ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

INGENIERÍA TÉCNICA EN INFORMÁTICA DE SISTEMAS

EVALUACIÓN DE POLÍTICAS DE REEMPLAZO MULTINIVEL EN CACHÉS WEB

Realizado por

JESÚS PULIDO VÁZQUEZ

Dirigido por

FRANCISCO JAVIER GONZÁLEZ CAÑETE

Departamento

TECNOLOGÍA ELECTRÓNICA

Tutor

GERARDO BANDERA BURGUEÑO

Departamento

ESTRUCTURA DE COMPUTADORES

UNIVERSIDAD DE MÁLAGA

MÁLAGA, Febrero 2006

UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

INGENIERÍA TÉCNICA EN INFORMÁTICA DE SISTEMAS

Reunido el tribunal examinador en el día de la fecha, constituido por:

Presidente Dº/Dª. _____

Secretario Dº/Dª. _____

Vocal Dº/Dª. _____

para juzgar el proyecto Fin de Carrera titulado:

_____

_____

del alumno Dº/Dª. _____

dirigido por Dº/Dª. _____

ACORDÓ POR _____ OTORGAR LA CALIFICACIÓN DE _____

Y PARA QUE CONSTE, SE EXTIENDE FIRMADA POR LOS COMPARECIENTES DEL TRIBUNAL, LA PRESENTE DILIGENCIA.

Málaga, a de del 200__

El Presidente El Secretario El Vocal

Fdo: Fdo: Fdo:

A mi madre.

Índice de contenido

Índice de figuras

Índice de tablas

1. Introducción

El uso de memorias caché es habitual desde hace mucho tiempo en todo tipo de sistemas para mejorar el rendimiento de los mismos. En los últimos años, el gran aumento del uso de Internet hace que el uso de técnicas de caché sean aplicadas para mejorar tanto el rendimiento como la calidad del servicio a los usuarios. Por ello, este estudio está centrado en analizar un tipo concreto de cachés, las cachés Web usadas en los servidores proxy de Internet, también llamadas proxy cachés [1].

Un proxy es un tipo de servidor Web, que tras recibir las peticiones Web de los usuarios, accede a los servidores remotos para dar una respuesta a dichas peticiones. Además, añade funcionalidades de seguridad adicional, como filtrado de tráfico, tanto entrante como saliente. Normalmente, el proxy es usado por usuarios dentro de una subred, lo cual hace posible aplicar eficientemente técnicas de caché para dar respuesta a las peticiones Web.

Las cachés Web pueden ser instaladas en tres lugares: en los usuarios, en los servidores Web o en los servidores proxy. Éstos distintos emplazamientos para las cachés Web pueden observarse en la Figura 1.

En el caso de los usuarios, la existencia de una caché reduce el tráfico a través de la red y el tiempo de respuesta, pero tiene el problema de la inconsistencia de los datos. En los servidores, reduce la el número de operaciones de entrada-salida, pero no reduce el tráfico ni prácticamente el tiempo de respuesta, y además conlleva un aumento de la carga del sistema.

En el caso de ser usadas en un proxy, que es el caso que se estudiará, tiene las mismas ventajas que en el usuario y además el problema de la inconsistencia de los datos se reduce, pero puede aparecer un problema de tiempo de respuesta causado por el mantenimiento de la caché.

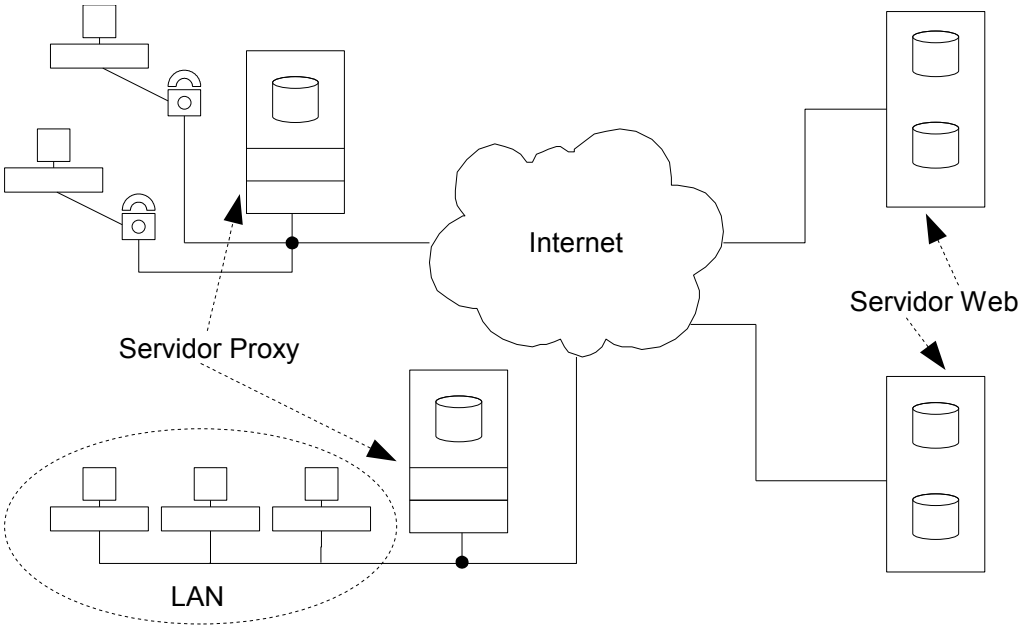

Figura 1: Tipos de cachés Web

Al usar una caché, el proxy almacena los documentos que ha enviado a los usuarios para que cuando vuelvan a pedirle esos mismos documentos, ya sea el mismo usuario u otro, no tenga que acceder a Internet a buscarlos, sino que directamente les devuelva los documentos desde dicha caché. Como ya se ha señalado anteriormente, con esto se consigue ganar no sólo en velocidad de respuesta al usuario, sino también en un mayor aprovechamiento del ancho de banda de la conexión a la red. Además, en el caso en el que reciba una petición para acceder a un servidor remoto que en ese momento no está en funcionamiento, si los documentos están en caché, podrá responder a dicha petición.

El problema aparece cuando los documentos de una petición no están en caché y no existe más espacio disponible en la misma, y por lo tanto, hay que eliminar algún documento existente en la caché para que ésta pueda albergar los nuevos documentos. De esto se encargan las políticas de reemplazo, algunas de las cuales serán objeto de estudio en este proyecto.

Las cachés Web tienen algunas características especiales que las diferencian de las demás. Por ejemplo, el tamaño de los documentos a procesar varía enormemente, ya que puede ir desde tan sólo algunos bytes en un documento de texto o HTML, a varios cientos de megabytes en archivos de audio o vídeo. Otro punto a tener en cuenta es que los documentos en

caché sólo pueden ser leídos por los usuarios, y no modificados, ya que los documentos proceden de los servidores Web.

En la actualidad, las soluciones software más extendidas en los proxy cachés para administrar cachés Web se basan en el algoritmo LRU (*Least Recently Used*), un algoritmo sencillo que reemplaza los documentos que no han sido requeridos en un largo periodo de tiempo, pero esto tiene sus limitaciones dada la complejidad del problema, ya que no tiene en cuenta otros factores como la frecuencia con la que han sido requeridos los documentos o el tamaño de los mismos, lo cual conlleva que no sea la mejor solución.

Existe una gran cantidad de políticas de reemplazo, y con la realización de este proyecto se tratará de sacar conclusiones acerca de qué políticas son más convenientes, concretamente se estudiarán las políticas de reemplazo multinivel o multicola, que son las que usan más de una lista o cola para almacenar los documentos. En concreto, se van a analizar un grupo de políticas, todas ellas variantes de LRU, que tienen en cuenta no sólo el tiempo que hace que se accedió al documento pedido, sino también la frecuencia con la que fueron solicitados, e incluso el tamaño de los documentos, además de basarse en la utilización de múltiples colas, cada una de ellas con una prioridad distinta. Las políticas de reemplazo y sus características principales son las siguientes:

- SLRU (*Segmented LRU*): utiliza dos colas, en las cuales guarda los documentos según la importancia de los mismos.
- 2Q (*Two Queue*): utiliza una cola para los elementos más importantes, a la que sólo acceden los elementos que han sido referenciados más de una vez en un cierto periodo de tiempo.
- FBR (*Frequency-Based Replacement*): elimina de entre los documentos que no han sido utilizados en un cierto tiempo, el que haya sido referenciado menos veces.
- LRFU (*Least Recently/Frequently Used*): ordena los elementos según una función que otorga un valor a cada documento dependiendo de cuándo fue referenciado por última vez y el número de referencias del mismo, y elimina el que tiene un menor valor.
- ML-LRU (*Multilevel LRU*): según una función decisión almacena cada documento en una de sus dos colas, una con más importancia que la otra.

- C-LRU (*Class-Based LRU*): realiza particiones de la caché según el tamaño de los objetos.

En estudios anteriores aplicados a varios entornos (comerciales, bases de datos, etc.) se ha mostrado que estas políticas de reemplazo multinivel consiguen normalmente un mayor número de aciertos en la caché que LRU. Para poder simular el comportamiento de dichas políticas en una caché Web y comparar fácilmente los resultados de dichas simulaciones, se ha realizado una aplicación capaz de evaluarlas y compararlas para distintos conjuntos de peticiones (trazas) realizadas al servidor proxy. Con dicho simulador y tras realizar las simulaciones correspondientes se han comparado los resultados obtenidos atendiendo a las métricas de rendimiento habituales: los aciertos en caché (*Hit Rate*) y los aciertos en bytes (*Byte Hit Rate*).

En el segundo capítulo se explican detalladamente las características de cada una de las políticas de reemplazo que se evaluarán. En el capítulo tercero se muestra el funcionamiento de la aplicación creada para realizar las simulaciones, y en el cuarto los detalles de implementación de la misma. Los resultados de dichas simulaciones se muestran, comparan y estudian en el capítulo quinto. El capítulo sexto queda reservado para las conclusiones y las líneas futuras del proyecto.

2. Políticas de reemplazo

En este capítulo se analizarán las políticas de reemplazo que se evaluarán posteriormente en el capítulo 5. Se explicará detalladamente el funcionamiento de cada una de las políticas, incluyendo en la explicación tanto los principios en los que se basan para intentar conseguir un mayor rendimiento de la caché como la forma en la que estructuran la caché, algo especialmente importante en las políticas multinivel, así como los parámetros de cada una de ellas.

En primer lugar se presentará la política *Least Recently Used* (LRU), que aunque no es multinivel, se incluye por ser la más usada para gestionar cachés y para compararla con las políticas multinivel. Posteriormente, se analizarán las políticas de reemplazo multinivel SLRU, 2Q, FBR, LRFU, ML-LRU y C-LRU.

2.1. LRU (*Least Recently Used)*

Esta política aprovecha el principio de la localidad temporal, para ello, los elementos están ordenados en caché según el tiempo que haga que fueron referenciados, es decir, el elemento referenciado más recientemente está en primera posición, y el menos reciente en la última. Para reemplazar un elemento se escoge el último de la caché, es decir, el que ha estado más tiempo en caché sin ser referenciado.

Para mantener el orden se suele utilizar una lista en la que se van introduciendo los elementos por el principio, y cuando hay un acierto en la caché, se mueve el elemento de su posición actual a la primera posición de la lista. Es fácil de implementar, pero para cachés muy grandes tiene un gran coste computacional.

2.2. SLRU (*Segmented Least Recently Used)*

En un estudio anterior [2] sobre técnicas de caché en discos se muestra que los documentos que son referenciados al menos dos veces tienden a ser reutilizados varias veces más. El problema es que usando LRU, la caché puede llenarse de documentos que son utilizados una sola vez, quitandole el sitio a estos documentos que son usados con más frecuencia. Con esta idea base se creó esta política, que divide la caché en dos segmentos. Aquí se aplicará dicha idea a las cachés Web.

La caché consta de dos segmentos o partes, es decir, estamos ante una política de reemplazo multinivel. Al primer segmento se le llama segmento protegido y al segundo, de prueba. Cada segmento está ordenado como LRU. Puede verse un esquema de la política en la Figura 2.

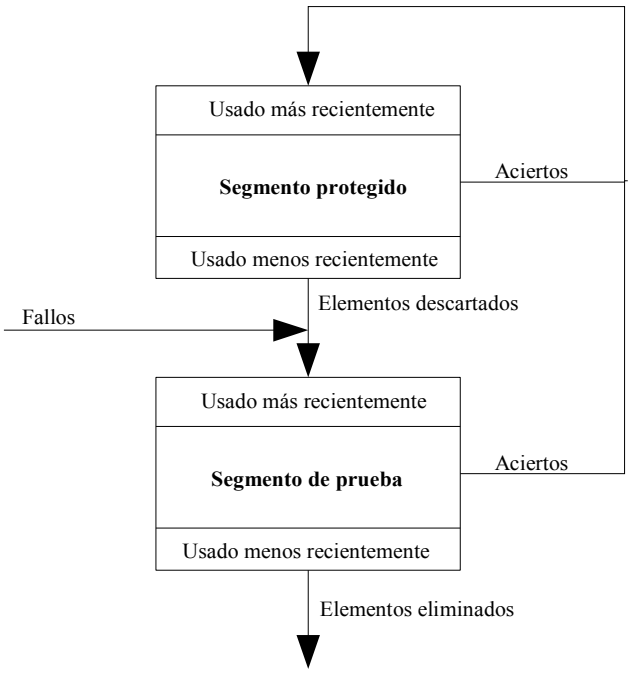

Figura 2: Esquema SLRU

Cuando hay un fallo en caché, el nuevo elemento se añade al comienzo de la parte de prueba. Cuando hay un acierto en cualquier posición, el elemento se mueve al inicio de la par-

te protegida. Con esto se consigue colocar en la parte protegida sólo a los elementos que hayan sido referenciados al menos dos veces.

El tamaño de la parte de prueba puede ser tan grande como el tamaño total de la caché, mientras que la parte protegida tiene un tamaño máximo delimitado que viene dado por un parámetro de la política SLRU, que podrá ser variado según convenga. Por tanto en caso de que la parte protegida esté llena, se moverá el último elemento de la misma a la parte de prueba, y en el caso de que el tamaño de la caché sea sobrepasado, se eliminará el último elemento de la parte de prueba.

Por tanto, SLRU plantea de forma sencilla una manera de organizar los elementos colocando en una parte a los que serán, seguramente, referenciados más veces, y en otra a los elementos que serán menos utilizados, eliminando antes estos últimos. Además, hay que destacar que los elementos de la parte protegida, en el caso de no ser requeridos en un cierto periodo de tiempo, perderán su posición en dicha parte de la caché para pasar a la parte de prueba, con lo que no habrá problemas de permanencia de un elemento en caché durante mucho tiempo si no es utilizado con cierta frecuencia. Como se verá en el capítulo quinto, esta política, sencilla pero efectiva, obtiene buenos resultados en las simulaciones que se han realizado.

2.3. 2Q (*Two Queue)*

LRU, por su propia forma de funcionar, para almacenar nuevos documentos, puede eliminar otros que a la larga serán utilizados en más ocasiones que estos, y que por tanto, quizá sería más conveniente mantenerlos en caché pensando en el futuro. Para mejorar la efectividad de una caché evitando que se produzcan estas situaciones, se propuso en [3] la política 2Q, que divide la caché en dos colas, Am y A1in. Am es la cola principal, donde se almacenan los elementos con mayor probabilidad de volver a ser usados en el futuro, para ello, en la primera referencia a un documento, este se guarda en la cola A1in, que es una cola FIFO. Si

un elemento de A1in vuelve a ser referenciado, este nuevo acceso sobre el documento se considera como una referencia correlativa en un corto periodo de tiempo y que posiblemente este documento no volverá a ser utilizado en el futuro, y por ello no se modifica su posición dentro de la caché, ya que dicho documento no se considera lo suficientemente importante como para pasar a la cola principal Am.

Cuando los elementos de A1in van siendo descartados, no se eliminan completamente, sino que se guarda información sobre los mismos (Identificadores: nombre, tamaño, etc., pero no los datos) en una cola FIFO especial llamada A1out. Con la utilización de esta nueva cola se intenta detectar los elementos que serán referenciados más veces en un largo periodo de tiempo, para así aumentar el rendimiento de la caché.

Si un elemento cuyos identificadores están en A1out es referenciado, se eliminan estos identificadores, y el documento vuelve a ser leído, en este caso, de Internet, para ser colocado en la cabeza de la cola Am. Asimismo, si se produce un acierto en Am, el documento referenciado es movido a la cabeza de la misma cola Am, para mantener los elementos ordenados según el tiempo que haga que han sido referenciados. En la Figura 3 se muestran las distintas partes de la caché y el funcionamiento de la misma.

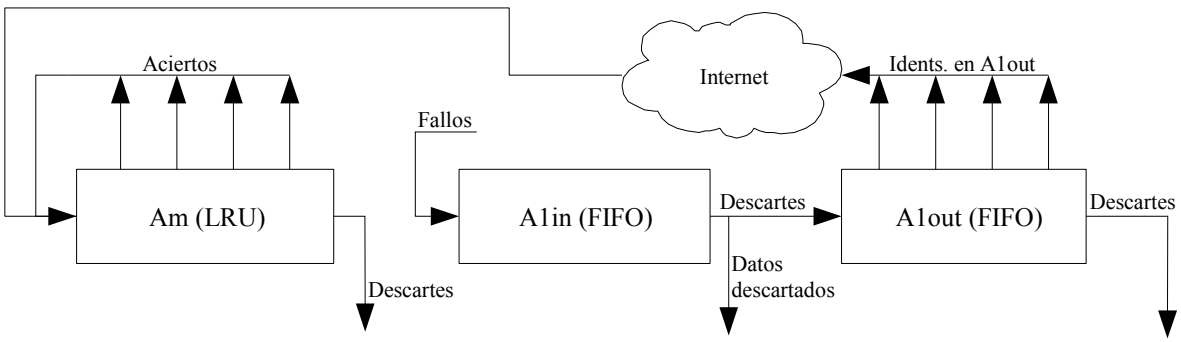

Figura 3: Esquema 2Q

El tamaño total de la caché se reparte entre Am, con tamaño máximo igual al máximo de la caché; A1in, de tamaño máximo determinado y dado por uno de los dos parámetros de la política, llamado Kin; y A1Out, de tamaño máximo Kout, que es el otro parámetro de la política.

2.4. FBR (*Frequency-Based Replacement)*

Esta política de reemplazo para cachés está basada en la idea de contabilizar el numero de veces que ha sido referenciado cada uno de los documentos que no han sido utilizados en un cierto periodo de tiempo. Con la ayuda de estos contadores, los elementos a reemplazar se escogen usando para la decisión una mezcla entre la frecuencia de uso y el tiempo que llevan dichos elementos en caché sin ser utilizados.

Al igual que otras políticas de reemplazo basadas en la frecuencia, como LFU (*Least Frequently Used*), FBR mantiene para cada documento en caché un contador de referencias, pero la diferencia es que no incrementa dicho contador en cada referencia al mismo, sino que solo lo hace en ciertas situaciones que más adelante se detallarán. El problema de incrementar siempre el contador es que documentos que en un corto periodo de tiempo son usados muchas veces puede que en el futuro no sean utilizados de nuevo, pero entonces su contador de referencias será alto y este documento se guardará en caché un largo tiempo, ocupando un espacio que podría estar ocupado por otros documentos más útiles en términos de rendimiento para la caché. Para solucionar esto, FBR actúa de la siguiente manera: cuando hay un acierto en caché, el contador de referencias sólo se incrementa si el documento no está en una parte de la caché llamada *sección nueva.*

La caché está ordenada como LRU, y como puede verse en la Figura 4 está dividida en varias partes a las que en [4] se les denomina secciones, y son en concreto tres, llamadas *nueva*, *media* y *antigua*. El tamaño de las distintas secciones viene dado por dos parámetros de la política: F_{old} y F_{new}, que (dados en tanto por uno) han de cumplir la propiedad de que $F_{old} \leq 1 - F_{new}$.

Con el uso de la *sección nueva* y el hecho de no incrementar los contadores de referencias cuando un documento referenciado esté en esta parte de la caché, se consigue, siempre que su tamaño sea suficientemente grande, eliminar el problema de que los contadores crezcan demasiado por un alto uso del mismo documento en un corto periodo de tiempo.

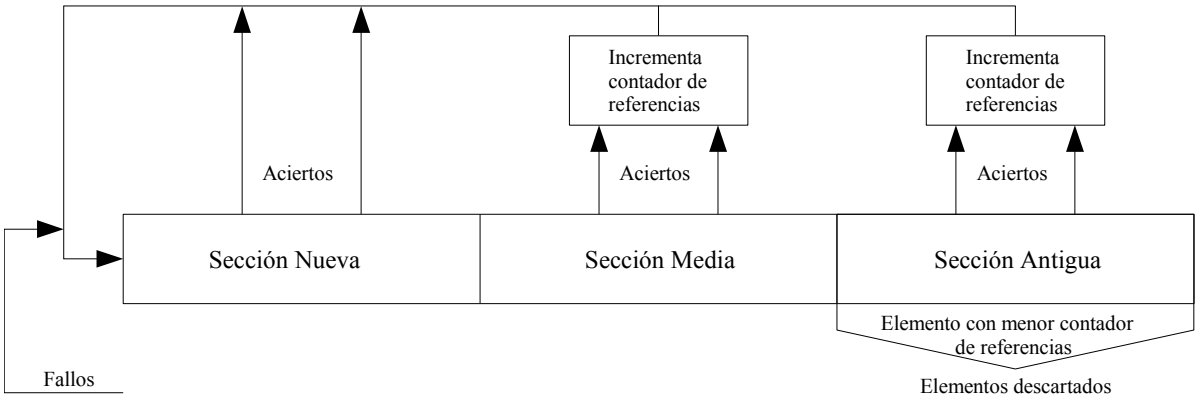

Figura 4: Esquema FBR

Cuando se produce un acierto, el elemento accedido se mueve, independientemente de la posición que ocupe en la caché, a la cabeza de la *sección nueva*, para así mantener los elementos en caché ordenados del más al menos usado recientemente.

El elemento a eliminar de la caché es el que tiene un menor contador de referencias y que está en la *sección antigua*, eligiendo en caso de empate el menos usado recientemente. En un principio, los creadores del algoritmo sólo incluyeron dos secciones en la caché, pero posteriormente observaron que incluyendo una sección intermedia, la *sección media,* se conseguía aumentar el rendimiento de la caché, ya que eliminaban el problema de que un elemento, al pasar de la *sección nueva* a la *sección antigua*, tenía bastantes posibilidades de ser el escogido para abandonar la caché, e incluyendo esta sección intermedia, el documento durante su permanencia en ella puede aumentar su contador de referencias si es un documento importante para la caché, y si no lo es, llegará a la *sección antigua,* donde optará a salir de la caché.

Para que no haya problemas con el mantenimiento de los contadores de referencias, se añaden dos parámetros más al algoritmo, uno, llamado C_{max} es un valor límite para dichos contadores, que una vez alcanzado ya no será incrementado más; y otro, denominado A_{max}, utilizado como límite para el valor medio de la suma de todos los contadores de referencias. Cuando dicha media de los contadores supera A_{max}, todos los contadores son reducidos a la mitad, para evitar que demasiados contadores lleguen al límite C_{max}.

2.5. LRFU (*Least Recently/Frequently Used)*

Estamos ante una política de reemplazo multinivel que en realidad no es sólo una, sino muchas, o como sus autores lo denominan en [5], son un espectro de políticas, que van desde LRU hasta LFU, es decir, desde dar más importancia a los documentos en caché referenciados más recientemente, hasta dársela a los que han sido utilizados un mayor número de veces, habiendo entre medio una gran variedad de posibles estrategias posibles a utilizar para decidir sobre el reemplazo de los elementos en caché. A este espectro de políticas se le llama *Least Recently/Frequently Used* (LRFU).

LFU y LRU tienen el problema que se basan sólo en la frecuencia o en lo recientemente que han sido utilizados los documentos, con lo cual funcionarán bien para ciertos tipos de cachés, pero no para aquellas que no tengan un patrón de referencias donde no sea claramente más efectiva una política u otra. LRFU posibilita establecer la importancia de cada una de las estrategias LRU y LFU, y así conseguir un mayor rendimiento para la caché.

La política LRFU asigna un valor para cada elemento de la caché. Este valor, denominado CRF (*Combined Recency and Frequency*), cuantifica la probabilidad de que un documento sea referenciado en el futuro. Cada referencia a un elemento en el pasado se tiene en cuenta para calcular el CRF del mismo, que se calcula con la ayuda de una función peso $F(x)$, donde x es el tiempo que ha pasado desde la última vez que el documento fue referenciado. La función F devuelve un valor acorde con la frecuencia y con lo recientemente que ha sido referenciado un documento. Por ejemplo, suponiendo que un elemento ha sido referenciado en los instantes de tiempo 2, 5, 8 y 9, y se está en el instante de tiempo 11, entonces el CRF del documento en el instante 11 será:

$$CRF_{11} = F(11-2) + F(11-5) + F(11-8) + F(11-9) = F(9) + F(6) + F(3) + F(2)$$

En general,

$$CRF_{t_{base}}(b) = \sum_{k}^{i=1} F\left(t_{base} - t_{b_i}\right)$$

donde *b* es el elemento cuyo CRF se quiere calcular, t_{base} es el instante actual y t_{b_i} los instantes de tiempo en los que el documento *b* fue referenciado.

Una vez conocido lo que significa el CRF de un documento, ya se puede decir cuál es la decisión que toma LRFU para reemplazar un elemento de caché. LRFU reemplaza el elemento que tiene el menor CRF.

El funcionamiento de LRFU hace que haya que guardar el historial completo de cada una de las referencias que se han producido a lo largo del tiempo para así poder calcular los valores CRF de cada elemento, pero guardar dicho historial haría que la política LRFU no pudiera ser implementada, ya que se necesitaría una memoria infinita. Para solventar este problema, se demuestra en [5] que si cumple que $F(x+y) = F(x) \cdot F(y)$ ó $F(x+y) = F(x) + F(y)$ para todo *x* e *y*, entonces el CRF en un instante dado puede ser calculado a partir del CRF en un instante anterior, y por lo tanto, no será necesario almacenar todo el historial de referencias, sino tan sólo información sobre la última vez que fue referenciado. La expresión que calcula el CRF será la siguiente:

$$CRF_{t_{act}}(b) = F(0) + F(t_{act} - t_{ult}) \cdot CRF_{t_{ult}}$$

donde b es elemento cuyo CRF se quiere calcular, t_{act} es el instante actual y t_{ult} es el instante en el que el elemento fue referenciado por última vez. $F(0)$ es el resultado de aplicar a la función *F* el valor 0, en concreto y como podrá verse a continuación, $F(0) = 1$ para la función *F* escogida.

La función *F* escogida es $F(x) = \left(\frac{1}{p}\right)^{\lambda \cdot x}$,con $p \geq 2$, $0 \leq \lambda \leq 1$.

Para la implementación se ha tomado p = 2. Dicha función cumple la propiedad $F(x+y) = F(x) \cdot F(y)$. Cuando el parámetro λ (que es el que propicia que aparezcan el espectro de políticas ya comentado anteriormente) se acerque a 0, dará mayor importancia a la frecuencia con la que son usados los documentos, y si λ es más próximo a 1, dará más importan-

cia a lo recientemente que hayan sido referenciados los elementos en caché.

Como LRFU reemplaza el documento con menor CRF, es necesario que los documentos estén ordenados según los valores CRF de los mismos. Esto sería un problema, ya que habría que actualizar todos los CRF continuamente al transcurrir el tiempo, pero con la función *F* anteriormente explicada, no es necesario, ya que el orden entre cada par de elementos no varía hasta que alguno de ellos sea referenciado, y por consiguiente, sólo habrá que reordenar los elementos en esa situación.

A la hora de implementar la política de reemplazo, se suele usar una estructura tipo heap para mantener los elementos ordenados, y una lista enlazada para los elementos con menor CRF de la caché (normalmente CRF = $F(0)$), y así conseguir que el algoritmo tenga un menor coste computacional. Por esta forma de estructurar la caché, que se puede observar en la Figura 5, hemos considerado la política de reemplazo LRFU como multinivel, aunque quizás no lo sea en su sentido más estricto.

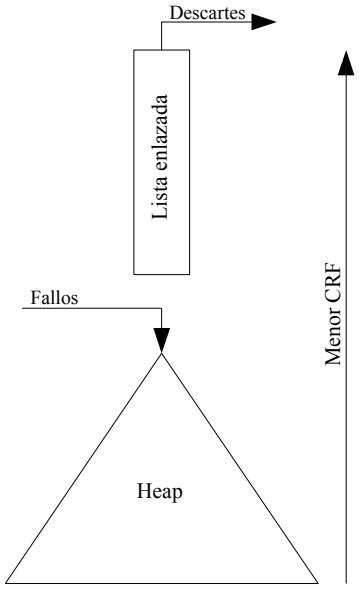

Figura 5: Estructura LRFU

2.6. ML-LRU (*MultiLevel LRU)*

MultiLevel LRU es una política de reemplazo para cachés Web propuesta en [6] como respuesta a la necesidad de diferenciar los servicios que ofrecen las cachés Web, es decir, ofrecer una mayor calidad de servicio (QoS – *Quality of Service*) dando mayor importancia a ciertos tipos de documentos en función de su contenido, valor económico, popularidad o procedencia, según convenga.

ML-LRU divide la caché en un cierto número M de listas conectadas entre sí, cada una de ellas administradas utilizando la estrategia LRU. Cuando un elemento se elimina del final de una de las listas dicho elemento se mueve a la primera posición de la siguiente lista.

En la Figura 6 se muestra la política para M=2, que será la que se va a estudiar. Cuando un nuevo documento entra en caché se coloca en una de las listas en función de la clasificación que se quiera realizar de los mismos.

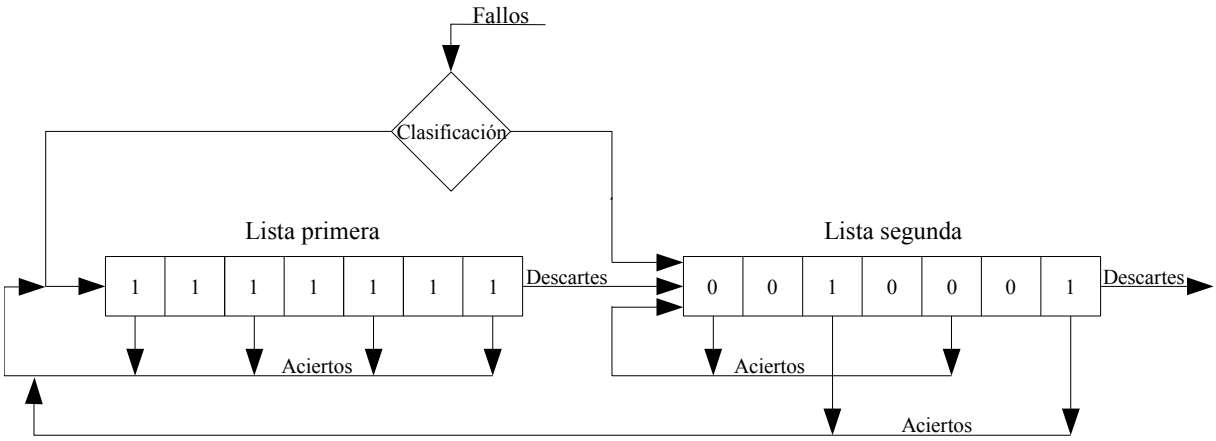

Figura 6: Esquema ML-LRU para M=2

Por tanto, la caché tiene dos listas, en la primera de ellas se almacenarán los documentos más importantes y en la segunda los menos importantes según los clasifiquemos. Para realizar dicha clasificación pueden utilizarse, por ejemplo, algunos de los siguientes factores:

- Dominio de procedencia.
- Tipo de contenido o de documento: texto, imágenes, vídeos, etc.
- Cantidad de dinero que hay que pagar por acceder a los documentos.
- Popularidad
- Tamaño.

El funcionamiento de la caché es el siguiente. Tanto la primera como la segunda lista usan la estrategia LRU. Cuando un elemento entra por primera vez en caché se le asigna un valor , que será 1 cuando en la clasificación previa se decida que el elemento ha de estar en la lista primera, y 0 en caso contrario.

Si un elemento cuyo valor es 0 es referenciado, dicho elemento se coloca en la cabeza de la segunda lista, y si es 1, se moverá a la cabeza de la primera, sea cual sea su posición en la caché. Por tanto, en la lista primera sólo podrá haber documentos clasificados con valor 1, es decir, los documentos más importantes, mientras que en la segunda lista podrá haber elementos de cualquier tipo, pero el número de documentos con valor 1 en esta segunda lista no podrá ser ilimitado, sino oque tendrá un límite. El límite viene dado por el parámetro β, que, por tanto, establece el número máximo de elementos que puede haber en la segunda lista que hayan estado anteriormente en la primera.

Para establecer el tamaño de cada lista, se utiliza un parámetro adicional que indica el tamaño máximo para la primera lista en relación al tamaño total.

En definitiva, ML-LRU plantea una forma distinta de administrar las cachés Web, no orientada tanto a conseguir un buen rendimiento general de la caché en términos absolutos de aciertos, sino que trata de mejorar el rendimiento de la caché para ciertos tipos de documentos que son más importantes, y que por lo tanto revierten en una mejora de la calidad del servicio ofrecido al usuario.

2.7. C-LRU (*Class-Based LRU)*

El rango de los tamaños de los documentos de Internet es muy amplio, y normalmente los documentos de menor tamaño son más populares y son usados con más frecuencia, lo que permite aumentar el número de aciertos obtenidos. Por otra parte, los documentos de mayor tamaño son utilizados con menos frecuencia, pero su almacenamiento en caché conlleva un mayor índice de aciertos en bytes. Por tanto, para conseguir unos buenos resultados en el conjunto de ambas métricas de rendimiento habría que conseguir en la caché un cierto balance entre la cantidad de documentos de poco y de mucho tamaño.

Usando *Class-Based* LRU se divide la caché en clases o particiones, cada una de las cuales almacenará documentos en un rango de tamaño determinado. Cada una de estas particiones se administra utilizando la estrategia LRU. Por tanto, cuando un documento va a ser almacenado en caché hay que determinar en primer lugar la partición a la que va a pertenecer.

Cada clase tendrá un tamaño determinado que se expresa normalmente como fracción del tamaño total de la caché. Observando los patrones de acceso de la caché y variando el nú-

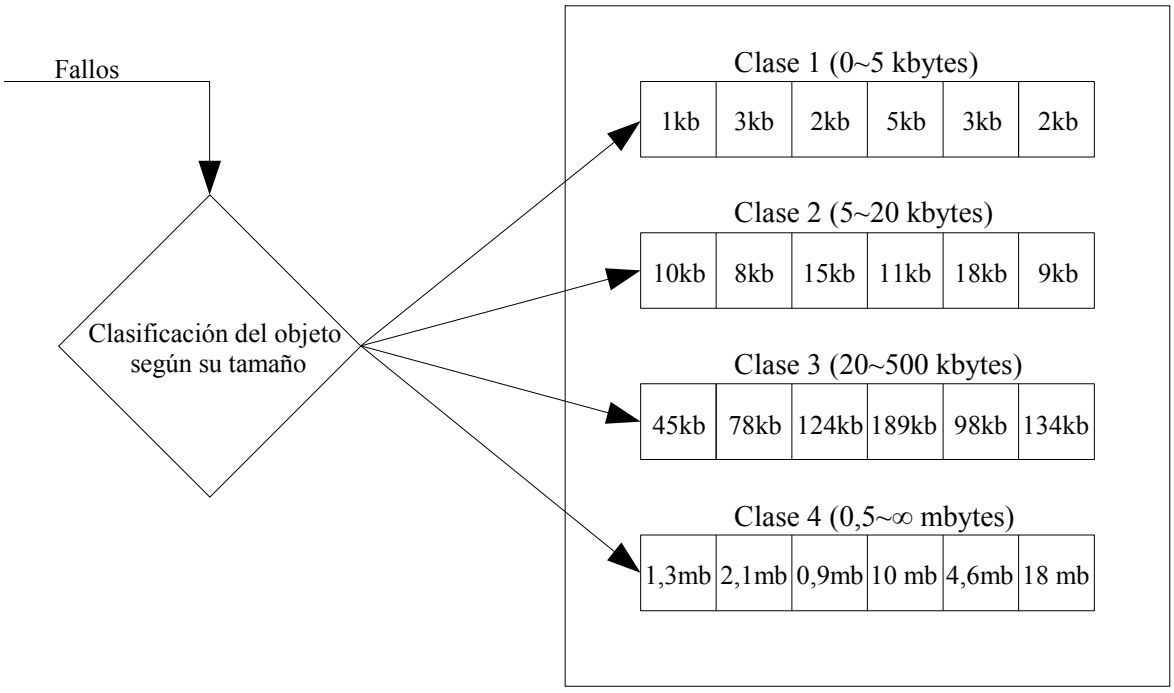

Figura 7: Ejemplo C-LRU

mero de particiones, su rango de tamaño de documentos y su proporción de tamaño respecto al de la caché se puede lograr un alto índice tanto de aciertos como de aciertos en bytes.

El ejemplo mostrado en la Figura 7 se muestra una caché C-LRU con cuatro clases, cada una de las cuales almacena documentos de un cierto rango de tamaños. Cuando se produce un fallo, se decide a qué clase pertenece el nuevo elemento y se inserta en la caché.

3. Simulador de políticas de reemplazo en cachés Web: Manual de usuario

Se ha creado una aplicación para realizar simulaciones de cachés Web utilizando las distintas políticas de reemplazo multinivel que se han detallado en el capítulo 2. Para realizar dichas simulaciones, se utilizan trazas procedentes de servidores proxy. En estas trazas aparecen las sucesión de peticiones de documentos realizadas por los usuarios al servidor, así como información sobre las mismas como por ejemplo su tamaño o su tipo.

El funcionamiento es sencillo, basta conocer las políticas de reemplazo que se quieran utilizar en las simulaciones, puesto que hay que indicar los parámetros de las mismas para llevarlas a cabo, y una vez realizadas dichas simulaciones, se pueden visualizar los resultados obtenidos y generar archivos para su poder compararlos.

La aplicación permite programar una serie de simulaciones para su ejecución en lotes, y para mayor seguridad, guarda automáticamente en un fichero las simulaciones realizadas hasta el momento cada vez que termina la ejecución de una simulación, y así poder recuperar los documentos en el caso de que ocurriera algún problema inesperado que detuviera la ejecución de la aplicación.

Hay que tener en cuenta las siguientes consideraciones sobre la aplicación:

- Los ficheros que contienen la traza han de ser de texto plano y deben tener una petición por cada línea en la que aparecerá en primer lugar el tamaño de la petición en bytes y en segundo lugar un identificador numérico de la petición, estando estos dos campos separados por uno o varios espacios.
- Si entre dos peticiones sucesivas a un mismo documento que está alojado en la caché el tamaño del mismo ha cambiado menos del 5% entonces se asumirá que el documento ha sido modificado y ha de ser actualizado, y por lo tanto se procesará como un fa-

llo [7]. En caso contrario se considerará que ha ocurrido un corte de conexión, y por lo tanto se tratará como un acierto y se mantendrá el documento que ya estaba en la caché.

3.1. Pantalla principal

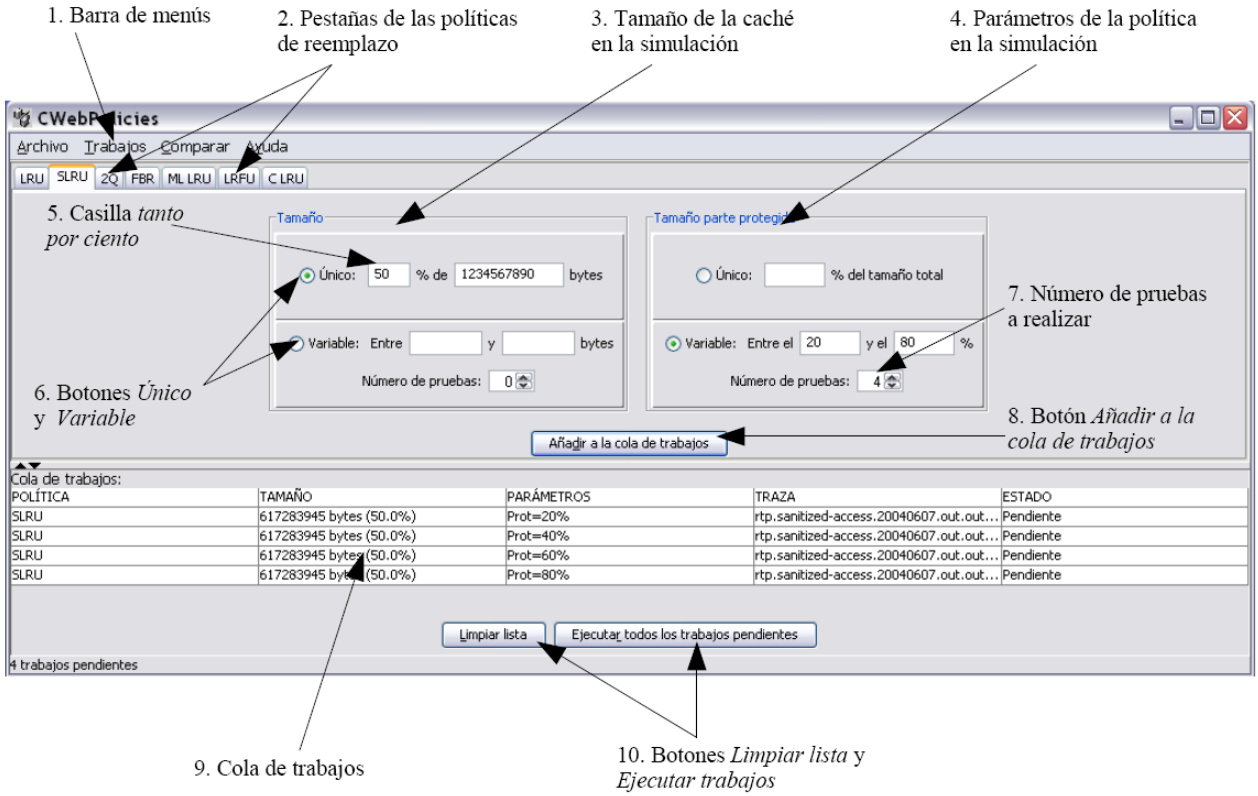

Figura 8: Aspecto general de la aplicación

En primer lugar se explicarán brevemente las funciones de los botones y menús de la aplicación que aparecen en la Figura 8.

1. Barra de menús. Incluye los menús de la aplicación: *Archivo*, *Trabajos*, *Comparar, Ayuda*. Las funciones de cada menú serán explicadas detalladamente en el apartado 3.4.
2. Pestañas de las políticas de reemplazo. Cada una de las pestañas muestra las opciones de las distintas políticas de reemplazo que se han implementado.

3. Tamaño de la caché en la simulación. Permite indicar el tamaño máximo que tendrá la caché en la simulación.
4. Parámetros de la política en la simulación. Permite indicar el valor para cada uno de los parámetros de cada política de reemplazo. Dependiendo de la política habrá más o menos parámetros. En el ejemplo se muestra la política SLRU, que tiene un sólo parámetro.
5. Casilla *tanto por ciento*. En ocasiones se querrán hacer simulaciones para una caché cuya tamaño sea un tanto por ciento de un tamaño dado. En ese caso se indicará aquí dicho porcentaje. Por ejemplo, podría indicarse 23.4 % de 1000000 bytes.
6. Botones *Único* y *Variable*. Seleccionando *Único*, se realizará una sola simulación con los valores de los parámetros elegidos. Si se selecciona *Variable*, se realizarán varias simulaciones para el intervalo de valores del parámetro, por ejemplo: Entre 2 y 10, Número de pruebas: 5, entonces se harán simulaciones con valores 2, 4, 6, 8 y 10 del parámetro en cuestión.
7. Número de pruebas a realizar. Indica las simulaciones que se harán para el intervalo de valores dado.
8. Botón *Añadir a la cola de trabajos*. Al pulsar este botón, se creará en la cola de trabajos una o varias tareas, según se haya seleccionado, con los datos que hayamos indicado, para su posterior ejecución.
9. Cola de trabajos. Se muestra información sobre las tareas que se han añadido a la misma. Aparecen cinco columnas por cada trabajo, que indican la política de reemplazo a la que pertenecen, el tamaño de la caché, los parámetros adicionales de la política, los nombres de los ficheros traza y el estado de dicho trabajo, que podrá estar pendiente, en ejecución o ya finalizada su ejecución.
10. Botones *Limpiar lista* y *Ejecutar trabajos*. *Limpiar lista* borra todas las tareas en la cola de trabajos, y *Ejecutar trabajos* comienza las simulaciones pendientes de la cola de trabajos.

3.2. Cómo realizar simulaciones paso a paso

1º. Cargar una traza. Pulsar sobre el menú *Archivo* → *Cargar traza*. Aparecerá un diálogo como el de la Figura 9 en el que se pueden escoger el/los fichero/s traza. Tras seleccionar la traza pulsar *Abrir*. Una barra de progreso similar a la que aparece en la Figura 10 mostrará el avance de la operación, que podrá tardar más o menos dependiendo del tamaño de la traza, y cuando concluya la carga de la traza, si no ocurre ningún fallo aparecerá un mensaje de confirmación como se muestra en la Figura 11.

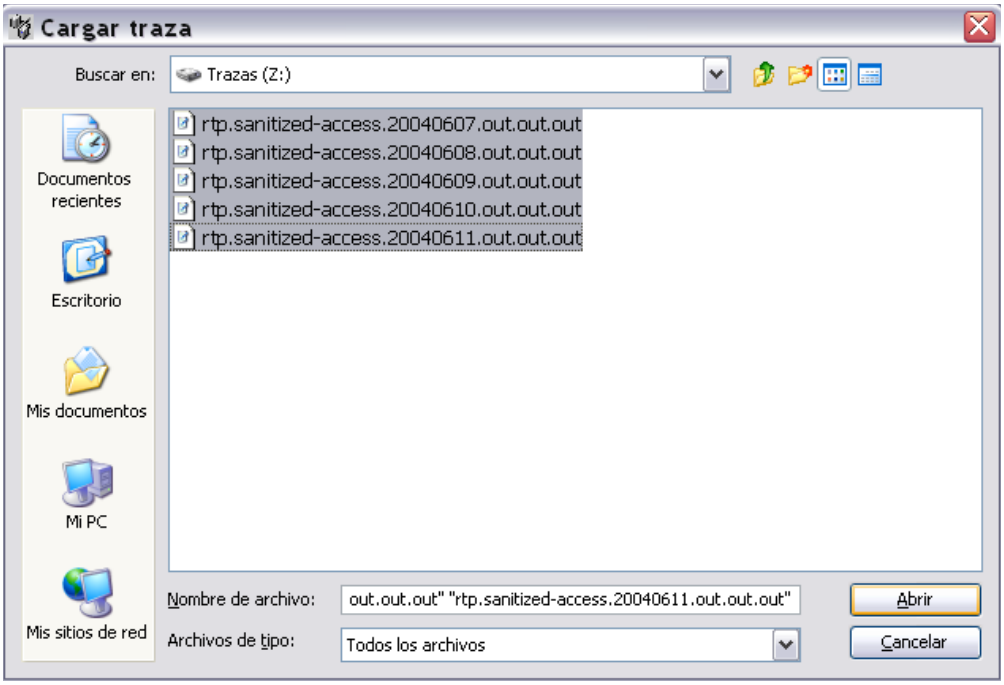

Figura 9: Diálogo para cargar una traza

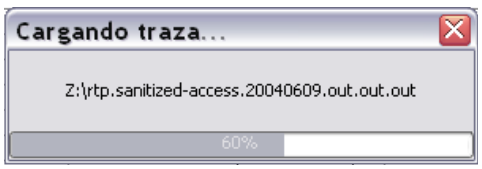

Figura 10: Progreso de la carga de la traza

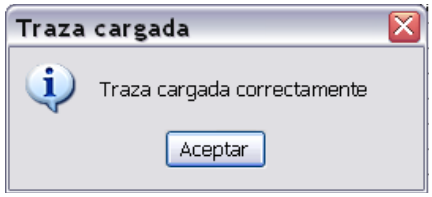

Figura 11: Carga de traza finalizada

2º. (Opcional) Una vez cargada correctamente la traza, si está compuesta por más de un fi-

chero, puede comprobarse el orden correlativo de los mismos y cambiarlo si no es el correcto (por defecto se cargan los ficheros por orden alfabético). Para ello hay que ir al menú *Archivo → Orden de los ficheros traza*, con lo que aparecerá una pantalla como la que muestra la Figura 12 donde se podrá seleccionar el orden correcto de los ficheros.

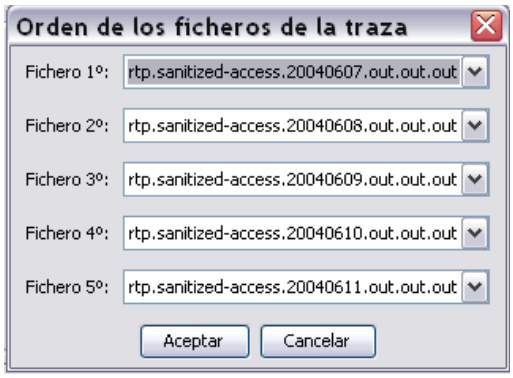

Figura 12: Orden de los ficheros de la traza

3º. Pulsar la pestaña de la política de reemplazo que se quiera simular e introducir los datos correspondientes al tamaño de la caché y a los parámetros de la política. Para más información sobre la función de los botones ir al apartado 3.3.

4º. Una vez introducidos los datos pulsar el botón *Añadir a la cola de trabajos*. Si los datos son correctos, aparecerá en la cola de trabajos una nueva línea por cada simulación que hayamos añadido.

5º. Para ejecutar las simulaciones pulsar sobre el botón *Ejecutar todos los trabajos pendientes* o seleccionar las simulaciones que se quieran ejecutar y pulsar sobre la opción *Ejecutar seleccionados* del menú *Trabajos* o del menú contextual (que aparece al pulsar el botón derecho del ratón cuando está situado sobre la cola de trabajos). Nótese que el menú contextual tiene las mismas funciones que el menú *Trabajo*.

6º. Mientras se ejecutan las simulaciones puede verse el porcentaje que llevan ejecutado. Cuando una simulación concluya su ejecución podrá observarse los resultados haciendo doble click sobre la misma en la cola de trabajos o pulsando menú *Trabajos → Ver resultados*. Asimismo, pueden generarse ficheros con los datos de las simulaciones para poder compararlos mediante otras aplicaciones, como por ejemplo Matlab, lo cual será explicado en el apartado 3.4.11.

3.3. Políticas de reemplazo

A continuación se comentará el significado de los parámetros que aparecen en cada una de las pestañas correspondientes a las políticas de reemplazo incluidas en la aplicación. Aunque el funcionamiento y el significado se las políticas de reemplazo y sus parámetros ya fueron explicados anteriormente (véase capítulo 2), aquí se aclararán algunas dudas que puedan surgir a la hora de introducir dichos parámetros en la aplicación para realizar las simulaciones.

3.3.1. LRU

Sólo es necesario introducir el tamaño de la caché, que viene expresado en bytes. El tamaño se introduce de igual modo para todas las políticas, de forma similar al ejemplo mostrado Figura 13, por lo que de aquí en adelante se omitirá nombrarlo.

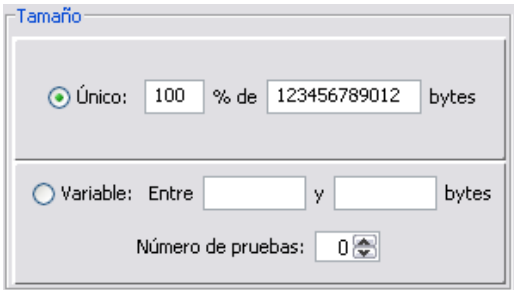

Figura 13: Tamaño máximo de la caché

3.3.2. SLRU

SLRU tiene un parámetro adicional que determina el tamaño de la parte protegida de la caché, y se indica en tanto por ciento en relación al tamaño total de la misma. Puede observarse la pantalla correspondiente en la Figura 14.

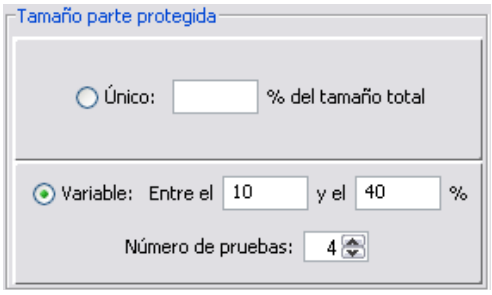

Figura 14: Tamaño de la parte protegida

3.3.3. 2Q

En primer lugar aparece el parámetro kIn que sirve para determinar el tamaño de A1In, que se indica en tanto por ciento respecto al tamaño total de la caché.

En segundo lugar está el parámetro kOut, que indica el tamaño de A1Out en número de entradas que puede almacenar, siendo una entrada capaz de almacenar los identificadores de una petición. Para especificar este parámetro se ha de introducir en la aplicación un porcentaje que expresa el tanto por ciento de entradas respecto al máximo de peticiones que podría almacenar la caché (este máximo se calcula dividiendo el tamaño máximo de la caché entre el tamaño medio de las peticiones de la caché). Por ejemplo si se selecciona el 90% del máximo de entradas para una caché de tamaño 1.000.000 bytes cuyo tamaño medio de peticiones es 1.000 bytes, entonces A1Out podrá almacenar el 90% de $\frac{1000000}{1000} = 900$ entradas.

En las figuras 15 y 16 se muestran ejemplos para los parámetros de la política 2Q.

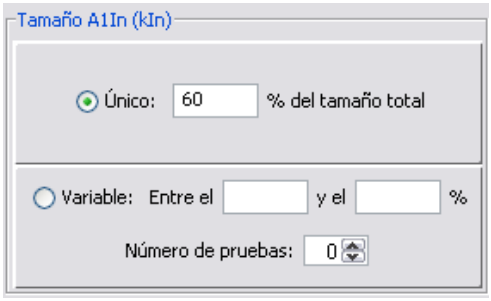

Figura 15: Tamaño A1In

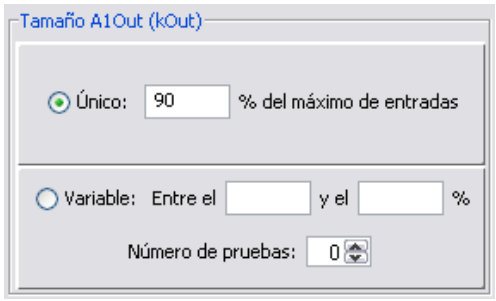

Figura 16: Tamaño A1Out

3.3.4. FBR

En primer lugar hay que indicar el tamaño de las partes *Nueva* y *Antigua*, que se ha de expresar en tanto por ciento respecto al tamaño de la caché. Hay que tener en cuenta que la suma de dichos tamaños ha de ser menor o igual que el 100%. Los otros dos parámetros son los umbrales C_{max} y A_{max}, cuyos significados ya fueron explicados en el apartado 2.4. Las figuras 17 y 18 muestran ejemplos para el tamaño de la parte *Antigua* y para C_{max}, respectivamente.

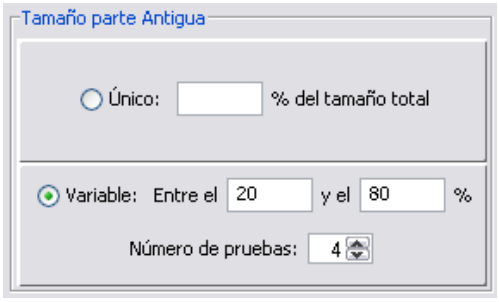

Figura 17: Tamaño parte Antigua

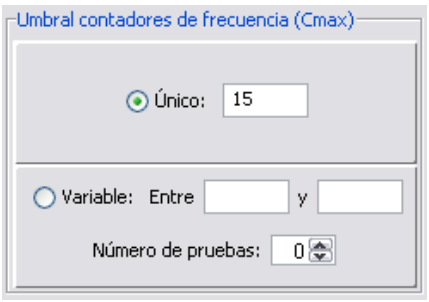

Figura 18: Parámetro C_{max}

3.3.5. LRFU

En la pestaña LRFU se debe indicar el valor de λ en la función *F* usada para calcular el CRF de los objetos como se explica en el apartado 2.5. Dicho valor ha de estar entre 0 y 1, tal y como se muestra en la Figura 19.

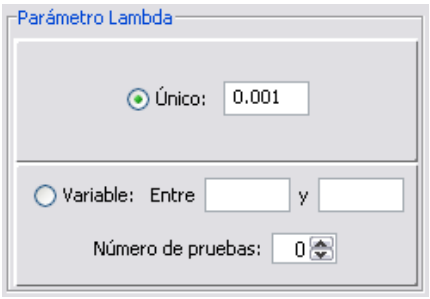

Figura 19: Parámetro Lambda

3.3.6. ML-LRU

Para realizar una simulación con esta política hay que introducir el tamaño de la parte *Primera* de la caché indicado en tanto por ciento respecto al tamaño total de la misma de como se muestra en el ejemplo de la Figura 20 y asimismo se ha de introducir el valor para el parámetro umbral β, indicado en tanto por ciento de elementos de la parte *Segunda*.

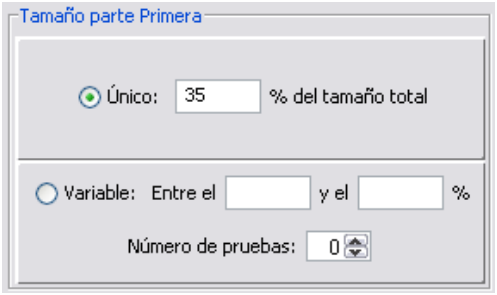

Figura 20: Tamaño parte Primera

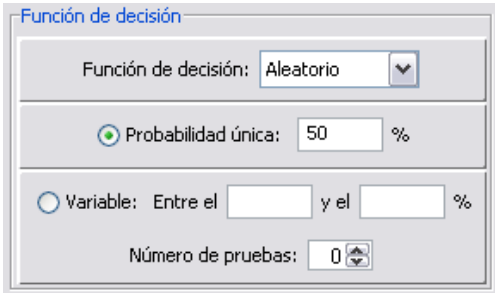

Figura 21: Tamaño parte Primera

Además especificaremos la función decisión que se quiera utilizar, tal y como se muestra en el ejemplo de la Figura 21. En la aplicación se han incluido dos funciones de decisión: la función aleatoria y la función tamaño, ambas con un parámetro que hay que introducir. El parámetro de la función aleatoria indica la probabilidad de que una nueva petición sea almacenada en la parte *Primera*, aunque la decisión de qué peticiones irán a una u otra parte de la caché se efectúa de forma aleatoria. Por otra parte, el parámetro de la función tamaño indica el tamaño máximo para que una petición sea almacenada en la parte *Primera*, es decir, si la petición es menor o igual que dicho tamaño se colocará inicialmente en la parte *Primera*, y si es mayor en la parte *Segunda* de la caché.

3.3.7. C-LRU

En la pestaña correspondiente a C-LRU (ver Figura 22) debe indicarse en primer lugar, pulsando sobre los botones "+" y "-", el número particiones que tendrá la caché. Posteriormente se han de introducir para cada una de dichas particiones los tamaños de las

peticiones que se almacenarán en cada una de ellas y el tamaño de cada partición en relación al tamaño total de la caché. A la hora de introducir los datos hay que tener en cuenta que las peticiones han de abarcar todos los tamaños de peticiones posibles y que la suma de los tamaños de todas las particiones ha de ser igual al 100% del tamaño de la caché.

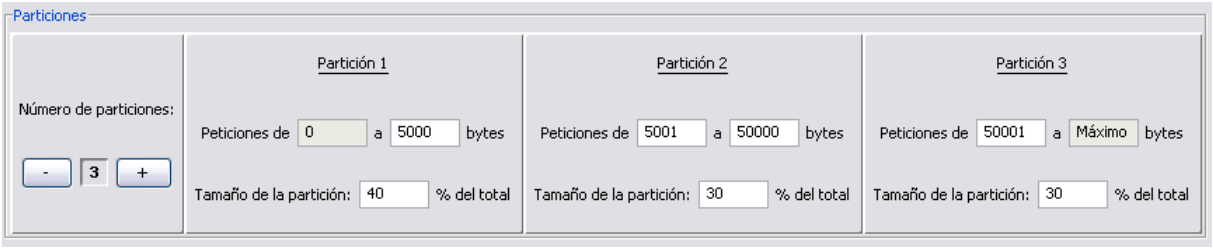

Figura 22: Datos de las particiones

3.4. Funcionamiento detallado de la aplicación

A continuación se explica detalladamente el funcionamiento de las distintas acciones que podemos realizar con la aplicación.

3.4.1. Opciones de la aplicación

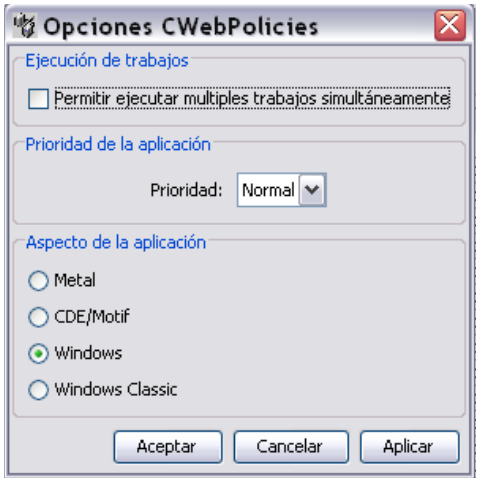

Figura 23: Opciones

Pulsando sobre el menú *Archivo → Opciones* aparecerá el diálogo que se muestra en la Figura 23, que tiene tres valores a configurar:

- Ejecución de trabajos. Por defecto, *Permitir ejecutar múltiples trabajos simultáneamente* no está marcado, con lo que las simulaciones se ejecutarán de una en una. Si marcamos esta opción, se podrán ejecutar varias simulaciones al mismo tiempo. Esto sólo es recomendable si el equipo en el que se está ejecutando la aplicación es multiprocesador.
- Prioridad de la aplicación. Permite seleccionar la prioridad de las hebras que ejecuten las simulaciones. Tiene tres posibles valores: alta, normal y baja.
- Aspecto de la aplicación. Permite escoger el aspecto de la aplicación entre los disponibles (pueden variar según el sistema operativo y la versión de la consola Java).

3.4.2. Cargar una traza

Como se explicó en el apartado 3.2-1º, hay que pulsar sobre el menú *Archivo → Cargar traza*. Aparecerá entonces un diálogo como el de la Figura 9 en el que se podrán escoger el/los fichero/s traza.

3.4.3. Cambiar el orden de los ficheros traza

Como se mostró en el apartado 3.2-2º, hay que acceder al menú *Archivo → Orden de los ficheros traza*, donde se seleccionará el orden relativo correcto de los ficheros que compongan la traza.

3.4.4. Visualizar la información de una traza

Para poder realizar esta acción se necesita en primer lugar cargar la traza. Una vez cargada, acceder al menú *Archivo → Información de la traza*. Se mostrará en una pantalla similar a la de la Figura 24 el número de ficheros que componen la traza, el número de peticiones totales y únicas, el tamaño total, medio y máximo de dichas peticiones y el tiempo empleado en cargar la traza.

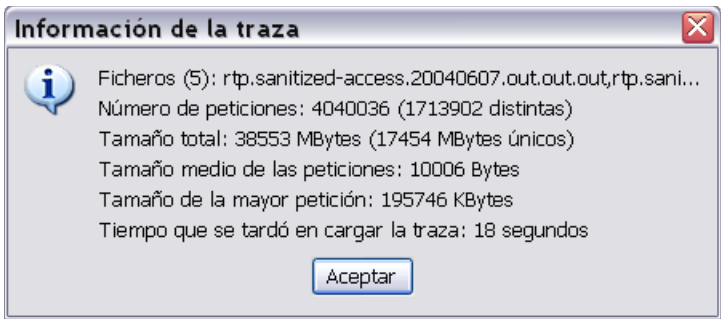

Figura 24: Información de una traza

3.4.5. Guardar la cola de trabajos actual

Con esta acción se guarda en un fichero la información de todos los trabajos que aparezcan en la cola de trabajos. Se guardan tanto los trabajos pendientes como los ejecutados y los resultados obtenidos en las simulaciones, lo cual permite que posteriormente se pueda volver a rescatar dicha información. Para crear dicho fichero pulsar el menú *Trabajos → Guardar trabajos* e introducir un nombre para el archivo a crear y pulsar guardar. Si no hay ningún problema se mostrará un mensaje indicando que se han guardado los trabajos correctamente.

3.4.6. Cargar una cola de trabajos

Para cargar una cola de trabajos que haya sido anteriormente guardada como se indicó en el apartado 3.4.5, hay que pulsar sobre la opción *Cargar trabajos* del menú *Trabajos*, seleccionar el fichero que contiene los trabajos y pulsar el botón *Cargar*. Si el fichero es correcto se mostrarán en la cola de trabajos de la aplicación los trabajos cargados, y un mensaje de

información con el número de trabajos que se han añadido a la cola de trabajos.

Si ya existieran trabajos en la cola de trabajos actual antes de cargar los trabajos desde el archivo, dichos trabajos se mantendrán, y se añadirán los procedentes del archivo. En el caso de que se quiera cargar algún trabajo que sea igual (misma política, tamaño, parámetros y traza) que alguno que ya esté en la cola de trabajos actual, dicho trabajo no se cargará y se mantendrá el que que ya estaba en la cola de trabajos actual.

3.4.7. Recuperar una cola de trabajos

La aplicación guarda automáticamente en un fichero la cola de trabajos actual cada vez que termina la ejecución de una simulación por si ocurriera algún problema inesperado que detuviera la ejecución de la aplicación. Dichos ficheros son creados en el mismo directorio en el que se encuentre la aplicación con el nombre *autosaveDIA-MES_HhMmSs.tra*. Para saber cómo cargar dichos ficheros ver el apartado 3.4.6.

3.4.8. Eliminar trabajos de la cola de trabajos

Si se quiere borrar uno o varios trabajos, en primer lugar hay que seleccionarlos pulsando sobre ellos con el ratón, y en segundo lugar pulsar sobre menú *Trabajos → Borrar seleccionados*, o bien pulsar la tecla *Delete*. También pueden borrarse todos los trabajos pulsando el botón *Limpiar lista* o en *Trabajos → Borrar todos*.

3.4.9. Ejecutar trabajos

Para ejecutar todos los trabajos que aún no han sido ejecutados pulsar el botón *Ejecutar todos los trabajos pendientes* o menú *Trabajos → Ejecutar todos*. También pueden ejecutarse sólo los trabajos que se escojan, para ello hay que seleccionarlos y pulsar la opción *Ejecutar seleccionados* del menú *Trabajos*.

3.4.10. Detener la ejecución de trabajos en ejecución

Con la opción *Cancelar ejecución de todos los trabajos* del menú *Trabajos* se detendrá la ejecución de todos los trabajos que se estén ejecutando en dicho momento y de los pendientes de ejecutarse, si los hay.

Si se ha habilitado la opción para permitir ejecutar varios trabajos simultáneamente se podrá además detener la ejecución de uno o varios trabajos en ejecución. Para ello se han de seleccionar los trabajos correspondientes y después pulsar sobre el menú *Trabajos* → *Cancelar ejecución de los seleccionados*.

3.4.11. Visualizar los resultados de las simulaciones

Para poder visualizar los resultados de una simulación, la columna *Estado* del trabajo correspondiente a dicha simulación ha de indicar que la ejecución ha terminado correctamente, lo cual se indica con el mensaje *Ejecutado 100%*. Entonces se podrán ver los resultados haciendo doble click con el ratón sobre dicho trabajo o pulsando menú *Trabajos* → *Ver resultados*. En los resultados de la simulación se muestran los siguientes datos:

- Aciertos. Indica el número de aciertos obtenidos y el total de peticiones procesadas, así como el tanto por ciento de aciertos (*Hit Rate*).
- Aciertos en bytes (*Byte Hits*). Muestra el número de bytes correspondiente a los aciertos que se han conseguido y la suma de los tamaños de todas las peticiones procesadas. Asimismo se indica el tanto por ciento de aciertos en bytes (*Byte Hit Rate*).
- Media de elementos en caché. Indica el número medio de elementos que residieron en caché durante la simulación.
- Número de elementos en caché al finalizar. Muestra el número de elementos que había en la caché al finalizar la simulación, es decir, después de haber procesado todas las peticiones de la traza.

- Tamaño medio ocupado. Indica el número medio de bytes de la caché que han estado ocupados durante la simulación.
- Tamaño ocupado al finalizar. Indica el número de bytes de la caché ocupados al finalizar la simulación.
- Número de peticiones actualizadas. Número de veces que se ha actualizado un documento de la caché por cumplir las propiedades que ya se explicaron anteriormente en el capítulo 3.
- Adicionalmente y según la política de reemplazo de la que se trate pueden aparecer más datos similares a los que se han explicado anteriormente correspondientes a los distintos niveles de la caché.

3.4.12. Comparar resultados de simulaciones

La aplicación permite comparar los resultados obtenidos en las simulaciones, para ello genera un archivo que posteriormente puede ser procesado con una aplicación externa, por ejemplo Matlab, para realizar la comparación.

Para generar los archivos de las comparaciones en primer lugar se han de seleccionar en la cola de trabajos las simulaciones que se quieran comparar. Lógicamente, las simulaciones deben haber sido ejecutadas para poder ser comparadas, por lo que si se se selecciona alguna que no haya sido ejecutada no será tenida en cuenta para realizar la comparación. Una vez realizada la selección, hay que pulsar sobre el menú *Comparar*, y posteriormente seleccionar el parámetro que se quiera comparar, que podrá ser un parámetro particular de una política o bien el tamaño de la caché en la simulación. Cuando se haya seleccionado el parámetro a comparar habrá que elegir el nombre del fichero a generar.

En el fichero generado permitirá comparar los resultados obtenidos en las simulaciones en función del parámetro que hayamos seleccionado en el menú *Comparar*. Los datos que se guardan en el fichero generado son los siguientes:

- Porcentaje de aciertos (*Hit Rate*).

- Porcentaje de aciertos en bytes (*Byte Hit Rate*).
- Media de elementos almacenados en caché.
- Porcentaje medio de tamaño ocupado del total de la caché.

4. Simulador de políticas de reemplazo en cachés Web: Implementación

La aplicación ha sido implementada en el lenguaje de programación orientado a objetos Java bajo el entorno de desarrollo NetBeans, y tiene el objetivo de poder realizar con facilidad y en el menor tiempo posible simulaciones de cachés Web utilizando para ello una serie de peticiones realizadas por un grupo de usuarios a un servidor proxy durante un cierto periodo de tiempo. La información de dichas peticiones se recoge en uno o varios ficheros, que son los que la aplicación utilizará para las simulaciones. Una vez realizadas las simulaciones, los resultados pueden verse y compararse fácilmente, para ello la aplicación genera ficheros con los resultados obtenidos para poder compararlos, concretamente se ha creado un pequeño fichero *script* para Matlab que a partir de dichos ficheros dibuja gráficas comparando los resultados.

La aplicación se divide claramente en dos partes. Por un lado está la parte dedicada a la interfaz gráfica con la que el usuario interactuará para realizar las simulaciones y compararlas, y por otro lado está la parte que realiza las simulaciones de las cachés.

La interfaz gráfica se ha desarrollado con la ayuda de NetBeans, y se ha buscado la mayor simplicidad posible para que el usuario pueda trabajar con la aplicación de forma intuitiva y sin problemas desde el primer momento. Con el uso de excepciones se controla en todo momento los posibles errores que puedan ocurrir, mostrando los mensajes de error oportunos en el caso de que alguno ocurriera.

Las clases que realizan las simulaciones se han desarrollado para conseguir la mayor eficiencia, y así realizar las simulaciones en el menor tiempo posible, para así poder realizar un gran número de simulaciones con los que poder evaluar satisfactoriamente el rendimiento de las distintas políticas de reemplazo objetos de este estudio.

4.1. Descripción general de las clases

Las simulaciones son realizadas por las clases correspondiente a cada una de las políticas de reemplazo. Como puede verse en la Figura 25, existe una clase con el nombre de cada política. Estas clases contienen las estructuras de la caché necesarias para realizar las simulaciones. Las cachés se han implementado mediante las Colecciones de Java, concretamente LinkedList y ArrayList, en las cuales cada uno de los elementos es un objeto de la clase PeticionWeb, que representa a un documento que ha sido pedido por un usuario al servidor proxy. Cada caché está compuesta de varias colecciones de objetos si la política de reemplazo es multinivel, o de una única colección en el caso de LRU. Además, para mejorar la eficiencia de las simulaciones, para cada parte de la caché se ha creado una tabla hash, con lo que rápidamente se puede saber si un documento está o no almacenado en la caché.

La clase Trabajo especifica los parámetros y la traza de la simulación que se quiera llevar a cabo, y se simulan a través del método *simular* de la clase correspondiente a la política de reemplazo en cuestión. El método *simular* lee las peticiones mediante la clase LectorPeticiones, que accede a los ficheros de la traza para extraer una a una las peticiones, que son definidas mediante la clase PeticionWeb. Cada vez que se lee una petición, se procesa según la política. El funcionamiento de las simulaciones será explicado con mayor detalle en el apartado 4.3.

La clase principal de la interfaz gráfica es VentanaPrincipal muestra y administra la pantalla principal de la aplicación e incluye todos los menús de la misma. Las acciones correspondientes a los eventos originados en dicha ventana principal relacionados con las simulaciones son ejecutadas por la clase AccionesVentana, que actúa a modo de interfaz entre las clases de la interfaz gráfica y las clases que llevan a cabo las simulaciones. Por otra parte existen una serie de clases auxiliares a las que accede VentanaPrincipal que muestran y ejecutan las acciones de los diálogos auxiliares a la ventana principal, como por ejemplo las opciones de la aplicación, el orden de los ficheros de la traza o los resultados de las simulaciones. Pueden observarse las clases en el diagrama de la Figura 26.

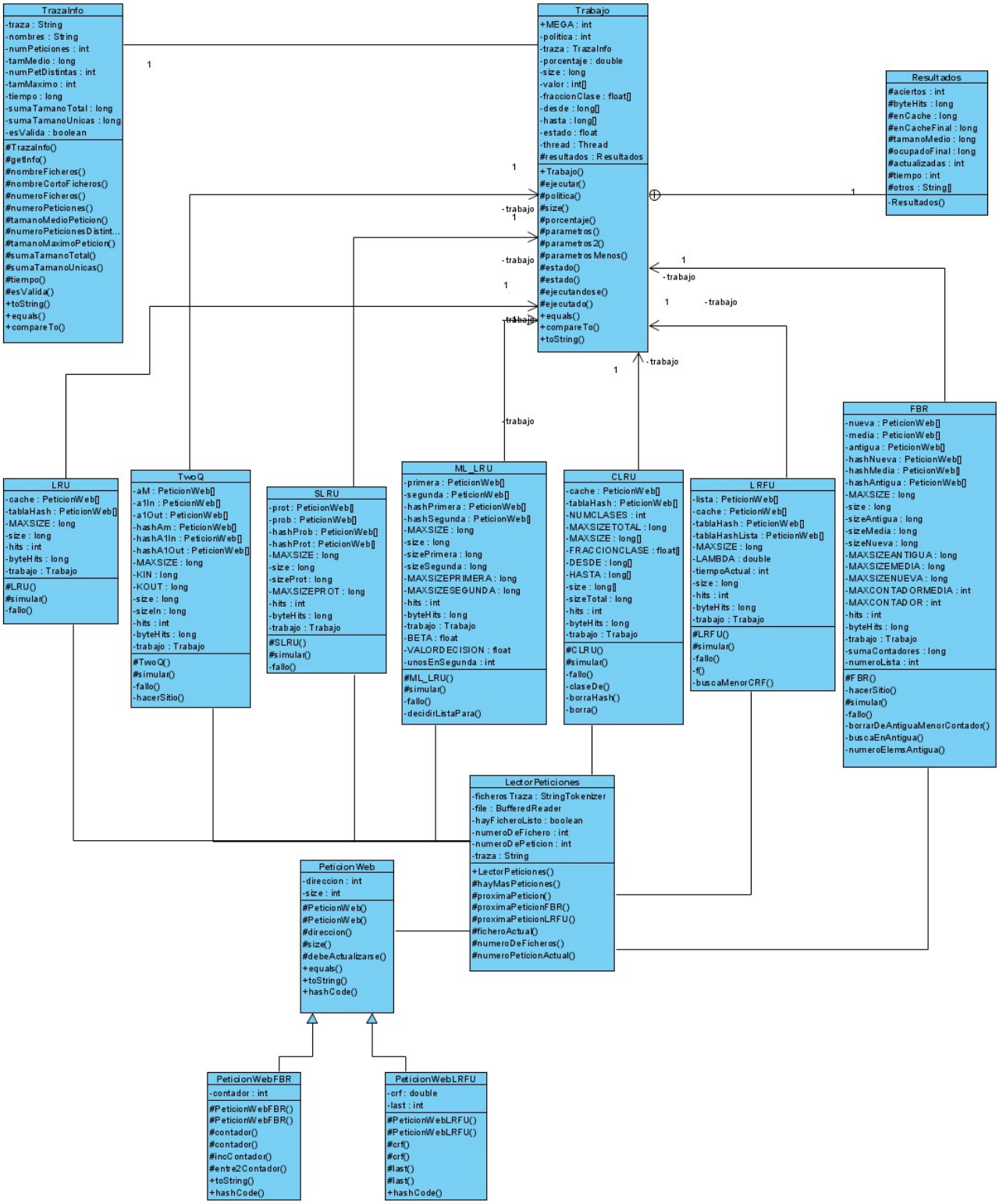

Figura 25: Diagrama Clases Simulaciones

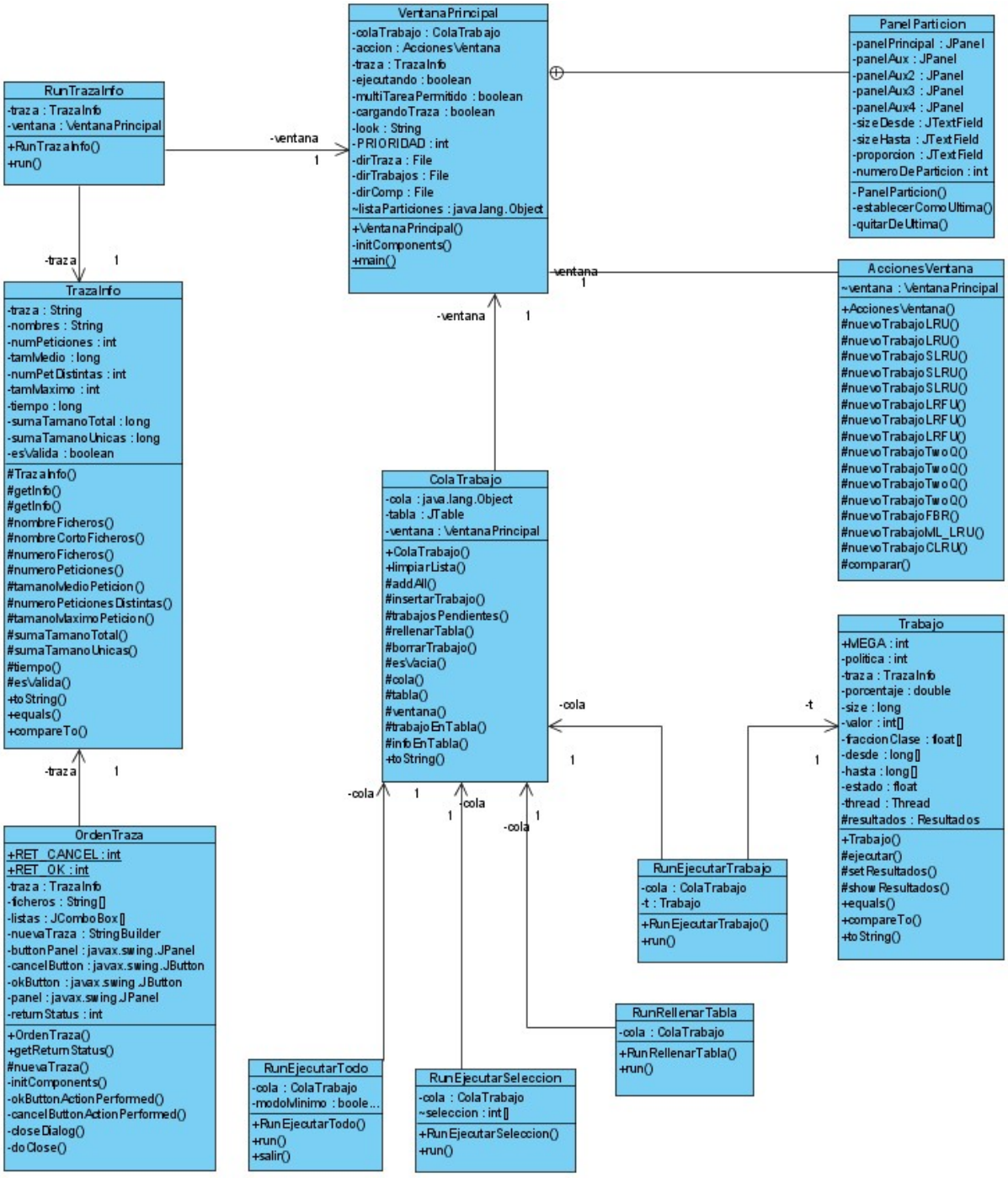

Figura 26: Diagrama Clases GUI

4.2. Descripción detallada de las clases

A continuación se describe detalladamente el funcionamiento de cada una de las clases que componen la aplicación, para ello se muestra el significado y la función de las principales variables y métodos de las mismas.

4.2.1. Clase Trabajo

Almacena todos los datos correspondientes a una simulación, desde la política a la que pertenece y los parámetros de la misma hasta los resultados obtenidos en la simulación.

4.2.1.1. Variables

-política: indica la política de reemplazo a la que pertenece el trabajo.
-traza: almacena la traza que será utilizada en la simulación.
-size: tamaño máximo de la caché en bytes.
-valor: array que almacena los parámetros de la política de reemplazo.
-estado: valor de 0 a 100 que indica el progreso de la simulación (0: la simulación no ha comenzado a ejecutarse. 100: simulación finalizada. Otro valor: progreso de la simulación).
-porcentaje: valor que se indicó en la aplicación en tanto por ciento.
-fraccionClase: sólo usado para la política C-LRU, es un array que indica los tamaños de cada una de las particiones respecto al tamaño total de la caché.
-desde, -hasta: sólo usados para la política C-LRU, son arrays que indican los rangos de tamaños de documentos que almacenarán cada una de las particiones.
-thread: hebra que ejecuta la simulación.
#resultados: almacena los resultados obtenidos en la simulación. Es una subclase que contiene los siguientes campos:

#aciertos: número de aciertos obtenidos en la simulación

#byteHits: número de aciertos en bytes obtenidos.

#enCache: media de elementos almacenados en caché durante la simulación.
#enCacheFinal: número de elementos en caché al finalizar la simulación.
#tamanoMedio: tamaño medio ocupado de la caché en bytes durante la simulación.
#ocupadoFinal: tamaño ocupado de la caché al finalizar la simulación.
#actualizadas: número de peticiones que se han actualizado.
#tiempo: tiempo que ha tardado en ejecutarse la simulación.
#otros: indica resultados adicionales en función de la política de reemplazo.

4.2.1.2. Métodos

#ejecutar(): ejecuta la simulación con los parámetros dados.
#setResultados(): guarda en la clase Resultados los resultados obtenidos en la simulación.
#showResultados(): devuelve una cadena tipo String con los resultados de la simulación.
#valorUnicoTrabajosIgualesMenosSize(): devuelve un valor único para todos los trabajos que tengan todos los parámetros iguales excepto el tamaño de la caché. Útil para generar los archivos de comparaciones.
#valorUnicoTrabajosIgualesMenosValor(): devuelve un valor único para todos los trabajos que tengan todos los parámetros iguales excepto un parámetro dado. Útil para generar los archivos de comparaciones.
+compareTo(): compara dos trabajos para tenerlos ordenados dentro de la cola de trabajos.

4.2.2. Clase TrazaInfo

Contiene la información de una traza, es decir, de una sucesión de peticiones realizadas por un grupo de usuarios al servidor proxy. La traza ha de estar compuesta por uno o varios ficheros de texto plano que deben tener una petición en cada línea donde se indique en primer lugar el tamaño de la petición en bytes y en segundo lugar un identificador numérico de la petición, estando estos dos campos separados por uno o varios espacios.

4.2.2.1. Variables

-traza: nombre de los ficheros que contienen la traza con la ruta completa. Si hay más de un fichero estarán separados por ";".
-nombres: nombre de los ficheros que contienen la traza. Si hay más de un fichero estarán separados por ";".
-numPeticiones: número de peticiones de la traza.
-tamMedio: tamaño medio en bytes de las peticiones.
-numPetDistintas: número de peticiones distintas de la traza.
-tamMaximo: tamaño en bytes de la mayor petición de la traza.
-tiempo: tiempo que tardó en ejecutarse la carga de la traza.
-sumaTamanoTotal: almacena la suma de los tamaño de todas las peticiones.
-sumaTamanoUnicas: almacena la suma de los tamaño de las peticiones únicas.
-esValida: indica si la traza es válida, es decir, si los ficheros existen y su formato es el correcto.

4.2.2.2. Métodos

#TrazaInfo(): constructor de la clase que necesita como parámetro un array de tipo java.io.File. con los ficheros que componen la traza.
#getInfo(): lee completamente la traza para comprobar que su formato es el correcto y para obtener datos de la misma como el número de peticiones que contiene o el tamaño de las mismas.

4.2.3. Clase PeticionWeb

Representa una petición que tras ser leída de la traza será almacenada en la caché. Contiene información sobre la misma y proporciona mecanismos para leerlas de los ficheros traza. Existen otras clases derivadas de PeticionWeb, en concreto PeticionWebFBR y PeticionWebLRFU, que añaden funciones adicionales necesarias para realizar las simulaciones de

los algoritmos de las políticas de reemplazo FBR y LRFU.

4.2.3.1. Variables

-direccion: almacena el identificador único de la petición indicado en la traza.
-size: indica el tamaño de la petición.
PeticionWebFBR incluye además:

-contador: lleva la cuenta del número de veces que se referenciado la petición estando en la parte *media* o *antigua*.

PeticionWebLRFU incluye además:

-crf: guarda el valor CRF de la petición.

-last: guarda el último momento en el que fue referenciada la petición.

4.2.3.2. Métodos

#PeticionWeb(): hay dos constructores. El primero crea una nueva instancia de la clase dados un identificador y un tamaño. El segundo crea una nueva petición a partir de un fichero del tipo java.io.BufferedReader.
#debeActualizarse(): indica si se tiene que actualizar un documento respecto por uno nuevo.
PeticionWebFBR incluye además:

#incContador(): incrementa en uno el valor del contador de referencias siempre que no supere el umbral C_{max}.

#entre2Contador(): reduce el contador de referencias a la mitad. Utilizado cuando se supera el umbral A_{max}.

4.2.4. Clase LectorPeticiones

Controla la lectura de peticiones de los ficheros de una traza. Comprueba el orden de los ficheros y de las peticiones, así como la integridad de las mismas.

4.2.4.1. Variables

-ficherosTraza: nombres de los ficheros traza separados por ";".
-file: fichero de la traza del cual se están leyendo peticiones actualmente.
-hayFicheroListo: indica si hay un fichero preparado para ser leído.
-numeroDeFichero: indica el numero de fichero del que se está leyendo.
-numeroDePeticion: indica el numero de la petición actual.
-traza: nombre de los ficheros traza.

4.2.4.2. Métodos

#proximaPeticion(): devuelven la siguiente petición (objeto de la clase PeticionWeb) de la traza. Existen también los métodos proximaPeticionFBR() y proximaPeticionLRFU() con la misma función pero que devuelven objetos de las clases PeticionWebFBR y PeticionWebLRFU, respectivamente.

4.2.5. Clase LRU

Realiza simulaciones de cachés administradas por la política de reemplazo *Least Recently Used* (LRU). Para ello, en primer lugar ha de ser inicializada con los datos necesarios, los cuales procederán normalmente de la interfaz gráfica de la aplicación donde el usuario los ha especificado, para poder crear las estructuras que almacenarán y gestionarán la caché. Una vez creada correctamente la instancia de la clase, se podrá proceder a la simulación de la caché. Dicho procedimiento es el mismo para el resto de clases que implementan las distintas políticas de reemplazo, por lo que a partir de ahora se omitirá explicarlo en cada una de ellas.

4.2.5.1. Variables

-cache: lista que almacena los documentos en caché.

-tablaHash: tabla hash para la caché.
-MAXSIZE: tamaño máximo de la caché en bytes.
-size: tamaño actual de la cache en bytes.
-hits: contador del número de aciertos durante la simulación.
-byteHits: contador del número de aciertos en bytes.
-trabajo: trabajo a ejecutar en la simulación.

Las variables -MAXSIZE, -size, -hits, -byteHits y -trabajo son comunes a todas las clases que implementan las simulaciones (clases SLRU, TwoQ, FBR, LRFU, ML_LRU y CLRU), por lo que de aquí en adelante se omitirá nombrarlas.

4.2.5.2. Métodos

#LRU(): constructor que inicializa las estructuras dado un trabajo para simular.
#simular(): ejecuta la simulación de la caché con los datos indicados por el trabajo dado.
-fallo(): utilizado en simular() para procesar un fallo.

4.2.6. Clase SLRU

Realiza las simulaciones de cachés utilizando la política de reemplazo multinivel *Segmented LRU* (SLRU).

4.2.6.1. Variables

-prot: listas que almacena los documentos en la parte protegida de la caché.
-prob: listas que almacena los documentos en la parte de prueba de la caché.
-hashProb, -hashProt: tablas hash de la parte de prueba y protegida, respectivamente.
-sizeProt: tamaño actual de la parte protegida.
-MAXSIZEPROT: tamaño máximo de la parte protegida.

4.2.6.2. Métodos

#SLRU(): constructor que inicializa las estructuras dado un trabajo para simular.
#simular(): ejecuta la simulación de la caché con los datos indicados por el trabajo dado.
-fallo(): utilizado en simular() para procesar un fallo.

4.2.7. Clase TwoQ

Realiza las simulaciones de cachés utilizando la política de reemplazo multinivel Two Queue (2Q).

4.2.7.1. Variables

-aM, -a1In, -a1Out: partes de la caché.
-hashAm, -hashA1In, -hashA1Out: tablas hash correspondientes a cada parte de la caché.
-KIN, -KOUT: tamaño máximo de A1In y A1Out, respectivamente.

4.2.7.2. Métodos

#TwoQ(): constructor que inicializa las estructuras dado un trabajo para simular.
#simular(): ejecuta la simulación de la caché con los datos indicados por el trabajo dado.
-fallo(): utilizado en simular() para procesar un fallo.
-hacerSitio(): utilizado en simular() para eliminar documentos de la caché hasta que haya espacio suficiente para una nueva petición dada.

4.2.8. Clase FBR

Realiza las simulaciones de cachés utilizando la política de reemplazo multinivel *Fre-*

quency-Based Replacement (FBR).

4.2.8.1. Variables

-nueva, media: listas que almacenan los documentos de las partes de la caché.
-antigua: parte antigua de la caché. Está dividida en varias listas, una para cada valor posible de los contadores de referencias de las peticiones.
-hashNueva, -hashMedia, -hashAntigua: tablas hash correspondientes a las partes de la caché.
-sizeAntigua, -sizeMedia, -sizeNueva: tamaño actual de las partes *Antigua*, *Media* y *Nueva.*
-MAXSIZEANTIGUA, -MAXSIZEMEDIA, -MAXSIZENUEVA: tamaño máximo de las partes *Antigua*, *Media* y *Nueva.*
-MAXCONTADORMEDIA, -MAXCONTADOR: valor máximo permitido para la media de los contadores y para un contador en particular, es decir A_{max} y C_{max}.
-sumaContadores: guarda la suma de los contadores de las peticiones actualmente en caché.
-numeroLista: utilizada para guardar el número de la lista *Antigua* en la que está la petición buscada.

4.2.8.2. Métodos

#FBR(): constructor que inicializa las estructuras dado un trabajo para simular.
#simular(): ejecuta la simulación de la caché con los datos indicados por el trabajo dado.
-fallo(): utilizado en simular() para procesar un fallo.
-hacerSitio(): utilizado en simular() para eliminar documentos de la caché hasta que haya espacio suficiente para una nueva petición dada.
-borrarDeAntiguaMenorContador(): elimina de la parte *Antigua* de la caché el documento con el menor contador.
-buscaEnAntigua(): busca una petición en la parte *Antigua*.
-numeroElemsAntigua(): devuelve el número de elementos de la parte *Antigua*.

4.2.9. Clase LRFU

Realiza simulaciones de cachés utilizando la política de reemplazo multinivel *Least Recently/Frequently Used* (LRFU).

4.2.9.1. Variables

-lista: almacena los documentos de la caché con menor CRF (normalmente CRF = 0).
-cache: lista no ordenada que almacena los documentos de la caché con CRF > 0.
-tablaHash, -tablaHashLista: tablas hash correspondientes a las dos partes de la cachés.
-LAMBDA: parámetro λ utilizado en la función que calcula el CRF.
-tiempoActual: cuenta el número de peticiones procesadas (simula el tiempo). No se ha utilizado el tiempo real por no disponer del tiempo transcurrido entre las distintas peticiones.

4.2.9.2. Métodos

#LRFU(): constructor que inicializa las estructuras dado un trabajo para simular.
#simular(): ejecuta la simulación de la caché con los datos indicados por el trabajo dado.
-fallo(): utilizado en simular() para procesar un fallo.
-f(): calcula el valor de la función *F*.
-buscaMenorCRF(): busca el documento con menor valor CRF dentro de la parte de la caché almacenada en la variable -cache.

4.2.10. Clase ML_LRU

Realiza las simulaciones de cachés utilizando la política de reemplazo multinivel *Multilevel LRU* (ML-LRU).

4.2.10.1. Variables

-primera, segunda: listas que almacenan los documentos de cada una de las partes de la caché.
-hashPrimera, -hashSegunda: tablas hash correspondientes a las dos partes de la caché.
-sizePrimera, -sizeSegunda: tamaño actual de las partes de la caché.
-MAXSIZEPRIMERA, -MAXSIZESEGUNDA: tamaño máximo de las partes de la caché.
-BETA: máximo de documentos en tanto por uno en la parte *Segunda* que han estado anteriormente en la parte *Primera*.
-VALORDECISION: guarda el parámetro de la función de decisión.
-unosEnSegunda: número de documentos almacenados actualmente en la parte *Segunda* han estado con anterioridad en la parte *Primera*.

4.2.10.2. Métodos

#ML_LRU(): constructor que inicializa las estructuras dado un trabajo para simular.
#simular(): ejecuta la simulación de la caché con los datos indicados por el trabajo dado.
-fallo(): utilizado en simular() para procesar un fallo.
-decidirListaPara(): dependiendo de la función decisión utilizada decide a qué parte de la caché ha de ir una nueva petición.

4.2.11. Clase CLRU

Realiza las simulaciones de cachés utilizando la política de reemplazo multinivel *Class-Based LRU* (C-LRU).

4.2.11.1. Variables

-cache: array compuesto de una lista por cada partición que guarda los documentos en caché.
-tablaHash: array de tablas hash correspondiente a las particiones de la caché.

-NUMCLASES: número de clases o particiones.
-MAXSIZETOTAL: tamaño máximo de la caché en bytes.
-MAXSIZE: array que indica el tamaño máximo de cada clase de la caché en bytes.
-FRACCIONCLASE: array con las fracciones del tamaño total de la caché asignado a cada clase.
-DESDE, -HASTA: arrays que indican los límites de los rangos de los tamaños de documentos permitidos en cada clase.
-size : array que indica el tamaño actual de cada clase de la cache.
-sizeTotal: tamaño actual de la caché.

4.2.11.2. Métodos

#CLRU(): constructor que inicializa las estructuras dado un trabajo para simular.
#simular(): ejecuta la simulación de la caché con los datos indicados por el trabajo dado.
-fallo(): utilizado en simular() para procesar un fallo.
-claseDe(): devuelve la clase a la que pertenece una petición en función de su tamaño.
-borraHash(): busca una petición en las tablas hash. Si está la elimina de la tabla hash y devuelve el número de la clase en la que estaba almacenada.

4.2.12. Clase VentanaPrincipal

Es la clase encargada de crear y administrar la interfaz gráfica de la aplicación, así como de capturar los eventos generados por las acciones del usuario y ejecutarlos.

4.2.12.1. Variables

-colaTrabajo: cola de trabajos correspondiente con la que aparece en la interfaz gráfica de la aplicación.
-accion: ejecuta las acciones relacionadas con las simulaciones.
-traza: indica la traza que está cargada actualmente en la aplicación.

-ejecutando: indica si se está ejecutando actualmente algún trabajo.
-multiTareaPermitido: indica si se pueden ejecutar varios trabajos a la vez.
-cargandoTraza: indica si se está cargando una traza.
-look: almacena el nombre del aspecto de la aplicación.
-PRIORIDAD: indica la prioridad de las hebras que ejecutan los trabajos.
-listaParticiones: lista con los paneles de la interfaz gráfica para las particiones de C-LRU.

4.2.12.2. Métodos

+VentanaPrincipal(): constructor que inicializa las estructuras, carga las opciones guardadas anteriormente o las opciones por defecto si es la primera ejecución y activa la ventana principal de la aplicación.
-botonXXXXActionPerformed(): es un grupo de funciones, donde XXXX es el nombre de una política de reemplazo. Capturan los eventos que se crean al pulsar los botones *Añadir a la* cola de trabajos. Leen los parámetros de las simulaciones que el usuario ha introducido y mediante la variable -accion de la clase AccionesVentana crea los correspondientes trabajos en la cola de trabajos.
-compararXXXXActionPerformed(): capturan los eventos lanzados cuando en la aplicación se pide generar un archivo para comparar, donde XXXX indica el tipo de comparación seleccionada. Accede a la clase AccionesVentana para crear dichos ficheros.
-cargarTrabajosActionPerformed(), -guardarTrabajosActionPerformed(): carga y guarda los trabajos en el sistema de ficheros.
-ejecutarTodosActionPerformed(), -ejecutarSeleccionActionPerformed(): capturan los eventos que piden ejecutar las simulaciones.
-borrarTodosActionPerformed(), -borrarSeleccionActionPerformed(): capturan los eventos que solicitan borrar uno o varios trabajos de la cola de trabajos.
-verResultadosActionPerformed(): muestra los resultados de una simulación.
-opcionesActionPerformed(): muestra un diálogo con las opciones de la aplicación y gestiona los cambios que se realicen.
#actualizaEstado(): actualiza la barra de estado escribiendo el número de trabajos pendientes.
#escribeEstado(): establece un mensaje en la barra de estado de la aplicación.

4.2.13. Clase ColaTrabajo

Gestiona la cola de trabajos mostrada en la aplicación.

4.2.13.1. Variables

-cola: conjunto ordenado de los trabajos o simulaciones que componen la cola de trabajos.
-tabla: almacena la tabla que muestra la lista de los trabajos en la aplicación.
-ventana: indica la ventana a la que pertenece la cola de trabajos.

4.2.13.2. Métodos

+limpiarLista(): borra todos los trabajos de la cola de trabajos.
#insertarTrabajo(): añade un nuevo trabajo.
#borrarTrabajo(): elimina un trabajo.
#addAll(): añade a la cola de trabajos actual una cola de trabajos obtenida normalmente de un fichero.
#rellenarTabla(): actualiza todos los valores mostrados en la tabla de la cola de trabajos.
#trabajoEnTabla(): devuelve el trabajo que está una cierta fila de la tabla que representa la cola de trabajos.
#infoEnTabla(): devuelve la información mostrada en una celda de la tabla.

4.2.14. Clase AccionesVentana

Ejecuta las acciones correspondientes a los eventos originados en la ventana principal de la aplicación relacionados con las simulaciones, es decir, actúa a modo de interfaz entre las clases de la interfaz gráfica y las clases que llevan a cabo las simulaciones.

4.2.14.1. Variables

-ventana: indica la ventana principal de la aplicación de la cual se realizarán las acciones.

4.2.14.2. Métodos

#nuevoTrabajoXXXX(): para cada política de reemplazo, dada por XXXX, crea uno o varios trabajos de simulaciones a ejecutar en función de los parámetros leídos de la interfaz gráfica.
#comparar(): crea los ficheros para realizar comparaciones con los resultados obtenidos en las simulaciones.

4.2.15. Clase RunEjecutarXXXX

Grupo de clases que implementan la clase Runnable cuya misión es crear nuevas hebras que dependiendo de lo que indique XXXX ejecutarán una, varias o todas las simulaciones pendientes de ser ejecutadas.

4.2.15.1. Variables

-cola: indica la cola de trabajos cuyos trabajos se van a ejecutar. Dependiendo del caso se ejecutarán una, varias o todas las simulaciones pendientes.

4.2.15.2. Métodos

-run(): lanza las ejecuciones de las simulaciones y guarda automáticamente la cola de trabajos después de finalizar cada simulación.

4.2.16. Clase RunTrazaInfo

Crea una nueva hebra que ejecuta la carga de los ficheros de una traza.

4.2.16.1. Variables

-traza: indica la traza a cargar.
-ventana: indica la ventana principal de la aplicación en la que cargar la traza.

4.2.16.2. Métodos

-run(): ejecuta el proceso de carga de una traza.

4.2.17. Clase RunRellenarTabla

Crea una hebra que existirá de forma indefinida hasta que se finalice la ejecución de la aplicación para actualizar el contenido de la tabla que representa a la cola de trabajos en la interfaz gráfica.

4.2.17.1. Variables

-cola: indica la cola de trabajos cuya tabla correspondiente tiene que actualizar.

4.2.17.2. Métodos

-run(): en un bucle infinito actualiza el contenido de la tabla periódicamente, es decir, cada cierto tiempo, para así ahorrar el tener que actualizarla constantemente.

4.3. Funcionamiento de las simulaciones

El funcionamiento de las simulaciones puede verse en el diagrama de flujo mostrado en la Figura 27.

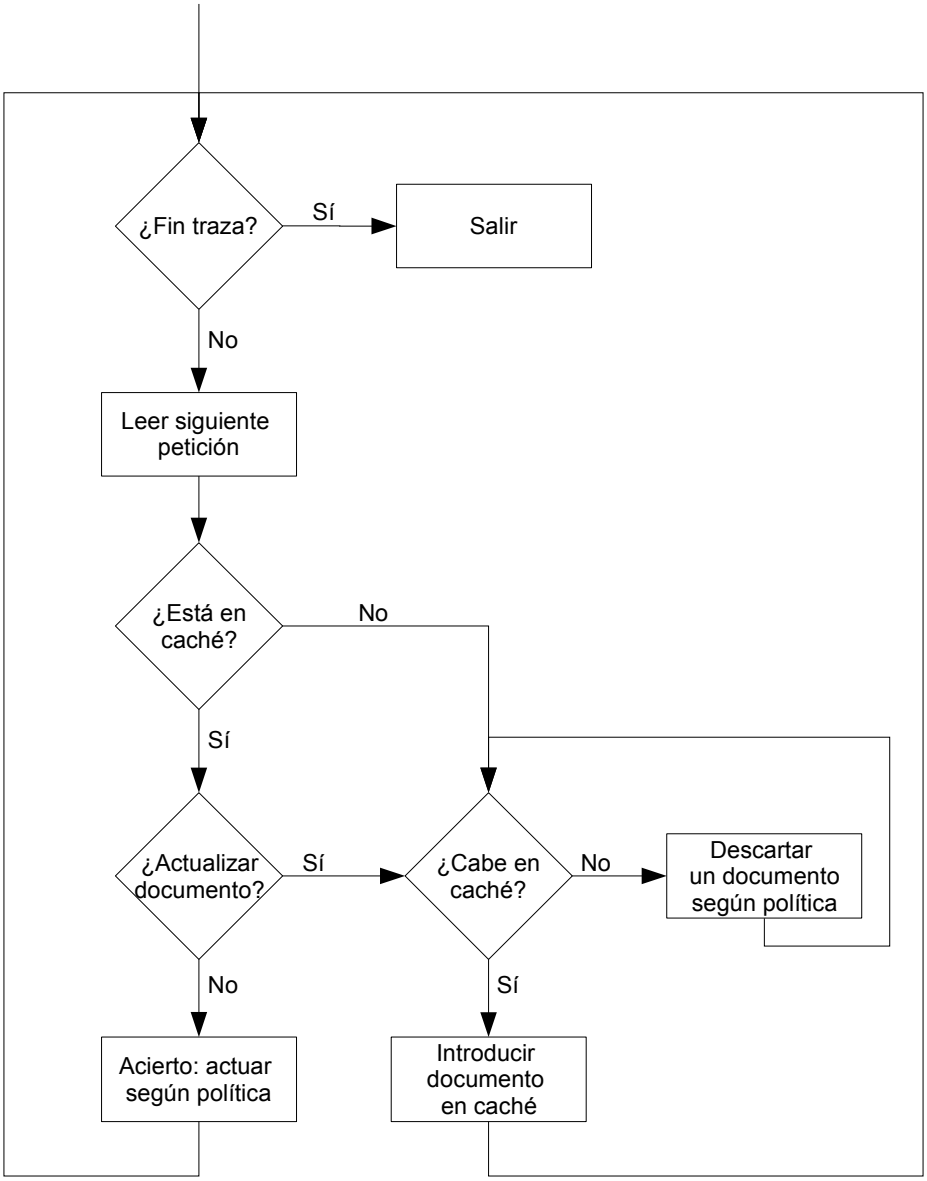

Figura 27: Diagrama Flujo Simulaciones

En primer lugar, y partiendo de una traza válida, el método *simular* lee una petición de la traza, y se busca en caché el documento de dicha petición. La búsqueda del documento en la caché se realiza mediante las tablas hash habilitadas para ello, y puesto que las cachés están divididas en varias partes, a excepción de LRU, se ha de buscar el documento en todas las tablas hash correspondientes.

Si el documento está almacenado en caché hay que comprobar si hay que actualizar el documento existente por el nuevo o no. Se considera que hay que actualizar el documento cuando su tamaño haya variado menos del 5%, en caso contrario se considera como corte de conexión y el documento no se actualiza.

Por tanto, si el documento no tiene que ser actualizado, se considera como un acierto en la caché, y se procederá a realizar las acciones pertinentes en función de la política de reemplazo que se esté simulando, por ejemplo, en el caso de LRU habría que mover el documento referenciado a la primera posición de la caché.

En el caso de tener que actualizar el documento, estamos en un fallo, y se procede eliminando el documento ya existente y si hay espacio suficiente para almacenar el nuevo documento, se introduce en caché. Si no hubiera espacio libre suficiente para el nuevo documento, habría que eliminar uno o varios elementos de la caché en función de la política de reemplazo que se esté simulando, por ejemplo, en el caso de LRU se descartaría el último elemento de la caché. Una vez que hubiera espacio para el nuevo documento se introduciría en la caché.

A continuación se volvería a intentar leer otra petición de la traza. En el caso de que no hubiera más peticiones, la simulación concluiría. Si la traza contiene más peticiones por leer se volverá a realizar el proceso ya explicado.

5. Evaluación de políticas de reemplazo

Utilizando la aplicación creada se han realizado simulaciones de cachés Web utilizando las distintas políticas de reemplazo explicadas en el capítulo 2. En esta sección se mostrarán y compararán los resultados obtenidos en las más de 500 simulaciones realizadas, para ello se usarán las métricas habituales para medir el rendimiento de las cachés, como son los aciertos (*Hit Rate*) y los aciertos en bytes (*Byte Hit Rate*). La traza utilizada para las simulaciones es RTP[8], que ha sido obtenida de los servidores del Research Triangle Park (RTP) de Carolina del Norte en EE.UU. entre los días 7 y 11 de junio de 2004, ambos inclusive. RTP es un parque tecnológico de uso tanto público como privado creado en 1.959. Los datos principales de la traza escogida son los siguientes:

- Número de total de peticiones: 4.040.036
- Número de peticiones únicas: 1.713.902
- Tamaño total de las peticiones: 37,64 Gbytes
- Tamaño de las peticiones únicos: 17,044 Gbytes
- Tamaño medio de las peticiones: 10.006 bytes

Se han hecho las simulaciones para tamaños de caché correspondientes al 2%, 5%, 10%, 30% y 50% del tamaño máximo que sería ocupado en una caché de tamaño infinito al procesar la traza RTP. Este tamaño se ha determinado que es 18.297.752.326 bytes, y para dicho tamaño se obtienen 1.953.800 aciertos (48,36%) que suponen 11.593.782.535 bytes acertados (28,68%). Por tanto, los tamaños de caché en las distintas simulaciones serán 365.955.046 bytes (2% de 18.297.752.326 bytes), 914.887.616 bytes (5%), 1.829.775.232 bytes (10%), 5.489.325.697 bytes (30%) y 9.148.876.163 bytes (50%).

Se evaluarán en primer lugar la influencia de los parámetros específicos de las políticas de reemplazo en los resultados obtenidos y posteriormente se compararán los mejores resultados de cada política para cada uno de los tamaños de caché probados, para lo cual se

escogerán las configuraciones de cada política que hayan obtenido un mayor índice de aciertos y un mayor índice de bytes acertados.

5.1. SLRU

5.1.1. Tamaño de la parte protegida

Se han realizado simulaciones para tamaños de la parte protegida iguales al 5, 20, 35, 50, 65, 80 y 95 por ciento del tamaño total de la caché. Los resultados obtenidos se muestran en la Figura 28.

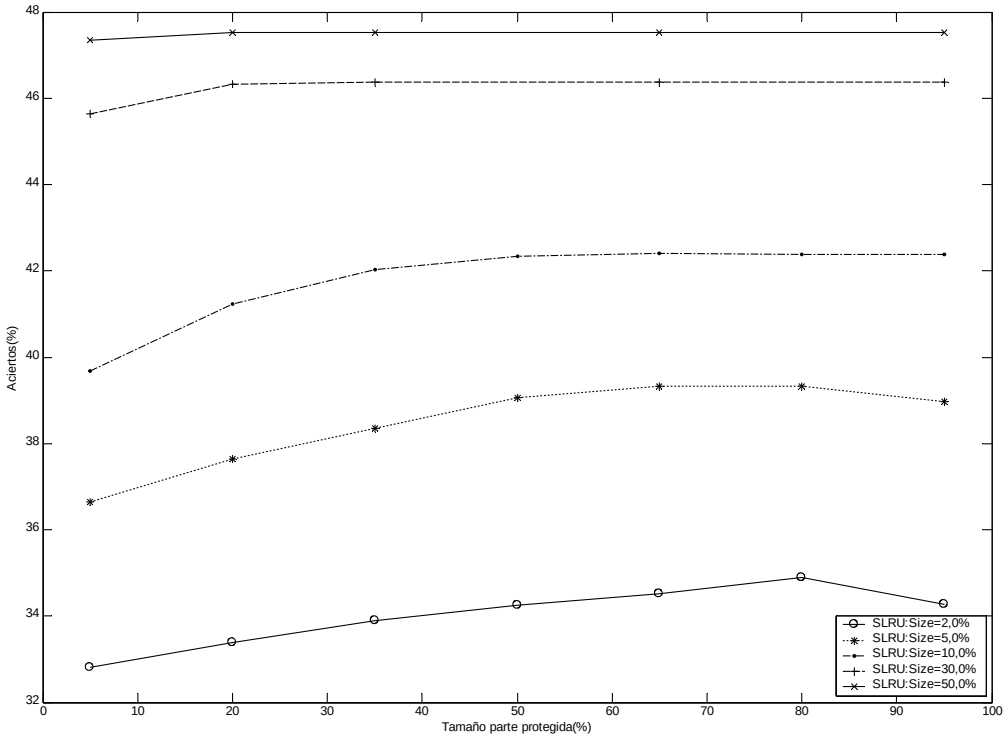

Figura 28: SLRU: Tamaño Parte Protegida vs Hit Rate

Se puede observar en general que a mayor tamaño de la parte protegida, mayor índice de aciertos, cuyo máximo se produce concretamente con un tamaño de la parte protegida entre el 65 y el 80% en cachés pequeñas, y para tamaños de caché mayores con un tamaño de la

parte protegida a partir del 35%. El tamaño óptimo de la parte protegida desciende cuando aumenta el tamaño de la caché a consecuencia de que a mayor tamaño de la caché los elementos que más veces son referenciados, que son los que se almacenan en la parte protegida, representan una proporción menor en relación al total de elementos almacenados en la caché.

A continuación se presentan los resultados correspondientes a los aciertos en bytes en la Figura 29, donde se muestra que para los tamaños pequeños de caché, el porcentaje acertado varía enormemente. Estas grandes fluctuaciones de los aciertos en bytes obtenidos con el uso de SLRU y también con el uso de otras políticas, como veremos en apartados posteriores, es debido a que son políticas de reemplazo orientadas a conseguir el mayor número de aciertos posible y por ello los elementos sobre los que se han producido dichos aciertos pueden variar entre las distintas simulaciones, y por consiguiente variará también el número de bytes acertados.

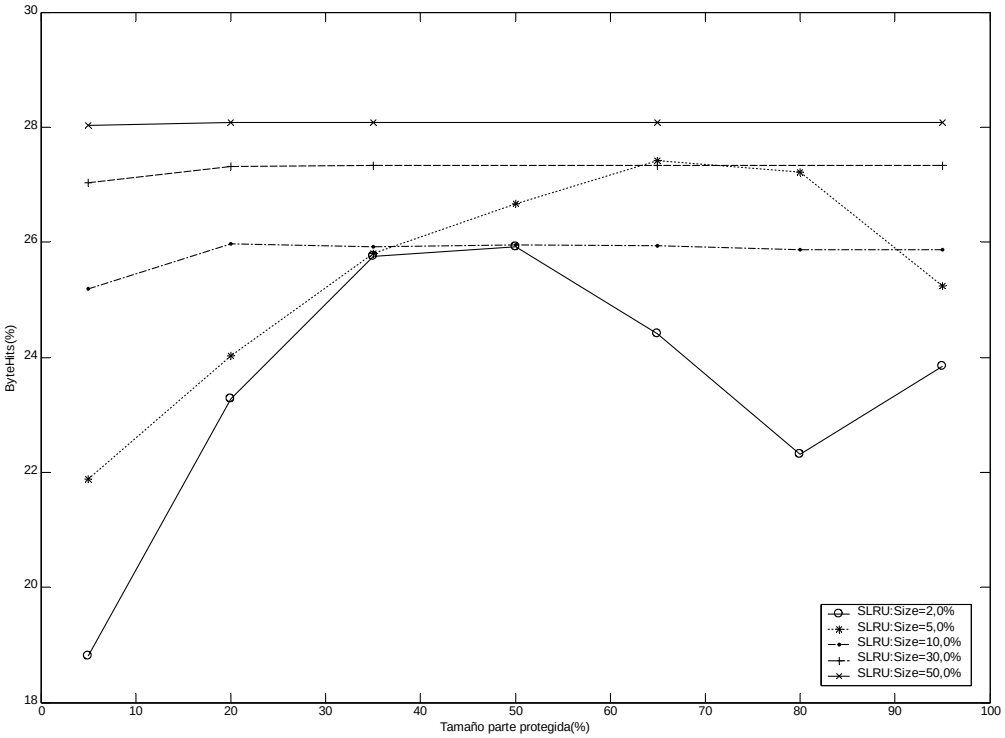

Figura 29: SLRU: Tamaño Parte Protegida vs Byte Hit Rate

Puede observarse que el máximo se consigue con un tamaño de parte protegida sobre

el 50-65%, mientras que para tamaños de caché a partir del 10%, el tamaño de la parte protegida no influye demasiado, siempre que sea de un mínimo del 20% del total.

En las tablas 1 y 2 se muestran los resultados obtenidos en las simulaciones. Se han marcado en negrita los mejores resultados para cada tamaño de caché.

Parte protegida / **Tamaño caché**	**5 %**	**20 %**	**35 %**	**50 %**	**65 %**	**80 %**	**95 %**
2 %	32.81	33.37	33.90	34.24	34.50	**34.88**	34.26
5 %	36.63	37.64	38.34	39.05	**39.32**	39.31	38.97
10 %	39.68	41.22	42.02	42.34	**42.39**	42.38	42.38
30 %	45.63	46.32	**46.37**	**46.37**	**46.37**	**46.37**	**46.37**
50 %	47.34	**47.52**	**47.52**	**47.52**	**47.52**	**47.52**	**47.52**

Tabla 1: SLRU Hit Rate

Parte protegida / **Tamaño caché**	**5 %**	**20 %**	**35 %**	**50 %**	**65 %**	**80 %**	**95 %**
2 %	18.81	23.27	25.76	**25.91**	24.40	22.32	23.84
5 %	21.88	24.02	25.81	26.67	**27.42**	27.22	25.23
10 %	25.18	**25.97**	25.93	25.96	25.93	25.88	25.88
30 %	27.03	**27.32**	**27.32**	**27.32**	**27.32**	**27.32**	**27.32**
50 %	28.03	**28.08**	**28.08**	**28.08**	**28.08**	**28.08**	**28.08**

Tabla 2: SLRU Byte Hit Rate

5.1.2. Conclusiones

Los resultados de las simulaciones realizadas muestran que la mayor eficiencia de SLRU se produce cuando el único parámetro de la política, que indica el tamaño de la parte protegida, se sitúa alrededor del 65% para todos los casos estudiados salvo algunas excepciones, siendo el tamaño óptimo de la parte protegida muy similar tanto para conseguir el mejor índice de aciertos como para el de bytes acertados.

5.2. 2Q

5.2.1. Tamaño de A1In (kIn)

En las simulaciones que se han llevado a cabo se han tomado como tamaños de A1In el 5, 20, 35, 50, 65, 80 y 95 por ciento del tamaño total de la caché. Los resultados han sido que aparecen en la Figura 30.

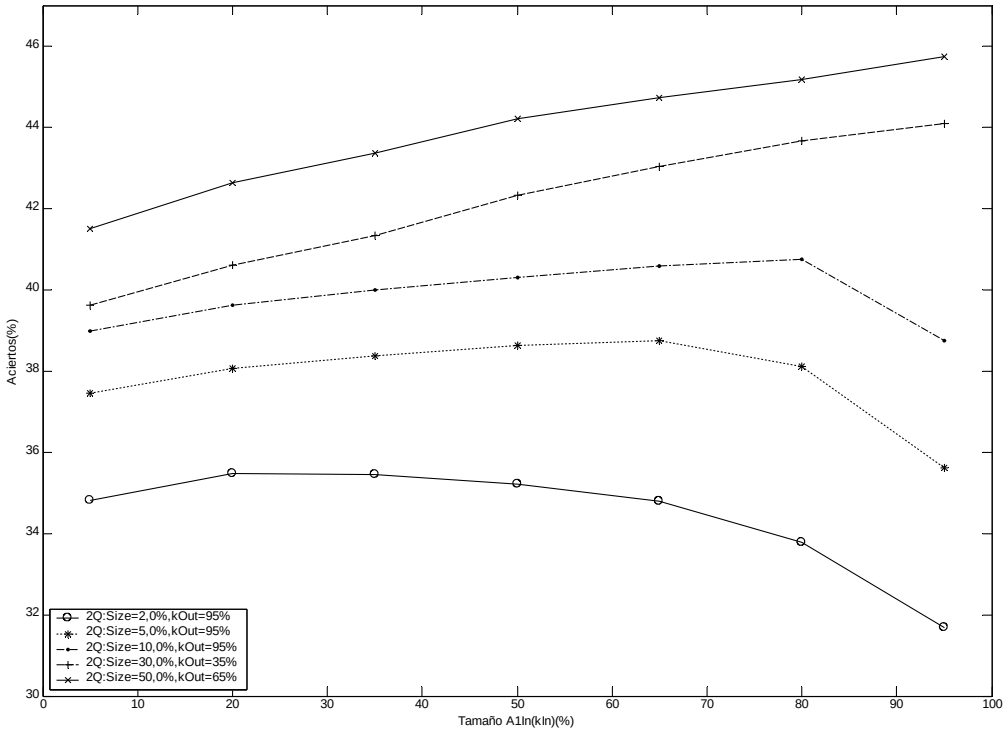

Figura 30: 2Q: Tamaño A1In vs Hit Rate

En los resultados se observa que el mejor tamaño de A1In para obtener un mayor índice de aciertos crece cuando crece también el tamaño de la caché, algo que ya vimos que ocurría también en la política SLRU debido a que la proporción de elementos que son más referenciados en un largo periodo de tiempo, que son los que se almacenan en la cola Am, disminuye respecto al total de elementos almacenados en la caché, y por tanto también disminuye la porción de tamaño necesario para almacenarlos. Concretamente, el valor óptimo de kIn

es del 20%, 65%, 80% para tamaños de caché del 2%, 5% y 10% respectivamente, y del 95% para cachés de tamaños 30% y 50%. Los valores de kOut que se muestran en los resultados son los que han obtenido los mejores resultados en lo que a número de aciertos se refiere para cada uno de los tamaños de caché, como podrá comprobarse en el apartado 5.2.2. A continuación se muestra la Tabla 3, donde aparecen los resultados obtenidos.

kIn / **Tamaño caché**	**5 %**	**20 %**	**35 %**	**50 %**	**65 %**	**80 %**	**95 %**
2 %	34.82	**35.48**	35.45	35.23	34.78	33.79	31.68
5 %	37.46	38.07	38.36	38.62	**38.74**	38.10	35.63
10 %	38.97	39.62	40.00	40.31	40.58	**40.74**	38.75
30 %	39.61	40.60	41.34	42.32	43.02	43.66	**44.09**
50 %	41.51	42.64	43.36	44.21	44.74	45.17	**45.74**

Tabla 3: 2Q (kIn) Hit Rate

En cuanto a los bytes acertados los resultados obtenidos en función del tamaño de A1In son muy variables para tamaños de caché de hasta el 10% como puede verse en la Tabla 4 y en la Figura 31, donde se observa también que para tamaños mayores de caché los aciertos en bytes obtenidos tienden a crecer cuando también lo hace kIn.

kIn / **Tamaño caché**	**5 %**	**20 %**	**35 %**	**50 %**	**65 %**	**80 %**	**95 %**
2 %	18.38	24.86	**26.06**	23.33	24.09	22.67	21.32
5 %	23.22	26.20	26.68	25.17	26.10	**28.23**	23.97
10 %	25.50	22.06	24.19	**26.86**	24.26	24.86	24.72
30 %	24.45	24.45	25.09	25.26	26.15	**27.37**	26.57
50 %	23.86	24.87	25.78	25.86	26.55	27.25	**27.48**

Tabla 4: 2Q (kIn) Byte Hit Rate

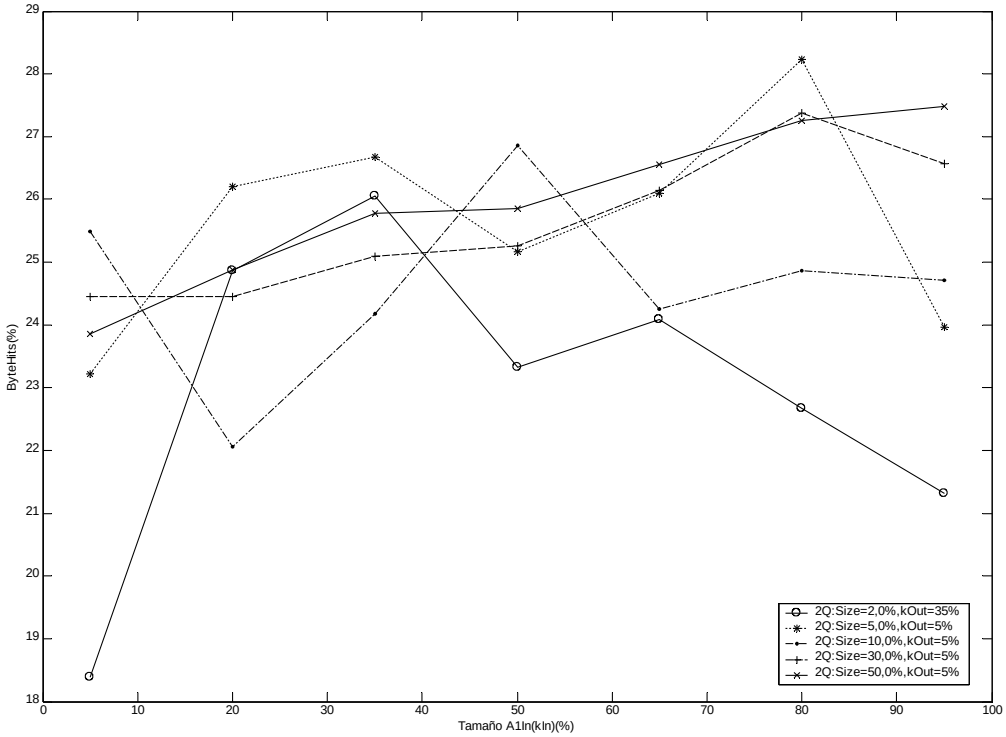

Figura 31: 2Q: Tamaño A1In vs Byte Hit Rate

5.2.2. Tamaño A1Out (kOut)

Se han realizado simulaciones con tamaños del 5, 35, 65 y 95 por ciento. El significado de dicho porcentaje se explicó en el capítulo 3.3.3.

A continuación se muestra en la Figura 32 y en la Tabla 5 los resultados obtenidos en función de kOut. Se observa que los aciertos tienden a aumentar cuando también aumenta el tamaño de A1Out, aunque esto ocurre en menor medida para los tamaños de caché más grandes, para los cuales estableciendo un valor de kOut a partir del 35% el número de aciertos varía muy poco.

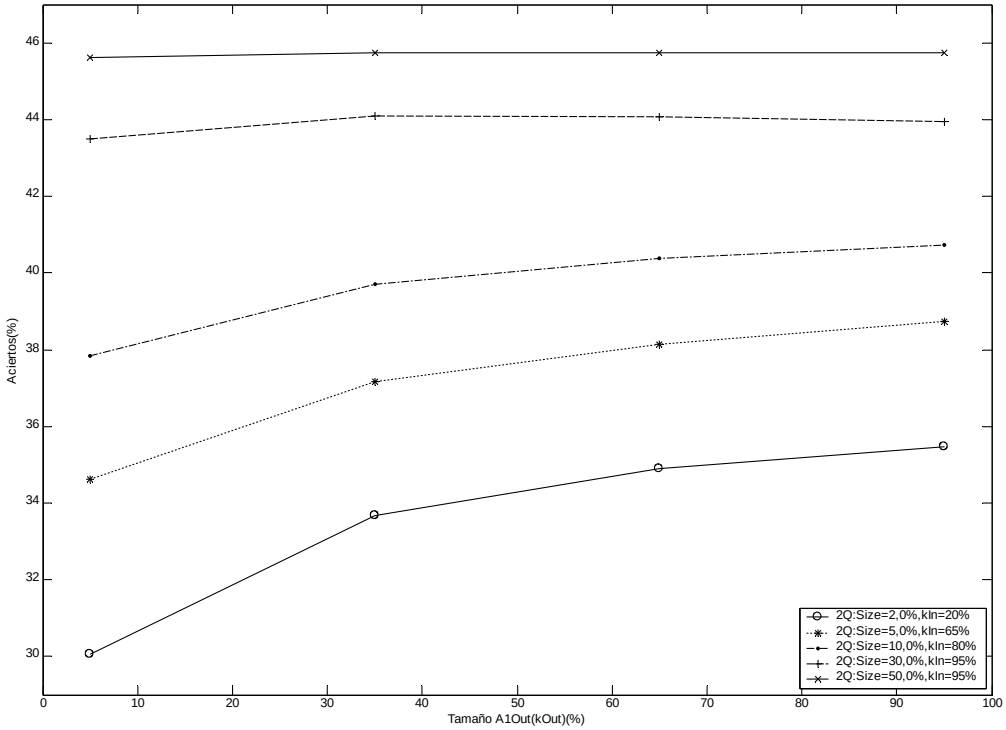

Figura 32: 2Q: Tamaño A1Out vs Hit Rate

kOut / Tamaño caché	5 %	35 %	65 %	95 %
2 %	30.06	33.68	34.89	**35.48**
5 %	34.62	37.18	38.14	**38.74**
10 %	37.84	39.71	40.37	**40.74**
30 %	43.50	**44.09**	44.07	43.94
50 %	45.60	45.73	**45.74**	**45.74**

Tabla 5: 2Q (kOut) Hit Rate

Ahora se van a estudiar los aciertos en bytes que se han obtenido en las simulaciones realizadas en función del tamaño de A1Out, que aparecen en la Figura 33.

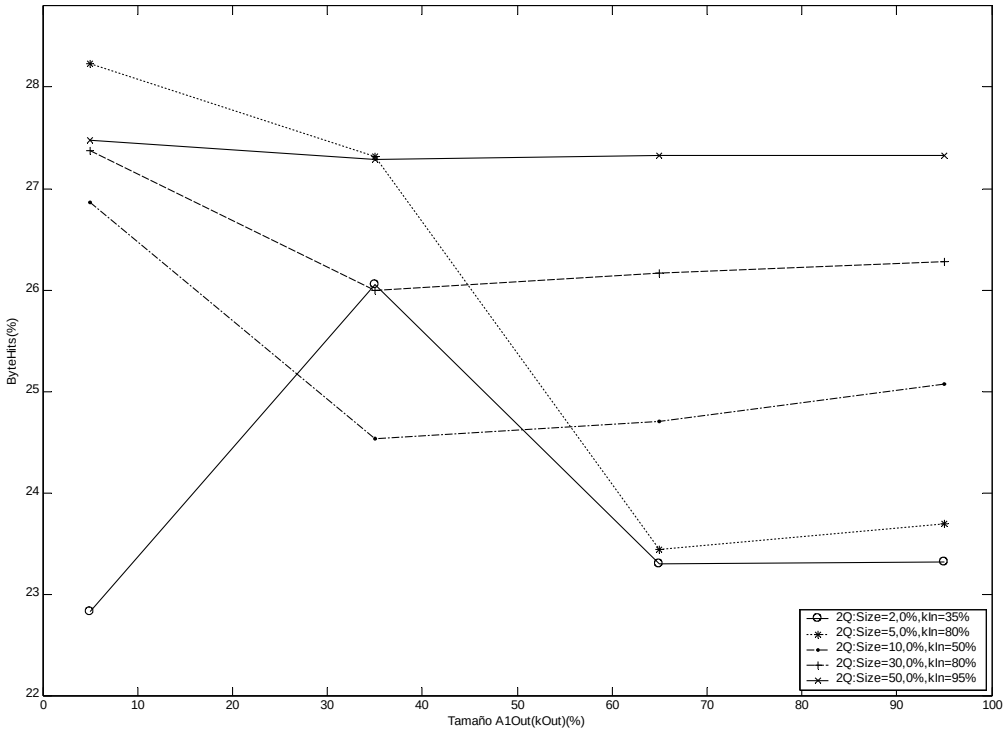

Figura 33: 2Q: Tamaño A1Out vs Byte Hit Rate

Puede observarse cómo los bytes acertados en cachés del 2% y 5% llegan a superar incluso a los acertados en cachés de tamaños superiores. El tamaño óptimo de A1Out en este caso es del 5% para todos los tamaños de caché probados excepto para el 2%, donde los mejores resultados se consiguen con un valor de kOut del 35%. Estos valores de kOut son prácticamente los opuestos a los que consiguen los mejores índices de aciertos. Los resultados obtenidos se muestran en la Tabla 6.

kOut / **Tamaño caché**	**5 %**	**35 %**	**65 %**	**95 %**
2 %	22.83	**26.06**	23.30	23.32
5 %	**28.23**	27.32	23.44	23.70
10 %	**26.86**	24.53	24.71	25.08
30 %	**27.37**	26.00	26.17	26.28
50 %	**27.48**	27.29	27.33	27.32

Tabla 6: 2Q (kOut) Byte Hit Rate

5.2.3. Conclusiones

Las simulaciones realizadas muestran que el tamaño óptimo de A1In crece cuando también lo hace el tamaño de la caché, exceptuando el caso en el que para conseguir el mayor índice de bytes acertados en una caché del 5% habría que establecer kIn en un 80%. En cuanto a kOut, hay que diferenciar entre si lo que se busca es obtener un mayor número de aciertos, en cuyo caso el valor óptimo es del 95% excepto para la caché de tamaño 30% que es del 35%, o si lo que se quiere es conseguir el mayor número posible de aciertos en bytes, en cuyo caso, y según los resultados obtenidos, el tamaño óptimo de A1Out sería del 5% excepto para la caché de tamaño 2%, donde se sitúa en el 35%.

5.3. FBR

5.3.1. C_{max}

Se han realizado simulaciones con valores de C_{max} de 10 y 20. Los resultados que aparecen en la Tabla 7 y en la figuras 34 y 35 muestran que este parámetro apenas tiene efecto sobre el rendimiento de la caché excepto en un caso en el que 10 obtiene mejores resultados.

C_{max}	**10**	**20**
Tamaño caché	HR / BHR	HR / BHR
2 %	**35.36** / **26.19**	**35.36** / 25.87
5 %	**39.41** / **28.48**	**39.41** / **28.48**
10 %	**42.14** / **26.39**	**42.14** / **26.39**
30 %	**46.20** / **27.32**	**46.20** / **27.32**
50 %	**47.42** / **28.05**	**47.42** / **28.05**

Tabla 7: FBR (C_{max}) Resultados obtenidos

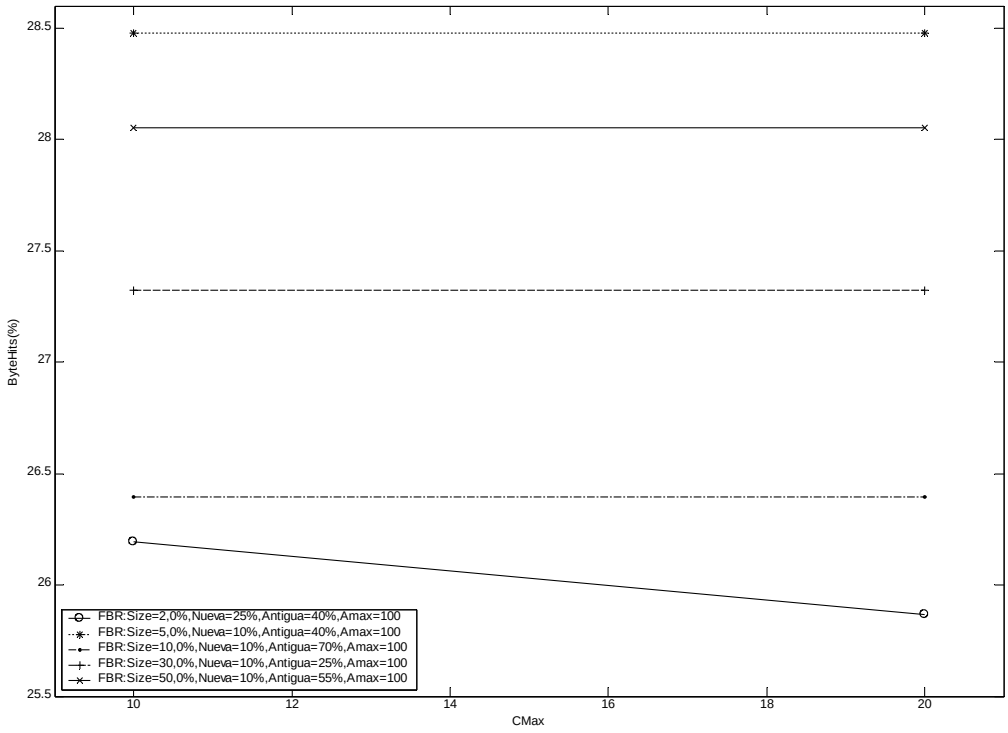

Figura 34: Figura 35: FBR: Cmax vs Byte Hit Rate

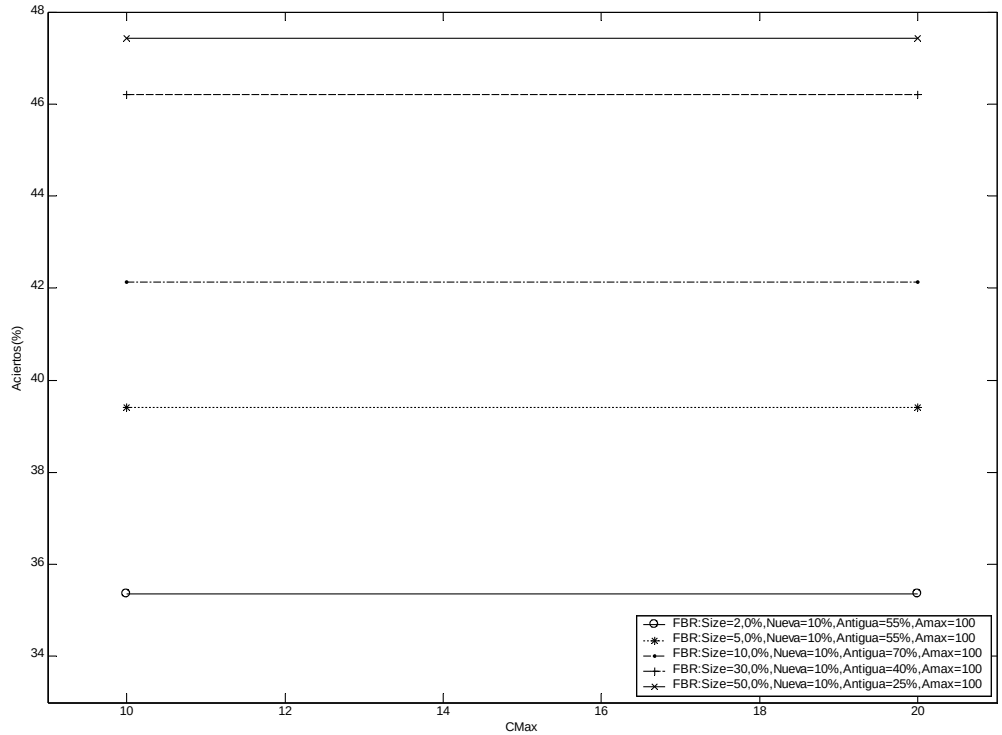

Figura 35: Figura 34: FBR: Cmax vs Hit Rate

A partir de ahora, en el resto de comparaciones de la política de reemplazo FBR se omitirá especificar el valor de C_{max}, ya que siempre será 10.

5.3.2. A_{max}

Se han realizado simulaciones con valores de A_{max} 2, 4 y 100. Los resultados muestran que para cachés de tamaños 2, 5 y 10 por ciento, el valor óptimo de A_{max} es 100 (o cualquier otro valor siempre que $A_{max} > C_{max}$), mientras que en cachés mayores variaciones en A_{max} no influyen en el rendimiento de la caché.

A continuación, las figuras 36 y 37 y la Tabla 8 muestran los resultados obtenidos en las simulaciones en función del valor de A_{max}. Nótese que la escala del eje que indica el valor de A_{max} es logarítmica y no lineal como en el resto de gráficas, para poder diferenciar mejor las diferencias.

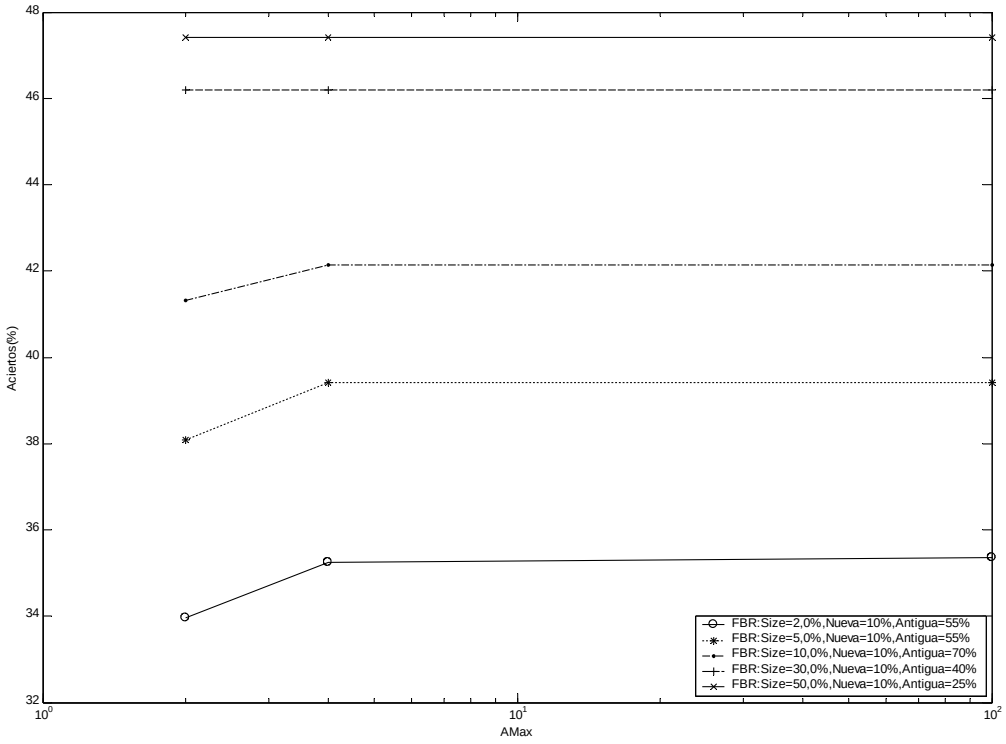

Figura 36: FBR: A_{max} vs Hit Rate

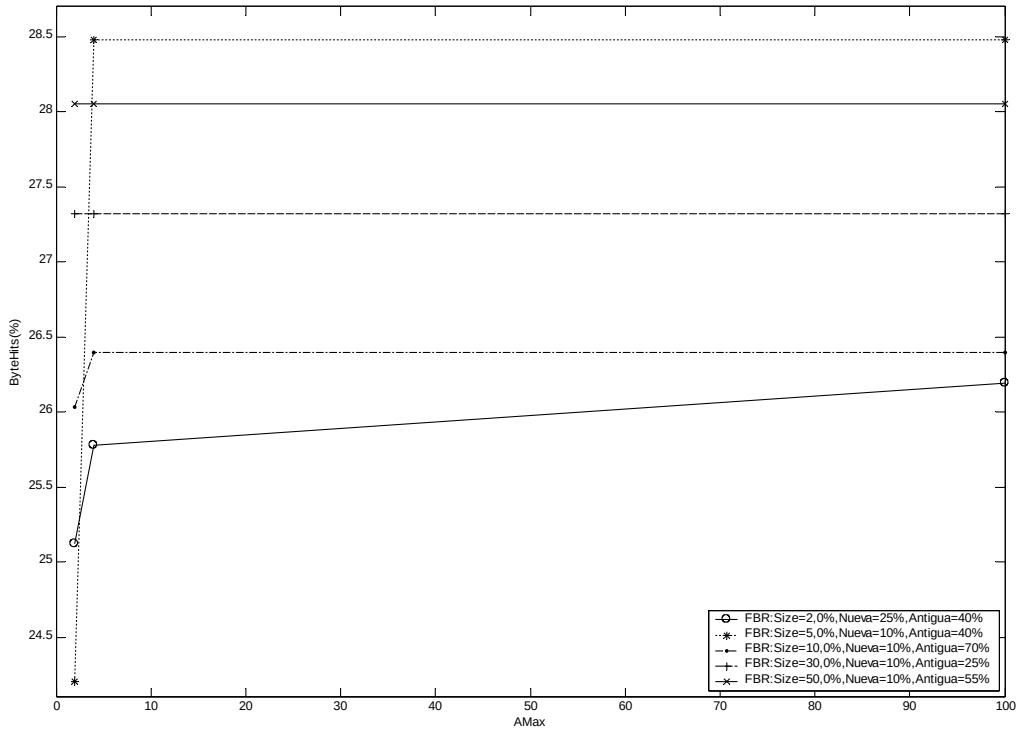

Figura 37: FBR: A_{max} vs Byte Hit Rate

A_{max}	**2**	**4**	**100**
Tamaño caché	HR / BHR	HR / BHR	HR / BHR
2 %	33.95 / 25.12	35.25 / 25.78	**35.36** / **26.20**
5 %	38.09 / 24.20	**39.41** / **28.48**	**39.41** / **28.48**
10 %	41.30 / 26.03	**42.14** / **26.39**	**42.14** / **26.39**
30 %	**46.20** / **27.32**	**46.20** / **27.32**	**46.20** / **27.32**
50 %	**47.42** / **28.05**	**47.42** / **28.05**	**47.42** / **28.05**

Tabla 8: FBR (A_{max}) Resultados obtenidos

A partir de ahora, en el resto de comparaciones de la política de reemplazo FBR se omitirá especificar el valor de A_{max}, ya que siempre será 100.

5.3.3. Tamaño parte *Nueva*

Se han realizado simulaciones con tamaños de la parte *Nueva* de la caché iguales al 10, 25, 40, 55 y 70 por ciento del tamaño total de la caché.

A continuación se muestran los aciertos obtenidos para los distintos tamaños de la parte *Nueva*. Como en todos los resultados que se muestran, se han establecido los valores de los otros parámetros de la política, en este caso, el tamaño de la parte *Antigua* que hace que se obtengan los mejores resultados en cada caso estudiado. Los resultados aparecen en la Figura 38 y en la Tabla 9. Se puede observar cómo para todos los tamaños de caché probados, los mayores índices de aciertos se producen cuando el tamaño de la parte *Nueva* es el 10% del tamaño total de la caché. El número de aciertos desciende cuanto mayor sea el tamaño de la parte *Nueva*.

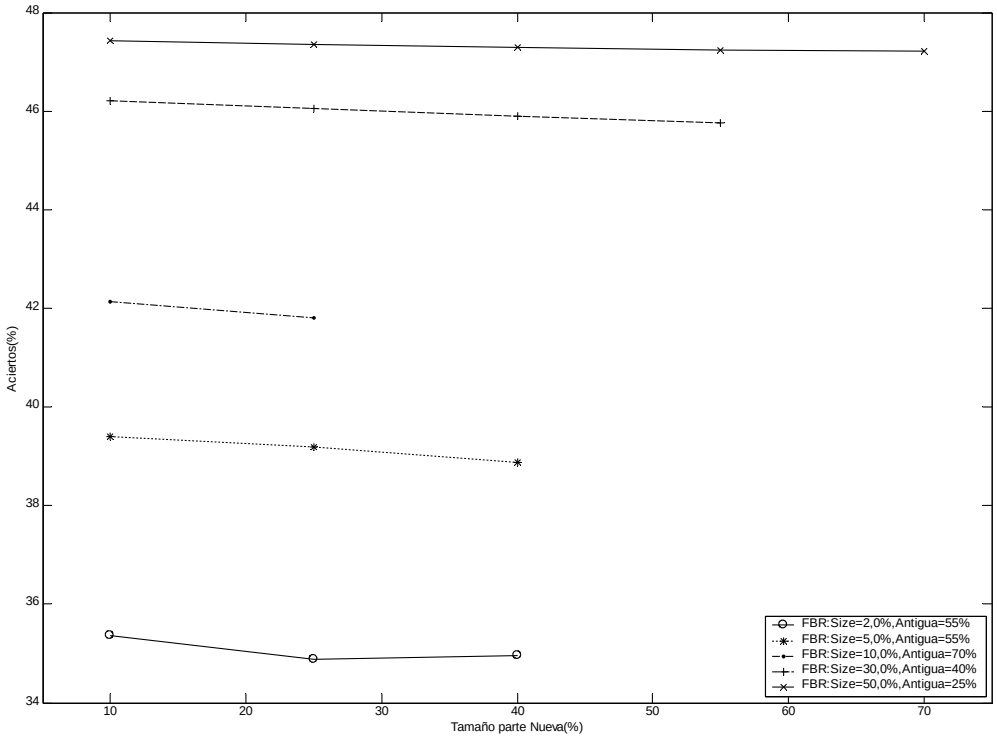

Figura 38: FBR: Tamaño Parte Nueva vs Hit Rate

En la Figura 38 puede verse que aparecen algunos huecos, lo cual ocurre cuando por ejemplo, el tamaño de la parte *Antigua* es del 70%, lo cual hace que la suma de los tamaños de las partes *Nueva* y Media no pueda ser mayor del 30%.

Parte Nueva / **Tamaño caché**	**10 %**	**25 %**	**40 %**	**55 %**	**70%**
2 %	**35.37**	34.88	34.96	-	-
5 %	**39.41**	39.18	38.87	-	-
10 %	**42.14**	41.81	-	-	-
30 %	**46.20**	46.05	45.91	45.76	-
50 %	**47.42**	47.35	47.28	47.23	47.21

Tabla 9: FBR (Parte Nueva) Hit Rate

En cuanto al índice de aciertos en bytes, los resultados son similares, puesto que de nuevo se obtienen mejores resultados con un tamaño de la parte nueva del 10% excepto para un tamaño de caché del 2% donde el mayor número de bytes acertados se produce cuando la parte *Nueva* ocupa el 25% del tamaño total de la caché.

Al igual que ocurría con las políticas anteriores, se producen grandes variaciones en los bytes acertados, obteniéndose en ocasiones mejores resultados con tamaños de caché inferiores que con tamaños mayores. Dichos resultados se muestran en la Tabla 10 y en la Figura 39.

Parte Nueva / **Tamaño caché**	**10 %**	**25 %**	**40 %**	**55 %**	**70%**
2 %	24.01	**26.19**	23.59	19.90	-
5 %	**28.48**	24.62	24.58	23.04	-
10 %	**26.39**	26.30	-	-	-
30 %	**27.32**	27.27	27.19	27.13	27.08
50 %	**28.05**	28.02	28.01	-	-

Tabla 10: FBR (Parte Nueva) Byte Hit Rate

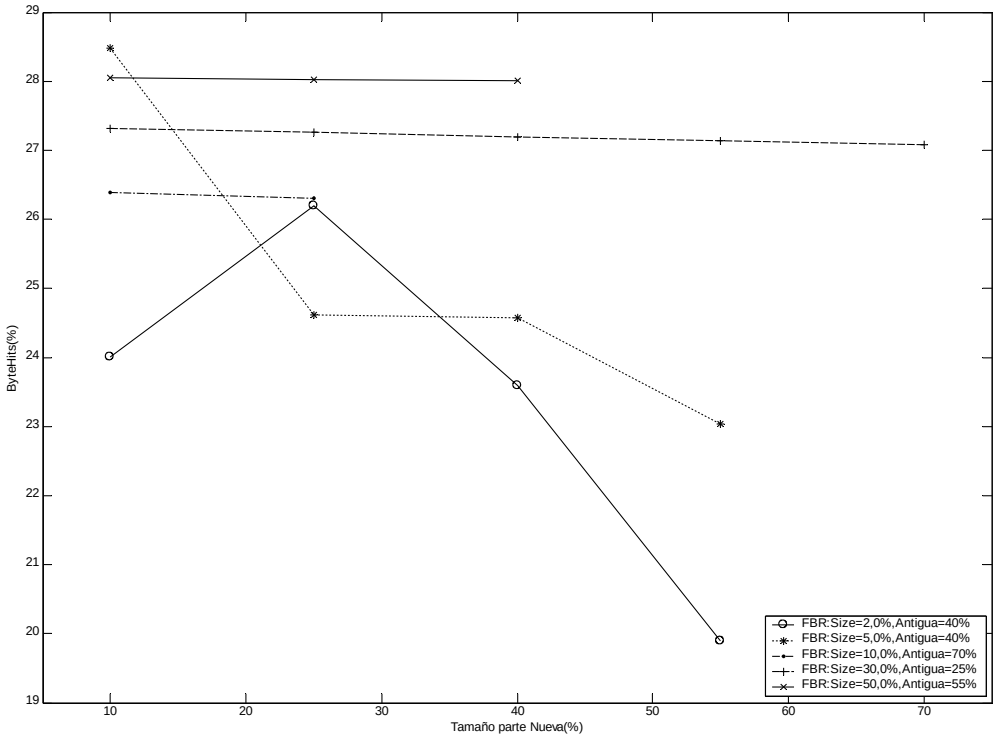

Figura 39: FBR: Tamaño Parte Nueva vs Byte Hit Rate

5.3.4. Tamaño parte A*ntigua*

Se han realizado simulaciones con tamaños de la parte *Antigua* de la caché iguales al 10, 25, 40, 55 y 70 por ciento del tamaño total de la caché.

En la Figura 40 se comparan los resultados obtenidos según el tamaño de la parte *Antigua*, fijando el tamaño de la parte *Nueva* al 10% del total. Se puede observar que a partir del 25%, el tamaño de la parte *Antigua* no tiene mucha influencia en el rendimiento de la caché, siendo la diferencia de eficiencia en algunas ocasiones mínima o incluso inexistente. Dichos datos pueden comprobarse igualmente en la Tabla 11.

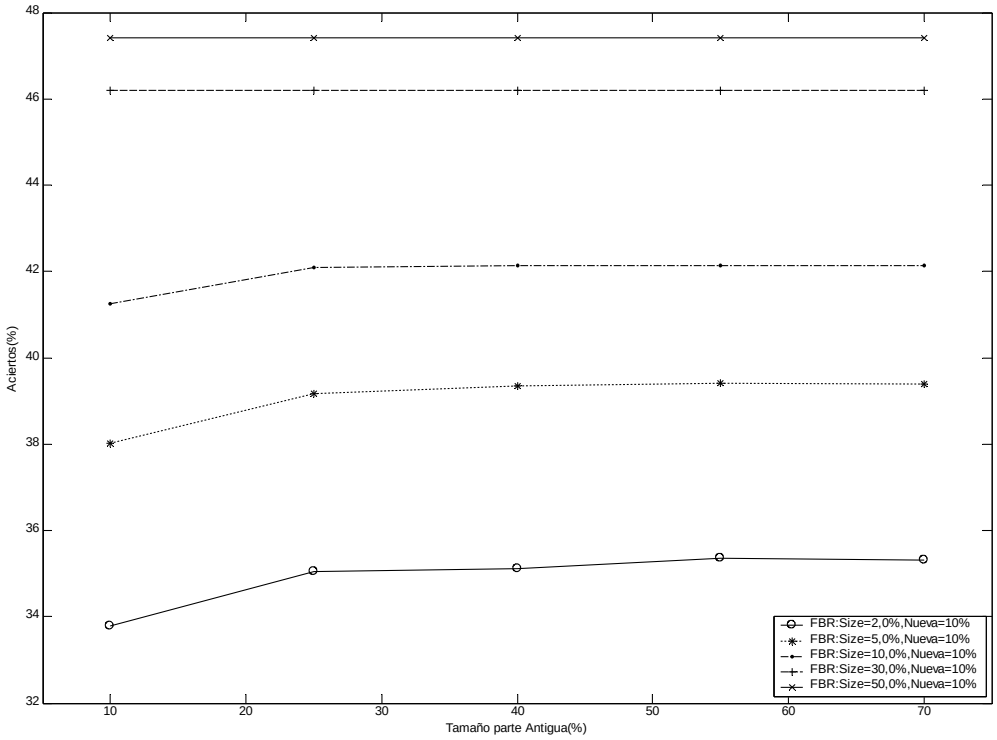

Figura 40: FBR: Tamaño Parte Antigua vs Hit Rate

Parte Antigua / **Tamaño caché**	**10 %**	**25 %**	**40 %**	**55 %**	**70%**
2 %	33.78	35.04	35.11	**35.37**	35.32
5 %	38.03	39.16	39.35	**39.41**	39.40
10 %	41.24	42.09	**42.14**	**42.14**	**42.14**
30 %	**46.20**	**46.20**	**46.20**	**46.20**	**46.20**
50 %	**47.42**	**47.42**	**47.42**	**47.42**	**47.42**

Tabla 11: FBR (Parte Antigua) Hit Rate

A continuación se muestra la Figura 41, que compara los aciertos en bytes obtenidos en las simulaciones.

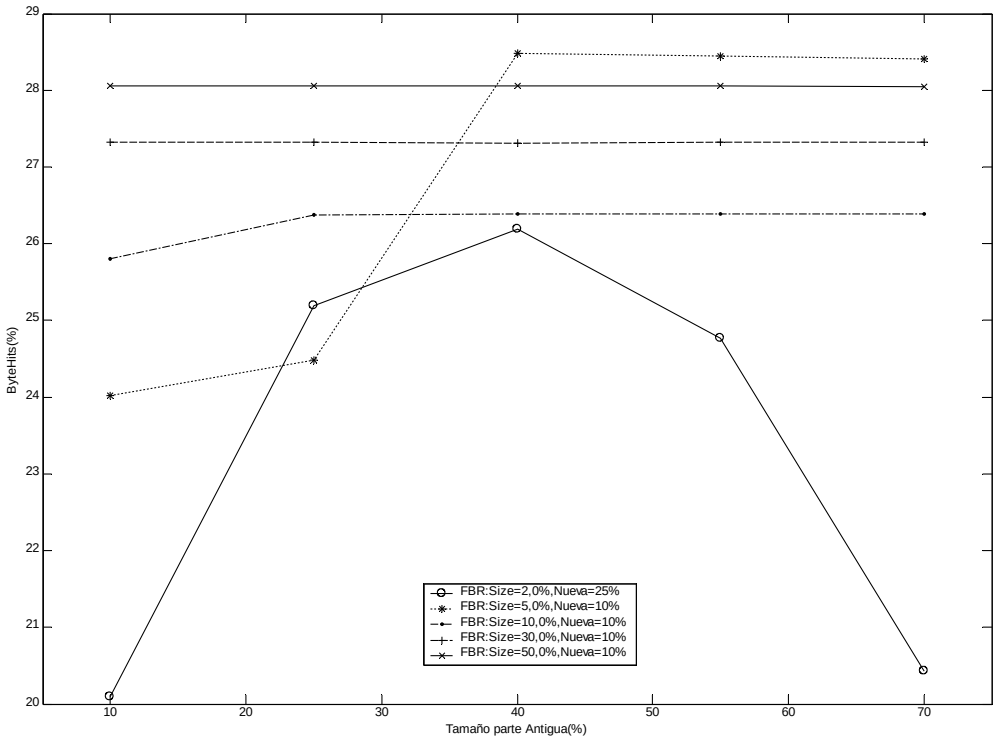

Figura 41: FBR: Tamaño Parte Antigua vs Byte Hit Rate

Para tamaños de caché pequeños (2 y 5%) se producen grandes variaciones en los resultados, consiguiéndose en ambos casos el máximo cuando el tamaño de la parte *Antigua* es el 40% del total. Para tamaños de caché mayores, no se producen apenas variaciones en los resultados obtenidos, como puede verse también en la Tabla 12.

Parte Antigua / **Tamaño caché**	**10 %**	**25 %**	**40 %**	**55 %**	**70%**
2 %	20.10	25.19	**26.19**	24.77	20.42
5 %	24.02	24.48	**28.48**	28.44	28.41
10 %	25.80	26.37	**26.39**	**26.39**	**26.39**
30 %	**27.32**	**27.32**	**27.31**	**27.32**	**27.32**
50 %	**28.05**	**28.05**	**28.05**	**28.05**	**28.05**

Tabla 12: FBR (Parte Antigua) Byte Hit Rate

5.3.5. Conclusiones

Según las simulaciones realizadas, la configuración más eficiente de FBR se consigue estableciendo C_{max} = 10 y A_{max}=100, un tamaño de la parte *Nueva* del 10% salvo una excepción, y un tamaño de la parte *Antigua* del 55% para obtener el máximo de aciertos o del 40% para conseguir el máximo de bytes acertados. Cabe destacar que para los tamaños de caché más grandes la variación del tamaño de la parte *Antigua* no hacen variar el rendimiento de la caché.

5.4. LRFU

5.4.1. Lambda

Para las simulaciones se han tomado como valores de λ los siguientes: 0, 0.0001, 0.001, 0.01, 0.03125, 0.0625, 0.125, 0.25, 0.5, 0.75 y 1.

Como se verá a continuación en la Tabla 13 y en la Figura 42, la influencia de λ en el número de aciertos obtenidos es inexistente en todas las simulaciones realizadas excepto en las que la caché tiene un tamaño del 2% y del 5%. En estos dos últimos casos, sí se observa una fluctuación de los resultados obtenidos, llegándose al máximo de aciertos cuando λ es 0.

λ / T. caché	**0**	**0.0001**	**0.001**	**0.01**	**0.03125**	**0.0625**	**0.125**	**0.25**	**0.5**	**0.75**	**1**
2 %	**34.21**	30.62	30.39	30.92	31.50	31.53	32.15	32.49	32.92	33.25	33.11
5 %	**38.78**	37.42	37.59	38.52	38.63	38.69	38.72	38.72	38.74	38.74	38.74
10 %	**42.38**	**42.38**	**42.38**	**42.38**	**42.38**	**42.38**	**42.38**	**42.38**	**42.38**	**42.38**	**42.38**
30 %	**46.37**	**46.37**	**46.37**	**46.37**	**46.37**	**46.37**	**46.37**	**46.37**	**46.37**	**46.37**	**46.37**
50 %	**47.52**	**47.52**	**47.52**	**47.52**	**47.52**	**47.52**	**47.52**	**47.52**	**47.52**	**47.52**	**47.52**

Tabla 13: LRFU (Lambda) Hit Rate

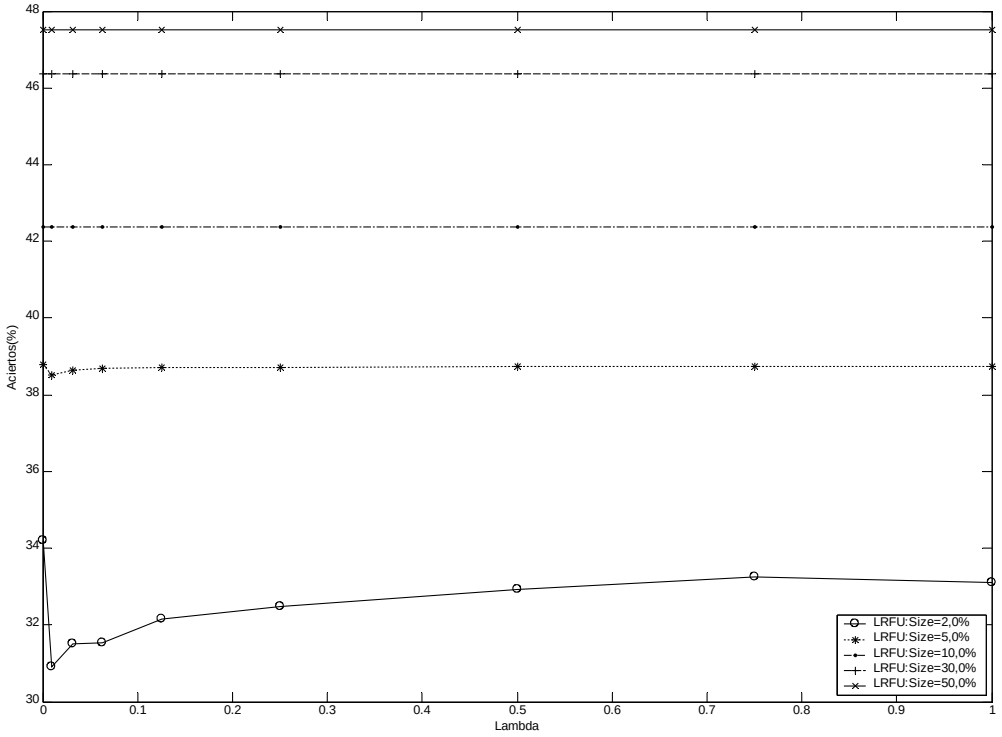

Figura 42: LRFU: Lambda vs Hit Rate

En cuanto al número de bytes que se han acertado los resultados obtenidos son similares a los del número de aciertos, ya que, como puede verse en la Tabla 14 y en la Figura 43, el valor de λ sólo produce variaciones en los resultados obtenidos cuando la caché es de tamaño 2% ó 5%.

λ T. caché	**0**	**0.0001**	**0.001**	**0.01**	**0.03125**	**0.0625**	**0.125**	**0.25**	**0.5**	**0.75**	**1**
2 %	**22.87**	18.33	18.91	18.79	18.81	18.94	18.99	19.11	21.47	21.49	22.59
5 %	**24.62**	23.05	24.05	24.47	24.58	24.60	24.61	24.61	24.62	24.61	24.62
10 %	**25.88**	**25.88**	**25.88**	**25.88**	**25.88**	**25.88**	**25.88**	**25.88**	**25.88**	**25.88**	**25.88**
30 %	**27.32**	**27.32**	**27.32**	**27.32**	**27.32**	**27.32**	**27.32**	**27.32**	**27.32**	**27.32**	**27.32**
50 %	**28.08**	**28.08**	**28.08**	**28.08**	**28.08**	**28.08**	**28.08**	**28.08**	**28.08**	**28.08**	**28.08**

Tabla 14: LRFU (Lambda) Byte Hit Rate

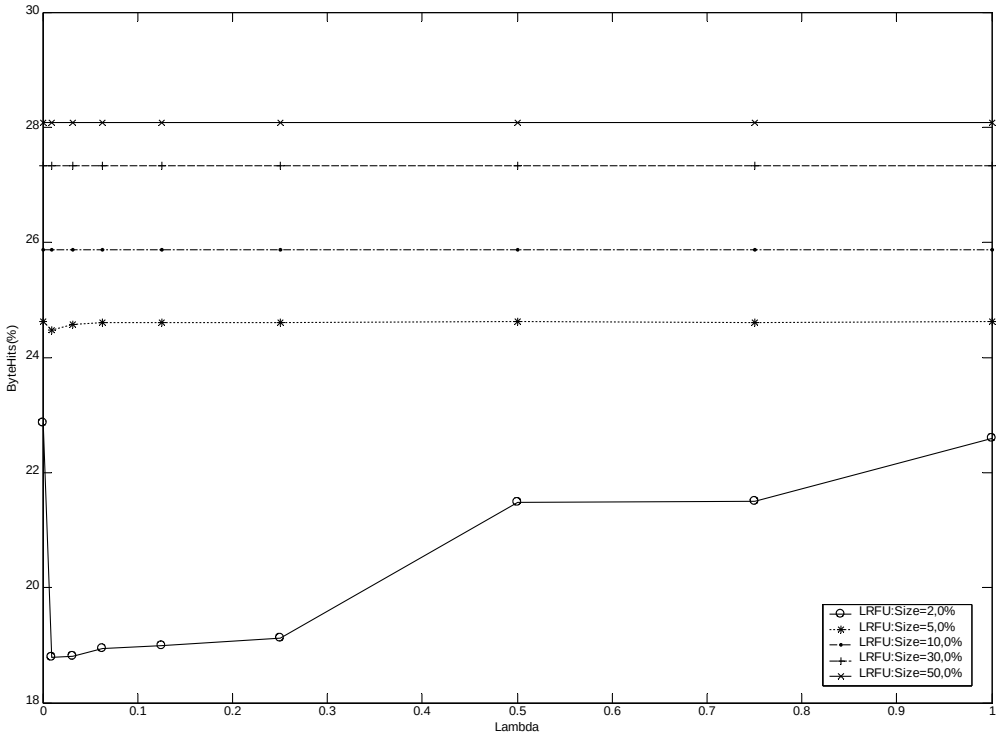

Figura 43: LRFU: Lambda vs Byte Hit Rate

5.4.2. Conclusiones

Se han mostrado los resultados en los que se analizaba el comportamiento de LRFU en función del valor del parámetro λ, y se concluye que el valor óptimo de dicho parámetro es 0 para tamaños de caché del 2 y 5 por ciento, mientras que para tamaños de caché mayores no influye en el rendimiento de la caché. Ésto hace que la política LRFU sea fácilmente optimizable, a diferencia de otras políticas en las que la búsqueda de los valores óptimos de cada parámetro no es tan sencilla.

5.5. ML-LRU

Las simulaciones se han realizado utilizando como función decisión la función aleatoria, asignando la misma probabilidad a un elemento de ir a una u a otra parte de la caché.

5.5.1. Tamaño parte *Primera*

Se han realizado simulaciones con tamaños de la parte *Primera* de la caché iguales al 20, 35, 50, 65 y 80 por ciento del tamaño total de la caché. En la Figura 44 se muestran los resultados de las simulaciones.

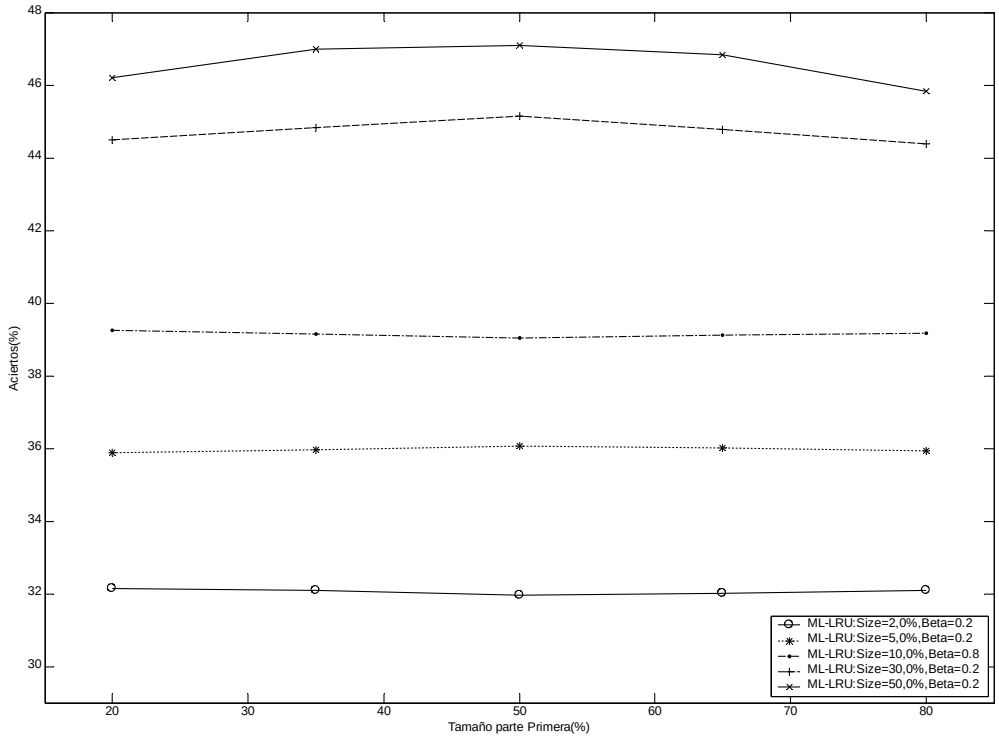

Figura 44: ML-LRU: Tamaño Parte Primera vs Hit Rate

El tamaño óptimo de la parte *Primera* de la caché varía entre el 20% (para tamaños de caché del 2% y 10%) y el 50% (para tamaños de caché del 5%, 30% y 50%). No se producen

grandes variaciones en el número de aciertos excepto en los tamaños de caché más grandes. Los resultados aparecen también en la Tabla 15.

Parte Primera / **Tamaño caché**	**20 %**	**35 %**	**50 %**	**65 %**	**80%**
2 %	**32.14**	32.09	31.97	32.01	32.11
5 %	35.89	35.96	**36.07**	36.02	35.92
10 %	**39.25**	39.15	39.05	39.11	39.17
30 %	44.49	44.84	**45.14**	44.78	44.37
50 %	46.20	47.00	**47.08**	46.83	45.83

Tabla 15: ML-LRU (Parte Primera) Hit Rate

A continuación se estudiarán los aciertos en bytes obtenidos en función del tamaño de la parte *Primera* de la caché, que aparecen en la Figura 45.

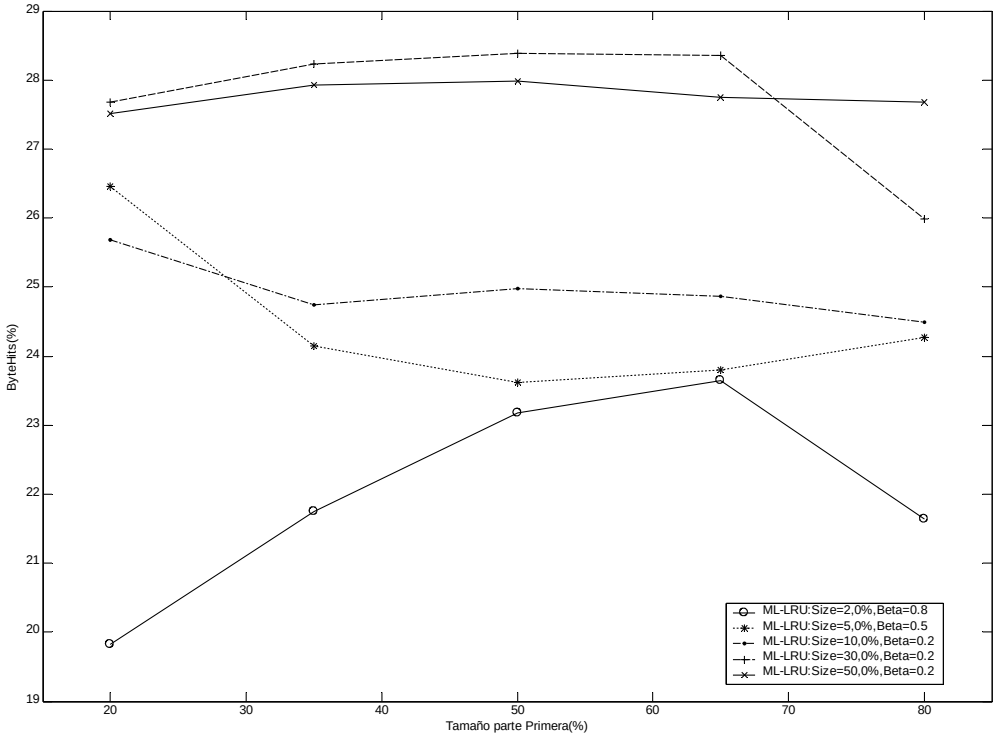

Figura 45: ML-LRU: Tamaño Parte Primera vs Byte Hit Rate

La influencia del tamaño de las dos partes de la caché en el rendimiento es clara, ya

que hace variar enormemente los aciertos en bytes obtenidos. Los mejores resultados se consiguen con tamaños de la parte *Primera* del 65% para una caché de tamaño 2%, 20% para cachés de 5% y 10%, y 50% para cachés de 30% y 50%. Los resultados se muestran también en la Tabla 16.

Parte Primera / **Tamaño caché**	**20 %**	**35 %**	**50 %**	**65 %**	**80%**
2 %	19.82	21.75	23.18	**23.64**	21.63
5 %	**26.45**	24.15	23.61	23.79	24.27
10 %	**25.68**	24.74	24.97	24.87	24.49
30 %	27.67	28.24	**28.38**	28.35	25.99
50 %	27.51	27.93	**27.98**	27.74	27.68

Tabla 16: ML-LRU (Parte Primera) Byte Hit Rate

5.5.2. Beta

Se han realizado simulaciones para $\beta = 0.2$, 0.5 y 0.8, los resultados son los que se muestran a continuación. Para cada tamaño de caché se muestra la evolución de los resultados en función del parámetro β. Los tamaños de la parte Primera escogidos son los que han obtenido mejores resultados.

En la Figura 46 se observa que el parámetro β no influye significativamente en los aciertos conseguidos, al menos con el uso de la función decisión aleatoria, que es el que se ha probado. Se consigue un mayor número de aciertos con $\beta = 0.2$ para todos los tamaños de caché excepto para el 10%, donde lo más conveniente es establecer $\beta = 0.8$.

En el número de aciertos en bytes sí se producen mayores variaciones en los resultados en función del parámetro β especificado, como puede verse en la Figura 47. Para un tamaño de caché del 2%, se consigue un mayor índice de aciertos en bytes con $\beta = 0.8$, y para un tamaño del 5% con $\beta = 0.5$, mientras que para tamaños mayores los resultados conseguidos son mejores con $\beta = 0.2$.

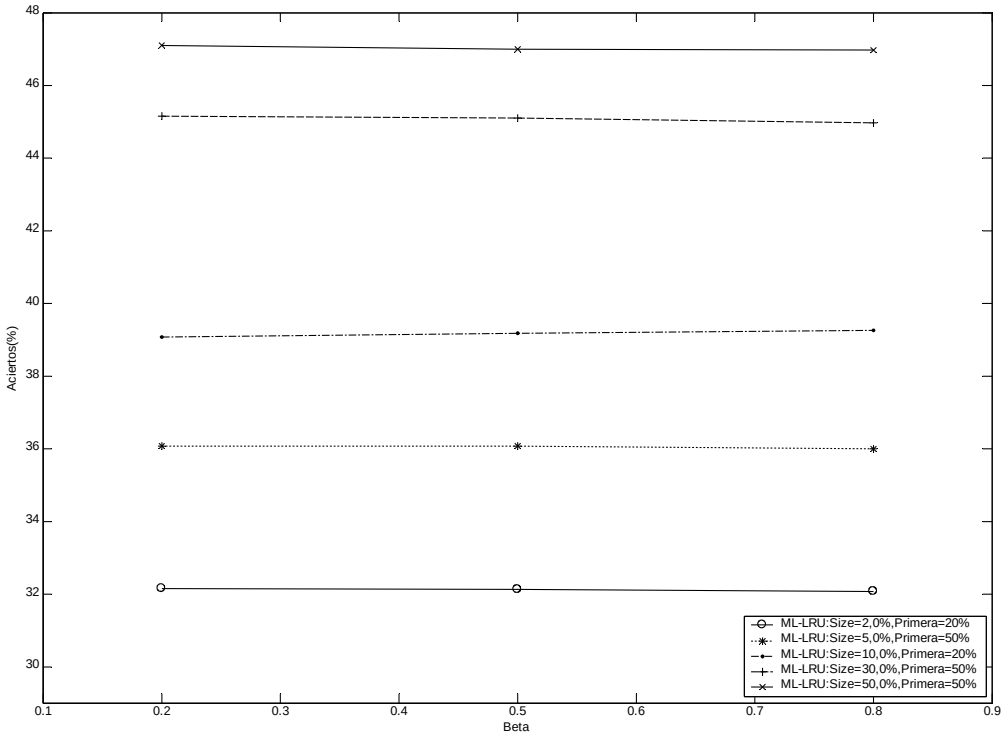

Figura 46: ML-LRU: Beta vs Hit Rate

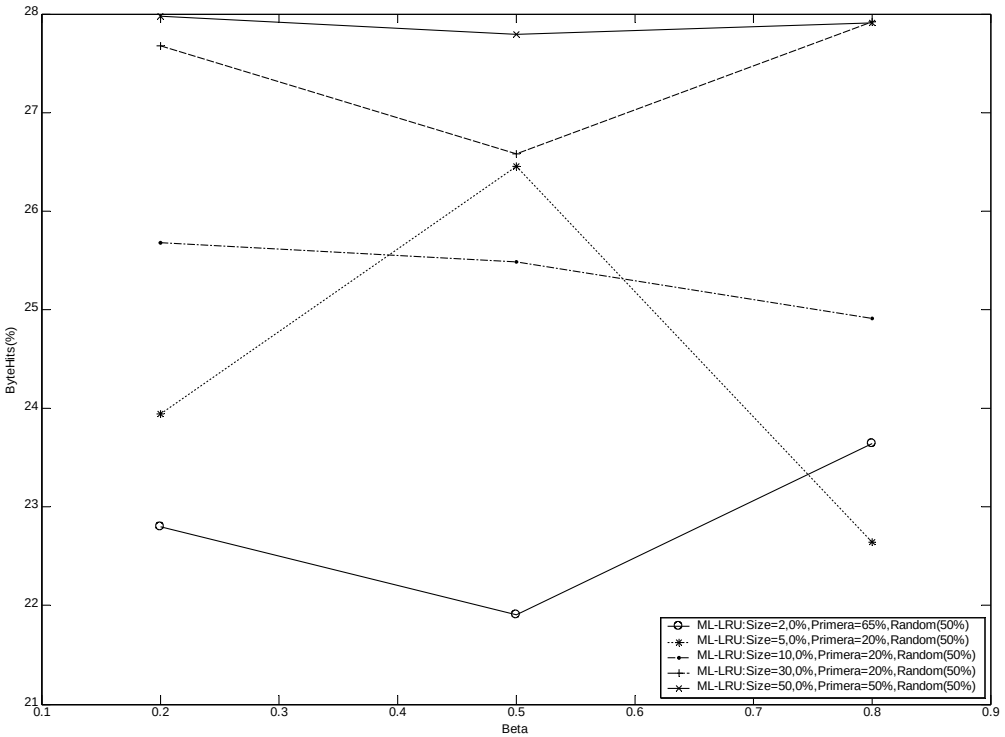

Figura 47: ML-LRU: Beta vs Byte Hit Rate

A continuación se muestra la Tabla 17, que recoge los resultados tanto para los aciertos (HR) como para los aciertos en bytes (BHR) en función del parámetro β.

β	**0.2**	**0.5**	**0.8**
Tamaño caché	HR / BHR	HR / BHR	HR / BHR
2 %	**32.14** / 22.79	32.11 / 21.90	32.06 / **23.64**
5 %	**36.07** / 23.94	36.07 / **26.45**	35.97 / 22.64
10 %	39.06 / **25.67**	39.16 / 25.48	**39.24** / 24.90
30 %	**45.14** / **28.37**	45.10 / 26.82	44.95 / 26.84
50 %	**47.08** / **27.97**	46.98 / 27.79	46.96 / 27.90

Tabla 17: ML-LRU (Beta) Resultados obtenidos

5.5.3. Conclusiones

Se ha analizado el rendimiento de la política de reemplazo multinivel ML-LRU, y se ha mostrado que el tamaño óptimo de la parte *Primera* varía sin un patrón claro de comportamiento entre el 20 y el 65 por ciento en función del tamaño de la caché, lo cual es debido al uso de la función aleatoria como función decisión. En cuanto al parámetro β, el valor óptimo es 0.2 salvo algunas excepciones. El rendimiento de la política es bueno, pero para sacar más conclusiones habría que aplicar una función decisión distinta.

5.6. C-LRU

Se han realizado simulaciones con las configuraciones de particiones que se muestran en la Tabla 18, que son las mismas que aparecen en [9] y [10]. Dicha tabla muestra para cada partición los tamaños de peticiones en bytes que almacena y el tamaño de la partición en relación al tamaño total de la caché. Posteriormente se compararán los resultados obtenidos en las simulaciones utilizando dichas configuraciones.

Número	Partición 1	Partición 2	Partición 3	Partición 4
1	0-2048 bytes 1/3	2049-6144 bytes 1/3	6145 bytes-Máx 1/3	-
2	0-2048 bytes 1/6	2049-6144 bytes 1/3	6145 bytes-Máx 1/2	-
3	0-2048 bytes 10%	2049-6144 bytes 20%	6145 bytes-Máx 70%	-
4	0-6468 bytes 48,6%	6469-55484 bytes 30,2%	55485 -375755 bytes 18%	375756 bytes-Máx 3,2%
5	0-6468 bytes 16%	6469-55484 bytes 38,2%	55485-375755 bytes 16,4%	375756 bytes-Máx 29,4%
6	0-6468 bytes 65%	6469-55484 bytes 32,1%	55485-375755 bytes 2,7%	375756 bytes-Máx 0,2%

Tabla 18: Particiones C-LRU

Los aciertos obtenidos son mayores para tamaños de caché del 2 al 10 por ciento con la configuración número 1, la cual divide la caché en tres partes cada una de ellas del mismo tamaño, tal y como muestran la Tabla 19 y la Figura 48. Para tamaños de caché del 30% y 50% se consiguen los mayores aciertos con las configuraciones 3 y 4 respectivamente. Éstas dos configuraciones y la número 6 consiguen prácticamente la misma cantidad de aciertos a pesar de que tienen un planteamiento bastante distinto entre sí, lo cual muestra que las formas de conseguir los mejores resultados utilizando esta política de reemplazo son múltiples.

Configuración Tamaño caché	1	2	3	4	5	6
2 %	**38.86**	36.85	35.08	36.42	32.43	37.49
5 %	**43.43**	41.36	39.21	40.67	36.36	42.16
10 %	**45.85**	45.03	43.10	44.67	39.43	45.56
30 %	47.18	47.43	47.51	**47.74**	45.59	47.73
50 %	47.51	47.93	**48.09**	48.08	47.38	48.04

Tabla 19: C-LRU Hit Rate

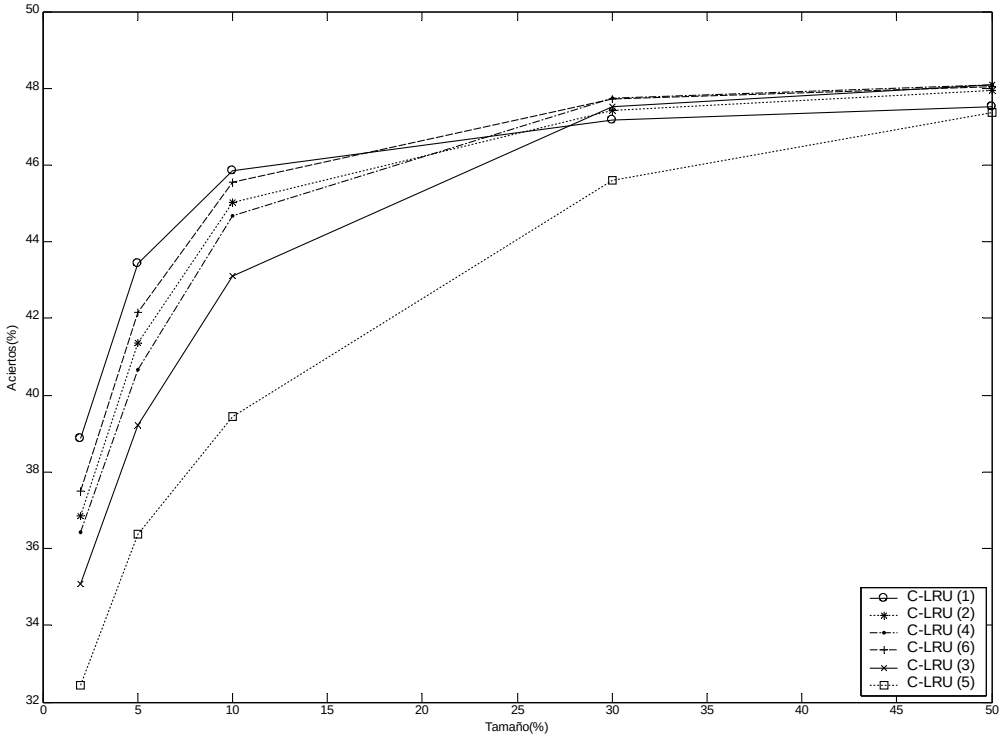

Figura 48: Particiones C-LRU vs Hit Rate

En cuanto al número de bytes acertados la situación, como era de esperar, varía mucho, ya que aquí destacan las configuraciones que están claramente orientadas a conseguir un mayor índice de aciertos en bytes.

Las configuraciones que mejores resultados obtienen son la 5 y la 3, por este orden, ya que la 5 supera a la 3 en todos los tamaños de caché probados excepto en el 5%. La configuración 3 ya consiguió muy buenos resultados en lo que al índice de aciertos se refiere excepto en tamaños de caché pequeños, con lo que podemos decir que es la configuración más idónea para esta política, ya que consigue muy buenos resultados en ambos apartados, algo que no ocurre con la configuración 5, ya que su rendimiento medido en número de aciertos fue de los peores entre todas las distintas configuraciones probadas. A continuación, la Figura 49 y la Tabla 20 muestran dichos resultados.

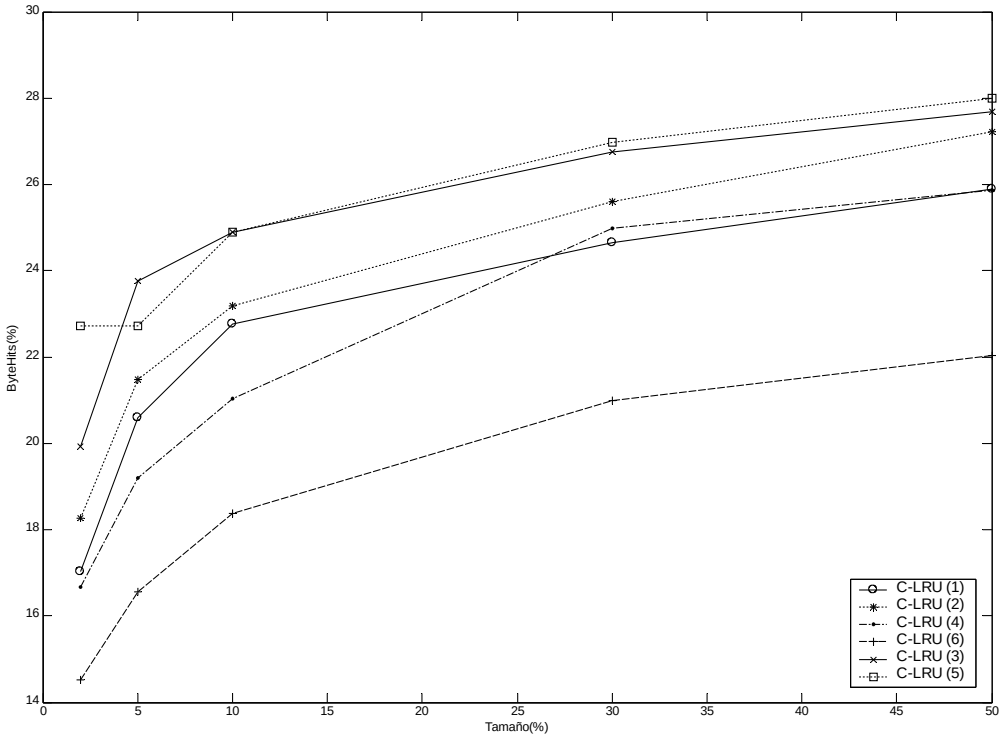

Figura 49: Particiones C-LRU vs Byte Hit Rate

Configuración / Tamaño caché	1	2	3	4	5	6
2 %	17.03	18.26	19.93	16.67	**22.72**	14.52
5 %	20.59	21.49	**23.75**	19.20	22.72	16.55
10 %	22.76	23.18	**24.89**	21.04	**24.89**	18.39
30 %	24.65	25.60	26.76	24.98	**26.98**	20.98
50 %	25.90	27.23	27.69	25.86	**28.00**	22.03

Tabla 20: C-LRU Byte Hit Rate

5.6.1. Conclusiones

Se ha mostrado cómo con un planteamiento sencillo como es clasificar los objetos en caché en función de su tamaño se consiguen muy buenos resultados. Las configuraciones que consiguen un mayor índice de aciertos son la número 1 para tamaños de caché de hasta el

10%, y las configuraciones 4 y 5 para tamaños mayores, mientras que las que obtienen mejores índices de aciertos en bytes son las número 5 y 3.

5.7. Comparación entre políticas

A continuación se muestran tanto de forma gráfica como numérica los resultados obtenidos en las simulaciones realizadas utilizando cada una de las políticas de reemplazo estudiadas.

Para cada resultado a comparar, es decir, aciertos y aciertos en bytes, se han escogido de cada política de reemplazo las configuraciones que han obtenido los resultados mejores para cada tamaño de caché.

5.7.1. Aciertos

La Figura 50 muestra claramente que la política C-LRU es la que obtiene los mejores resultados para todos los tamaños de caché que se han probado, llegando a obtener un índice de aciertos de hasta 4 puntos porcentuales mayor a la siguiente política que obtiene mejores resultados, disminuyendo esta diferencia cuando aumenta el tamaño de la caché.

En cuanto a las demás políticas puede verse que SLRU, FBR y LRFU consiguen resultados bastante similares, siendo mejores para cachés de tamaños 2 y 5 por ciento los obtenidos usando FBR y a partir del 10% son mejores los que se consiguen tanto con SLRU como con LRFU, que obtienen idénticos resultados para cachés de tamaños 30 y 50 por ciento.

2Q es la política más adecuada cuando la caché es menor, es decir, cuando es de tamaño 2%, quedando sólo por detrás de C-LRU y por delante de todas las demás. Para cachés mayores, 2Q deja de ser eficiente y llega a estar por debajo incluso de LRU cuando la caché es grande.

ML-LRU obtiene resultados similares a LRU, algo lógico debido a que en las simulaciones usamos la función aleatoria, mientras que LRU consigue índices de aciertos muy inferiores a los de las políticas de reemplazo multinivel estudiadas, siendo la diferencia de hasta 7 puntos en los tamaños de caché más pequeños, aunque después disminuye en tamaños mayores.

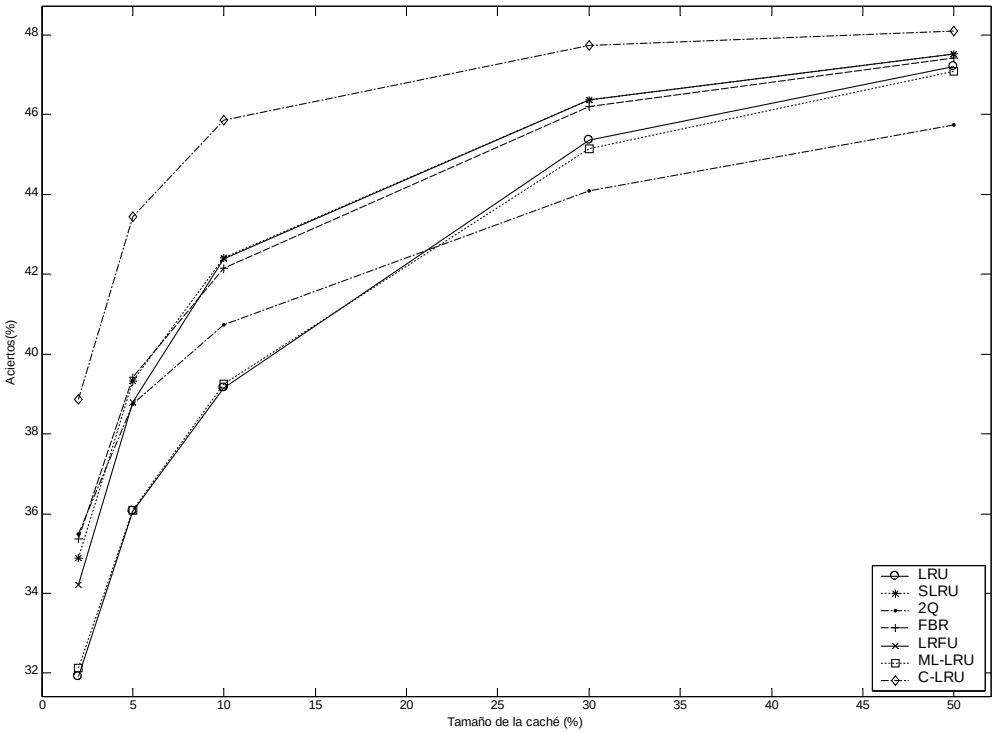

Figura 50: Todas las políticas: Tamaño Caché vs Hit Rate

Por tanto, se ven claramente los beneficios de las políticas de reemplazo multinivel sobre la política más extendida en la actualidad, LRU, especialmente cuando la caché es de menor tamaño, que es cuando realmente se comprueba la capacidad de una política de reemplazo para conseguir un mayor rendimiento.

La Tabla 21 muestra los aciertos obtenidos en las simulaciones para todas las políticas de reemplazo estudiadas.

Tamaño caché / Política	2%	5%	10%	30%	50%
LRU	31.91	36.06	39.15	45.35	47.21
SLRU	34.89	39.32	42.40	46.37	47.52
2Q	35.48	38.74	40.74	44.09	45.74
FBR	35.37	39.41	42.14	46.20	47.42
LRFU	34.21	38.78	42.38	46.37	47.52
ML-LRU	32.14	36.07	39.25	45.14	47.08
C-LRU	**38.86**	**43.43**	**45.85**	**47.74**	**48.09**

Tabla 21: Todas las Políticas: Hit Rate

5.7.2. Aciertos en bytes

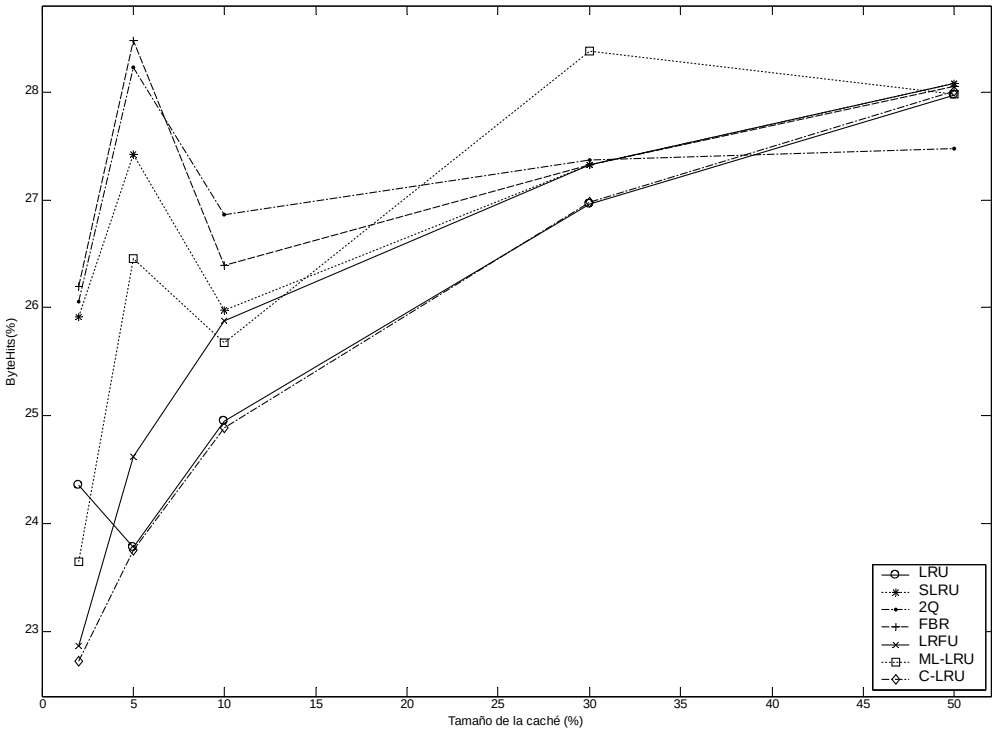

Figura 51: Todas las políticas: Tamaño Caché vs Byte Hit Rate

Lo primero se observa en la Figura 51 de los resultados es la gran subida que tiene el

índice de aciertos en bytes obtenidos cuando la caché es del 5%, volviendo a bajar posteriormente. Esto puede ser debido a la existencia de un documento de gran tamaño que no cabe en las cachés del 2% ni del 5%, pero que sí cabe en la caché de tamaño 10%, lo cual puede provocar que al introducir dicho documento en la caché haya que descartar un gran número de documentos de la caché, lo que hace disminuir el índice de aciertos en bytes.

La política con la que se consiguen los mejores resultados cuando la caché es de tamaño 2 y 5 por ciento es FBR, y tras ella se sitúan 2Q y SLRU, en ese orden, con una diferencia en el índice de aciertos en bytes de algo más de una décima de punto entre las dos primeras.

Para un tamaño del 10% se invierte el orden, y es 2Q la que obtiene mejores resultados, seguida de FBR y SLRU, siendo en este caso mayor la diferencia entre las mismas.

Con un tamaño de caché del 30% es con ML-LRU como se consiguen mejores resultados, aunque esto es algo ficticio debido a la aleatoriedad usada. De las demás políticas la mejor es de nuevo 2Q, que consigue apenas media décima más en el índice de bytes acertados que SLRU, FBR y LRFU, que obtienen resultados prácticamente idénticos.

Para la caché del 50% los mejores resultados se obtienen utilizando LRFU y SLRU, ya que ambas igualan en el índice de aciertos en bytes obtenido, y consiguen una pequeña mejoría sobre FBR y algo mayor sobre C-LRU, ML-LRU y 2Q.

Por tanto, se concluye que las políticas FBR, 2Q, SLRU y LRFU obtienen muy buenos resultados, exceptuando 2Q cuando la caché es del 50% y LRFU cuando es menor del 10%, consiguiendo siempre un mayor índice de bytes acertados que LRU, siendo la diferencia de 5 puntos porcentuales cuando la caché es del 5%, 3 puntos cuando es del 2 y del 10 por ciento y de algunas décimas cuando la caché es mayor. Tan sólo C-LRU consigue unos resultados algo peores que las demás políticas multinivel, siendo estos similares a los obtenidos usando LRU, lo cual contrasta con los excelentes resultados que se consiguieron en el número de aciertos.

A continuación aparece la Tabla 22, que muestra los aciertos en bytes obtenidos para las mejores configuraciones de cada una de las políticas de reemplazo que han sido objeto de

estudio.

Tamaño caché / Política	2%	5%	10%	30%	50%
LRU	24.35	23.78	24.95	26.96	27.97
SLRU	25.91	27.42	25.97	27.32	**28.08**
2Q	26.06	28.23	**26.86**	27.37	27.48
FBR	**26.19**	**28.48**	26.39	27.32	28.05
LRFU	22.87	24.62	25.88	27.32	**28.08**
ML-LRU	23.64	26.45	25.68	**28.38**	27.98
C-LRU	22.72	23.75	24.89	26.98	28.00

Tabla 22: Todas las Políticas: Byte Hit Rate

6. Conclusiones y líneas futuras

En el proyecto que se ha realizado se ha mostrado que existe una gran variedad de políticas de reemplazo multinivel que pueden ser usadas para administrar las cachés Web como alternativa a la política de reemplazo LRU (*Least Recently Used*), que es en la actualidad la solución más extendida para la administración de cachés Web. Dichas políticas de reemplazo multinivel dividen la caché en varios niveles, cada uno con una prioridad distinta, y a diferencia de LRU consideran no sólo el tiempo que hace que un documento ha sido referenciado, sino también parámetros como la frecuencia con la que son referenciados los documentos o el tamaño de los mismos. Las políticas multinivel estudiadas son SLRU (*Segmented* LRU), 2Q (*Two Queue*), FBR (*Frequency-Based Replacement*), LRFU (*Least Recently/Frequently Used*), ML-LRU (*Multilevel* LRU) y C-LRU (*Class-Based* LRU).

Se ha creado una aplicación multiplataforma y sencilla de utilizar con la que poder evaluar fácilmente el rendimiento de cachés Web en función de la política de reemplazo y los parámetros escogidos. Esta aplicación realiza simulaciones de cachés Web a partir de unos ficheros traza que contienen los accesos de los usuarios a un servidor proxy durante un cierto periodo de tiempo. Pueden realizarse simulaciones variando el tamaño de la caché y los parámetros de cada una de las políticas de reemplazo, y posteriormente los resultados obtenidos en las simulaciones pueden visualizarse, almacenarse y compararse entre sí.

A diferencia de otros estudios anteriores, en lugar de utilizar los parámetros que los autores de las políticas de reemplazo recomiendan en sus artículos, se ha realizado un análisis exhaustivo de cada una de las políticas para conseguir encontrar los valores óptimos de los parámetros de cada una de ellas y así poder realizar una comparación más precisa entre las mismas. Las métricas de rendimiento utilizadas han sido el índice de aciertos (*Hit Rate*) y el índice de aciertos en bytes (*Byte Hit Rate*).

Los resultados de las simulaciones realizadas para determinar los valores óptimos de

los parámetros de cada política de reemplazo muestran que el mayor rendimiento de la política SLRU se produce cuando el tamaño de la parte protegida, que es su único parámetro, se establece sobre el 65% salvo en algunas excepciones. Los valores óptimos de los parámetros de la política 2Q, salvo excepciones, varían en función del tamaño de la caché, siendo el valor óptimo de kIn mayor cuando también es mayor el tamaño de la caché, variando entre el 5 y el 80 por ciento, mientras que los valores óptimos de kOut son el 95 y el 5 por ciento según se busque obtener el máximo índice de aciertos o de bytes acertados, respectivamente. Para FBR, los valores óptimos de los parámetros son C_{max} = 10, A_{max}=100, un tamaño de la parte *Nueva* del 10% y un tamaño de la parte *Antigua* del 55% (para obtener el máximo de aciertos) o del 40% (para máximo de aciertos en bytes). En cuanto a LRFU, el parámetro λ no hace variar los resultados cuando las cachés son grandes, mientras que para cachés pequeñas la mayor eficiencia se consigue estableciendo λ = 0. Para ML-LRU, el valor óptimo del parámetro β es 0.2 salvo alguna excepción, mientras que el tamaño óptimo de la parte *Primera* se sitúa entre el 20 y el 65 por ciento, según sea el tamaño de la caché. Las configuraciones de C-LRU que consiguen un mayor índice de aciertos son la número 1 (ver Tabla 18) para cachés pequeñas, y las 4 y 5 en cachés grandes, mientras que las número 5 y 3 obtienen los mayores índices de aciertos en bytes.

En las simulaciones realizadas se ha comprobado que con las políticas de reemplazo estudiadas se puede conseguir un rendimiento de la caché mucho mayor que usando políticas tradicionales como LRU, y además se consigue también en muchos casos una menor carga computacional del sistema, ya que aunque sean algoritmos más complejos, el hecho de dividir la caché en varias partes hace que sea más rápido el proceso de búsqueda, inserción y borrado de los elementos de la misma.

Los resultados muestran que C-LRU es la política que obtiene los mayores índices de aciertos para todos los tamaños de caché que se han probado, consiguiendo un rendimiento significativamente superior a los obtenidos con las demás políticas. Para el menor tamaño de caché probado, 2Q es la segunda política con mayor índice de aciertos, pero con tamaños mayores es poco eficiente. FBR, SLRU y LRFU obtienen resultados bastante similares entre sí, siendo mejores los conseguidos con FBR en cachés pequeñas, y los obtenidos tanto con SLRU como con LRFU en cachés grandes. El rendimiento de LRU queda muy por debajo del

de las políticas de reemplazo multinivel, excepto de ML-LRU, que obtiene resultados muy similares a LRU.

Los mayores índices de aciertos en bytes en las cachés de menor tamaño se consiguen con el uso de las políticas FBR y 2Q, que obtienen resultados claramente mejores al resto. Para tamaños de caché grandes los aciertos en bytes obtenidos varían menos entre las distintas políticas, siendo SLRU, LRFU, FBR y 2Q e incluso ML-LRU soluciones eficientes para dichas cachés. C-LRU y LRU obtienen resultados similares en lo que a aciertos en bytes se refiere, siempre muy inferiores a los obtenidos por el resto de políticas de reemplazo multinivel.

Por tanto, se ha mostrado que para mejorar el rendimiento de las cachés Web, las políticas de reemplazo pueden tener cada una distintos planteamientos o estrategias, como por ejemplo tener en cuenta la frecuencia, el tamaño o lo recientemente que han sido referenciados los documentos, siendo todas ellas en conjunto o por separado estrategias válidas, aunque lógicamente unas consigan mejores resultados que otras.

En el futuro podrían realizarse nuevos estudios para evaluar las políticas de reemplazo existentes utilizando distintas muestras procedentes de otros servidores proxy para así corroborar la validez de las políticas aquí mostradas no sólo en el entorno probado, sino también en otros distintos. Asimismo, podría ampliarse el número de políticas de reemplazo a evaluar e incluso compararlas con políticas de reemplazo que no sean multinivel.

Por otra parte, se podría llegar a diseñar y crear una aplicación para administrar cachés Web en el mundo real, es decir, para instalarlas en servidores proxy y así proporcionarles la capacidad de utilizar cachés Web basadas en el uso de políticas de reemplazo multinivel para mejorar el rendimiento.

Bibliografía

[1] Ari Luotonen, Kevin Altis. *World Wide Web Proxies*. Computer Networks and ISDN Systems. 1994.

[2] Ramakrishna Karedla, J. Spencer Love, Bradley G. Wherry. *Caching Strategies to Improve Disk System Performance*. IEEE. 1994.

[3] Theodore Jonson, Dennis Shasha. *2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm*. Proceedings of the Twentieth International Conference on Very Large Databases, Santiago, Chile. 1994.

[4] John T. Robinson, Murthy V. Devarakonda. *Data Cache Management Using Frequency-Based Replacement*. Proceedings of the 1990 ACM SIGMETRICS conference on Measurement and modeling of computer systems. 1990.

[5] Donghee Lee, Jongmoo Choi, Jong-Hun Kim, Sam H. Noh, Sang Lyul Min, Yookun Cho, Chong Sang Kim. *On the Existence of a Spectrum of Policies that Subsumes the Least Recently Used (LRU) and Least Frequently Used (LFU) Policies*. Measurement and Modeling of Computer Systems. Proceedings of the 1999 ACM SIGMETRICS Conference, Atlanta, Georgia. 1999.

[6] Michael Feldman, John Chuang. *Service Differentiation in Web Caching and Content Distribution*.

[7] Christoph Lindemann, Oliver P. Waldhorst. *Evaluating the Impact of Different Document Types on the Performance of Web Cache Replacement Schemes*. University of Dortmund.

[8] IRCache project. http://www.ircache.net.

[9] *A Class-Based Least-Recently Used Caching Algorithm for WWW Proxies*. Submission for ACM Sigmetrics. 2002.

[10]Cristina Duarte Murta, Virgilio A. F. Almeida, Wagner Meira Jr. *Analyzing Performance of Partitioned Caches for the WWW*. Federal University of Minas Gerais, Brazil.