# Irrevocabilidad Relajada para Memoria Transaccional Hardware

Ricardo Quislant, Eladio Gutiérrez, Emilio L. Zapata y Óscar Plata[1]

*Resumen*— **Los sistemas comerciales que ofrecen memoria transaccional (TM) implementan un sistema hardware best-effort (BE-HTM) con limitaciones. Es necesario programar un fallback software basado en cerrojos para asegurar el progreso de la aplicación, lo que añade complejidad al paradigma.**

**En este artículo se propone un nuevo tipo de irrevocabilidad hardware (un modo transaccional que marca las transacciones como no abortables) para hacer frente a las limitaciones de los sistemas BE-HTM de una manera más eficiente, y para liberar al usuario de tener que programar un fallback. Se basa en el concepto de suscripción relajada utilizada en el contexto de la programación de fallbacks basada en cerrojos, donde la transacción se suscribe al cerrojo al final de la misma en lugar de al principio. El mecanismo de irrevocabilidad relajada hardware no involucra cambios en el protocolo de coherencia y se compara con su homólogo software, que proponemos como un fallback con suscripción relajada de espera escapada. También proponemos la irrevocabilidad relajada con anticipación, un mecanismo que no se puede implementar en software, y que mejora el rendimiento de las aplicaciones con múltiples reemplazos de bloques transaccionales de caché.**

**La evaluación de las propuestas se lleva a cabo con el simulador Simics/GEMS junto con la suite de benchmarks STAMP, y se obtiene una mejora de rendimiento sobre el fallback del 14% al 28% para algunos benchmarks.**

*Palabras clave*— **Memoria Transaccional Hardware; Best-effort; Irrevocabilidad Relajada; Fallback**

## I. Introducción

La memoria transaccional (TM) es un paradigma de programación cuyo propósito es facilitar la programación concurrente y maximizar el paralelismo en multiprocesadores de memoria compartida. Después de veinte años de su publicación por Herlihy y Moss [1], la TM hardware (HTM) se incluye en los procesadores Intel Haswell [2] e IBM BlueGene/Q [3] en 2013. IBM también lanza dos sistemas HTM diferentes en sus procesadores System z [4] y Power 8 [5]. Estos sistemas son una forma best-effort de HTM (BE-HTM) donde las transacciones que no encuentran impedimentos se ejecutan en hardware, pero las que encuentran alguna limitación, como fallos de capacidad, interrupciones o conflictos persistentes, tienen que ser ejecutadas por un fallback para asegurar el progreso de la aplicación.

El fallback de una transacción tiene que ser programado por el usuario en la mayoría de estos sistemas, lo que va en contra del propósito principal del paradigma TM, la simplicidad. Sin embargo, BlueGene/Q elimina la necesidad de fallback por medio de un runtime software que implementa irrevocabilidad [6]. Siempre que una transacción no pueda finalizar después de un número de reintentos se hace irrevocable para que no pueda ser abortada. En este modo, el sistema puede ser incapaz de mantener información transaccional de la transacción por lo que los demás hilos de ejecución deben parar.

En este artículo se propone la *irrevocabilidad relajada* como un modo de irrevocabilidad mejorado basado en el concepto de suscripción relajada [7] que se usa en la programación de fallbacks en sistemas BE-HTM [8]. En una implementación de fallback simple con un solo cerrojo global las transacciones se suscriben al cerrojo leyéndolo al principio. La suscripción relajada se hace al final de la transacción permitiendo más paralelismo. La ejecución es segura ya que la suscripción al cerrojo, el aislamiento de las transacciones y el aislamiento fuerte con respecto al código no transaccional [9] abortan las transacciones que hacen algún acceso indebido.

Las contribuciones de este artículo son las siguientes:

- Un mecanismo de irrevocabilidad relajada: Se permite paralelismo entre las transacciones normales y la irrevocable, pero las primeras han de parar antes de finalizar y esperar a que acabe la irrevocable. De esta manera se preserva el cómputo realizado a no ser que un conflicto las haga abortar. El modo irrevocable se arbitra por un protocolo basado en token independiente del protocolo de coherencia.
- Un fallback de suscripción relajada con espera escapada: Es el homólogo software del mecanismo anterior. Se trata del fallback propuesto en [8] con una espera escapada que retrasa la finalización de la transacción hasta que acabe la transacción que está en el fallback. El sistema transaccional ha de permitir instrucciones escapadas dentro de las transacciones [10].
- Un mecanismo de irrevocabilidad relajada con anticipación de reemplazo: Se realizan ligeras modificaciones al protocolo de coherencia para anticipar un reemplazo de un bloque transaccional de caché, de manera que se pide la irrevocabilidad antes de que se produzca. Este mecanismo no se puede implementar en software.

Con estas propuestas se pretende tener una solución BE-HTM que no requiera un esfuerzo extra por parte del usuario y que maximice el paralelismo. La evaluación con el sistema simulado Simics [11]/GEMS [12] y con la suite de benchmarks STAMP [13] muestra un rendimiento mejorado para ciertos benchmarks de hasta el 28%.

[1]Dpto. de Arquitectura de Computadores, Univ. Málaga, e-mail: {quislant, eladio, zapata, oplata}@uma.es.

## II. Trabajo Relacionado

Intel Haswell [2] e IBM System z [4] y Power 8 [5] incluyen sistemas BE-HTM que requieren un fallback software para garantizar el progreso de las aplicaciones transaccionales. Estos sistemas usan las cachés privadas para mantener los datos transaccionales y se basan en el protocolo de coherencia para detectar conflictos. En cuanto a las instrucciones de escape [10], sólo Power 8 permite una forma de instrucciones no transaccionales dentro de transacciones a través de su modo transaccional suspendido [14].

IBM Blue Gene/Q [3] usa una solución diferente que utiliza la caché L2 para almacenar los datos transaccionales, gracias a su caché asociativa por conjuntos con multiversión, donde un bloque puede estar a su vez en forma transaccional y no transaccional. Para asegurar el progreso de las transacciones utiliza un modo de irrevocabilidad implementado en un runtime software. Si una transacción supera un número de abortos dado entra en irrevocabilidad, y su identificador se inserta en una tabla hash para que sucesivas ejecuciones de la misma sólo permitan un aborto antes de entrar en irrevocabilidad. BG/Q aborta la transacción en un fallo de capacidad a diferencia de nuestro sistema con anticipación de reemplazo. Además, puede sufrir el problema de contención debido al cerrojo con el que implementa la irrevocabilidad.

En lo que se refiere a fallbacks software, Calciu et al. [8] proponen el fallback con suscripción relajada para sistemas como Haswell, que proporcionan aislamiento fuerte, y hardware sandboxing. Su fallback favorece el paralelismo entre las transacciones que no han tomado el fallback y aquella que lo ha tomado. Pero si a la hora de finalizar una transacción se comprueba en la suscripción al cerrojo que el fallback sigue en ejecución se aborta la transacción.

La irrevocabilidad en el contexto de HTM fue propuesta en un principio para asegurar el progreso en el sistema BE-HTM TCC [15]. Blundell et al. [16] la introducen para ejecutar las transacciones que exceden los recursos. Su sistema OneTM-Serialized es una implementación simplificada que no permite la ejecución de transacciones que no han excedido los recursos ni la ejecución de código transaccional en paralelo con la irrevocable. Ofrecen otra versión de su sistema llamada OneTM-Concurrent que permite más concurrencia, pero para su implementación necesitan incluir información transaccional en memoria principal para que se puedan detectar conflictos entre el código (no) transaccional y la transacción irrevocable. Nuestra propuesta de irrevocabilidad consigue una concurrencia similar sin necesidad de una modificación tan drástica.

## III. Arquitectura del Sistema

La arquitectura del sistema que hemos utilizado en este artículo se muestra en la Fig. 1. El sistema usa la caché L1 privada para almacenar los valores nuevos de las escrituras transaccionales, mientras que los

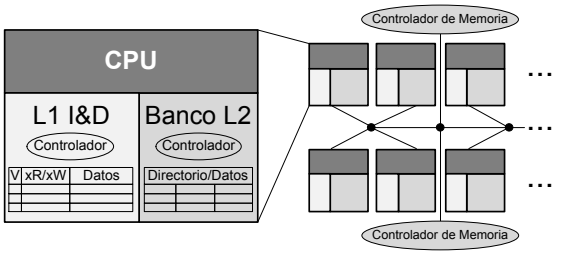


Fig. 1. Arquitectura del sistema base BE-HTM.

valores antiguos se mantienen en la L2. Se guardan un par de bits de lectura y escritura transaccionales (xR/xW) por bloque de caché L1. Estos bits se pueden borrar en un solo ciclo cuando se aborta o se finaliza una transacción. En caso de aborto, los bloques cuyo bit de escritura transaccional está a 1 también son invalidados. El protocolo de caché mantiene aislamiento fuerte [9] e implementa una política de detección de conflictos eager. La política de resolución de conflictos es requester-wins, es decir, el que pide el dato aborta al que lo tiene.

El protocolo de caché está basado en directorio y está modificado para soportar el sistema transaccional. Las modificaciones son:

- *Copiar en la primera escritura transaccional*: Un bloque modificado en L1 tiene que ser guardado en L2 antes de que una transacción escriba en él para que la L2 mantenga la copia antigua más reciente.
- *Abortar en reemplazos*: Un reemplazo de un bloque transaccional supone abortar la transacción, incluso si el reemplazo en L1 es debido a un reemplazo en L2 por la propiedad de inclusión.
- *Servir los datos antiguos desde la L2*: Si una transacción requiere un dato de una transacción que fue abortada debe de ser servido por la caché L2. Un paquete reenviado desde la L2 a una L1 será devuelto por esta a la L2 para que sirva el dato.

Por último, el sistema BE-HTM permite instrucciones de escape [10] y es implícitamente transaccional, lo que quiere decir que todas las instrucciones incluidas en una transacción son tomadas como transaccionales, excepto aquellas que estén escapadas.

## IV. Irrevocabilidad Relajada

El mecanismo de irrevocabilidad relajada implica cambios en los procedimientos de inicio y finalización de una transacción, e involucra un protocolo de comunicación de irrevocabilidad para indicar a todos los núcleos del multiprocesador el cambio a modo irrevocable por parte de una transacción. Sin embargo, no se modifica el protocolo de coherencia ni el conjunto de instrucciones.

Al iniciar una transacción primero hemos de comprobar si la transacción se tiene que ejecutar en modo

irrevocable. Usualmente, esto consiste en verificar que la transacción haya llegado a su límite de abortos. Si la transacción tiene que entrar en irrevocabilidad se pide a través del protocolo de comunicación (Véase la Sección IV-B). Otra transacción en modo irrevocable puede hacernos esperar. Una vez que se nos permite entrar en modo irrevocable la transacción se ejecuta con el sistema de aislamiento transaccional desactivado, es decir, como si fuera código no transaccional. De esta manera no se puede abortar. Cuando se llega al final de la transacción lo único que tenemos que hacer es comunicar el final de la irrevocabilidad.

En caso de que no tengamos que entrar en irrevocabilidad todavía, el inicio de la transacción se ejecuta con normalidad, activando el aislamiento transaccional, lo que llevará a que cada lectura y escritura marque su respectivo bit de bloque de la caché (xR/xW). Cuando la transacción llega a la fase de finalización se comprueba la existencia de una transacción irrevocable en el sistema. Si la hay, la transacción se para hasta que finalice la irrevocable. Nótese que una transacción puede ser abortada estando en el estado de espera.

### A. Correctitud

El mecanismo de irrevocabilidad relajada asegura una ejecución correcta debido a que se cumplen tres propiedades en el sistema BE-HTM:

- Aislamiento transaccional: esta propiedad asegura que cualquier escritura transaccional sólo será visible por la transacción que la ejecutó.
- Aislamiento fuerte: define la relación entre los accesos transaccionales y el código no transaccional. Con aislamiento fuerte, si una instrucción no transaccional escribe una posición de memoria que está en el conjunto de datos de una transacción, se abortará la transacción.
- Finalización postergada: el mecanismo de irrevocabilidad posterga la finalización de la transacción siempre que haya una transacción irrevocable en el sistema.

La Fig. 2 muestra los cuatro posibles casos de ejecución de una transacción normal e irrevocable. El primero muestra una transacción irrevocable en el primer hilo (`Th1`) que empieza después de una normal en el segundo (`Th2`). Pero la transacción de `Th2` finaliza antes que la irrevocable. En este caso se debería abortar la transacción de `Th2` ya que ha leído la posición Y (`LD Y`) escrita por la irrevocable (`ST Y`) y finaliza antes, lo que compromete la serializabilidad [17]. Sin embargo, a diferencia del fallback con suscripción relajada descrito en la Sección II, nuestro mecanismo de irrevocabilidad no aborta la transacción, si no que posterga su finalización hasta que acabe la irrevocable. En el peor caso, la transacción será abortada un poco más tarde debido a un conflicto con otra transacción o con la irrevocable por aislamiento fuerte. Con suerte, la transacción podrá finalizar mejorando así la concurrencia.

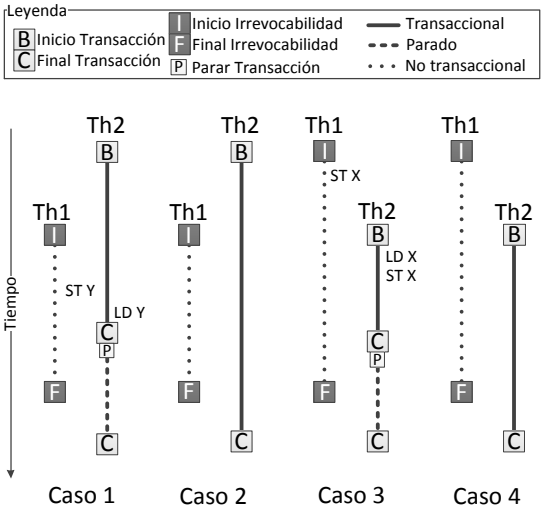


Fig. 2. Casos de ejecución para evaluar la correctitud de la irrevocabilidad relajada.

El Caso 2 muestra un escenario correcto donde la irrevocable acaba antes que la transacción normal. De hecho, el Caso 1 es una adaptación artificial al Caso 2 gracias a la finalización postergada. La misma adaptación se realiza con el Caso 3 y el 4 donde la irrevocable empieza antes que la normal. De nuevo, la serializabilidad podría romperse si `Th1` modifica `X` antes de que la transacción de `Th2` lo lea. Postergar la finalización soluciona el problema.

### B. Protocolo de Comunicación de Irrevocabilidad

Para la comunicación de la irrevocabilidad proponemos un protocolo basado en token donde sólo el núcleo que posee el token puede ejecutar una transacción en modo irrevocable. Cada núcleo tiene un bit, I, que indica si hay una transacción irrevocable corriendo en el sistema. Se necesita otro bit, T, para señalizar si se posee el token. Junto con este par de bits (I,T), cada núcleo posee un contador (C) que mantiene el número de reintentos de la transacción en curso. El núcleo pide la irrevocabilidad cuando C llega a cero.

Dependiendo del valor del par de bits (I,T) el controlador de L1 de cada núcleo actúa en consecuencia:

- *(I,T) = (0,0)*: No hay transacciones irrevocables y no tenemos el token. Si tenemos que pedir irrevocabilidad se difunde un mensaje de petición de token que será respondido por el núcleo que lo posee. Si ese núcleo acaba de iniciar una transacción irrevocable no mandará el token y quedaremos a la espera. Si recibimos el token ponemos el bit T a 1 y difundimos un mensaje de irrevocabilidad para que el resto de núcleos ponga su bit I a 1. Cuando estén a 1 podremos continuar en modo irrevocable, (1,1). En caso de encontrar (0,0) al finalizar una transacción es que no hay transacciones irrevocables y no tenemos que postergar la finalización.

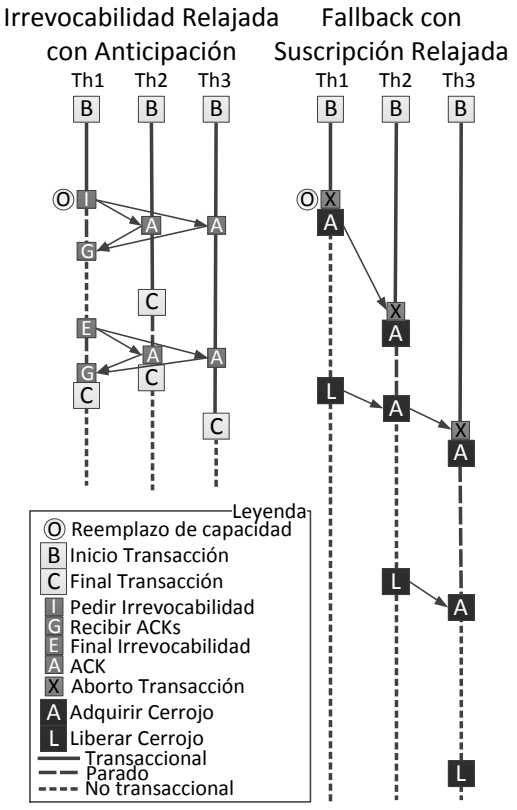


Fig. 3. Escenario de ejecución de irrevocabilidad relajada con anticipación contra fallback con suscripción relajada.

- *(I,T) = (0,1)*: El núcleo tiene el token por lo que puede informar de que va a entrar en irrevocabilidad directamente ya que nadie más está en modo irrevocable.
- *(I,T) = (1,0)*: Otro núcleo está en modo irrevocable. Si tenemos que pedir irrevocabilidad pararemos hasta que I sea 0. De la misma manera, si tenemos que finalizar una transacción también pararemos hasta recibir el mensaje de fin de la irrevocabilidad. En ese momento se finalizará la transacción.

El caso en el que varias transacciones piden el token al mismo tiempo se arbitra por medio de las colas de mensajes que poseen los routers de la red de interconexión y de la cola del controlador de memoria del núcleo que posee el token.

El uso de este protocolo de comunicación evita el efecto de la contención por cerrojo (véase la Sección VII-C). Además, no es necesario alojar el cerrojo y el contador en la L1, lo que podría causar abortos innecesarios.

## V. Irrevocabilidad Relajada con Anticipación de Reemplazo

La anticipación a un reemplazo de un bloque transaccional consiste en pedir irrevocabilidad antes de que se produzca el reemplazo para continuar la transacción en modo irrevocable sin tener que perder el cómputo hecho hasta el momento. Esta optimización no se puede realizar en fallbacks software.

La Fig. 3 muestra cómo se comportaría la irrevocabilidad relajada con anticipación de reemplazo en contraste con el fallback con suscripción relajada. El hilo 1, `Th1`, encuentra un reemplazo de caché que causaría un aborto. Pero en el caso de la irrevocabilidad se pide irrevocabilidad y se retiene el reemplazo. En cuanto al fallback, `Th1` aborta y reintenta la transacción tras adquirir el cerrojo del fallback. Las otras transacciones continúan debido a la suscripción relajada. Sin embargo, `Th2` finaliza antes que el fallback por lo que tiene que abortar. Después de abortar trata de adquirir el cerrojo pero tiene que esperar a que lo libere `Th1`. La transacción de `Th3` finaliza después de la de `Th1` y `Th2` pero tiene que abortar porque `Th2` adquirió el cerrojo un poco antes. Con la irrevocabilidad relajada se espera hasta recibir el mensaje de final de irrevocabilidad, que es cuando `Th2` finaliza su transacción. En este caso, la transacción de `Th3` que finalizaba después de las anteriores no tiene que esperar.

### A. Implementación

La implementación de la anticipación de reemplazo requiere ligeras modificaciones del protocolo de coherencia de caché puesto que hay que intervenir las acciones realizadas en los eventos de reemplazo. La Tabla I muestra dichas modificaciones sombreadas. El reemplazo de bloques no transaccionales, $\neg(xR \vee xW)$, se realiza sin modificación. Sin embargo, si el bloque es transaccional, $xR \vee xW$, se comprueba el contador de reintentos (C) para ver si la transacción ha llegado a su límite. Si no ha llegado, la transacción aborta y se decrementa el contador. El bloque cambia su estado a inválido, I. Nótese que un reemplazo de caché puede no ser un evento persistente. Por lo tanto, la transacción se reintenta hasta que llega al límite menos uno, $C \leq 1$, de manera que se anticipa el último reintento y se pide irrevocabilidad. El mensaje que causó el reemplazo es reciclado en la cola de la CPU hasta que entremos en irrevocabilidad, parando así al procesador. Un paso importante que tiene que realizar el protocolo de irrevocabilidad es limpiar los bits xR y xW como si se tratará de finalizar la transacción. De esta manera, el mensaje reciclado que causó el reemplazo transaccional causará ahora un reemplazo no transaccional.

Los reemplazos de bloques de L1 debidos a reemplazos en L2 y la propiedad de inclusión se muestran como `Reempla L2` en la Tabla I. Los reemplazos de L2 no transaccionales se dejan igual, se reconoce el reemplazo y se invalida el dato. Se manda el dato también en caso de que se haya modificado. Sin embargo, cuando el dato a reemplazar por la L2 es transaccional en la L1, $xR \vee xW$, abortamos la transacción y decrementamos el contador de reintentos siempre que no hayamos llegado al límite de abortos o haya otra transacción en modo irrevocable, (I,T)=(1,0). Esta última condición, que no está presente en los reemplazos de L1, se necesita para abortar la transacción, ya que puede haber un interbloqueo si es la irrevocable la que necesita reemplazar

TABLA I

MODIFICACIONES DEL PROTOCOLO DE COHERENCIA DE CACHÉ L1 PARA LA ANTICIPACIÓN DE REEMPLAZO.

| Estado | Eventos | | | | | |
|---|---|---|---|---|---|---|
| | Reempla L1 ¬(xR∨xW) | Reempla L1 (xR∨xW)∧(C>1) | Reempla L1 (xR∨xW)∧(C≤1) | Reempla L2 ¬(xR∨xW) | Reempla L2 (xR∨xW) (1,0)∨(C>1) | Reempla L2 (xR∨xW) (0,-)∧(C≤1) |
| I | – | – | – | ACK | – | – |
| S | – /I | Aborto, C-1 /I | Irre, Z | ACK /I | Aborto, C-1 /I | Irre, Zz |
| E | PUT(no datos) /I | Aborto, C-1 /I | Irre, Z | ACK /I | Aborto, C-1 /I | Irre, Zz |
| M | PUT+Datos /I | Aborto, C-1 /I | Irre, Z | ACK+Data /I | Aborto, C-1 /I | Irre, Zz |

Irre: pedir irrevocabilidad.
Z, Zz: reciclar las colas de CPU y de red, respectivamente.
(#,#): par de bits (I,T).

el bloque de L2 para continuar. De esta manera, un hilo sólo puede esperar a entrar en irrevocabilidad si su transacción llegó al límite de reintentos y ninguna transacción del sistema está en modo irrevocable, (0,-)∧(C≤1). En el caso de que una transacción acabe de pedir irrevocabilidad pero todavía no hemos sido informados, permaneceremos parados reciclando el mensaje de reemplazo y cuando los bits (I,T)=(0,-) cambien a (1,0) se lanzará el evento anterior y nuestra transacción será abortada.

## VI. Fallbacks Software

La Fig. 4 muestra el código de un fallback con suscripción relajada. Definimos dos primitivas para empezar una transacción: (i) `TAKE_XACT_CHECKPOINT` toma un checkpoint para guardar el punto al que volver tras un aborto, pero no empieza el aislamiento transaccional (línea 3); y (ii) `BEGIN_XACT` inicia el sistema transaccional de manera que cada lectura y escritura sea aislada adecuadamente (línea 8). Así es posible insertar código no transaccional entre las dos primitivas para comprobar si tenemos que ejecutar el fallback.

El procedimiento de inicio de una transacción (líneas 2–10) comienza tomando el punto de retorno e incrementando la variable local que mantiene el número de reintentos de la transacción (línea 4). Si la transacción llegó al límite de reintentos definido globalmente en `RETRY_LIMIT` se intenta adquirir el cerrojo global (línea 6). Si no, el hilo ejecuta de forma transaccional (línea 8). El procedimiento de finalización (líneas 11–17) comprueba la variable local que almacena el número de reintentos para saber si se adquirió el cerrojo. Si no, se realiza la suscripción al cerrojo (líneas 13 y 14) comprobando si el cerrojo está a cero. Se trata de la suscripción relajada descrita en la Sección II. Si está a uno abortamos explícitamente. Si venimos de ejecutar el fallback liberamos el cerrojo (línea 16).

En la Fig. 4 se puede ver nuestra propuesta de fallback con suscripción relajada con espera escapada que es el homólogo software de la irrevocabilidad relajada sin anticipación de reemplazo. Consiste en sustituir la cláusula condicional por un bucle en el que se permanecerá hasta que el cerrojo se libere. Pero esta espera debe estar escapada para no incluir el cerrojo en el conjunto de lectura de la transacción, ya que de otro modo la liberación del cerrojo abortaría la transacción debido al aislamiento fuerte.

```
1  localRetries = 0;
2  void beginTransaction(&localRetries) {
3    TAKE_XACT_CHECKPOINT; //Punto de retorno
4    localRetries++; //Incrementar reintentos
5    if (localRetries > RETRY_LIMIT) { //Fallback?
6      while(!lockAcquire(globalLock)) ; //Adquirir cerrojo
7    } else { // Ejecución transaccional
8      BEGIN_XACT;
9    }
10 }
11 void endTransaction(localRetries) {
12   if (localRetries <= RETRY_LIMIT) {
13     if(lock != 0) //Suscripción relajada
14       ABORT_XACT;
15     COMMIT_XACT;
16   } else lockRelease(globalLock);
17 }

       BEGIN_ESCAPE; //Espera escapada
       while(lock != 0) ;
       END_ESCAPE;
```

Fig. 4. Código de fallback con suscripción relajada y con espera escapada.

## VII. Evaluación

### A. Metodología

Hemos usado el simulador Simics [11] junto con el módulo GEMS [12] que implementa el simulador de memoria de un multiprocesador. Hemos simulado el BE-HTM de la sección III con las siguientes características: 16 núcleos escalares, con caché L1 privada de 32KB y 4 vías. Caché L2 unificada y compartida dividida en 16 bancos de 512KB y 8 vías. Directorio de vector de bits completo y red Garnet [18]. El número de hilos está limitado a 15 para que el sistema no haga migraciones.

Los benchmarks son de la suite de la universidad de Stanford STAMP [13]. La Tabla II muestra los parámetros que se han usado para cada benchmark y las principales características transaccionales como el número de transacciones, el porcentaje de tiempo dentro de transacción y la media del tamaño de los conjuntos de lectura y escritura de las transacciones, en bloques de caché.

### B. Resultados

En esta sección se analizan los resultados obtenidos con nuestros mecanismos de irrevocabilidad, comparándolos con los resultados del fallback homólogo al mecanismo de irrevocabilidad sin anticipación. También se analizan los resultados del fallback con suscripción relajada. La Fig. 5 muestra

TABLA II
BENCHMARKS: PARÁMETROS Y CARACTERÍSTICAS TRANSACCIONALES.

| **Benchmark** | **Entrada** | **# Xact** | **Tiempo en Xact** | $avg$\|**RS**\| | $avg$\|**WS**\| |
|---|---|---|---|---|---|
| Bayes | -v32 -r1024 -n2 -p20 -i2 -e2 -s1 | 654 | 94% | 87.64 | 48.91 |
| Genome | -g512 -s32 -n32768 | 19496 | 85% | 23.34 | 3.58 |
| Intruder | -a10 -l16 -n4096 -s1 | 54933 | 92% | 9.87 | 3.06 |
| Kmeans-high | -m15 -n15 -t0.05 -i random-n2048-d16-c16 | 8235 | 46% | 6.23 | 1.75 |
| Kmeans-low | -m40 -n40 -t0.05 -i random-n2048-d16-c16 | 10980 | 12% | 6.23 | 1.75 |
| Labyrinth | -i random-x32-y32-z3-n96 | 222 | 100% | 139.34 | 95.12 |
| SSCA2 | -s14 -i1.0 -u1.0 -l9 -p9 | 93721 | 13% | 3.00 | 2.00 |
| Vacation-high | -n4 -q60 -u90 -r16384 -t4096 | 4095 | 95% | 63.20 | 10.16 |
| Vacation-low | -n2 -q90 -u98 -r16384 -t4096 | 4095 | 93% | 48.17 | 8.60 |
| Yada | -a20 -i633.2 | 5447 | 100% | 62.45 | 38.21 |

los resultados de rendimiento sobre la aplicación secuencial para los sistemas mencionados . Todos los experimentos fueron realizados con el contador de reintentos configurado a cinco repeticiones, que es un valor frecuentemente usado en la bibliografía [4], [2].

Los resultados muestran tres benchmarks que no escalan: Bayes, Labyrinth y Yada. Estos benchmarks dan un rendimiento como el de la aplicación secuencial. Además, con un sólo hilo los resultados son incluso peores que el secuencial. Esta degradación se debe al número de reintentos antes de la irrevocabilidad. Las transacciones que desbordan la caché abortan cinco veces hasta que se entra en irrevocabilidad lo que hace que se rinda peor que el secuencial. El mecanismo de irrevocabilidad que anticipa el último aborto rinde un poco mejor que los demás debido a que hay un aborto menos.

El siguiente grupo de benchmarks escala adecuadamente y muestra poca diferencia entre los distintos métodos estudiados. Este grupo se compone de Kmeans-low, SSCA2 y Vacation-low. Estos benchmarks tienen transacciones pequeñas en media, Kmeans-low y SSCA2 pasan al rededor de un 13% del tiempo dentro de transacción (véase la Tabla II) y todos ellos muestran baja contención. La Tabla III muestra el número de transacciones irrevocables desglosadas por causa y podemos ver que el porcentaje de transacciones irrevocables está en torno al 5% para 15 hilos. Además, Kmeans y SSCA2 no muestran transacciones irrevocables debido a reemplazos de caché por lo que el sistema con anticipación no produce ningún beneficio. Por el contrario, Vacation sí muestra transacciones irrevocables debido a reemplazos, debido a su conjunto de lectura más grande, pero no son suficientes como para suponer una mejora de rendimiento. Sin embargo, con respecto al fallback relajado nuestras propuestas muestran una mejora notable para 15 hilos ya que al haber baja contención los abortos del fallback relajado penalizan el rendimiento para ningún benchmark.

El último grupo de benchmarks muestra una mejora en el rendimiento cuando se usa la irrevocabilidad en lugar del fallback software: Genome, Intruder, Kmeans-high y Vacation-high. Se puede observar que la irrevocabilidad relajada con y sin anticipación pueden dar un resultado muy parecido

TABLA III
NÚMERO DE TRANSACCIONES IRREVOCABLES POR CAUSA: DEBIDO A UN REEMPLAZO DE L1, L2, O CONFLICTO.

| **Bench** | **# Transacciones Irrevocables (L1/L2/Conflicto)** | |
|---|---|---|
| | **8 th's** | **15 th's** |
| Bayes | **211**(62/0/149) | **226**(34/0/191) |
| Genome | **1119**(922/0/197) | **1434**(1165/0/268) |
| Intruder | **6397**(42/0/6355) | **16619**(89/0/16530) |
| Kmeans-high | **1043**(0/0/1043) | **1986**(0/0/1986) |
| Kmeans-low | **291**(0/0/291) | **567**(0/0/567) |
| Labyrinth | **106**(29/0/77) | **117**(22/0/96) |
| SSCA2 | **283**(0/0/283) | **527**(0/0/527) |
| Vacation-high | **336**(268/0/68) | **428**(302/0/126) |
| Vacation-low | **103**(69/0/33) | **247**(181/3/63) |
| Yada | **1863**(581/0/1282) | **1883**(540/2/1342) |

entre sí y mejor que el fallback con espera escapada, homólogo de la irrevocabilidad sin anticipación, sobre todo para Intruder, Kmeans-high y Vacation-high con un 14%, un 25% y un 13% de mejora respectivamente. Esto ocurre cuando la causa principal de entrar en irrevocabilidad son los conflictos, y los reemplazos no tienen una posición dominante, como se puede ver en la Tabla III. Vacation-high, sin embargo, muestra un mayor número de transacciones irrevocables debidas a reemplazos, y el mecanismo de irrevocabilidad con anticipación realmente muestra mejores resultados que el que no tiene anticipación, pero no parece algo significativo. La Sección VII-C explica el efecto que la contención de cerrojos tiene en el fallback homólogo a nuestro mecanismo de irrevocabilidad, que hace que el fallback pierda rendimiento. Por último, Genome muestra una mejora de rendimiento notable de un 28% con respecto al fallback con espera escapada y al mecanismo de irrevocabilidad sin anticipación. En este caso, las transacciones irrevocables de Genome son principalmente debidas a fallos de capacidad, ya que tiene un numeroso grupo de transacciones con un conjunto de lectura grande. En cualquier caso, nuestros mecanismos de irrevocabilidad no penalizan el rendimiento.

### *C. Efecto de la Contención de Cerrojos*

Hemos visto como el mecanismo de irrevocabilidad relajada sin anticipación puede dar mejores resultados que su homólogo software, incluso cuando se im-

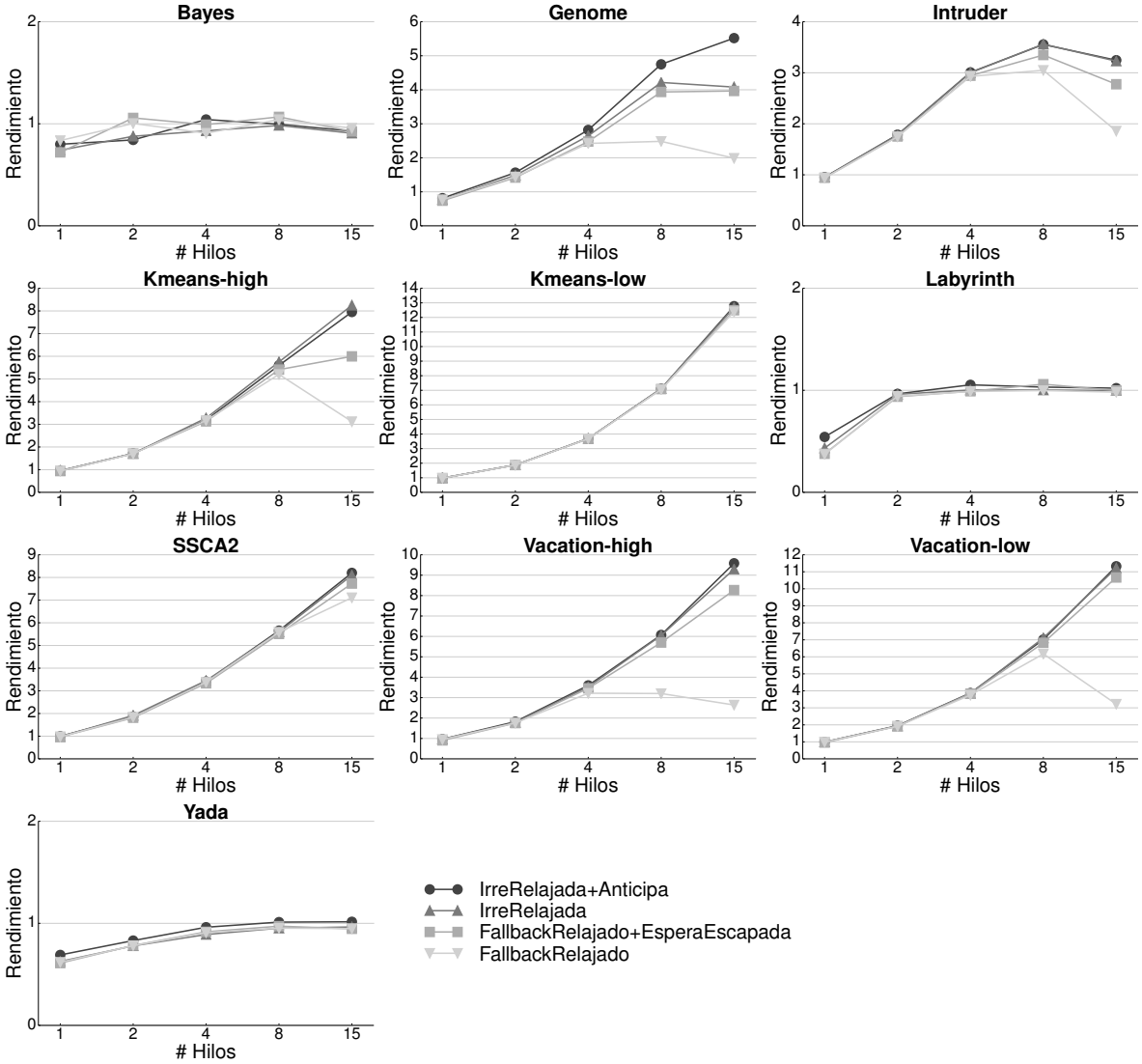


Fig. 5. Rendimiento sobre la aplicación secuencial de los mecanismos de irrevocabilidad relajada con y sin anticipación, junto con los fallbacks con suscripción relajada con y sin espera escapada.

plementan esencialmente de la misma manera. Sin embargo, el fallback utiliza cerrojos que introducen un efecto de contención del que carece nuestro sistema de irrevocabilidad, que utiliza el protocolo de comunicación basado en token.

Con el uso de cerrojos se puede dar la situación de que un grupo de transacciones esté esperando para finalizar sus transacciones debido a la transacción que está en el fallback. Están postergando su finalización con la espera escapada. Por otro lado podemos tener un grupo de transacciones que está esperando a que termine la transacción que está en el fallback, para entrar uno de ellos. Cuando la transacción que está en el fallback libera el cerrojo, se invalidan todas las copias privadas del mismo que residen en las cachés L1, y los dos grupos de transacciones esperando por distinto motivo piden acceso al cerrojo. Si la que obtiene el acceso primero es una de las que quería entrar al fallback, esta pondrá el cerrojo a uno y todas las demás transacciones encontrarán el cerrojo adquirido nuevamente. Por lo tanto, aquellas que estaban postergando su finalización, seguirán en la espera escapada, con el riesgo de poder ser abortadas.

Este efecto de contención debido al cerrojo no se da en nuestro protocolo de comunicación de irrevocabilidad ya que las transacciones que estaban postergando la finalización finalizan inmediatamente al recibir el mensaje de fin de irrevocabilidad. El efecto de contención es más probable a medida que se incrementa el número de hilos y en consecuencia la contención transaccional.

## VIII. Conclusiones

Recientemente podemos encontrar en el mercado multiprocesadores con extensiones transaccionales best-effort que necesitan de un fallback software para garantizar el progreso de las aplicaciones y superar las limitaciones hardware. En este artículo proponemos un nuevo tipo de irrevocabilidad que garantiza el progreso de las aplicaciones transaccionales y esconde las limitaciones del hardware al usuario a la vez que maximiza el paralelismo.

Se propone la irrevocabilidad relajada basada en el concepto de suscripción relajada de cerrojos en fall-

backs software. El mecanismo favorece la concurrencia postergando la finalización de las transacciones hasta la finalización de la transacción irrevocable y no necesita modificar el protocolo de coherencia. Proponemos un fallback homólogo basado en cerrojo con espera escapada que obtiene un rendimiento peor en algunos casos debido al efecto de la contención de cerrojos. Nuestro mecanismo de irrevocabilidad evita ese efecto con un protocolo de comunicación de irrevocabilidad basado en token.

También se propone un mecanismo de irrevocabilidad con anticipación de reemplazo que requiere ligeras modificaciones al protocolo de coherencia y que no puede ser implementado en software. Esta solución mejora el rendimiento de aquellas aplicaciones cuyas transacciones irrevocables sean causadas principalmente por reemplazos de caché.

La evaluación de las propuestas se lleva a cabo con el sistema simulado Simics/GEMS y con la suite de benchmarks STAMP, y obtenemos mejoras de rendimiento de hasta el 28% sobre las soluciones basadas en fallback.

## Agradecimientos

Este trabajo ha sido realizado gracias a los proyectos TIN2013-42253-P, del Ministerio de Economía y Competitividad del gobierno de España, y P12-TIC-1470, de la Junta de Andalucía.

## Referencias

[1] M. Herlihy and J.E.B. Moss, "Transactional memory: Architectural support for lock-free data structures," in *20th Ann. Int'l. Symp. on Computer Architecture (ISCA'93)*, 1993, pp. 289–300.

[2] Richard M Yoo, Christopher J Hughes, Konrad Lai, and Ravi Rajwar, "Performance Evaluation of Intel Transactional Synchronization Extensions for High-performance Computing," in *Int'l Conf. on High Performance Computing, Networking, Storage and Analysis (SC'13)*, 2013, pp. 19:1–19:11.

[3] Amy Wang, Matthew Gaudet, Peng Wu, José Nelson Amaral, Martin Ohmacht, Christopher Barton, Raul Silvera, and Maged Michael, "Evaluation of Blue Gene/Q hardware support for transactional memories," in *21st Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT'12)*, 2012, pp. 127–136.

[4] Christian Jacobi, Timothy Slegel, and Dan Greiner, "Transactional Memory Architecture and Implementation for IBM System z," in *45th Ann. Int'l. Symp. on Microarchitecture (MICRO'12)*, Dec. 2012, pp. 25–36.

[5] Allon Adir, Charles Meissner, Amir Nahir, Randall R. Pratt, Mike Schiffli, Brett St. Onge, Brian Thompto, Elena Tsanko, Avi Ziv, Dave Goodman, Daniel Hershcovich, Oz Hershkovitz, Bryan Hickerson, Karen Holtz, Wisam Kadry, Anatoly Koyfman, John Ludden, and Brett St Onge, "Verification of Transactional Memory in POWER8," in *51st Ann. Design Automation Conf. (DAC'14)*, 2014, pp. 1–6.

[6] Adam Welc, Saha Bratin, and Ali-Reza Adl-Tabatabai, "Irrevocable transactions and their applications," in *20th ACM Symp. on Parallelism in Algorithms and Architectures (SPAA'08)*, june 2008, pp. 285–296.

[7] Luke Dalessandro, François Carouge, Sean White, Yossi Lev, Mark Moir, Michael L. Scott, and Michael F. Spear, "Hybrid NOrec: A Case Study in the Effectiveness of Best Effort Hardware Transactional Memory," *ACM SIGPLAN Notices*, vol. 47, no. 4, pp. 39, jun 2012.

[8] Irina Calciu, Tatiana Shpeisman, Gilles Pokam, and Maurice Herlihy, "Improved Single Global Lock Fallback for Best-effort Hardware Transactional Memory," in *9th Workshop on Transactional Computing (TRANSACT'14)*, 2014.

[9] Milo M K Martin, Colin Blundell, and E Lewis, "Subtleties of Transactional Memory Atomicity Semantics," *IEEE Computer Architecture Letters*, vol. 5, no. 2, pp. 17, 2006.

[10] Michelle J Moravan, Jayaram Bobba, Kevin E Moore, Luke Yen, Mark D Hill, Ben Liblit, Michael M Swift, and David A Wood, "Supporting Nested Transactional Memory in logTM," in *12th Int'l. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS'06)*, 2006, pp. 359–370.

[11] P.S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, B. Werner, and B. Werner, "Simics: A full system simulation platform," *IEEE Computer*, vol. 35, no. 2, pp. 50–58, 2002.

[12] M.M.K. Martin, D.J. Sorin, B.M. Beckmann, M.R. Marty, M. Xu, A.R. Alameldeen, K.E. Moore, M.D. Hill, and D.A. Wood, "Multifacet's general execution-driven multiprocessor simulator GEMS toolset," *ACM SIGARCH Computer Architecture News*, vol. 33, no. 4, pp. 92–99, 2005.

[13] C.C. Minh, J. Chung, C. Kozyrakis, and K. Olukotun, "STAMP: Stanford Transactional Applications for Multi-Processing," in *IEEE Int'l Symp. on Workload Characterization (IISWC'08)*, 2008, pp. 35–46.

[14] Harold W Cain, Maged M Michael, Brad Frey, Cathy May, Derek Williams, and Hung Le, "Robust Architectural Support for Transactional Memory in the Power Architecture," in *40th Ann. Int'l. Symp. on Computer Architecture (ISCA'13)*, 2013, pp. 225–236.

[15] L. Hammond, V. Wong, M. Chen, B.D. Carlstrom, J.D. Davis, B. Hertzberg, M.K. Prabhu, H. Wijaya, C. Kozyrakis, and K. Olukotun, "Transactional memory coherence and consistency," in *31th Ann. Int'l. Symp. on Computer Architecture (ISCA'04)*, 2004, pp. 102–113.

[16] Colin Blundell, Joe Devietti, E Christopher Lewis, and Milo M K Martin, "Making the Fast Case Common and the Uncommon Case Simple in Unbounded Transactional Memory," in *34th Ann. Int'l. Symp. on Computer Architecture (ISCA'07)*, 2007, ISCA '07, pp. 24–34.

[17] Tim Harris, James Larus, and Ravi Rajwar, *Transactional Memory, 2nd edition*, Morgan & Claypool Publishers, 2010.

[18] N. Agarwal, T. Krishna, Li-Shiuan Peh, and N.K. Jha, "GARNET: A detailed on-chip network model inside a full-system simulator," in *IEEE Int'l. Symp. on Performance Analysis of Systems and Software (ISPASS'09)*, April 2009, pp. 33–42.