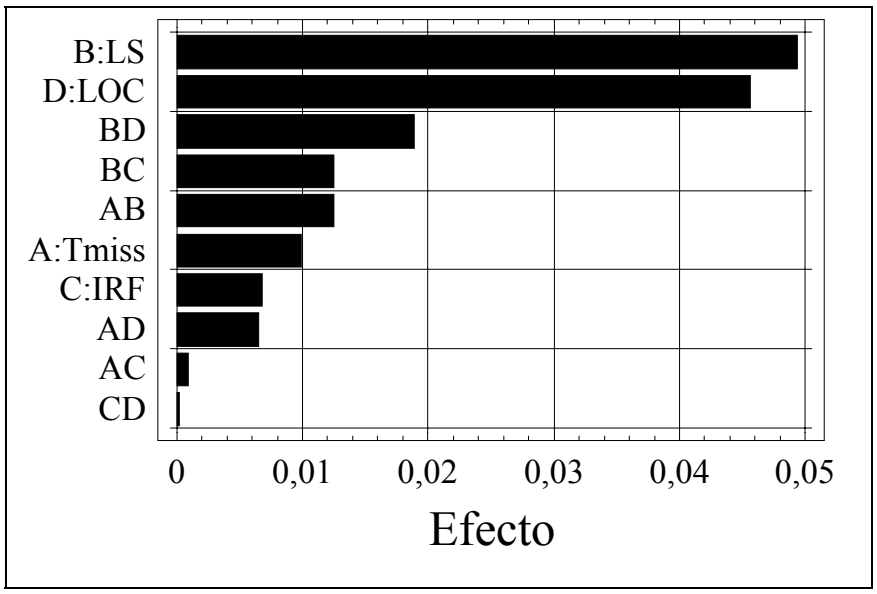


Figura 118. Gráfico de Pareto para $\pi = U_{cd}/U_{ldf}$. Espacio ϵ_3.

En el *anova* se ha obtenido un valor de R-Squared del 97,19%, indicando que los cuatro factores incluidos en el análisis son la causa determinante de la variación observada en los datos. Concretamente, seis factores tanto simples como interacciones dobles se muestran estadísticamente significativos. De estos seis, el gráfico de Pareto muestra que los más importantes con diferencia son LS y LOC, seguidos de lejos, en cuanto a importancia del efecto, por la interacción de ambos. Es de destacar que el efecto de T_{miss} aunque significativo es casi nulo, mientras que el efecto de IRF ni siquiera se muestra como significativo.

El factor LS aparece con coeficiente positivo, indicando que al aumentar el tamaño de la línea de *cache*, el uso dinámico de *locking cache* mejorará sus *prestaciones* frente al empleo de *cache*s convencionales. Esto es debido a que las *cache*s convencionales no sufren, en este espacio, ningún tipo de interferencia, ni intrínseca ni extrínseca, ya que todo el código de todas las tareas puede ser ubicado simultáneamente en *cache*. Sin embargo, el uso dinámico de *locking cache* no ha sido diseñado para detectar esta situación, por lo que cada vez que una tarea

entre en ejecución recargará los contenidos de *cache,* aunque estos ya se encuentren en cache. Al aumentar el tamaño de la línea de *cache* se reduce el número de bloques que deben ser transferidos, por lo que la *locking cache* sufrirá una menor penalización debida a la recarga de la *cache*.

Factor	F-Ratio	P-Value
A:T_{miss}	9,27	0,0062
B:LS	232,67	0
C:IRF	4,31	0,0503
D:LOC	198,22	0
AB	19,68	0,0002
AC	0,09	0,7619
AD	3,96	0,0597
BC	19,71	0,0002
BD	33,8	0
CD	0	0,9805

Tabla 114. *Anova* del diseño de experimentos para $\pi = U_{cf}/U_{ldf}$. Espacio ϵ_3.

El factor LOC se muestra con coeficiente positivo, indicando que la *locking cache* presentará mejores *prestaciones* para aquellos sistemas con un alto grado de localidad. Estos sistemas compensarán, con un mejor aprovechamiento de los contenidos cargados en *cache*, el tiempo de carga empleado por la *locking cache*.

La Figura 119 muestra el efecto de la interacción LS*LOC, donde se puede apreciar que el efecto del factor LS será más pronunciado para aquellos sistemas con un bajo nivel de localidad, ya que el tiempo de carga de los contenidos de *cache* es, en relación al tiempo de ejecución, mayor, pero manteniendo el signo del efecto independientemente del grado de localidad del sistema.

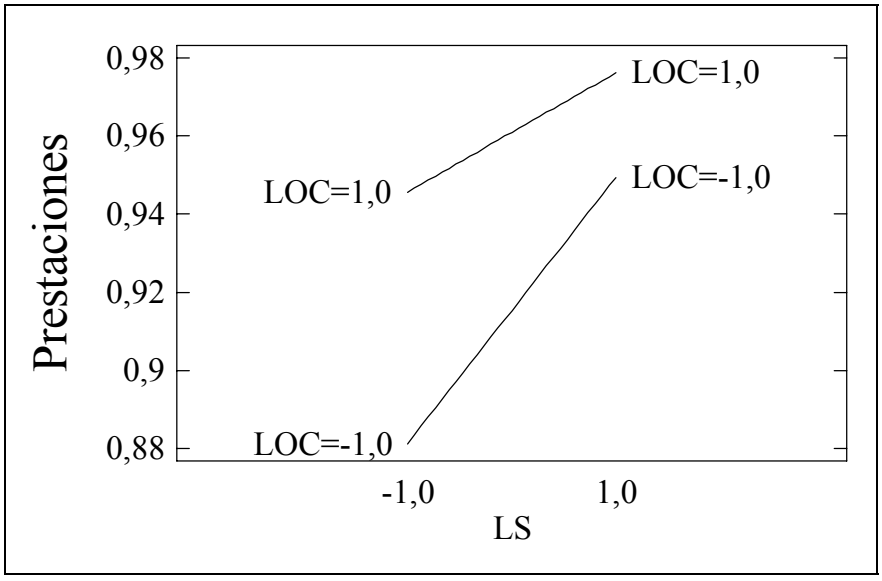


Figura 119. Efecto de los factores LOC y LS para $\pi = U_{cf}/ U_{ldf}$. Espacio ϵ_3.

6.4 Conclusiones del capítulo

Los análisis estadísticos realizados en este capítulo han permitido estimar el comportamiento de las *prestaciones* del uso dinámico de la *locking cache* en función de un conjunto de características *software* y *hardware* de los sistemas de tiempo real estudiados. Estas *prestaciones* se han definido como la relación entre la utilización planificable obtenida por el uso dinámico de *locking cache* frente a la utilización planificable obtenida en el uso de *cache*s convencionales, definiendo esta relación como $\pi = U_{cf}/U_{ldf}$, por lo que se ha estudiado la ganancia o pérdida de *prestaciones* sufrida por el uso dinámico de *locking cache* frente al empleo de *cache*s convencionales.

Nuevamente, tal como sucedió con el uso estático de *locking cache*, la relación entre el tamaño de la *cache* y el tamaño del sistema –SSR- se ha mostrado como una característica fundamental a la hora de determinar las *prestaciones* ofrecidas por el esquema de *cache* estudiado en este capítulo. Aunque otros factores, como el grado de localidad de las tareas o el número de tareas en el sistema influyen en la ganancia o pérdida de *prestaciones*, el factor SSR define, por sí sólo, diferentes escenarios en los que las *prestaciones* se distribuyen en rangos claramente definidos y delimitados. El efecto de SSR alcanza también a los factores *hardware*, ya que la variabilidad que estos producen sobre las *prestaciones* depende del valor de SSR del sistema, siendo esta influencia no sólo a nivel de intensidad o significación del efecto, sino también modificando el signo del efecto.

La agrupación de los datos en espacios ha permitido comparar, de forma muy precisa, las *prestaciones* ofrecidas por el uso dinámico de *locking cache* frente al uso estático de *locking cache*. De esta comparación se han podido extraer los siguientes resultados:

- El uso dinámico de *locking cache* presenta mejores *prestaciones* que el uso estático siempre y cuando el tamaño de la *cache* sea menor que el tamaño del sistema, es decir, que la suma de los tamaños de las tareas que forman el sistema.
- El comportamiento de las *prestaciones* del uso dinámico de *locking cache* es, en general, el contrario al mostrado por las *prestaciones* del uso estático. Aquellos factores o interacciones que aparecían en los modelos de ambos usos de *cache* presentaban signos opuestos en sus coeficientes estimados. Esto manifiesta que el uso dinámico de *locking cache* presenta un comportamiento muy similar al de las *cache*s convencionales, sin olvidar que además aportan determinismo.
- Al contrario que en el uso estático de *locking cache*, el nivel de interferencia extrínseca –IRF- del sistema no influye en las *prestaciones* ofrecidas por el uso dinámico respecto de las *caches* convencionales.
- Para determinados sistemas –aquellos enmarcados en el espacio B- el error cometido por el algoritmo genético al estimar los tiempos de respuesta de las tareas no es constante ni presenta una distribución uniforme, por lo que puede distorsionar la evaluación de la ganancia o pérdida de *prestaciones* sufrida por el uso dinámico de *locking cache* frente a la utilización de *cache*s convencionales.
- Los factores *hardware*, aunque influyen en las *prestaciones* obtenidas por la *locking cache*, lo hacen de manera poco intensa, especialmente la relación entre el tiempo de fallo y el tiempo de acierto –T_{miss}-.

Pese a que el uso dinámico de *locking cache* se desarrolla con el objetivo de mejorar las *prestaciones* al mismo tiempo que se obtiene una *cache* determinista, los resultados obtenidos en este capítulo muestran que dicho objetivo no se alcanza en todas las circunstancias, ya que cuando el tamaño de la cache iguala o supera al tamaño del código del sistema, el uso dinámico presenta unas prestaciones significativamente menores que el uso estático. Por ello, el diseñador de sistemas de tiempo real deberá utilizar los modelos desarrollados para los dos posibles usos de *locking cache* y determinar cuál de ellos es el más adecuado.

Capítulo 7

Conclusiones

Las conclusiones particulares y en detalle han sido presentadas al final de cada capítulo de esta tesis, por lo que en este último capítulo se presentan únicamente las conclusiones generales y más relevantes, así como las líneas de investigación que pueden surgir a partir de los trabajos descritos

7.1 Conclusiones

En la introducción de esta tesis se mostró que la utilización de memorias *cache* en sistemas de tiempo real genera problemas complejos a la hora de determinar los tiempos de ejecución y de respuesta de las tareas, y por tanto, a la hora de determinar la planificabilidad del sistema. Esta complejidad surge, principalmente, del comportamiento dinámico, adaptativo y poco predecible de este tipo de memoria. Este comportamiento es precisamente la razón del gran aumento de prestaciones que se obtiene al incluir la *cache* en la jerarquía de memoria. Pero no sólo esta es la causa de los problemas presentados por el uso de memorias *cache*. Su interacción con otras mejoras estructurales, como la segmentación, la predicción de saltos o la ejecución desordenada hacen que el uso combinado de estas técnicas convierta la estimación precisa del tiempo de respuesta de las tareas en un problema casi irresoluble.

Múltiples soluciones y propuestas se han presentado en los últimos años y siguen presentándose en la actualidad. Contrariamente a lo que podría pensarse, la existencia de una gran variedad de técnicas para utilizar de forma eficaz y segura las memorias *cache* en los sistemas de tiempo real no significa que el problema se encuentre resuelto, sino todo lo contrario. Las soluciones existentes no son completamente satisfactorias, por lo que nuevas soluciones son presentadas, intentando sobrepasar las limitaciones de los trabajos realizados anteriormente.

El inconveniente principal y común a casi todas las soluciones existentes hasta el momento se encuentra en que éstas resuelven una de las dos caras del problema, ignorando la otra. Esto sucede tanto para las técnicas de análisis como en las diferentes modificaciones propuestas al esquema convencional de *cache*, tanto *hardware* como *software*. De este modo, las diferentes soluciones pueden dividirse en dos grupos independientes: soluciones para la interferencia intrínseca y cálculo del tiempo de ejecución de las tareas, y soluciones para la interferencia extrínseca. Añadido a esto, la complejidad y requisitos para la puesta en práctica de algunas soluciones limita de forma importante su ámbito de aplicación.

En este trabajo se ha presentado una solución que, a través de la propuesta de una arquitectura de *cache* predecible, alcanza un triple objetivo:

- Utilización de técnicas sencillas y practicables para la estimación del tiempo de ejecución y de respuesta de las tareas. Estas técnicas son pequeñas modificaciones a algoritmos ya existentes, sencillos y bien conocidos, por lo que su utilización no presenta serios problemas ni elevada complejidad. Una ventaja derivada directamente de la consecución de este objetivo es la posibilidad de aplicar todas las técnicas desarrolladas para el análisis de sistemas de tiempo real con determinadas particularidades, como la consideración de rutas imposibles en las tareas, la existencia de desfases en los lanzamientos de las tareas o la inclusión de tareas aperiódicas y esporádicas.
- Solución global y unificada a las dos vertientes del problema: interferencias intrínseca y extrínseca. La información necesaria, en cuanto al modelado de la *cache*, para la obtención de los tiempos de respuesta de las tareas es la misma que la utilizada para la obtención de los tiempos de ejecución. Dicha información se comparte para solucionar los dos problemas. Esta unificación del problema presenta básicamente dos ventajas: la no

necesidad de modelar dos veces la memoria *cache*, y la reducción de la sobreestimación resultante de utilizar dos modelos distintos.

- Mantenimiento de las prestaciones del nuevo esquema en niveles similares a las de las arquitecturas convencionales. Este nivel de prestaciones se obtiene, por una parte, por la arquitectura propuesta, y por otra parte, por la utilización de algoritmos genéticos para la selección de los contenidos de la *cache*. El desarrollo de un algoritmo genético ha permitido obtener las mejores prestaciones posibles al mismo tiempo que mantiene el coste computacional de la selección de contenidos dentro de unos límites aceptables.

La estructura de *cache* propuesta es una variación de las *cache*s con bloqueo o *locking cache*s, existentes en múltiples procesadores comerciales. Sin embargo, las implementaciones concretas de cada fabricante no garantizan el total determinismo del sistema, por lo que las modificaciones propuestas se dirigen fundamentalmente a la obtención de un esquema totalmente predecible, al mismo tiempo que se introducen elementos para intentar mantener las mismas prestaciones que los esquemas de *cache* convencionales. Las modificaciones propuestas son, en cuanto a complejidad *hardware*, mínimas, por lo que su realización física no presenta ningún problema.

Las *cache*s con bloqueo permiten mantener de forma constante e invariados los contenidos de *cache*. Dos formas de utilizar esta cualidad se han descrito con detalle, junto con las herramientas necesarias para determinar la planificabilidad del sistema: uso estático de *locking cache* y uso dinámico de *locking cache*.

En el uso estático de *locking cache*, los contenidos se cargan y bloquean en *cache* antes de lanzar a ejecución las tareas, y se mantienen sin variación durante todo el funcionamiento del sistema. El principal logro de este modo de utilización de la *locking cache* es el altísimo grado de determinismo alcanzado, avalado por los resultados experimentales, que permite estimar con exactitud los tiempos de respuesta de las tareas. La principal desventaja es la obtención de unas prestaciones sensiblemente inferiores a las ofrecidas por las *cache*s convencionales en determinados sistemas, aunque de forma general esta pérdida de prestaciones es mínima.

El uso dinámico de *locking cache*, donde los contenidos de *cache* se modifican durante el funcionamiento del sistema cada vez que una tarea entra en ejecución, ofrece unos niveles de determinismo menores que el uso estático, pero por contra, presenta mejores prestaciones de forma generalizada, aunque en determinados casos dichas prestaciones siguen siendo inferiores a las ofrecidas por las *cache*s convencionales.

Las prestaciones de la *locking cache* se han comparado con los resultados de la simulación de la ejecución de los experimentos considerando la utilización de *cache*s convencionales. Estas simulaciones ofrecen, como resultado, la cota máxima de prestaciones que se puede obtener, pero que en muchos casos, las técnicas de análisis propuestas hasta el momento no son capaces de alcanzar debido al error cometido al modelar el comportamiento de estas memorias. Por tanto, la pérdida de prestaciones que presenta tanto el uso estático como dinámico de *locking cache* corresponde al caso más desfavorable para el esquema propuesto. La comparación frente a las prestaciones ofrecidas por las diferentes técnicas de análisis podría reducir, en muchos casos, esta pérdida de prestaciones.

Los diferentes análisis estadísticos realizados para cada una de las propuestas de uso de la *locking cache* han permitido obtener un conjunto de modelos del comportamiento de las prestaciones. Estos modelos, que consideran diferentes características *software* y *hardware* del sistema, permiten predecir la pérdida o ganancia de prestaciones que se obtendrá al emplear una *locking cache*, por lo que el diseñador de sistemas de tiempo real podrá evaluar, sin necesidad de llevar a cabo ningún experimento, el esquema de *cache* más conveniente para los requisitos de su sistema. Pero el estudio de los resultados obtenidos de dichos análisis ha permitido extraer una serie de conclusiones, en cuanto a las prestaciones presentadas por la *locking cache*, que se resumen a continuación:

- El comportamiento del uso estático de *locking cache* es independiente del tipo de planificador utilizado, ya que los modelos desarrollados para planificadores de prioridades fijas y de prioridades dinámicas presentan una altísimo grado de coincidencia. Las ligeras diferencias observadas son debidas, posiblemente, a la forma de obtener la utilización para cada una de las políticas de planificación.
- De todos los factores considerados en los análisis, tanto *hardware* como *software*, la relación entre el tamaño de la *cache* y el tamaño del sistema se ha mostrado como el más importante de todos, explicando por sí sólo una parte significativa de la variabilidad observada en las prestaciones de la *locking cache*. Más allá aún, este factor -SSR- determina el efecto que el resto de características del sistema ejercen sobre las prestaciones.
- La utilización de SSR para la creación de grupos de sistemas ha permitido estudiar con detalle el comportamiento de las prestaciones, identificando diferentes espacios o conjuntos de sistemas con tendencias claramente definidas.
- Otros factores se han mostrado significativos de forma generalizada pero supeditados al valor de SSR para los dos usos de la *locking cache*. Estos factores son el número de tareas, el tiempo de acceso cuando se produce un fallo en *cache* y el tamaño de la línea de *cache*. Sin embargo, el efecto de estas características no es constante para todos los espacios definidos por el factor SSR.
- El nivel de localidad de las tareas que forman el sistema y el grado de interferencia entre las tareas también determinan, de forma general, tendencias en el comportamiento de las prestaciones de la *locking cache*. De este modo, los sistemas pueden clasificarse según estas características, sirviendo también esta clasificación para una estimación rápida de la posible pérdida o ganancia de prestaciones que presentará la *locking cache* para un determinado sistema.
- Aunque el uso dinámico presenta mejores prestaciones, de forma generalizada, que el uso estático, la comparación a través de los grupos generados a partir de SSR ha mostrado que esto no siempre es así. Por ello, en determinadas situaciones, claramente definidas por el valor de SSR y que corresponden a tamaños de cache iguales o mayores que el tamaño del código del sistema, es más conveniente desde el punto de vista de las prestaciones el uso estático de *locking cache*.

7.2 Trabajo futuro

De los trabajos desarrollados en esta tesis y de los resultados experimentales se derivan pocas líneas de investigación en cuanto a la obtención de determinismo, ya que el esquema presentado es completamente predecible tanto en el ámbito de la interferencia intrínseca como en el de la extrínseca. El problema más interesante en el ámbito del determinismo es la consideración combinada de la existencia de *locking cache* y otras mejoras estructurales, como la segmentación o la predicción de saltos. Aunque la utilización de memorias *cache* con bloqueo proporciona un sistema totalmente determinista y predecible, éste no se encuentra libre de posibles interacciones con otras estructuras del procesador, por lo que deben llevarse a cabo los experimentos necesarios para identificar, modelar e incluir en los análisis dichas interacciones.

Donde sí existe un gran número de direcciones en las que desarrollar nuevas investigaciones es en el ámbito de las prestaciones. Aunque el uso dinámico de *locking cache* proporciona unos niveles de prestaciones similares a los de las *cache*s convencionales, en algunos casos existe una pérdida de prestaciones significativa, que aunque puede ser superada mediante la utilización de procesadores más potentes, determinadas aplicaciones, especialmente las relacionadas con productos de gran consumo, no pueden asumir.

En los casos en que hay pérdida de prestaciones en el uso dinámico de *locking cache*, esta pérdida se debe a dos causas. Por un lado, aunque el espacio de *cache* se utiliza de forma dinámica, los reemplazos se realizan de forma muy estricta y limitada. Por otro lado, el coste

temporal de reemplazar una línea de *cache* es del orden de cuatro veces superior al que presenta una *cache* convencional, ya que es necesario ejecutar un bucle con múltiples accesos a memoria para precargar la memoria *cache*. Es por esta última razón que el uso dinámico presenta un mal comportamiento cuanto mayor es el tamaño de la *cache*, ya que el número de bloques que se mueven en cada reemplazo es superior. Reducir este tiempo de carga, mediante la inclusión de mecanismos *hardware* que detecten de forma automática los bloques que deben cargarse y bloquearse en *cache* mejoraría de forma general las prestaciones, y especialmente en aquellos casos en que el número de bloques que deben transferirse desde memoria principal a memoria *cache* es elevado. Así mismo, aumentar el dinamismo de la *locking cache*, permitiendo el reemplazo de sus contenidos en determinados puntos bien seleccionados ofrecería también una mejora generalizada.

Por otro lado, las prestaciones presentadas en este trabajo se han obtenido gracias a la utilización de algoritmos genéticos, que ofrecen una solución pseudo-óptima. Como se comentó, es casi imposible desarrollar un algoritmo que encuentre la solución óptima en tiempo acotado, pero sí es posible desarrollar algoritmos con un coste computacional menor que los genéticos que ofrezcan prestaciones similares, aumentando de esta forma el atractivo de las *cache*s con bloqueo. En esta dirección ya existen trabajos publicados [Puaut02].

Finalmente, tanto el uso estático como dinámico de *locking cache* presenta en algunos casos unas prestaciones superiores a las de las memorias *cache* convencionales. Aunque la problemática de los sistemas de propósito general es muy diferente a la de los sistemas de tiempo real, las técnicas descritas en este trabajo, debidamente modificadas y adaptadas, podrían mejorar las prestaciones en determinados entornos, como los sistemas empotrados, que aun no siendo de tiempo real, están formados por una carga *software* bastante estable y que permiten un análisis estático de su comportamiento.

Capítulo 8

Bibliografía

[Arnold94] Robert Arnold, Frank Mueller, David Whalley and Marion G. Harmon. **Bounding Worst-Case Instruction Cache Performance.** Proceedings of the 15th IEEE Real-Time Systems Symposium. pp. 172-181. San Juan, Puerto Rico December 1994

[Audsley91a] N. Audsley, A. Burns, M. Richardson and A. Wellings. **Hard Real-Time Scheduling: The Deadline Monotonic Approach.** Proceedings of the 8th IEEE Workshop on Real-Time Operating Systems and Software. Atlanta, GA May 1991

[Audsley93a] N. Audsley, K. W. Tindell and A. Burns. **The End of the Road for Static Cyclic Scheduling.** Proceedings of the 5th Euromicro Workshop on Real-Time Systems. pp. 36-41. Oulu, Finland 1993

[Audsley93b] N. Audsley, A. Burns, M. Richardson, K. Tindell and A. Wellings. **Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling.** Software Engineering Journal, Vol. **8**, Num. 5. pp. 284-292. September 1993

[Bacon94] David F. Bacon, Susan L. Graham and Oliver J. Sharp. **Compiler Transformations for High-Performance Computing.** ACM Computing Surveys, Vol. **26**, Num. 4. pp. 345-420. December 1994

[Baker90] T. P. Baker. **A Stack-Based Resource Allocation Policy for Realtime Processes.** Proceedings of the 11th IEEE Real-Time Systems Symposium. pp. 191-200. Lake Buena Vista, FL December 1990

[Basumallick94] Swagato Basumallick and Kelvin D. Nilsen. **Cache Issues in Real-Time Systems.** ACM SIGPLAN Workshop on Language, Compiler, and Tool Support for Real-Time Systems. June 1994

[Bernat98] Guillem Bernat Nicolau. **Specification and Analysis of Weakly Hard Real-Time Systems.** Tesis doctoral. Departament de Ciències Matemàtiques i Informàtica. 1998

[Bershad94] Brian N. Bershad, Dennis Lee, Theodore H. Romer and J. B. Chen. **Avoiding Conflict Misses Dynamically in Large Direct-Mapped Caches.** Sixth International Conference on Architectural Support for Programming Languages and Operating Systems. San Jose, CA October 1994

[Boland94] Keith Boland and Apostolos Dollas. **Predicting and Precluding Problems with Memory Latency.** IEEE Micro, Vol. **14**, Num. 4. pp. 59-67. August 1994

[Burns95a] A. Burns, N. Hayes and M.F. Richardson. **Generating Feasible Cyclic Schedules.** Control Engineering Practice, Vol. **3**, Num. 2. pp. 151-162. 1995

[Bursky96c] Dave Bursky. **Advanced CPUs, Multimedia ICs Deliver Top Throughputs.** Electronic Design, Vol. **44**, Num. 2. pp. 55-74. 19 February 1996

[Busquets93] Jose Vicente BusquetsMataix and Jeremy Jones. **XmAnalyse Software.** Internal Report, Computer Architecture Group. Dublin, July 1993

[Busquets95a] Jose Vicente BusquetsMataix and Juan Jose Serrano. **The Impact of Extrinsic Cache Performance on Predictability of Real-Time Systems.** Proceedings of the Second International Workshop on Real-Time Computing Systems and Applications. pp. 8-15. Tokyo, Japan October 1995

[Busquets95b] Jose Vicente BusquetsMataix and Andy J. Wellings. **Adding Instruction Cache Effect to Schedulability Analysis of Preemptive Real-Time Systems.** Department of Computer Science, YCS 260. pp. 1-15. University of York, UK October 1995

[Busquets96a] Jose Vicente BusquetsMataix, Andy J. Wellings, Juan Jose Serrano, Rafael Ors and Pedro Gil. **Adding Instruction Cache Effect to an Exact Schedulability Analysis of Preemptive Real-Time Systems.** 8th Euromicro Workshop on Real-Time Systems. pp. 271-276. June 1996

[Busquets96b] Jose Vicente BusquetsMataix, Andy J. Wellings, Juan Jose Serrano, Rafael Ors and Pedro Gil. **Adding Instruction Cache Effect to Schedulability Analysis of Preemptive Real-Time Systems.** IEEE Real-Time Technology and Applications Symposium. pp. 204-212. June 1996

[Busquets97] Jose Vicente BusquetsMataix, Andy J. Wellings and Juan Jose Serrano. **Hybrid Instruction Cache Partitioning for Preemptive Real-Time Systems.** 9th Euromicro Workshop on Real-Time Systems. pp. 271-276. Toledo, Spain June 11-12, 1997

[Busquets97b] J. V. Busquets-Mataix. **Técnicas para la Utilización Eficaz de Memorias Cache en Sistemas de Tiempo Real.** Tesis doctoral. Departamento de Ingeniería de Sistemas y Automática. 1997

[Caironi96] P. V. C. Caironi, L. Mezzalira and M. Sami. **Context Reorder Buffer: an Architectural Support for Real-Time Processing on RISC Architectures.** Proceedings of the 8th Euromicro Workshop on Real-Time Systems. pp. 262-270. 1996

[Chirivella99] Vicente Chirivella González. **Estadística.** Universidad Politécnica de Valencia. 1999

[Colin00] A. Colin and I. Puaut. **Worst Case Execution Time Analysis for a Processor with Branch Prediction.** Real-Time Systems, Vol. **18**, Num. 2. pp. 249-274. April 2000

[Corti00] Matteo Corti, Roberto Brega and Thomas Gross. **Approximation of Worst-Case Execution Time for Preemptive Multitasking Systems.** ACM SIGPLAN Workshop on Languages, Compilers, and Tools for Embedded Systems. pp. 178-198. Vancouver, Canada June 2000

[Davis95b] Robert Davis, Sasikumar Punnekkat, Neil Audsley and Alan Burns. **Flexible Scheduling for Adaptable Real-Time Systems.** Proceedings of IEEE Real-Time Technology and Applications Symposium. pp. 230-239. Chicago, Illinois May 1995

[Ferdinand99] Christian Ferdinand, Florian Martin, Reinhard Wilhelm and Martin Alt. **Cache Behavior Prediction by Abstract Interpretation.** Science of Computer Programming, Vol. **35**, Num. 2/3. pp. 163-189. November 1999

[Ferrari76] Domenico Ferrari. **The Improvement of Program Behavior.** IEEE Computer, Vol. **9**, Num. 11. November 1976

[Flynn87] M. J. Flynn, C. L. Mitchell and J. M. Mulder. **And Now a Case for More Complex Instruction Sets.** IEEE Computer, Vol. **20**, Num. 9. pp. 71-83. September 1987

[Furht91] B. Furht, D. Grostick, D. Gluch, G. Rabbat, J. Parker and M. McRoberts. **REAL-TIME UNIX SYSTEMS Design and Applicacion Guide.** Kluwer Academic Publishers. 1991

[Goldberg89] David E. Goldberg. **Genetic Algorithms in Search, Optimization and Machine Learning.** Addison-Wesley Pub. Co. 1989

[Goodman83] James R. Goodman. **Using Cache Memory to Reduce Processor-Memory Traffic.** Proceedings of the 10th International Symposium of Computer Architecture. Pp. 124-131. June 1983

[Hamacher87] V. C. Hamacher, Z. G. Vranesic and S. G. Zaky. **Organizacion de Computadoras.** Mc Graw Hill. 1987

[Harter84] P. K. Harter. **Response Times in Level Structured Systems.** CU-CS-269-94 Departament of Computer Science. 1984

[Healy96] C.A. Healy, David Whalley and Marion G. Harmon. **Integrating the Timing Analysis of Pipelining and Instruction Caching.** Proceedings of the 16th IEEE Real-Time Systems Symposium. pp. 288-297. Pisa, Italy December 1996

[Healy99] Christopher A. Healy, Robert D. Arnold, Frank Mueller, David B. Whalley and Marion G. Harmon. **Bounding Pipeline and Instruction Cache Performance.** IEEE Transaction on Computers Vol. **48**, Num. 1. pp. 53-70. February 1999

[Healy00] Christopher A. Healy, Mikael Sjödin, Viresh Rustagi, David B. Whalley and Robert van Engelen. **Supporting Timing Analysis by Automatic Bounding of Loop Iterations.** Real-Time Systems, Vol. **18**, Num. 2/3. pp. 129-156. May 2000

[Hennessy96] John L. Hennessy and David A. Patterson. **Computer Architecture: A Quantitative Approach.** Morgan Kaufmann Publishers, Inc. 1996

[Hill84] Mark D. Hill and Alan Jay Smith. **Experimental Evaluation of On-Chip Microprocessor Cache Memories.** Proceedings of the 11th International Symposium of Computer Architecture. pp. 158-166. Ann Arbor MI June 1984

[Hill88] Mark D. Hill. **A Case for Direct-Mapped Caches.** IEEE Computer, Vol. **21**, Num. 12. pp. 25-40. December 1988

[Hillary02] Nat Hillary and Ken Madsen. **You Can't Control what you Can't Measure, or Why it's Close to Impossible to Guarantee Real-Time Software Performance on a CPU with On-Chip Cache.** 2nd Euromicro Workshop on Worst Case Execution Time Analysis. Vienna, Austria June 2002

[Huang95] T. Y. Huang and J. Liu. **Predicting the worst-case execution time of the concurrent execution of instructions and cycle-stealing DMA I/O operations.** Second ACM SIGPLAN Workshop on Languages, Compilers and Tools for Real-Time Systems. La Jolla, CA June 1995

[Hur96] Y. Hur and Company. **Worst Case Timing Analyis of RISC Processors: R3000/R3010 Case Study.** Proceedings of the 16th IEEE Real-Time Systems Symposium. pp. 308-319. Pisa, Italy December 1996

[IDT00] Integrated Device Technolgy, inc. **RC64574/RC64575 User Reference Manual.** March 2000

[Intel89] Intel Corporation. **80960CA User's Manual.** 80960CA User's Manual. 1989

[Jain91] Raj Jain. **The Art of Computer Systems Performance Analysis.** John Wiley & Sons, Inc. 1991

[Joseph86] M. Joseph and P. Pandya. **Finding Response Times in a Real-Time System.** The Computer Journal (British Computer Society), Vol. **29**, Num. 5. pp. 390-395. October 1986

[Jouppi89] Norman P. Jouppi and David W. Wall. **Available Instruction-Level Parallelism for Superscalar and Superpipelined Machines.** Third International Conference on Architectural Support for Programming Languages and Operating Systems. pp. 272-281. April 1989

[Jourdan95a] Stéphan Jourdan, Pascal Sainrat and Daniel Litaize. **Exploring Configurations of Functional Units in an Out-of-Order Superscalar Processor.** Procceedings of the 22nd Annual International Symposium on Computer Architecture. pp. 117-125. Santa Margherita Ligure, Italy June 1995

[Junghickel99] Dieter Junghickel. **Graphs, networks and algorithms.** Springer. 1999

[Kenny90] Kevin B. Kenny and Kwei-Jay Lin. **A Measurement-Based Performance Analyzer for Real-Time Programs.** University of Illinois at Urbana-Champaign Report UIUCDCS-R-90-1606. 1990

[Kim96] Sung-Kwan Kim, S. L. Min and R. Ha. **Efficient Worst Case Timing Analysis of Data Caching.** Proceedings of the 2nd IEEE Real-Time Technology and Applications Symposium. pp. 230-240. 1996

[Kirk89] David B. Kirk. **SMART (Strategic Memory Allocation for Real-Time) Cache Design.** Proceedings of the 10th IEEE Real-Time Systems Symposium. pp. 229-237. December 1989

[Kirk90] David B. Kirk and J.K. Strosnider. **SMART (Strategic Memory Allocation for Real-Time) Cache Design using the MIPS R3000.** Proceedings of the 11th IEEE Real-Time Systems Symposium. pp. 322-330. Lake Buena Vista, FL December 1990

[Kirk91a] David B. Kirk. **Allocating SMART Cache Segments for Schedulability.** Foundations of Real-Time Computing: Scheduling and Resource Management. pp. 251-275. 1991

[Kirk91b] David B. Kirk. **SMART Cache Design for Predictable Multi-Processing.** Workshop on Architecture Aspects of Real-Time Systems at the Real-Time Systems Symposium. San Antonio, TX December 1991

[Kogge81] P. M. Kogge. **The architecture of Pipelined Computers.** Heisphere Publishing Corp. 1981

[Kountouris96] Apostolos A. Kountouris. **Safe and Efficient Elimination of Infeasible Execution Paths in WCET Estimation.** Proceedings of 3rd International Workshop on Real-Time Computing Systems and Applications. Seoul, Korea October 1996

[Krick91] Rober F. Krick and Apostolos Dollas. **The Evolution of Instruction Sequencing.** IEEE Computer, Vol. **24**, Num. 4. pp. 5-15. April 1991

[Lee96] Chang Gun Lee, Joosun Hahn, Yang-Min Seo, Sang Lyul Min, Rhan Ha, Seongsoo Hong, Chan Yun Park, Minsuk Lee and Chong Sang Kim. **Analysis of Cache-related Preemption Delay in Fixed-priority Preemptive Scheduling.** Proceedings of the 17th IEEE Real-Time Systems Symposium. pp. Whasington D.C. Whasington D.C. December 1996

[Lee96] C. G. Lee, J. Hahn, Y. M. Seo, S. L. Min, R. Ha, S. Hong, C. Y. Park, M. Lee and C. S. Kim. **Analysis of Cache-Related Preemption Delay in Fixed-Priority Preemptive Scheduling.** Proceedings of the 17th IEEE Real-Time Systems Symosium. pp. 264-274. 1996

[Lee97a] C. G. Lee, J. Hahn, Y. M. Seo, S. L. Min, R. Ha, S. Hong, C. Y. Park, M. Lee and C. S. Kim. **Analysis of Cache-Related Preemption Delay in Fixed-Priority Preemptive Scheduling.** IEEE Transactions on Computers, Vol. **47**, Num. 6. pp. 700-713. June 1997

[Lee97b] C. G. Lee, J. Hahn, Y. M. Seo, S. L. Min, R. Ha, S. Hong, C. Y. Park, M. Lee and C. S. Kim. **Enhaced Analysis of Cache-Related Preemption Delay in Fixed-Priority Preemptive Scheduling.** Proceedings of the 18th IEEE Real-Time Systems Symposium. pp. 187-198. December 1997

[Lee99] S. Lee, S. L. Min, C. S. Kim, C.-G. Lee, and M. Lee. **Cache-Conscious Limited Preemptive Scheduling.** Real-Time Systems, Vol. **17**, Num. 2/3. pp. 257-282. November 1999

[Lee01] C. G. Lee, J. Hahn, Y. M. Seo, S. L. Min, R. Ha, S. Hong, C. Y. Park, M. Lee and C. S. Kim. **Bounding Cache-Related Preemption Delay for Real-Time Systems.** IEEE Transactions on Software Engineering, Vol. **27**, Num. 9. pp. 805-826. September 2001

[Lehoczky89] J. P. Lehoczky, L. Sha and Y. Ding. **The Rate-Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior.** Proceedings of the 10th IEEE Real-Time Systems Symposium. pp. 166-171. December 1989

[Leoczky87] J. P. Leoczky, Lui Sha and J. K. Strosnider. **Enhancing Aperiodic Responsiveness in Hard Real-Time Environment.** Proceedings of the 8th IEEE Real-Time Systems Symposium. pp. 261-270. December 1987

[Li93] Yau-Tsun Steven Li and Andrew Wolfe. **An Experimental Implementation of Software-Based Cache Partitioning.** Computer Engineering Technical Report CE-A93-2. pp. 1-18. Princenton University, NJ. June 1993

[Li95] Y. Li and S. Malik. **Performance analysis of embedded software using implicit path enumeration.** Second ACM SIGPLAN Workshop on Languages, Compilers and Tools for Real-Time Systems. La Jolla, CA June 1995

[Li95b] Y. T. Li, S. Malik and A. Wolfe. **Performance Estimation of Embedded Software with Instruction Cache Modelling.** Proceedings of the IEEE/ACM International Conference on Computer-Aided Design. pp. 380-387. 1995

[Li96] Y. T. Li, S. Malik and A. Wolfe. **Efficient Microarchitecture Modeling and Path Analysis for Real-Time Software.** Proceedings of the 16th IEEE Real-Time Systems Symposium. pp. 298-307. Pisa, Italy December 1996

[Li96b] Y. T. Li, S. Malik and A. Wolfe. **Cache Modeling for Real-Time Software: Beyond Direct Mapped Instruction Caches.** Proceedings of Real-Time Systems Symposium. 1996

[Liedtke97] J. Liedtke, H. Härtig and M. Hohmuth. **OS-Controlled Cache Predictability for Real-Time Systems.** Proceedings of the 3rd IEEE Real-Time Technology and Applications Symposium. Montreal, Canada June 1997

[Lilja94] David J. Lilja and Peter L. Bird. **The Interaction of Compilation Technology and Computer Architecture.** Kluwer Academic Publishers. 1994

[Lim94a] Sung-Soo Lim, Sang Lyul Min, Minsuk Lee, Chan Yun Park, Young Hyun Bae, Heonshik Shin and Chong Sang Kim. **An Accurate Worst Case Timing Analysis Technique for Real-Time Systems.** Proceedings of the Workshop on Architectures for Real-Time Applications, International Symposium on Computer Architectures. Chicago April 1994

[Lim94b] Sung-Soo Lim, Young Hyun Bae, Gyu Tae Jang, Byung Do Rhee, Sang Lyul Min, Chan Yun Park, Heonshik Shin, Kunsoo Park and Chong Sang Kim. **An Accurate Worst Case Timing Analysis Technique for RISC Processors.** Proceedings of the 15th IEEE Real-Time Systems Symposium. pp. 97-108. San Juan, Puerto Rico December 1994

[Liu73] C.L.Liu and James W.Layland. **Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment.** Journal of the ACM, Vol. **20**, Num. 1. pp. 46-61. 1973

[Locke92] C. D. Locke. **Software Architecture for Hard Real-Time Applications : Cyclic Executives vs. Fixed Priority Executives.** Journal of Real-Time Systems, Vol. **4** pp. 37-53. March 1992

[Luculli97] G. Luculli, M. Di Natale. **A Cache-Aware Scheduling Algorithm For Embedded Systems.** Proceedings of the 18th IEEE Real-Time Systems Symposium. pp. 199-209. San Francisco, CA December 1997

[Lundqvist99] Thomas Lundqvist and Per Stenström. **Timing Anomalies in Dynamically Scheduled Microprocessors.** Proceedings of the 20th IEEE Real-Time Systems Symposium. pp. 12-21. Phoenix, AZ December 1999

[Martí98] A. Martí Campoy. **Las mejoras estructurales de los modernos procesadores en los sistemas de tiempo real.** Trabajo de doctorado de 2 créditos. Universidad Politécnica de Valencia Septiembre 1998

[Martí00] A. Martí Campoy, X. Molero, A. Perles, F. Rodríguez and J.V. Busquets. **Combined Intrinsic-Extrinsic Cache Analysis for Preemptive Real-Time Systems.** Proceedings of the 25th IFAC Workshop on Real-Time Programming WRTP'2000. pp. 49-55. Pergamon. Palma de Mallorca May 2000

[Martí01a] A. Martí Campoy, A. Perez Jimenez, A. Perles Ivars and J.V. Busquets Mataix. **Using Genetic Algorithms in Content Selection for Locking-Caches.** Proceedings of the IASTED International Symposia Applied Informatics. pp. 271-276. Acta Press. Innsbruck, Austria February 2001

[Martí01b] A. Marti Campoy, A. Perles Ivars and J. V. Busquets Mataix. **Using Locking Caches In Preemptive Real-Time Systems.** Proceedings of the 12th IEEE Real Time Congress on Nuclear And Plasma Sciencies. pp. 157-159. Valencia June 2001

[Martí01c] A. Martí Campoy, A. Perles Ivars and J. V. Busquets Mataix. **Static Use Of Locking Caches In Multitask, Preemptive Real-Time Systems.** Proceedings of the IEEE Real-Time Embedded System Workshop. London, UK December 2001

[Martí02] A. Martí Campoy, A. Perles Ivars and J. V. Busquets Mataix. **Dynamic Use Of Locking Caches In Multitask, Preemptive Real-Time Systems.** Proceedings of the 15th World Congress of the International Federation of Automatic Control. Elsevier Science. Barcelona July 2002

[Martí03a] A. Martí Campoy, S. Sáez, A. Perles and J.V.Busquets. **Schedulability Analysis in the EDF Scheduler with Cache Memories.** The 9th International Conference on Real-Time and Embedded Computing Systems and Applications. Tainan, TW February 2003

[Martí03b] Antonio Martí Campoy, Angel Perles, Sergio Sáez and Jose Vicente Busquets-Mataix. **Performance Comparison of Use of Locking Caches Under Static and Dynamic Schedulers.** Proceedings of the 27th IFAC/IFIP/IEEE Workshop in Real-Time Programming. Lagow, Poland May 2003

[Mitchell96] Melanie Mitchell. **An Introduction to Genetic Algorithms.** Mit Press. 1996

[Mogul91] Jeffrey C. Mogul and Anita Borg. **The Effect of Context Switches on Cache Performance.** Fourth International Conference on Architectural Support for Programming Languages and Operating Systems. pp. 75-84. Santa Clara, CA 1991

[Motorola99] Motorola Inc. **MPC7400 RISC Microprocessor User's Manual.** 1999

[Mueller95] Frank Mueller. **Compiler Support for Software-Based Cache Partitioning.** Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation. La Jolla, CA June 1995

[Mueller95b] Frank Mueller and David B. Whalley. **Fast Instruction Cache Analysis via Static Cache Simulation.** Proceedings of the 28th Annual Simulation Symposium. pp. 105-114. 1995

[Mueller97] Frank Mueller. **Generalizing Timing Predictions to Set-Associative Caches.** In Proceedings of the 9th Euromicro Workshop on Real-Time Systems. pp. 64-71. June 1997

[Mueller97b] Frank Mueller. **Timing Predictions for Multi-Level Caches.** In ACM SIGPLAN Workshop on Leguaje, Compiler and Tool Suport for Real-Time Systems. pp. 29-36. June 1997

[Mueller98] F. Mueller and J. Wegener. **A Comparison of Static Analysis and Evolutionary Testing for the Verification of Timing Constraints.** Proceedings of the 4th IEEE Real-Time Technology and Applications Symposium. pp. 179-188. June 1998

[Mueller00] F. Mueller. **Timing Analysis for Instruction Caches.** Real-Time Systems Journal, Vol. **18**, Num. 2/3. pp. 209-239. May 2000

[Nilsen95a] Kelvin Nilsen. **Real-Time is No Longer a Small Specialized Niche.** Proceedings of the Fifth Workshop on Hot Topics in Operating Systems (HotOS-V). Orcas Island, Washington 1995

[Panda97] Preeti Ranjan Panda, Nikil D. Dutt and Alexandru Nicolau. **Efficient Utilization of Scratch-Pad Memory in Embedded Processor Applications.** The 13th European Design and Test Conference. pp. 7-11. Paris, France March 1997

[Park91] Chang Yun Park and Alan C. Shaw. **Experiments with a Program Timing Tool Based on Source-Level Timing Schema.** IEEE Computer, Vol. **24**, Num. 5. pp. 48-57. May 1991

[Park93] Chang Yun Park. **Predicting Program Execution Times by Analyzing Static and Dynamic Program Paths.** Real-Time Systems, Vol. **5**, Num. 1. pp. 31-62. 1993

[Patterson85] David A. Patterson. **Reduced Instruction Set Computers.** Communications of the ACM, Vol. **28**, Num. 1. pp. 8-21. January 1985

[Patterson00] David A. Patterson and John L. Hennessy. **Estructura y Diseño de Computadores: Interficie Circuitería/Programación.** Editorial Reverté, S.A. 2000

[Puaut02] I. Puaut and D. Decotigny. **Low-Complexity Algorithms for Static Cache Locking in Multitasking Hard Real-Time Systems.** Proceedings. of the 23rd IEEE International Real-Time Systems Symposium. Austin, TX December 2002

[Rajkumar88] R. Rajkumar, Lui Sha, J. P. Lehoczky and K. Ramamithram. **An Optimal Priority Inheritance Protocol for Real-Time Synchronization.** COINS Technical Report 88-98. October 1988

[Rajkumar89] R. Rajkumar, Lui Sha and J. P. Lehoczky. **An Experimental Investigation of Synchronization Protocols.** Proceedings of the 6th IEEE Workshop on Real-Time Operating Systems and Software. pp. 11-17. May 1989

[Ramamritham94] Krithi Ramamritham and John A. Stankovic. **Scheduling Algorithms and Operating Systems Support for Real-Time Systems.** Proceedings of The IEEE, Vol. **82**, Num. 1. pp. 55-67. January 1994

[Rhee94a] B. Rhee, S. L. Min and H. Shin. **Retargetable Timing Analyzer for Risc Processors.** Proceedings of the 1st International Workshop on Real-Time Computing Systems and Applications. pp. 76-79. 1994

[Rhee94b] B. Rhee, S. Lim, S. L. Min, H. Shin and C. Y. Park. **Issues of Advanced Architectural Features in the Design of a Timing Tool.** Proceedings of the 11th Workshop on Real-Time Operating Systems and Software. pp. 59-62. 1994

[Ripoll96] I. Ripoll, A. Crespo and A. Mok. **Improvements in feasibility testing for real-time tasks.** The Journal of Real-Time Systems, Vol. **11** pp. 19-39. 1996

[Romer94] T. H. Romer, J. B. Chen, D. Lee and B. N. Bershad. **Dynamic Page Mapping Policies for Cache Conflict Resolution on Standard Hardware.** Proceedings of the First USENIX Symposium on Operating Systems Design and Implementation. pp. 255-256. Monterey, CA November 1994

[Romero93] Rafael Romero Villafranca y Luisa Rosa Zúnica Ramajo. **Estadística: Conceptos básicos, análisis de la varianza, diseño de experimentos, modelos de regresión,.** Valencia 1993

[Rymarczyk82] James W. Rymarczyk. **Coding Guidelines for Pipelined Processors.** First International Conference on Architectural Support for Programming Languages and Operating Systems. pp. 12-19. 1982

[Sarkar89] Vivek Sarkar. **Determining Average Program Execution Times and their Variance.** ACM SIGPLAN Conference on Programming Language Design and Implementation. pp. 298-312. June 1989

[Sasinowski93] J. E. Sasinowski and J. K. Strosnider. **A Dynamic-Programming Algorithm for Cache Memory Partitioning for Real-Time Systems.** IEEE Transactions on Computers, Vol. **42**, Num. 8. pp. 997-1001. August 1993

[Sha87] Lui Sha, R. Rajkumar and J. P. Lehoczky. **Priority Inheritance Protocols: an Aproach to Real-Time Synchronization.** Technical Report CMU-CS-87-181. December 1987

[Sha90b] Lui Sha, R. Rajkumar and J. P. Lehoczky. **Priority Inheritance Protocols: an Aproach to Real-Time Synchronization.** IEEE Transactions on Computers, Vol. **39**, Num. 9. pp. 1175-1185. September 1990

[Shaw89] A. Shaw. **Reasoning about time in higher-level language software.** IEEE Transactions on Software Engineering, Vol. **15**, Num. 7. pp. 875-889. July 1989

[Smith82] Alan Jay Smith. **Cache Memories.** Computing Surveys, Vol. **14**, Num. 3. pp. 473-530. September 1982

[So97] A. T. P. So and W. L. Chan. **Comprehensive Dynamic Zoning Algorithms.** Elevator World, Vol. **XLV**, Num. 8. pp. 99-109. Elevator World, Inc. September 1997

[Stankovic88] John A. Stankovic. **Misconceptions about Real-Time Computing: a Serious Problem for Next-Generation Systems.** IEEE Computer, Vol. **21**, Num. 10. pp. 10-19. October 1988

[Steininger91] A. Steininger and H. Schweinzer. **Can the Advantages of RISC be Utilized in Real-Time Systems?** Proceedings of the 3th Euromicro Workshop on Real-Time Systems. pp. 30-35. 1991

[Stone93] Harold S. Stone. **High-Performance Computer Architecture.** Addison Wesley. 1993

[Temam94] O. Temam, C. Fricker and W. Jalby. **Cache Interference Phenomena.** SIGMETRICS. pp. 261-271. Santa Clara, CA May 1994

[Tindell92a] K. Tindell. **An Extendible Approach for Analysing Fixed Priority Hard Real-Time Tasks.** Department of Computer Science, YCS 189. pp. 1-16. University of York, UK 1992

[Tindell92c] K. Tindell. **Using Offset Information to Analyse Static Priority Pre-Emptively Scheduled Task Sets.** Dept of Computer Science, YCS 182. pp. 1-17. University of York, UK August 1992

[Tindell94a] K. Tindell. **Adding Time-Offsets To Schedulability Analysis.** Dept of Computer Science, YCS 221. pp. 1-28. University of York, UK January 1994

[Tindell94b] K. Tindell, A. Burns and A.J. Wellings. **An Extendible Approach for Analysing Fixed Priority Hard Real-Time Tasks.** Real-Time Systems, Vol. **6**, Num. 2. pp. 133-151. March 1994

[Tomiyama97] H. Tomiyama and H. Yasuura. **Code Placement Techniques for Cache Miss Rate Reduction.** ACM Trans. Design Automation of Electronic Systems, Vol. **2**, Num. 4. pp. 410-429. October 1997

[White97] R. T. White, F. Mueller, C. A. Healy, D. V. Whalley and M. G. Harmon. **Timing Analysis for Data Caches and Set-Associative Caches.** Proceedings of the 3rd IEEE Real-Time Technology and Applications Symposium. pp. 192-202. June 1997

[Wolfe93] Andrew Wolfe. **Software-Based Cache Partitioning for Real-Time Applications.** Proceedings of the Third International Workshop on Responsive Computer Systems. September 1993

[Zhang93] N. Zhang, A. Burns and M. Nicholson. **Pipelined Processors and Worst Case Execution Times.** Journal of Real-Time Systems, Vol. **5**, Num. 4. pp. 319-343. October 1993